

Windows GDI

Article • 01/28/2023

Purpose

The Microsoft Windows graphics device interface (GDI) enables applications to use graphics and formatted text on both the video display and the printer. Windows-based applications do not access the graphics hardware directly. Instead, GDI interacts with device drivers on behalf of applications.

Where applicable

GDI can be used in all Windows-based applications.

Developer audience

This API is designed for use by C/C++ programmers. Familiarity with the Windows [message-driven architecture](#) is required.

Run-time requirements

For information on which operating systems are required to use a particular function, see the Requirements section of the documentation for the function.

In this section

- [Bitmaps](#)
- [Brushes](#)
- [Clipping](#)
- [Colors](#)
- [Coordinate Spaces and Transformations](#)
- [Device Contexts](#)
- [Filled Shapes](#)
- [Fonts and Text](#)
- [Lines and Curves](#)
- [Metafiles](#)
- [Multiple Display Monitors](#)
- [Painting and Drawing](#)

- [Paths](#)
- [Pens](#)
- [Printing and Print Spooler](#)
- [Rectangles](#)
- [Regions](#)

Related topics

[DirectX](#)

[GDI+](#)

[OpenGL](#)

[Windows Image Acquisition](#)

Security Considerations: GDI

Article • 01/07/2021

This topic provides information about security considerations related to GDI. This topic does not provide all you need to know about security issues. Instead, use it as a starting point and reference for this technology area.

GDI generally has few security concerns because it deals with display rather than input. However, here are a few issues that you should consider.

Bitmaps, metafiles, and fonts are complex structures that could become corrupted. It is good practice to try to ensure that these items are uncorrupted and from a trustworthy source.

An application can specify the security descriptor for some of the printing and spooling APIs. You should take care when setting the security descriptor.

Related topics

[Microsoft Security Central](#) ↗

[Security Developer Center](#) ↗

[Security TechCenter](#) ↗

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) ↗ | [Get help at Microsoft Q&A](#)

Bitmaps (Windows GDI)

Article • 01/07/2021

A *bitmap* is a graphical object used to create, manipulate (scale, scroll, rotate, and paint), and store images as files on a disk. This overview describes the bitmap classes and bitmap operations.

- [About Bitmaps](#)
 - [Using Bitmaps](#)
 - [Bitmap Reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Bitmaps

Article • 01/07/2021

A bitmap is one of the GDI objects that can be selected into a *device context* (DC).

[Device contexts](#) are structures that define a set of graphic objects and their associated attributes, and graphic modes that affect output. The table below describes the GDI objects that can be selected into a device context.

 [Expand table](#)

Graphic object	Description
Bitmaps	Creates, manipulates (scale, scroll, rotate, and paint), and stores images as files on a disk.
Brushes	Paints the interior of polygons, ellipses, and paths.
Fonts	Draws text on video displays and other output devices.
Logical Palette	A color palette created by an application and associated with a given device context.
Paths	One or more figures (or shapes) that are filled and/or outlined.
Pens	A graphics tool that an application uses to draw lines and curves.
Regions	A rectangle, polygon, or ellipse (or a combination of two or more of these shapes) that can be filled, painted, inverted, framed, and used to perform hit testing (testing for the cursor location).

From a developer's perspective, a bitmap consists of a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

A bitmap size is related to the type of image it contains. Bitmap images can be either monochrome or color. In an image, each pixel corresponds to one or more bits in a bitmap. Monochrome images have a ratio of 1 bit per pixel (bpp). Color imaging is more

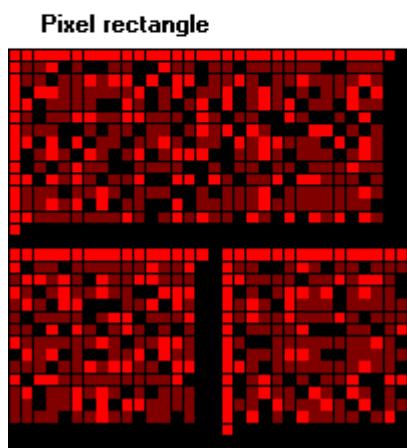
complex. The number of colors that can be displayed by a bitmap is equal to two raised to the number of bits per pixel. Thus, a 256-color bitmap requires 8 bpp ($2^8 = 256$).

Control Panel applications are examples of applications that use bitmaps. When you select a background (or wallpaper) for your desktop, you actually select a bitmap, which the system uses to paint the desktop background. The system creates the selected background pattern by repeatedly drawing a 32-by-32 pixel pattern on the desktop.

The following illustration shows the developer's perspective of the bitmap found in the file Redbrick.bmp. It shows a palette array, a 32-by-32 pixel rectangle, and the index array that maps colors from the palette to pixels in the rectangle.

Palette-index

```
row 0, scanline 31  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
row 1, scanline 30  00 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
row 2, scanline 29  11 11 01 19 11 01 10 10 09 09 01 09 11 11 01 90  
.  
. .  
row 31, scanline 0  99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 90
```



Color-palette

- 0 Black
- 1 Dark red
- 2 Dark green
- 3 Dark yellow
- 4 Dark blue
- 5 Dark magenta
- 6 Dark cyan
- 7 Dark gray
- 8 Light gray
- 9 Light red
- A Light green
- B Light yellow
- C Light blue
- D Light magenta
- E Light cyan
- F White

In the preceding example, the rectangle of pixels was created on a VGA display device using a palette of 16 colors. A 16-color palette requires 4-bit indexes; therefore, the array that maps palette colors to pixel colors is composed of 4-bit indexes as well. (For more information about logical color-palettes, see [Colors](#).)

ⓘ Note

In the above bitmap, the system maps indexes to pixels beginning with the bottom scan line of the rectangular region and ending with the top scan line. A *scan line* is a single row of adjacent pixels on a video display. For example, the first row of the array (row 0) corresponds to the bottom row of pixels, scan line 31. This is because the above bitmap is a bottom-up device-independent bitmap (DIB), a common

type of bitmap. In top-down DIBs and in device-dependent bitmaps (DDB), the system maps indexes to pixels beginning with the top scan line.

The following topics describe different areas of bitmaps.

- [Bitmap Classifications](#)
- [Bitmap Header Types](#)
- [JPEG and PNG Extensions for Specific Bitmap Functions and Structures](#)
- [Bitmaps, Device Contexts, and Drawing Surfaces](#)
- [Bitmap Creation](#)
- [Bitmap Rotation](#)
- [Bitmap Scaling](#)
- [Bitmaps as Brushes](#)
- [Bitmap Storage](#)
- [Bitmap Compression](#)
- [Alpha Blending](#)
- [Smooth Shading](#)
- [ICM-Enabled Bitmap Functions](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Classifications

Article • 01/07/2021

There are two classes of bitmaps:

- **Device-independent bitmaps** (DIB). The DIB file format was designed to ensure that bitmapped graphics created using one application can be loaded and displayed in another application, retaining the same appearance as the original.
- **Device-dependent bitmaps** (DDB), also known as GDI bitmaps, were the only bitmaps available in early versions of 16-bit Microsoft Windows (prior to version 3.0). However, as display technology improved and as the variety of available display devices increased, certain inherent problems surfaced which could only be solved using DIBs. For example, there was no method of storing (or retrieving) the resolution of the display type on which a bitmap was created, so a drawing application could not quickly determine whether a bitmap was suitable for the type of video display device on which the application was running.

Despite these problems, DDBs (also known as compatible bitmaps) are still useful for better GDI performance and for other situations.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device-Independent Bitmaps

Article • 11/19/2022

A device-independent bitmap (DIB) contains a *color table*. A color table describes how pixel values correspond to *RGB* color values, which describe colors that are produced by emitting light. Thus, a DIB can achieve the proper color scheme on any device. A DIB contains the following color and dimension information:

- The color format of the device on which the rectangular image was created.
- The resolution of the device on which the rectangular image was created.
- The palette for the device on which the image was created.
- An array of bits that maps red, green, blue (**RGB**) triplets to pixels in the rectangular image.
- A data-compression identifier that indicates the data compression scheme (if any) used to reduce the size of the array of bits.

The color and dimension information is stored in a **BITMAPINFO** structure, which consists of a **BITMAPINFOHEADER** structure followed by two or more **RGBQUAD** structures. The **BITMAPINFOHEADER** structure specifies the dimensions of the pixel rectangle, describes the device's color technology, and identifies the compression schemes used to reduce the size of the bitmap. The **RGBQUAD** structures identify the colors that appear in the pixel rectangle.

There are two varieties of DIBs:

- A bottom-up DIB, in which the origin lies at the lower-left corner.
- A top-down DIB, in which the origin lies at the upper-left corner.

If the height of a DIB, as indicated by the **Height** member of the bitmap information header structure, is a positive value, it is a bottom-up DIB; if the height is a negative value, it is a top-down DIB. Top-down DIBs cannot be compressed.

The color format is specified in terms of a count of color planes and color bits. The count of color planes is always 1; the count of color bits is 1 for monochrome bitmaps, 4 for VGA bitmaps, and 8, 16, 24, or 32 for bitmaps on other color devices. An application retrieves the number of color bits that a particular display (or printer) uses by calling the **GetDeviceCaps** function, specifying **BITSPixel** as the second argument.

The resolution of a display device is specified in pixels-per-meter. An application can retrieve the horizontal resolution for a video display, or printer, by following this three-step process.

1. Call the **GetDeviceCaps** function, specifying **HORZRES** as the second argument.

2. Call [GetDeviceCaps](#) a second time, specifying HORZSIZE as the second argument.
3. Divide the first return value by the second return value.

The application can retrieve the vertical resolution by using the same three-step process with different parameters: VERTRES in place of HORZRES, and VERTSIZE in place of HORZSIZE.

The palette is represented by an array of [RGBQUAD](#) structures that specify the red, green, and blue intensity components for each color in a display device's color palette. Each color index in the palette array maps to a specific pixel in the rectangular region associated with the bitmap. The size of this array, in bits, is equivalent to the width of the rectangle, in pixels, multiplied by the height of the rectangle, in pixels, multiplied by the count of color bits for the device. An application can retrieve the size of the device's palette by calling the [GetDeviceCaps](#) function, specifying NUMCOLORS as the second argument.

Windows supports the compression of the palette array for 8-bpp and 4-bpp bottom-up DIBs. These arrays can be compressed by using the run-length encoding (RLE) scheme. The RLE scheme uses 2-byte values, the first byte specifying the number of consecutive pixels that use a color index and the second byte specifying the index. For more information about bitmap compression, see the description of the [BITMAPINFOHEADER](#), [BITMAPFILEHEADER](#), [BITMAPV4HEADER](#), and [BITMAPV5HEADER](#) structures.

An application can create a DIB from a DDB by initializing the required structures and calling the [GetDIBits](#) function. To determine whether a device supports this function, call the [GetDeviceCaps](#) function, specifying RC_DI_BITMAP as the RASTERCAPS flag.

An application that needs to copy a bitmap can use [TransparentBlt](#) to copy all pixels in a source bitmap to a destination bitmap except those pixels that match the transparent color.

An application can use a DIB to set pixels on the display device by calling the [SetDIBitsToDevice](#) or the [StretchDIBits](#) function. To determine whether a device supports the [SetDIBitsToDevice](#) function, call the [GetDeviceCaps](#) function, specifying RC_DIBTODEV as the RASTERCAPS flag. Specify RC_STRETCHDIB as the RASTERCAPS flag to determine if the device supports [StretchDIBits](#).

An application that simply needs to display a pre-existing DIB can use the [SetDIBitsToDevice](#) function. For example, a spreadsheet application can open existing charts and display them in a window by using the [SetDIBitsToDevice](#) function. To repeatedly redraw a bitmap in a window, however, the application should use the [BitBlt](#) function. For example, a multimedia application that combines animated graphics with

sound would benefit from calling the **BitBlt** function because it executes faster than **SetDIBitsToDevice**.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device-Dependent Bitmaps

Article • 01/07/2021

Device-dependent bitmaps (DDBs) are described by using a single structure, the [BITMAP](#) structure. The members of this structure specify the width and height of a rectangular region, in pixels; the width of the array that maps entries from the device palette to pixels; and the device's color format, in terms of color planes and bits per pixel. An application can retrieve the color format of a device by calling the [GetDeviceCaps](#) function and specifying the appropriate constants. Note that a DDB does not contain color values; instead, the colors are in a device-dependent format. For more information, see [Color in Bitmaps](#). Because each device can have its own set of colors, a DDB created for one device may not display well on a different device.

To use a DDB in a device context, it must have the color organization of that device context. Thus, a DDB is often called a *compatible bitmap* and it usually has better GDI performance than a DIB. For example, to create a bitmap for video memory, it is best to use a compatible bitmap with the same color format as the primary display. Once in video memory, rendering to the bitmap and displaying it to the screen are significantly faster than from a system memory surface or directly from a DIB.

In addition to enabling better GDI performance, compatible bitmaps are used to capture images (see [Capturing an Image](#)) and to create bitmaps at run time for menus see "Creating the Bitmap" in (see [Using Menus](#)).

To transfer a bitmap between devices with different color organization, use [GetDIBits](#) to convert the compatible bitmap to a DIB and call [SetDIBits](#) or [StretchDIBits](#) to display the DIB to the second device.

There are two types of DDBs: discardable and nondiscardable. A discardable DDB is a bitmap that the system discards if the bitmap is not selected into a DC and if system memory is low. The [CreateDiscardableBitmap](#) function creates discardable bitmaps. The [CreateBitmap](#), [CreateCompatibleBitmap](#), and [CreateBitmapIndirect](#) functions create nondiscardable bitmaps.

An application can create a DDB from a DIB by initializing the required structures and calling the [CreateDIBitmap](#) function. Specifying CBM_INIT in the call to [CreateDIBitmap](#) 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. To determine whether a device supports the [SetDIBits](#) function, call the [GetDeviceCaps](#) function, specifying RC_DI_BITMAP as the RASTERCAPS flag.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Header Types

Article • 11/19/2022

The bitmap has four basic header types:

- [BITMAPCOREHEADER](#)
- [BITMAPINFOHEADER](#)
- [BITMAPV4HEADER](#)
- [BITMAPV5HEADER](#)

The four types of bitmap headers are differentiated by the **Size** member, which is the first **DWORD** in each of the structures.

The [BITMAPV5HEADER](#) structure is an extended [BITMAPV4HEADER](#) structure, which is an extended [BITMAPINFOHEADER](#) structure. However, the [BITMAPINFOHEADER](#) and [BITMAPCOREHEADER](#) have only the **Size** member in common with other bitmap header structures.

The [BITMAPCOREHEADER](#) and [BITMAPV4HEADER](#) formats have been superseded by [BITMAPINFOHEADER](#) and [BITMAPV5HEADER](#) formats, respectively. The [BITMAPCOREHEADER](#) and [BITMAPV4HEADER](#) formats are presented for completeness and backward compatibility.

The format for a DIB is the following (for more information, see [Bitmap Storage](#)):

- a [BITMAPFILEHEADER](#) structure
- either a [BITMAPCOREHEADER](#), a [BITMAPINFOHEADER](#), a [BITMAPV4HEADER](#), or a [BITMAPV5HEADER](#) structure.
- an optional color table, which is either a set of [RGBQUAD](#) structures or a set of [RGBTRIPLE](#) structures.
- the bitmap data
- optional Profile data

A color table describes how pixel values correspond to RGB color values. RGB is a model for describing colors that are produced by emitting light.

Profile data refers to either the profile file name (linked profile) or the actual profile bits (embedded profile). The file format places the profile data at the end of the file. The profile data is placed just after the color table (if present). However, if the function receives a packed DIB, the profile data comes after the bitmap bits, like in the file format.

Profile data will only exist for [BITMAPV5HEADER](#) structures where **bV5CSType** is **PROFILE_LINKED** or **PROFILE_EMBEDDED**. For functions that receive packed DIBs, the profile data comes after the bitmap data.

A palettized device is any device that uses palettes to assign colors. The classic example of a palettized device is a display running in 8 bit color depth (that is, 256 colors). The display in this mode uses a small color table to assign colors to a bitmap. The colors in a bitmap are assigned to the closest color in the palette that the device is using. The palettized device does not create an optimal palette for displaying the bitmap; it simply uses whatever is in the current palette. Applications are responsible for creating a palette and selecting it into the system. In general, 16-, 24-, and 32-bits-per-pixel (bpp) bitmaps do not contain color tables (a.k.a. optimal palettes for the bitmap); the application is responsible for generating an optimal palette in this case. However, 16-, 24-, and 32-bpp bitmaps can contain such optimal color tables for displaying on palettized devices; in this case the application just needs to create a palette based off the color table present in the bitmap file.

Bitmaps that are 1, 4, or 8 bpp must have a color table with a maximum size based on the bpp. The maximum size for 1, 4, and 8 bpp bitmaps is 2 to the power of the bpp. Thus, a 1 bpp bitmap has a maximum of two colors, the 4 bpp bitmap has a maximum of 16 colors, and the 8 bpp bitmap has a maximum of 256 colors.

Bitmaps that are 16-, 24-, or 32-bpp do not require color tables, but may have them to specify colors for palettized devices. If a color table is present for 16-, 24-, or 32-bpp bitmap, the **biClrUsed** member specifies the size of the color table and the color table must have that many colors in it. If **biClrUsed** is zero, there is no color table.

The red, green, and blue bitfield masks for **BI_BITFIELD** bitmaps immediately follow the [BITMAPINFOHEADER](#), [BITMAPV4HEADER](#), and [BITMAPV5HEADER](#) structures. The **BITMAPV4HEADER** and **BITMAPV5HEADER** structures contain additional members for red, green, and blue masks as follows.

[+] Expand table

Member	Meaning
RedMask	Color mask that specifies the red component of each pixel, valid only if the Compression member is set to BI_BITFIELDS .
GreenMask	Color mask that specifies the green component of each pixel, valid only if the Compression member is set to BI_BITFIELDS .
BlueMask	Color mask that specifies the blue component of each pixel, valid only if the Compression member is set to BI_BITFIELDS .

When the **biCompression** member of **BITMAPINFOHEADER** is set to **BI_BITFIELDS** and the function receives an argument of type **LPBITMAPINFO**, the color masks will immediately follow the header. The color table, if present, will follow the color masks. **BITMAPCOREHEADER** bitmaps do not support color masks.

By default, bitmap data is bottom-up in its format. Bottom-up means that the first scan line in the bitmap data is the last scan line to be displayed. For example, the 0th pixel of the 0th scan line of the bitmap data of a 10-pixel by 10-pixel bitmap will be the 0th pixel of the 9th scan line of the displayed or printed image. Run-length encoded (RLE) format bitmaps and **BITMAPCOREHEADER** bitmaps cannot be top-down bitmaps. The scan lines are **DWORD** aligned, except for RLE-compressed bitmaps. They must be padded for scan line widths, in bytes, that are not evenly divisible by four, except for RLE compressed bitmaps. For example, a 10- by 10-pixel 24-bpp bitmap will have two padding bytes at the end of each scan line.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

JPEG and PNG Extensions for Specific Bitmap Functions and Structures

Article • 11/19/2022

On certain versions of Microsoft Windows, the [StretchDIBits](#) and [SetDIBitsToDevice](#) functions allow JPEG and PNG images to be passed as the source image to printer devices. This extension is not intended as a means to supply general JPEG and PNG decompression to applications, but rather to allow applications to send JPEG- and PNG-compressed images directly to printers having hardware support for JPEG and PNG images.

The [BITMAPINFOHEADER](#), [BITMAPV4HEADER](#) and [BITMAPV5HEADER](#) structures are extended to allow specification of **biCompression** values indicating that the bitmap data is a JPEG or PNG image. These compression values are only valid for [SetDIBitsToDevice](#) and [StretchDIBits](#) when the *hdc* parameter specifies a printer device. To support metafile spooling of the printer, the application should not rely on the return value to determine whether the device supports the JPEG or PNG file. The application must issue **QUERYECSUPPORT** with the corresponding escape before calling [SetDIBitsToDevice](#) and [StretchDIBits](#). If the validation escape fails, the application must then fall back on its own JPEG or PNG support to decompress the image into a bitmap.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmaps, Device Contexts, and Drawing Surfaces

Article • 01/07/2021

A *device context* (DC) is a data structure defining the graphics objects, their associated attributes, and the graphics modes affecting output on a device. To create a DC, call the [CreateDC](#) function; to retrieve a DC, call the [GetDC](#) function.

Before returning a handle that identifies that DC, the system selects a drawing surface into the DC. If the application called the [CreateDC](#) function to create a device context for a VGA display, the dimensions of this drawing surface are 640-by-480 pixels. If the application called the [GetDC](#) function, the dimensions reflect the size of the client area.

Before an application can begin drawing, it must select a bitmap with the appropriate width and height into the DC by calling the [SelectObject](#) function. When an application passes the handle to the DC to one of the graphics device interface (GDI) drawing functions, the requested output appears on the drawing surface selected into the DC.

For more information, see [Memory Device Contexts](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Creation

Article • 01/07/2021

To create a bitmap, use the [CreateBitmap](#), [CreateBitmapIndirect](#), or [CreateCompatibleBitmap](#) function, [CreateDIBitmap](#), and [CreateDiscardableBitmap](#).

These functions allow you to specify the width and height, in pixels, of the bitmap. The [CreateBitmap](#) and [CreateBitmapIndirect](#) function also allow you to specify the number of color planes and the number of bits required to identify the color. On the other hand, the [CreateCompatibleBitmap](#) and [CreateDiscardableBitmap](#) functions use a specified device context to obtain the number of color planes and the number of bits required to identify the color.

The [CreateDIBitmap](#) function creates a device-dependent bitmap from a device-independent bitmap. It contains a color table that describes how pixel values correspond to RGB color values. For more information, see [Device-Dependent Bitmaps](#) and [Device-Independent Bitmaps](#).

After the bitmap has been created, you cannot change its size, number of color planes, or number of bits required to identify the color.

When you no longer need a bitmap, call the [DeleteObject](#) function to delete it.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Rotation

Article • 01/07/2021

To copy a bitmap into a parallelogram; use the [PIgBlt](#) function, which performs a bit-block transfer from a rectangle in a source device context into a parallelogram in a destination device context. To rotate the bitmap, an application must provide the coordinates, in world units, to be used for the corners of the parallelogram. (For more information about rotation and world units, see [Coordinate Spaces and Transformations](#).)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Scaling

Article • 01/07/2021

The [StretchBlt](#) function scales a bitmap by performing a bit-block transfer from a rectangle in a source device context into a rectangle in a destination device context. However, unlike the [BitBlt](#) function, which duplicates the source rectangle dimensions in the destination rectangle, [StretchBlt](#) allows an application to specify the dimensions of both the source and the destination rectangles. When the destination bitmap is smaller than the source bitmap, the system combines rows or columns of color data (or both) in the bitmap before rendering the corresponding image on the display device. The system combines the color data according to the specified stretch mode, which the application defines by calling the [SetStretchBltMode](#) function. When the destination bitmap is larger than the source bitmap, the system scales or magnifies each pixel in the resultant image accordingly.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Bitmaps as Brushes

Article • 01/07/2021

A number of functions use the brush currently selected into a device context to perform bitmap operations. For example, the [PatBlt](#) function replicates the brush in a rectangular region within a window, and the [FloodFill](#) function replicates the brush inside an area in a window bounded by the specified color (unlike [PatBlt](#), [FloodFill](#) does fill nonrectangular shapes).

The [FloodFill](#) function replicates the brush within a region bounded by a specified color. However, unlike the [PatBlt](#) function, [FloodFill](#) does not combine the color data for the brush with the color data for the pixels on the display; it simply sets the color of all pixels within the enclosed region on the display to the color of the brush that is currently selected into the device context.

Feedback

Was this page helpful?



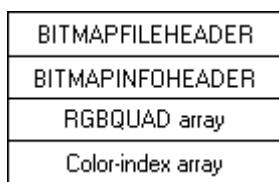
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Storage

Article • 11/19/2022

Bitmaps should be saved in a file that uses the established bitmap file format and assigned a name with the three-character .bmp extension. The established bitmap file format consists of a [BITMAPFILEHEADER](#) structure followed by a [BITMAPINFOHEADER](#), [BITMAPV4HEADER](#), or [BITMAPV5HEADER](#) structure. An array of [RGBQUAD](#) structures (also called a color table) follows the bitmap information header structure. The color table is followed by a second array of indexes into the color table (the actual bitmap data).

The bitmap file format is shown in the following illustration.



The members of the [BITMAPFILEHEADER](#) structure identify the file; specify the size of the file, in bytes; and specify the offset from the first byte in the header to the first byte of bitmap data. The members of the [BITMAPINFOHEADER](#), [BITMAPV4HEADER](#), or [BITMAPV5HEADER](#) structure specify the width and height of the bitmap, in pixels; the color format (count of color planes and color bits-per-pixel) of the display device on which the bitmap was created; whether the bitmap data was compressed before storage and the type of compression used; the number of bytes of bitmap data; the resolution of the display device on which the bitmap was created; and the number of colors represented in the data. The [RGBQUAD](#) structures specify the RGB intensity values for each of the colors in the device's palette.

The color-index array associates a color, in the form of an index to an [RGBQUAD](#) structure, with each pixel in a bitmap. Thus, the number of bits in the color-index array equals the number of pixels times the number of bits needed to index the [RGBQUAD](#) structures. For example, an 8x8 black-and-white bitmap has a color-index array of $8 * 8 * 1 = 64$ bits, because one bit is needed to index two colors. The Redbrick.bmp, mentioned in [About Bitmaps](#), is a 32x32 bitmap with 16 colors; its color-index array is $32 * 32 * 4 = 4096$ bits because four bits index 16 colors.

To create a color-index array for a top-down bitmap, start at the top line in the bitmap. The index of the [RGBQUAD](#) for the color of the left-most pixel is the first n bits in the color-index array (where n is the number of bits needed to indicate all of the [RGBQUAD](#) structures). The color of the next pixel to the right is the next n bits in the array, and so forth. After you reach the right-most pixel in the line, continue with the left-most pixel in

the line below. Continue until you finish with the entire bitmap. If it is a bottom-up bitmap, start at the bottom line of the bitmap instead of the top line, still going from left to right, and continue to the top line of the bitmap.

The following hexadecimal output shows the contents of the file Redbrick.bmp.

C++
0000 42 4d 76 02 00 00 00 00 00 00 00 00 76 00 00 00 28 00
0010 00 00 20 00 00 00 20 00 00 00 01 00 04 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80
0040 00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 00 80 80
0050 00 00 80 80 80 00 c0 c0 c0 00 00 00 ff 00 00 ff
0060 00 00 00 ff ff 00 ff 00 00 ff 00 ff 00 ff 00 ff ff
0070 00 00 ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 00
0090 00 00 00 00 00 00 11 11 01 19 11 01 10 10 09 09
00a0 01 09 11 11 01 90 11 01 19 09 09 91 11 10 09 11
00b0 09 11 19 10 90 11 19 01 19 19 10 10 11 10 09 01
00c0 91 10 91 09 10 10 90 99 11 11 11 19 00 09 01
00d0 91 01 01 19 00 99 11 10 11 91 99 11 09 90 09 91
00e0 01 11 11 11 91 10 09 19 01 00 11 90 91 10 09 01
00f0 11 99 10 01 11 11 91 11 11 19 10 11 99 10 09 10
0100 01 11 11 11 19 10 11 09 09 10 19 10 10 10 09 01
0110 11 19 00 01 10 19 10 11 11 01 99 01 11 90 09 19
0120 11 91 11 91 01 11 19 10 99 00 01 19 09 10 09 19
0130 10 91 11 01 11 11 91 01 91 19 11 00 99 90 09 01
0140 01 99 19 01 91 10 19 91 91 09 11 99 11 10 09 91
0150 11 10 11 91 99 10 90 11 01 11 11 19 11 90 09 11
0160 00 19 10 11 01 11 99 99 99 99 99 99 99 99 09 99
0170 99 99 99 99 99 99 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 90 00 00 00 00 00 00 00 00 00 00
0190 00 00 00 00 00 00 99 11 11 11 19 10 19 19 11 09
01a0 10 90 91 90 91 00 91 19 19 09 01 10 09 01 11 11
01b0 91 11 11 11 10 00 91 11 01 19 10 11 10 01 01 11
01c0 90 11 11 11 91 00 99 09 19 10 11 90 09 90 91 01
01d0 19 09 91 11 01 00 90 10 19 11 00 11 11 00 10 11
01e0 01 10 11 19 11 00 90 19 10 91 01 90 19 99 00 11
01f0 91 01 11 01 91 00 99 09 09 01 10 11 91 01 10 91
0200 99 11 10 90 91 00 91 11 00 10 11 01 10 19 19 09
0210 10 00 99 01 01 00 91 01 19 91 19 91 11 09 10 11
0220 00 91 00 10 90 00 99 01 11 10 09 10 10 19 09 01
0230 91 90 11 09 11 00 90 99 11 11 11 90 19 01 19 01
0240 91 01 01 19 09 00 91 10 11 91 99 09 09 90 11 91
0250 01 19 11 11 91 00 91 19 01 00 11 00 91 10 11 01
0260 11 11 10 01 11 00 99 99 99 99 99 99 99 99 99 99
0270 99 99 99 99 99 99 90

The following table shows the data bytes associated with the structures in a bitmap file.

Expand table

Structure	Corresponding bytes
BITMAPFILEHEADER	0x00 0x0D
BITMAPINFOHEADER	0x0E 0x36
RGBQUAD array	0x37 0x75
Color-index array	0x76 0x275

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Bitmap Compression

Article • 08/27/2021

Windows supports formats for compressing bitmaps that define their colors with 8 or 4 bits-per-pixel. Compression reduces the disk and memory storage required for the bitmap.

When the **Compression** member of the bitmap information header structure is BI_RLE8, a run-length encoding (RLE) format is used to compress an 8-bit bitmap. This format can be compressed in encoded or absolute modes. Both modes can occur anywhere in the same bitmap:

- *Encoded mode* consists of two bytes: the first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. In addition, the first byte of the pair can be set to zero to indicate an escape character that denotes the end of a line, the end of a bitmap, or a delta, depending on the value of the second byte. The interpretation of the escape depends on the value of the second byte of the pair, which can be one of the following values.

[] Expand table

Value	Meaning
0	End of line.
1	End of bitmap.
2	Delta. The 2 bytes following the escape contain unsigned values indicating the offset to the right and up of the next pixel from the current position.

- In *absolute mode*, the first byte is zero and the second byte is a value in the range 03H through FFH. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. When the second byte is two or less, the escape has the same meaning as encoded mode. In absolute mode, each run must be zero-padded to end on a 16-bit word boundary.

The following example shows the hexadecimal values of an 8-bit compressed bitmap:

C++

```
[03 04] [05 06] [00 03 45 56 67 00] [02 78] [00 02 05 01]
```

```
[02 78] [00 00] [09 1E] [00 01]
```

The bitmap expands as follows (two-digit values represent a color index for a single pixel):

C++

```
04 04 04  
06 06 06 06 06  
45 56 67  
78 78  
move current position 5 right and 1 up  
78 78  
end of line  
1E 1E 1E 1E 1E 1E 1E 1E  
end of RLE bitmap
```

When the **Compression** member is BI_RLE4, the bitmap is compressed by using a run-length encoding format for a 4-bit bitmap, which also uses encoded and absolute modes:

- In encoded mode, the first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte. The second byte contains two color indexes, one in its high-order 4 bits and one in its low-order 4 bits. The first of the pixels is drawn using the color specified by the high-order 4 bits, the second is drawn using the color in the low-order 4 bits, the third is drawn using the color in the high-order 4 bits, and so on, until all the pixels specified by the first byte have been drawn.
- In absolute mode, the first byte is zero. The second byte contains the number of color indexes that follow. Subsequent bytes contain color indexes in their high- and low-order 4 bits, one color index for each pixel. In absolute mode, each run must be aligned on a word boundary. The end-of-line, end-of-bitmap, and delta escapes described for BI_RLE8 also apply to BI_RLE4 compression.

The following example shows the hexadecimal values of a 4-bit compressed bitmap:

C++

```
03 04 05 06 00 06 45 56 67 00 04 78 00 02 05 01  
04 78 00 00 09 1E 00 01
```

The bitmap expands as follows (single-digit values represent a color index for a single pixel):

C++

```
0 4 0
0 6 0 6 0
4 5 5 6 6 7
7 8 7 8
move current position 5 right and 1 up
7 8 7 8
end of line
1 E 1 E 1 E 1 E 1
end of RLE bitmap
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Alpha Blending (Windows GDI)

Article • 06/30/2021

Alpha blending is used to display an alpha bitmap, which is a bitmap that has transparent or semi-transparent pixels. In addition to a red, green, and blue color channel, each pixel in an alpha bitmap has a transparency component known as its *alpha channel*. The alpha channel typically contains as many bits as a color channel. For example, an 8-bit alpha channel can represent 256 levels of transparency, from 0 (the entire bitmap is transparent) to 255 (the entire bitmap is opaque).

Alpha blending mechanisms are invoked by calling [AlphaBlend](#), which references the [BLENDFUNCTION](#) structure.

Alpha values per pixel are only supported for 32-bpp BI_RGB. This formula is defined as:

C++

```
typedef struct {
    BYTE   Blue;
    BYTE   Green;
    BYTE   Red;
    BYTE   Alpha;
};
```

This is represented in memory as shown in the following table.

31:24

23:16

15:08

07:00

Alpha

Red

Green

Blue

Bitmaps may also be displayed with a transparency factor applied to the entire bitmap. Any bitmap format can be displayed with a global constant alpha value by setting

SourceConstantAlpha in the **BLENDFUNCTION** structure. The global constant alpha value has 256 levels of transparency, from 0 (entire bitmap is completely transparent) to 255 (entire bitmap is completely opaque). The global constant alpha value is combined with the per-pixel alpha value.

For an example, see [Alpha Blending a Bitmap](#).

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Smooth Shading

Article • 01/07/2021

Smooth shading is a method of shading a region with a color gradient. Including color information, along with the bounds of drawing primitive, specifies the color gradient. GDI linearly interpolates the color of the inside of the primitive passed on the color endpoints. Color and vertex information is included with position information in the [TRIVERTEX](#) structure.

Use the [GradientFill](#) function to fill a triangle or rectangle structure. To fill a triangle with smooth shading, call [GradientFill](#) with the three triangle endpoints. To fill a rectangle with smooth shading, call [GradientFill](#) with the upper-left and lower-right rectangle coordinates. [GradientFill](#) references the [TRIVERTEX](#), [GRADIENT_RECT](#), and [GRADIENT_TRIANGLE](#) structures.

For an example, see [Drawing a Shaded Triangle](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ICM-Enabled Bitmap Functions

Article • 01/07/2021

Microsoft Image Color Management (ICM) ensures that a color image, graphic object, or text object is rendered as closely as possible to its original intent on any device, despite differences in imaging technologies and color capabilities between devices. Whether you are scanning an image or other graphic on a color scanner, downloading it over the Internet, viewing or editing it onscreen, or printing it on paper, film, or other media, ICM 2.0 helps you keep colors consistent and accurate. For more information about ICM, see [Windows Color System](#).

There are various functions in the graphics device interface (GDI) that use or operate on color data. The following bitmap functions are enabled for use with ICM:

- [BitBlt](#)
- [CreateDIBitmap](#)
- [CreateDIBSection](#)
- [MaskBlt](#)
- [SetDIBColorTable](#)
- [StretchBlt](#)
- [SetDIBits](#)
- [SetDIBitsToDevice](#)
- [StretchDIBits](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Using Bitmaps

Article • 01/07/2021

- [Capturing an Image](#)
- [Scaling an Image](#)
- [Storing an Image](#)
- [Alpha Blending a Bitmap](#)
- [Drawing a Shaded Rectangle](#)
- [Drawing a Shaded Triangle](#)
- [Testing a Printer for JPEG or PNG Support](#)
- [Sizing a JPEG or PNG Image](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Capturing an Image

Article • 01/07/2021

You can use a bitmap to capture an image, and you can store the captured image in memory, display it at a different location in your application's window, or display it in another window.

In some cases, you may want your application to capture images and store them only temporarily. For example, when you scale or zoom a picture created in a drawing application, the application must temporarily save the normal view of the image and display the zoomed view. Later, when the user selects the normal view, the application must replace the zoomed image with a copy of the normal view that it temporarily saved.

To store an image temporarily, your application must call [CreateCompatibleDC](#) to create a DC that is compatible with the current window DC. After you create a compatible DC, you create a bitmap with the appropriate dimensions by calling the [CreateCompatibleBitmap](#) function and then select it into this device context by calling the [SelectObject](#) function.

After the compatible device context is created and the appropriate bitmap has been selected into it, you can capture the image. The [BitBlt](#) function captures images. This function performs a bit block transfer that is, it copies data from a source bitmap into a destination bitmap. However, the two arguments to this function are not bitmap handles. Instead, [BitBlt](#) receives handles that identify two device contexts and copies the bitmap data from a bitmap selected into the source DC into a bitmap selected into the target DC. In this case, the target DC is the compatible DC, so when [BitBlt](#) completes the transfer, the image has been stored in memory. To redisplay the image, call [BitBlt](#) a second time, specifying the compatible DC as the source DC and a window (or printer) DC as the target DC.

Code example

This section contains a code example that captures an image of the entire desktop, scales it down to the current window size, and then saves it to a file (as well as displaying it in the client area).

To try out the code example, begin by creating a new project in Visual Studio based on the [Windows Desktop Application](#) project template. It's important to name the new project `GDI_CapturingAnImage` so that the code listing below will compile (for example, it

includes `GDI_CapturingAnImage.h`, which will exist in your new project if you name it as suggested).

Open the `GDI_CapturingAnImage.cpp` source code file in your new project, and replace its contents with the listing below. Then build and run. Each time you resize the window, you'll see the captured screenshot displayed in the client area.

C++

```
// GDI_CapturingAnImage.cpp : Defines the entry point for the application.
//


#include "framework.h"
#include "GDI_CapturingAnImage.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];             // the main window class
name

// Forward declarations of functions included in this code module:
ATOM           MyRegisterClass(HINSTANCE hInstance);
BOOL          InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                     _In_opt_ HINSTANCE hPrevInstance,
                     _In_ LPWSTR    lpCmdLine,
                     _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_GDICAPTURINGANIMAGE, szWindowClass,
MAX_LOADSTRING);
MyRegisterClass(hInstance);

    // Perform application initialization:
if (!InitInstance(hInstance, nCmdShow))
{
    return FALSE;
}

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_GDICAPTURINGANIMAGE));
```

```

MSG msg;

// Main message loop:
while (GetMessage(&msg, nullptr, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

return (int)msg.wParam;
}

// 
// FUNCTION: MyRegisterClass()
//
// PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW 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_GDICAPTURINGANIMATION));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_GDICAPTURINGANIMATION);
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

// 
// FUNCTION: InitInstance(HINSTANCE, int)
//
// PURPOSE: Saves instance handle and creates main window
//
// COMMENTS:
//
//      In this function, we save the instance handle in a global variable
and
//      create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

```

```
{  
    hInst = hInstance; // Store instance handle in our global variable  
  
    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,  
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance,  
        nullptr);  
  
    if (!hWnd)  
    {  
        return FALSE;  
    }  
  
    ShowWindow(hWnd, nCmdShow);  
    UpdateWindow(hWnd);  
  
    return TRUE;  
}  
  
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF  
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A  
// PARTICULAR PURPOSE.  
//  
// Copyright (c) Microsoft Corporation. All rights reserved  
  
//  
//     FUNCTION: CaptureAnImage(HWND hWnd)  
//  
//     PURPOSE: Captures a screenshot into a window ,and then saves it in a  
.bmp file.  
//  
//     COMMENTS:  
//  
//         Note: This function attempts to create a file called captureqwsx.bmp  
//  
int CaptureAnImage(HWND hWnd)  
{  
    HDC hdcScreen;  
    HDC hdcWindow;  
    HDC hdcMemDC = NULL;  
    HBITMAP hbmScreen = NULL;  
    BITMAP bmpScreen;  
    DWORD dwBytesWritten = 0;  
    DWORD dwSizeofDIB = 0;  
    HANDLE hFile = NULL;  
    char* lpbitmap = NULL;  
    HANDLE hDIB = NULL;  
    DWORD dwBmpSize = 0;  
  
    // Retrieve the handle to a display device context for the client  
    // area of the window.  
    hdcScreen = GetDC(NULL);  
    hdcWindow = GetDC(hWnd);
```

```
// Create a compatible DC, which is used in a BitBlt from the window DC.
hdcMemDC = CreateCompatibleDC(hdcWindow);

if (!hdcMemDC)
{
    MessageBox(hWnd, L"CreateCompatibleDC has failed", L"Failed",
MB_OK);
    goto done;
}

// Get the client area for size calculation.
RECT rcClient;
GetClientRect(hWnd, &rcClient);

// This is the best stretch mode.
SetStretchBltMode(hdcWindow, HALFTONE);

// The source DC is the entire screen, and the destination DC is the
// current window (HWND).
if (!StretchBlt(hdcWindow,
    0, 0,
    rcClient.right, rcClient.bottom,
    hdcScreen,
    0, 0,
    GetSystemMetrics(SM_CXSCREEN),
    GetSystemMetrics(SM_CYSCREEN),
    SRCCOPY))
{
    MessageBox(hWnd, L"StretchBlt has failed", L"Failed", MB_OK);
    goto done;
}

// Create a compatible bitmap from the Window DC.
hbmScreen = CreateCompatibleBitmap(hdcWindow, rcClient.right -
rcClient.left, rcClient.bottom - rcClient.top);

if (!hbmScreen)
{
    MessageBox(hWnd, L"CreateCompatibleBitmap Failed", L"Failed",
MB_OK);
    goto done;
}

// Select the compatible bitmap into the compatible memory DC.
SelectObject(hdcMemDC, hbmScreen);

// Bit block transfer into our compatible memory DC.
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;
    }

    // Get the BITMAP from the HBITMAP.
    GetObject(hbmScreen, sizeof(BITMAP), &bmpScreen);

    BITMAPFILEHEADER    bmfHeader;
    BITMAPINFOHEADER    bi;

    bi.biSize = sizeof(BITMAPINFOHEADER);
    bi.biWidth = bmpScreen.bmWidth;
    bi.biHeight = bmpScreen.bmHeight;
    bi.biPlanes = 1;
    bi.biBitCount = 32;
    bi.biCompression = BI_RGB;
    bi.biSizeImage = 0;
    bi.biXPelsPerMeter = 0;
    bi.biYPelsPerMeter = 0;
    bi.biClrUsed = 0;
    bi.biClrImportant = 0;

    dwBmpSize = ((bmpScreen.bmWidth * bi.biBitCount + 31) / 32) * 4 *
    bmpScreen.bmHeight;

    // Starting with 32-bit Windows, GlobalAlloc and LocalAlloc are
    // implemented as wrapper functions that
    // call HeapAlloc using a handle to the process's default heap.
    // Therefore, GlobalAlloc and LocalAlloc
    // have greater overhead than HeapAlloc.
    hDIB = GlobalAlloc(GHND, dwBmpSize);
    lpbitmap = (char*)GlobalLock(hDIB);

    // Gets the "bits" from the bitmap, and copies them into a buffer
    // that's pointed to by lpbitmap.
    GetDIBits(hdcWindow, hbmScreen, 0,
               (UINT)bmpScreen.bmHeight,
               lpbitmap,
               (BITMAPINFO*)&bi, DIB_RGB_COLORS);

    // A file is created, this is where we will save the screen capture.
    hFile = CreateFile(L"captureqwsx.bmp",
                       GENERIC_WRITE,
                       0,
                       NULL,
                       CREATE_ALWAYS,
                       FILE_ATTRIBUTE_NORMAL, NULL);

    // Add the size of the headers to the size of the bitmap to get the
    // total file size.
    dwSizeofDIB = dwBmpSize + sizeof(BITMAPFILEHEADER) +
    sizeof(BITMAPINFOHEADER);

    // Offset to where the actual bitmap bits start.
    bmfHeader.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) +
    (DWORD)sizeof(BITMAPINFOHEADER);

```

```

// Size of the file.
bmfHeader.bfSize = dwSizeofDIB;

// bfType must always be BM for Bitmaps.
bmfHeader.bfType = 0x4D42; // BM.

WriteFile(hFile, (LPSTR)&bmfHeader, sizeof(BITMAPFILEHEADER),
&dwBytesWritten, NULL);
WriteFile(hFile, (LPSTR)&bi, sizeof(BITMAPINFOHEADER), &dwBytesWritten,
NULL);
WriteFile(hFile, (LPSTR)lpbitmap, dwBmpSize, &dwBytesWritten, NULL);

// Unlock and Free the DIB from the heap.
GlobalUnlock(hDIB);
GlobalFree(hDIB);

// Close the handle for the file that was created.
CloseHandle(hFile);

// Clean up.
done:
DeleteObject(hbmScreen);
DeleteObject(hdcMemDC);
ReleaseDC(NULL, hdcScreen);
ReleaseDC(hWnd, hdcWindow);

return 0;
}

// 
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
// PURPOSE: Processes messages for the main window.
//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:

```

```

        DestroyWindow(hWnd);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
}
break;
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    CaptureAnImage(hWnd);
    EndPaint(hWnd, &ps);
}
break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}

```

Feedback

Was this page helpful?

 Yes

 No

Scaling an Image

Article • 01/07/2021

Some applications scale images; that is, they display zoomed or reduced views of an image. For example, a drawing application may provide a zoom feature that enables the user to view and edit a drawing on a pixel-by-pixel basis.

Applications scale images by calling the [StretchBlt](#) function. Like the [BitBlt](#) function, [StretchBlt](#) copies bitmap data from a bitmap in a source device context ([DC](#)) into a bitmap in a target DC. However, unlike the [BitBlt](#) function, [StretchBlt](#) scales the image based on the specified dimensions of the source and target rectangles. If the source rectangle is larger than the target rectangle, the resultant image will appear to have shrunk; if the source rectangle is smaller than the target rectangle, the resultant image will appear to have expanded.

If the target rectangle is smaller than the source rectangle, [StretchBlt](#) removes color data from the image according to a specified stretch mode as shown in the following table.

[+] [Expand table](#)

Stretch mode	Method
BLACKONWHITE	Performs a logical AND operation on the color data for the eliminated pixels and the color data for the remaining pixels.
WHITEONBLACK	Performs a logical OR operation on the color data for the eliminated pixels and the color data for the remaining pixels.
COLORONCOLOR	Eliminates the color data of the deleted pixels completely.
HALFTONE	Approximates the original (source) color data in the destination.

You set the stretch mode by calling the [SetStretchBltMode](#) function.

The following example code is taken from an application that demonstrates all four of the stretch modes available with the [StretchBlt](#) function.

C++

```
#include "stdafx.h"
#include "GDIBitmapScaling.h"
#include <commctrl.h>
```

```
#include <CommDlg.h>

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst; // current instance
TCHAR szTitle[MAX_LOADSTRING]; // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY _tWinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine,
                      int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.
    MSG msg;
    HACCEL hAccelTable;

    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_GDIBITMAPSCALING, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_GDIBITMAPSCALING));

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

```

        return (int) msg.wParam;
    }

    //

    // FUNCTION: MyRegisterClass()
    //

    // PURPOSE: Registers the window class.
    //

    // COMMENTS:
    //

    //      This function and its usage are only necessary if you want this code
    //      to be compatible with Win32 systems prior to the 'RegisterClassEx'
    //      function that was added to Windows 95. It is important to call this
    //      function
    //      so that the application will get 'well formed' small icons associated
    //      with it.
    //

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    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_GDIBITMAPSCALING));
    wcex.hCursor         = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground   = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCE(IDC_GDIBITMAPSCALING);
    wcex.lpszClassName   = szWindowClass;
    wcex.hIconSm          = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassEx(&wcex);
}

    //

    // FUNCTION: InitInstance(HINSTANCE, int)
    //

    // PURPOSE: Saves instance handle and creates main window
    //

    // COMMENTS:
    //

    //      In this function, we save the instance handle in a global variable
    //      and
    //      create and display the main program window.
    //

```

```

#define NEW_DIB_FORMAT(lpbih) (lpbih->biSize != sizeof(BITMAPCOREHEADER))
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;

    hInst = hInstance; // Store instance handle in our global variable

    hWnd = CreateWindow(szWindowClass, szTitle, WS_SYSMENU,
        CW_USEDEFAULT, 0, 1024, 768, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

static HCURSOR hcurSave;

WORD DIBNumColors (LPVOID lpv)
{
    INT             bits;
    LPBITMAPINFOHEADER lpbih = (LPBITMAPINFOHEADER)lpv;
    LPBITMAPCOREHEADER lpbch = (LPBITMAPCOREHEADER)lpv;

    /* With the BITMAPINFO format headers, the size of the palette
     * is in biClrUsed, whereas in the BITMAPCORE - style headers, it
     * is dependent on the bits per pixel (= 2 raised to the power of
     * bits/pixel).
     */
    if (NEW_DIB_FORMAT(lpbih)) {
        if (lpbih->biClrUsed != 0)
            return (WORD)lpbih->biClrUsed;
        bits = lpbih->biBitCount;
    }
    else
        bits = lpbch->bcBitCount;

    if (bits > 8)
        return 0; /* Since biClrUsed is 0, we dont have a an optimal palette
*/
    else
        return (1 << bits);
}

/* Macro to determine to round off the given value to the closest byte */
#define WIDTHBYTES(i) (((i)+31) >> 5) << 2
/********************* ****
 *
 *
 *  FUNCTION      : DIBInfo(HANDLE hbi, LPBITMAPINFOHEADER lpbih)

```

```

*
*
*
* PURPOSE      : Retrieves the DIB info associated with a CF_DIB
*
*                  format memory block.
*
*
*
* RETURNS      : TRUE  - if successful.
*
*                  FALSE - otherwise
*
*
*

*****
*/
BOOL DIBInfo (HANDLE hbi, LPBITMAPINFOHEADER lpbih)
{
    if (hbi){
        *lpbih = *(LPBITMAPINFOHEADER)hbi;

        /* fill in the default fields */
        if (NEW_DIB_FORMAT(lpbih)) {
            if (lpbih->biSizeImage == 0L)
                lpbih->biSizeImage = WIDTHBYTES(lpbih->biWidth*lpbih->biBitCount)
* lpbih->biHeight;

            if (lpbih->biClrUsed == 0L)
                lpbih->biClrUsed = DIBNumColors (lpbih);
        }

        return TRUE;
    }
    return FALSE;
}

/* flags for mmioSeek() */
#ifndef SEEK_SET
#define SEEK_SET      0          /* seek to an absolute position */
#define SEEK_CUR      1          /* seek relative to current position */
*/
#define SEEK_END      2          /* seek relative to end of file */
#endif /* ifndef SEEK_SET */

VOID ReadPackedFileHeader(HFILE hFile, LPBITMAPFILEHEADER lpbmfhdr, LPDWORD lpdwOffset)
{
    *lpdwOffset = _lseek(hFile, 0L, (UINT) SEEK_CUR);
    _hread(hFile, (LPSTR) &lpbmhdr->bfType, sizeof(WORD)); /* read in
bfType*/
    _hread(hFile, (LPSTR) &lpbmhdr->bfSize, sizeof(DWORD) * 3); /* read in
last 3 dwords*/
}

```

```

/* macro to determine if resource is a DIB */
#define ISDIB(bft) ((bft) == BFT_BITMAP)

/* Header signatures for various resources */
#define BFT_ICON    0x4349 /* 'IC' */
#define BFT_BITMAP  0x4d42 /* 'BM' */
#define BFT_CURSOR  0x5450 /* 'PT' */
HANDLE ReadDIBBitmapInfo (INT hFile)
{
    DWORD             dwOffset;
    HANDLE            hbi = NULL;
    INT               size;
    INT               i;
    WORD              nNumColors;
    LPRGBQUAD         lprgbq;
    BITMAPINFOHEADER  bih;
    BITMAPCOREHEADER  bch;
    LPBITMAPINFOHEADER lpbih;
    BITMAPFILEHEADER   bf;
    DWORD             dwDW_masks= 0;
    DWORD             dwWidth = 0;
    DWORD             dwHeight = 0;
    WORD              wPlanes, wBitCount;

    if (hFile == HFILE_ERROR)
        return NULL;

    /* Read the bitmap file header */
    ReadPackedFileHeader(hFile, &bf, &dwOffset);

    /* Do we have a RC HEADER? */
    if (!ISDIB (bf.bfType)) {
        bf.bfOffBits = 0L;
        _lseek(hFile, dwOffset, (UINT)SEEK_SET); /* seek back to beginning
of file */
    }

    if (sizeof(bih) != _hread(hFile, (LPSTR)&bih, (UINT)sizeof(bih)))
        return FALSE;

    nNumColors = DIBNumColors (&bih);

    /* Check the nature (BITMAPINFO or BITMAPCORE) of the info. block
     * and extract the field information accordingly. If a BITMAPCOREHEADER,
     * transfer it's field information to a BITMAPINFOHEADER-style block
     */
    switch (size = (INT)bih.biSize){
        case sizeof (BITMAPINFOHEADER):
            break;

        case sizeof (BITMAPCOREHEADER):
            bch = *(LPBITMAPCOREHEADER)&bih;

```

```

        dwWidth    = (DWORD)bch.bcWidth;
        dwHeight   = (DWORD)bch.bcHeight;
        wPlanes    = bch.bcPlanes;
        wBitCount  = bch.bcBitCount;

        bih.biSize           = sizeof(BITMAPINFOHEADER);
        bih.biWidth          = dwWidth;
        bih.biHeight         = dwHeight;
        bih.biPlanes         = wPlanes;
        bih.biBitCount       = wBitCount;
        bih.biCompression    = BI_RGB;
        bih.biSizeImage      = 0;
        bih.biXPelsPerMeter = 0;
        bih.biYPelsPerMeter = 0;
        bih.biClrUsed        = nNumColors;
        bih.biClrImportant   = nNumColors;

        _lseek(hFile, (LONG)sizeof(BITMAPCOREHEADER) - sizeof(BITMAPINFOHEADER), (UINT)SEEK_CUR);
        break;

    default:
        /* Not a DIB! */
        return NULL;
}

/* Fill in some default values if they are zero */
if (bih.biSizeImage == 0){
    bih.biSizeImage = WIDTHBYTES((DWORD)bih.biWidth * bih.biBitCount) *
bih.biHeight;
}
if (bih.biClrUsed == 0)
    bih.biClrUsed = DIBNumColors(&bih);

/* Allocate for the BITMAPINFO structure and the color table. */
if ((bih.biBitCount == 16) || (bih.biBitCount == 32))
    dwDwMasks = sizeof(DWORD) * 3;
hbi = GlobalAlloc (GPTR, (LONG)bih.biSize + nNumColors * sizeof(RGBQUAD)
+ dwDwMasks);
if (!hbi)
    return NULL;
lpbih = (LPBITMAPINFOHEADER)hbi;
*lpbih = bih;

/* Get a pointer to the color table */
lprgbq = (LPRGBQUAD)((LPSTR)lpbih + bih.biSize);
if (nNumColors){
    if (size == sizeof(BITMAPCOREHEADER)){
        /* Convert a old color table (3 byte RGBTRIPLEs) to a new
        * color table (4 byte RGBQUADS)
        */
        _hread(hFile, (LPSTR)lprgbq, (UINT)nNumColors *
sizeof(RGBTRIPLE));

```

```

        for (i = nNumColors - 1; i >= 0; i--){
            RGBQUAD rgbq;

            rgbq.rgbRed      = ((RGBTRIPLE*)lprgbq)[i].rgbtRed;
            rgbq.rgbBlue     = ((RGBTRIPLE*)lprgbq)[i].rgbtBlue;
            rgbq.rgbGreen    = ((RGBTRIPLE*)lprgbq)[i].rgbtGreen;
            rgbq.rgbReserved = (BYTE)0;

            lprgbq[i] = rgbq;
        }
    }
    else
        _hread(hFile, (LPSTR)lprgbq, (UINT)nNumColors *
sizeof(RGBQUAD));
} else
    if (dwDwMasks)
        _hread(hFile, (LPSTR)lprgbq, dwDwMasks);

if (bf.bfOffBits != 0L){
    _llseek(hFile, dwOffset + bf.bfOffBits, (UINT)SEEK_SET);
}

return hbi;
}
HGLOBAL GlobalFreeDIB(HGLOBAL hDIB)
{
    LPBITMAPINFOHEADER lpbi = (LPBITMAPINFOHEADER)hDIB;

    if (!lpbi->biClrImportant)
        return GlobalFree(hDIB);

    if (GlobalFlags((HGLOBAL)lpbi->biClrImportant) == GMEM_INVALID_HANDLE) {
        SetLastError(0);
        return GlobalFree(hDIB);
    } else
        return GlobalFree((HANDLE)lpbi->biClrImportant);
}
//****************************************************************************
***  

*  

*  

*   FUNCTION    : ColorTableSize(LPVOID lpv)  

*  

*  

*   PURPOSE     : Calculates the palette size in bytes. If the info. block  

*                  is of the BITMAPCOREHEADER type, the number of colors is  

*                  multiplied by 3 to give the palette size, otherwise the  

*                  number of colors is multiplied by 4.  

*  

*  

*
```

```

*   RETURNS    : Color table size in number of bytes.
*
*
*

*****
*/
WORD ColorTableSize (LPVOID lpv)
{
    LPBITMAPINFOHEADER lpbih = (LPBITMAPINFOHEADER)lpv;

    if (NEW_DIB_FORMAT(lpbih))
    {
        if (((LPBITMAPINFOHEADER)(lpbih))->biCompression == BI_BITFIELDS)
            /* Remember that 16/32bpp dibs can still have a color table */
            return (sizeof(DWORD) * 3) + (DIBNumColors (lpbih) * sizeof
(RGBQUAD));
        else
            return (DIBNumColors (lpbih) * sizeof (RGBQUAD));
    }
    else
        return (DIBNumColors (lpbih) * sizeof (RGBTRIPLE));
}

/*****
*** 
*
*
*   FUNCTION    :OpenDIB(LPSTR szFilename)
*
*
*
*   PURPOSE     :Open a DIB file and create a memory DIB -- a memory handle
*
*                   containing BITMAPINFO, palette data and the bits.
*
*
*
*   RETURNS     :A handle to the DIB.
*
*
*
***** */

HANDLE OpenDIB (LPSTR szFilename)
{
    HFILE             hFile;
    BITMAPINFOHEADER  bih;
    LPBITMAPINFOHEADER lpbih;
    DWORD             dwLen = 0;
    DWORD             dwBits;
    HANDLE            hDIB;
    HANDLE            hMem;
    OFSTRUCT          of;

```

```

/* Open the file and read the DIB information */
hFile = OpenFile(szFilename, &of, (UINT)OF_READ);
if (hFile == HFILE_ERROR)
    return NULL;

hDIB = ReadDIBBitmapInfo(hFile);
if (!hDIB)
    return NULL;
DIBInfo(hDIB, &bih);

/* Calculate the memory needed to hold the DIB */
dwBits = bih.biSizeImage;
dwLen = bih.biSize + (DWORD)ColorTableSize (&bih) + dwBits;

/* Try to increase the size of the bitmap info. buffer to hold the DIB */
hMem = GlobalReAlloc(hDIB, dwLen, GMEM_MOVEABLE);

if (!hMem){
    GlobalFreeDIB(hDIB);
    hDIB = NULL;
}
else
    hDIB = hMem;

/* Read in the bits */
if (hDIB){
    lpbih = (LPBITMAPINFOHEADER)hDIB;
    _hread(hFile, (LPSTR)lpbih + (WORD)lpbih->biSize +
ColorTableSize(lpbih), dwBits);
}
_lclose(hFile);

return hDIB;
}/* Macros to display/remove hourglass cursor for lengthy operations */
#define StartWait() hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT))
#define EndWait() SetCursor(hcurSave)
/********************* ****
 *
 *
 *   FUNCTION      : BitmapFromDIB(HANDLE hDIB, HPALETTE hPal)
 *
 *
 *   PURPOSE       : Will create a DDB (Device Dependent Bitmap) given a global
 *                   handle to a memory block in CF_DIB format
 *
 *
 *   RETURNS      : A handle to the DDB.
*
*

```

```

*
*****
*/
HBITMAP BitmapFromDIB (HANDLE hDIB, HPALETTE hPal)
{
    LPBITMAPINFOHEADER lpbih;
    HPALETTE           hPalOld;
    HDC                hDC;
    HBITMAP            hBitmap;

    StartWait();

    if (!hDIB)
        return NULL;

    lpbih = (LPBITMAPINFOHEADER)hDIB;

    if (!lpbih)
        return NULL;

    hDC = GetDC(NULL);

    if (hPal){
        hPalOld = SelectPalette(hDC, hPal, FALSE);
        RealizePalette(hDC);
    }

    hBitmap = CreateDIBitmap(hDC,
                            lpbih,
                            CBM_INIT,
                            (LPSTR)lpbih + lpbih->biSize + ColorTableSize(lpbih),
                            (LPBITMAPINFO)lpbih,
                            DIB_RGB_COLORS );

    if (hPal)
        SelectPalette(hDC, hPalOld, FALSE);

    ReleaseDC(NULL, hDC);

    EndWait();

    return hBitmap;
}
/*****
***  

*  

*   * FUNCTION    : DrawBitmap(HDC hDC, int x, int y,  

*  

*                           HBITMAP hBitmap, DWORD dwROP)  

*  

*   *  

*   * PURPOSE     : Draws bitmap <hBitmap> at the specified position in DC

```

```

<hDC> *
*
*
* RETURNS      : Return value of BitBlt()
*
*
* ****
****

*/
BOOL DrawBitmap (HDC hDC, INT x, INT y, HBITMAP hBitmap, DWORD dwROP)
{
    HDC      hDCBits;
    BITMAP   Bitmap;
    BOOL     bResult;

    if (!hDC || !hBitmap)
        return FALSE;

    hDCBits = CreateCompatibleDC(hDC);
    GetObject(hBitmap, sizeof(BITMAP), (LPSTR)&Bitmap);
    SelectObject(hDCBits, hBitmap);
    bResult = BitBlt(hDC, x, y, Bitmap.bmWidth, Bitmap.bmHeight, hDCBits, 0,
0, dwROP);
    DeleteDC(hDCBits);

    return bResult;
}

/*FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

PURPOSE: Processes messages for the main window.

WM_COMMAND      - process the application menu
WM_PAINT        - Paint the main window
WM_DESTROY      - post a quit message and return*/

```



```

#define ID_LOADBITMAP 1

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    //Handle to a GDI device context
    HDC hDC;
    //Handle to a DDB(device-dependent bitmap)
    HBITMAP hBitmap;

    HFONT hFont;
    NONCLIENTMETRICS ncm={0};
    ncm.cbSize= sizeof(NONCLIENTMETRICS);
}

```

```

static HWND hwndButton;
static HWND hwndButtonExit;
static HANDLE hDIB = NULL;

char szDirName[MAX_PATH];
char szFilename[MAX_PATH] = "\0";
char szBitmapName[MAX_PATH] = "\\Waterfall.bmp";
//char szBitmapName[MAX_PATH] = "\\tulips256.bmp";
OPENFILENAMEA ofn;

switch (message)
{
case WM_CREATE:

    //Creates a font from the current theme's caption font
    SystemParametersInfo(SPI_GETNONCLIENTMETRICS, NULL, &ncm, NULL);
    hFont = CreateFontIndirect(&ncm.lfCaptionFont);

    //Gets the device context for the current window
    hDC = GetDC(hWnd);

    //Gets the directory of the current project and loads Waterfall.bmp
    GetCurrentDirectoryA(MAX_PATH, szDirName);
    strcat_s(szDirName, szBitmapName);
    strcat_s(szFilename, szDirName);
    hDIB = OpenDIB(szFilename);
    hBitmap = BitmapFromDIB(hDIB, NULL);

    //Draws Waterfall.bmp as a device dependent bitmap
    DrawBitmap(hDC, 0, 0, hBitmap, SRCCOPY);
    InvalidateRect(hWnd, NULL, FALSE);
    ReleaseDC(hWnd, hDC);

    //Draws the "Load Bitmap" button
    hwndButton = CreateWindowW(TEXT("button"), TEXT("Load Bitmap"),
        WS_CHILD | WS_VISIBLE | WS_BORDER, 600, 200, 150, 50,
        hWnd,
        (HMENU)ID_LOADBITMAP,
        ((LPCREATESTRUCT) lParam)-> hInstance, NULL);

    //Set the font of the button to the theme's caption font
    SendMessage(hwndButton, WM_SETFONT, (WPARAM)hFont, TRUE );

return 0;

case WM_COMMAND:
    wmId     = LOWORD(wParam);
    wmEvent  = HIWORD(wParam);

```

```

//if Load Bitmap button is pressed
if (wmId == ID_LOADBITMAP && wmEvent == BN_CLICKED)
{

    //Get the current directory name, and store in szDirName
    GetCurrentDirectoryA(MAX_PATH, szDirName);

    //Set all structure members to zero.
    ZeroMemory(&ofn, sizeof(OPENFILENAMEA));

    //Initializing the OPENFILENAMEA structure
    ofn.lStructSize = sizeof(OPENFILENAMEA);
    ofn.hwndOwner = hWnd;
    ofn.lpstrFilter ="BMP Files (*.BMP)\0 *.BMP\0\0";
    ofn.lpstrFile = szFilename;
    ofn.nMaxFile = sizeof(szFilename);
    ofn.lpstrDefExt = "BMP";
    ofn.lpstrInitialDir = szDirName;
    ofn.Flags = OFN_EXPLORER | OFN_SHOWHELP | OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

    if (GetOpenFileNameA(&ofn)) {
        hDIB = OpenDIB((LPSTR)szFilename);
        if (!hDIB)
            MessageBox(hWnd, TEXT("Unable to load file!"),
TEXT("Oops"), MB_ICONSTOP);
        } else {
            if (strlen((const char *)szFilename) != 0)
                MessageBox(hWnd, TEXT("Unable to load file!"),
TEXT("Oops"), MB_ICONSTOP);

            return 0;
        }

        InvalidateRect(hWnd, NULL, FALSE);

    }

    break;
case WM_PAINT:
{
    //Initializing arrays for boxes
    POINT pRect1[5] = {{0,0},{400,0},{400,400},{0,400},{0,0}};
    POINT pRect2[5] = {{0,500}, {200, 500}, {200, 700}, {0,700},
{0,500}};
    POINT pRect3[5] = {{210,500}, {410, 500}, {410, 700}, {210,700},
{210,500}};
}

```

```

        POINT pRect4[5] = {{420,500}, {620, 500}, {620, 700}, {420,700},
{420,500}};
        POINT pRect5[5] = {{630,500}, {830, 500}, {830, 700}, {630,700},
{630,500}};

        //For the white background
        RECT clientRect;
        HRGN hRegion1;
        HRGN hRegion2;
        HRGN hRegion3;
        HBRUSH hBGBrush;

        //Handle to a logical font
        HFONT hFont;

        //Get the caption font that is currently in use
        SystemParametersInfo(SPI_GETNONCLIENTMETRICS, NULL, &ncm, NULL);
        hFont = CreateFontIndirect(&ncm.lfCaptionFont);

        //Begin drawing
        hDC = BeginPaint(hWnd, &ps);

        //Draw and fill rectangles for the background
        GetClientRect(hWnd, &clientRect);
        hRegion1 =
CreateRectRgn(clientRect.left,clientRect.top,clientRect.right,clientRect.bot
tom);
        hBGBrush = CreateSolidBrush(RGB(255,255,255));
        FillRgn(hDC, hRegion1, hBGBrush);

        //Create an HBITMAP(device dependent bitmap) to be drawn
        hBitmap = BitmapFromDIB(hDIB,NULL);
        //Draw the DDB
        DrawBitmap(hDC,0,0,hBitmap,SRCCOPY);

        if(hDIB)
        {
            hRegion2 = CreateRectRgn(401,0,clientRect.right,401);
            hRegion3 =
CreateRectRgn(0,401,clientRect.right,clientRect.bottom);
            FillRgn(hDC,hRegion2,hBGBrush);
            FillRgn(hDC,hRegion3,hBGBrush);
            //Set stretch mode as BLACKONWHITE and then copy from the
            //source rectangle into the smaller rectangle
            SetStretchBltMode(hDC,BLACKONWHITE);
            StretchBlt(hDC,0,500,200,200,hDC, 0,0,400,400,SRCCOPY);

            //Set stretch mode as WHITEONBLACK and then copy from the
            //source rectangle into the smaller rectangle
            SetStretchBltMode(hDC,WHITEONBLACK);
            StretchBlt(hDC,210,500,200,200, hDC, 0,0,400,400, SRCCOPY);

            //Set stretch mode as COLORONCOLOR and then copy from the
            //source rectangle into the smaller rectangle

```

```

        SetStretchBltMode(hDC,COLORONCOLOR);
        StretchBlt(hDC,420,500,200,200, hDC, 0,0,400,400, SRCCOPY);

        //Set stretch mode as HALFTONE and then copy from the
        //source rectangle into the smaller rectangle
        SetStretchBltMode(hDC,HALFTONE);
        StretchBlt(hDC,630,500,200,200, hDC, 0,0,400,400, SRCCOPY);

    }

    //Select the caption font created earlier
    SelectObject(hDC,hFont);

    //Create captions for each demonstration of color loss modes
    TextOut(hDC,50,480,TEXT("BLACKONWHITE"),12);
    TextOut(hDC,250,480,TEXT("WHITEONBLACK"),12);
    TextOut(hDC,460,480,TEXT("COLORONCOLOR"),12);
    TextOut(hDC,680,480,TEXT("HALFTONE"),8);
    DeleteObject(hFont);

    //Selecting the stock object pen to draw with
    SelectObject(hDC, GetStockObject(DC_PEN));
    //The pen is gray
    SetDCPenColor(hDC, RGB(80,80,80));

    //Polylines are drawn from arrays of POINTs
    Polyline(hDC, pRect1, 5);
    Polyline(hDC, pRect2, 5);
    Polyline(hDC, pRect3, 5);
    Polyline(hDC, pRect4, 5);
    Polyline(hDC, pRect5, 5);

    FillRgn(hDC,hRegion2,hBGBrush);
    EndPaint(hWnd, &ps);

    break;
}
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Storing an Image

Article • 11/19/2022

Many applications store images permanently as files. For example, drawing applications store pictures, spreadsheet applications store charts, CAD applications store drawings, and so on.

If you are writing an application that stores a bitmap image in a file, you should use the bitmap file format described in [Bitmap Storage](#). To store a bitmap in this format, you must use a [BITMAPINFOHEADER](#), a [BITMAPV4HEADER](#), or a [BITMAPV5HEADER](#) structure and an array of [RGBQUAD](#) structures, as well as an array of palette indexes.

The following example code defines a function that uses a [BITMAPINFO](#) structure and allocates memory for and initializes members within a [BITMAPINFOHEADER](#) structure. Note that the [BITMAPINFO](#) structure cannot be used with either a [BITMAPV4HEADER](#) or a [BITMAPV5HEADER](#) structure.

C++

```
PBITMAPINFO CreateBitmapInfoStruct(HWND hwnd, HBITMAP hBmp)
{
    BITMAP bmp;
    PBITMAPINFO pbmi;
    WORD    cClrBits;

    // Retrieve the bitmap color format, width, and height.
    if (!GetObject(hBmp, sizeof(BITMAP), (LPSTR)&bmp))
        errhandler("GetObject", hwnd);

    // Convert the color format to a count of bits.
    cClrBits = (WORD)(bmp.bmPlanes * bmp.bmBitsPixel);
    if (cClrBits == 1)
        cClrBits = 1;
    else if (cClrBits <= 4)
        cClrBits = 4;
    else if (cClrBits <= 8)
        cClrBits = 8;
    else if (cClrBits <= 16)
        cClrBits = 16;
    else if (cClrBits <= 24)
        cClrBits = 24;
    else cClrBits = 32;

    // Allocate memory for the BITMAPINFO structure. (This structure
    // contains a BITMAPINFOHEADER structure and an array of RGBQUAD
    // data structures.)

    if (cClrBits < 24)
        pbmi = (PBITMAPINFO) LocalAlloc(LPTR,
```

```

        sizeof(BITMAPINFOHEADER) +
        sizeof(RGBQUAD) * (1<< cClrBits));

    // There is no RGBQUAD array for these formats: 24-bit-per-pixel or 32-
    // bit-per-pixel

    else
        pbmi = (PBITMAPINFO) LocalAlloc(LPTR,
                                         sizeof(BITMAPINFOHEADER));

    // Initialize the fields in the BITMAPINFO structure.

    pbmi->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    pbmi->bmiHeader.biWidth = bmp.bmWidth;
    pbmi->bmiHeader.biHeight = bmp.bmHeight;
    pbmi->bmiHeader.biPlanes = bmp.bmPlanes;
    pbmi->bmiHeader.biBitCount = bmp.bmBitsPixel;
    if (cClrBits < 24)
        pbmi->bmiHeader.biClrUsed = (1<<cClrBits);

    // If the bitmap is not compressed, set the BI_RGB flag.
    pbmi->bmiHeader.biCompression = BI_RGB;

    // Compute the number of bytes in the array of color
    // indices and store the result in biSizeImage.
    // The width must be DWORD aligned unless the bitmap is RLE
    // compressed.
    pbmi->bmiHeader.biSizeImage = ((pbmi->bmiHeader.biWidth * cClrBits +31)
& ~31) /8
                                         * pbmi->bmiHeader.biHeight;
    // Set biClrImportant to 0, indicating that all of the
    // device colors are important.
    pbmi->bmiHeader.biClrImportant = 0;
    return pbmi;
}

```

The following example code defines a function that initializes the remaining structures, retrieves the array of palette indices, opens the file, copies the data, and closes the file.

C++

```

void CreateBMPFile(HWND hwnd, LPTSTR pszFile, PBITMAPINFO pbi,
                   HBITMAP hBMP, HDC hDC)
{
    HANDLE hf;           // file handle
    BITMAPFILEHEADER hdr; // bitmap file-header
    PBITMAPINFOHEADER pbih; // bitmap info-header
    LPBYTE lpBits;       // memory pointer
    DWORD dwTotal;       // total count of bytes
    DWORD cb;            // incremental count of bytes
    BYTE *hp;             // byte pointer
    DWORD dwTmp;

```

```

pbih = (PBITMAPINFOHEADER) pbi;
lpBits = (LPBYTE) GlobalAlloc(GMEM_FIXED, pbih->biSizeImage);

if (!lpBits)
    errhandler("GlobalAlloc", hwnd);

// Retrieve the color table (RGBQUAD array) and the bits
// (array of palette indices) from the DIB.
if (!GetDIBits(hDC, hBMP, 0, (WORD) pbih->biHeight, lpBits, pbi,
    DIB_RGB_COLORS))
{
    errhandler("GetDIBits", hwnd);
}

// Create the .BMP file.
hf = CreateFile(pszFile,
                GENERIC_READ | GENERIC_WRITE,
                (DWORD) 0,
                NULL,
                CREATE_ALWAYS,
                FILE_ATTRIBUTE_NORMAL,
                (HANDLE) NULL);
if (hf == INVALID_HANDLE_VALUE)
    errhandler("CreateFile", hwnd);
hdr.bfType = 0x4d42;           // 0x42 = "B" 0x4d = "M"
// Compute the size of the entire file.
hdr.bfSize = (DWORD) (sizeof(BITMAPFILEHEADER) +
                     pbih->biSize + pbih->biClrUsed
                     * sizeof(RGBQUAD) + pbih->biSizeImage);
hdr.bfReserved1 = 0;
hdr.bfReserved2 = 0;

// Compute the offset to the array of color indices.
hdr.bfOffBits = (DWORD) sizeof(BITMAPFILEHEADER) +
                pbih->biSize + pbih->biClrUsed
                * sizeof (RGBQUAD);

// Copy the BITMAPFILEHEADER into the .BMP file.
if (!WriteFile(hf, (LVOID) &hdr, sizeof(BITMAPFILEHEADER),
    (LPDWORD) &dwTmp, NULL))
{
    errhandler("WriteFile", hwnd);
}

// Copy the BITMAPINFOHEADER and RGBQUAD array into the file.
if (!WriteFile(hf, (LVOID) pbih, sizeof(BITMAPINFOHEADER)
    + pbih->biClrUsed * sizeof (RGBQUAD),
    (LPDWORD) &dwTmp, (NULL)))
    errhandler("WriteFile", hwnd);

// Copy the array of color indices into the .BMP file.
dwTotal = cb = pbih->biSizeImage;
hp = lpBits;
if (!WriteFile(hf, (LPSTR) hp, (int) cb, (LPDWORD) &dwTmp, NULL))
    errhandler("WriteFile", hwnd);

```

```
// Close the .BMP file.  
if (!CloseHandle(hf))  
    errhandler("CloseHandle", hwnd);  
  
// Free memory.  
GlobalFree((HGLOBAL)lpBits);  
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Alpha Blending a Bitmap

Article • 01/07/2021

The following code sample divides a window into three horizontal areas. Then it draws an alpha-blended bitmap in each of the window areas as follows:

- In the top area, constant alpha = 50% but there is no source alpha.
- In the middle area, constant alpha = 100% (disabled) and source alpha is 0 (transparent) in the middle of the bitmap and 0xff (opaque) elsewhere.
- In the bottom area, constant alpha = 75% and source alpha changes.

C++

```
void DrawAlphaBlend (HWND hWnd, HDC hdcwnd)
{
    HDC hdc;           // handle of the DC we will create
    BLENDFUNCTION bf; // structure for alpha blending
    HBITMAP hbitmap; // bitmap handle
    BITMAPINFO bmi;  // bitmap header
    VOID *pvBits;     // pointer to DIB section
    ULONG ulWindowWidth, ulWindowHeight; // window width/height
    ULONG ulBitmapWidth, ulBitmapHeight; // bitmap width/height
    RECT rt;          // used for getting window dimensions
    UINT32 x,y;       // stepping variables
    UCHAR ubAlpha;    // used for doing transparent gradient
    UCHAR ubRed;
    UCHAR ubGreen;
    UCHAR ubBlue;
    float fAlphaFactor; // used to do premultiply

    // get window dimensions
    GetClientRect(hWnd, &rt);

    // calculate window width/height
    ulWindowWidth = rt.right - rt.left;
    ulWindowHeight = rt.bottom - rt.top;

    // make sure we have at least some window size
    if ((!ulWindowWidth) || (!ulWindowHeight))
        return;

    // divide the window into 3 horizontal areas
    ulWindowHeight = ulWindowHeight / 3;

    // create a DC for our bitmap -- the source DC for AlphaBlend
    hdc = CreateCompatibleDC(hdcwnd);

    // zero the memory for the bitmap info
    ZeroMemory(&bmi, sizeof(BITMAPINFO));
```

```

// setup bitmap info
// set the bitmap width and height to 60% of the width and height of
// each of the three horizontal areas. Later on, the blending will occur in the
// center of each of the three areas.
bmi.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
bmi.bmiHeader.biWidth = ulBitmapWidth = ulWindowWidth -
(ulWindowWidth/5)*2;
bmi.bmiHeader.biHeight = ulBitmapHeight = ulWindowHeight -
(ulWindowHeight/5)*2;
bmi.bmiHeader.biPlanes = 1;
bmi.bmiHeader.biBitCount = 32;           // four 8-bit components
bmi.bmiHeader.biCompression = BI_RGB;
bmi.bmiHeader.biSizeImage = ulBitmapWidth * ulBitmapHeight * 4;

// create our DIB section and select the bitmap into the dc
hbitmap = CreateDIBSection(hdc, &bmi, DIB_RGB_COLORS, &pvBits, NULL,
0x0);
SelectObject(hdc, hbitmap);

// in top window area, constant alpha = 50%, but no source alpha
// the color format for each pixel is 0xaarrggb
// set all pixels to blue and set source alpha to zero
for (y = 0; y < ulBitmapHeight; y++)
    for (x = 0; x < ulBitmapWidth; x++)
        ((UINT32 *)pvBits)[x + y * ulBitmapWidth] = 0x000000ff;

bf.BlendOp = AC_SRC_OVER;
bf.BlendFlags = 0;
bf.SourceConstantAlpha = 0x7f; // half of 0xff = 50% transparency
bf.AlphaFormat = 0;           // ignore source alpha channel

if (!AlphaBlend(hdcwnd, ulWindowWidth/5, ulWindowHeight/5,
                ulBitmapWidth, ulBitmapHeight,
                hdc, 0, 0, ulBitmapWidth, ulBitmapHeight, bf))
    return;                  // alpha blend failed

// in middle window area, constant alpha = 100% (disabled), source
// alpha is 0 in middle of bitmap and opaque in rest of bitmap
for (y = 0; y < ulBitmapHeight; y++)
    for (x = 0; x < ulBitmapWidth; x++)
        if ((x > (int)(ulBitmapWidth/5)) && (x < (ulBitmapWidth-
ulBitmapWidth/5)) &&
            (y > (int)(ulBitmapHeight/5)) && (y < (ulBitmapHeight-
ulBitmapHeight/5)))
            //in middle of bitmap: source alpha = 0 (transparent).
            // This means multiply each color component by 0x00.
            // Thus, after AlphaBlend, we have a, 0x00 * r,
            // 0x00 * g, and 0x00 * b (which is 0x00000000)
            // for now, set all pixels to red
            ((UINT32 *)pvBits)[x + y * ulBitmapWidth] = 0x00ff0000;
        else
            // in the rest of bitmap, source alpha = 0xff (opaque)
            // and set all pixels to blue
            ((UINT32 *)pvBits)[x + y * ulBitmapWidth] = 0xff0000ff;
endif;

```

```

bf.BlendOp = AC_SRC_OVER;
bf.BlendFlags = 0;
bf.AlphaFormat = AC_SRC_ALPHA; // use source alpha
bf.SourceConstantAlpha = 0xff; // opaque (disable constant alpha)

if (!AlphaBlend(hdcwnd, ulWindowWidth/5,
ulWindowHeight/5+ulWindowHeight, ulBitmapWidth, ulBitmapHeight, hdc, 0, 0,
ulBitmapWidth, ulBitmapHeight, bf))
    return;

// bottom window area, use constant alpha = 75% and a changing
// source alpha. Create a gradient effect using source alpha, and
// then fade it even more with constant alpha
ubRed = 0x00;
ubGreen = 0x00;
ubBlue = 0xff;

for (y = 0; y < ulBitmapHeight; y++)
    for (x = 0; x < ulBitmapWidth; x++)
    {
        // for a simple gradient, base the alpha value on the x
        // value of the pixel
        ubAlpha = (UCHAR)((float)x / (float)ulBitmapWidth * 255);
        //calculate the factor by which we multiply each component
        fAlphaFactor = (float)ubAlpha / (float)0xff;
        // multiply each pixel by fAlphaFactor, so each component
        // is less than or equal to the alpha value.
        ((UINT32 *)pvBits)[x + y * ulBitmapWidth]
            = (ubAlpha << 24) | //0xaa000000
            ((UCHAR)(ubRed * fAlphaFactor) << 16) | //0x00rr0000
            ((UCHAR)(ubGreen * fAlphaFactor) << 8) | //0x0000gg00
            ((UCHAR)(ubBlue * fAlphaFactor)); //0x000000bb
    }

bf.BlendOp = AC_SRC_OVER;
bf.BlendFlags = 0;
bf.AlphaFormat = AC_SRC_ALPHA; // use source alpha
bf.SourceConstantAlpha = 0xbf; // use constant alpha, with
                           // 75% opaqueness

AlphaBlend(hdcwnd, ulWindowWidth/5,
           ulWindowHeight/5+2*ulWindowHeight, ulBitmapWidth,
           ulBitmapHeight, hdc, 0, 0, ulBitmapWidth,
           ulBitmapHeight, bf);

// do cleanup
DeleteObject(hbitmap);
DeleteDC(hdc);
}

```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing a Shaded Rectangle

Article • 01/07/2021

To draw a shaded rectangle, define a [TRIVERTEX](#) array with two elements and a single [GRADIENT_RECT](#) structure. The following code example shows how to draw a shaded rectangle using the [GradientFill](#) function with the GRADIENT_FILL_RECT mode defined.

C++

```
// Create an array of TRIVERTEX structures that describe
// positional and color values for each vertex. For a rectangle,
// only two vertices need to be defined: upper-left and lower-right.
TRIVERTEX vertex[2] ;
vertex[0].x      = 0;
vertex[0].y      = 0;
vertex[0].Red    = 0x0000;
vertex[0].Green  = 0x8000;
vertex[0].Blue   = 0x8000;
vertex[0].Alpha  = 0x0000;

vertex[1].x      = 300;
vertex[1].y      = 80;
vertex[1].Red    = 0x0000;
vertex[1].Green  = 0xd000;
vertex[1].Blue   = 0xd000;
vertex[1].Alpha  = 0x0000;

// Create a GRADIENT_RECT structure that
// references the TRIVERTEX vertices.
GRADIENT_RECT gRect;
gRect.UpperLeft  = 0;
gRect.LowerRight = 1;

// Draw a shaded rectangle.
GradientFill(hdc, vertex, 2, &gRect, 1, GRADIENT_FILL_RECT_H);
```

The following image shows the drawing output of the preceding code example.



Related topics

[Bitmaps Overview](#)

Bitmap Functions

Drawing a Shaded Triangle

EMRGRADIENTFILL

GRADIENT_RECT

GradientFill

TRIVERTEX

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing a Shaded Triangle

Article • 01/07/2021

To draw a shaded triangle, define a [TRIVERTEX](#) structure with three elements and a single [GRADIENT_TRIANGLE](#) structure. The following code example shows how to draw a shaded triangle using the [GradientFill](#) function with the GRADIENT_FILL_TRIANGLE mode defined.

C++

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {
        case WM_COMMAND:
            wmId     = LOWORD(wParam);
            wmEvent  = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:
            {hdc = BeginPaint(hWnd, &ps);
            // Create an array of TRIVERTEX structures that describe
            // positional and color values for each vertex.
            TRIVERTEX vertex[3];
            vertex[0].x      = 150;
            vertex[0].y      = 0;
            vertex[0].Red    = 0xff00;
            vertex[0].Green  = 0x8000;
            vertex[0].Blue   = 0x0000;
            vertex[0].Alpha  = 0x0000;

            vertex[1].x      = 0;
            vertex[1].y      = 150;
            vertex[1].Red    = 0x9000;
            vertex[1].Green  = 0x0000;
```

```

vertex[1].Blue  = 0x9000;
vertex[1].Alpha = 0x0000;

vertex[2].x      = 300;
vertex[2].y      = 150;
vertex[2].Red    = 0x9000;
vertex[2].Green  = 0x0000;
vertex[2].Blue   = 0x9000;
vertex[2].Alpha  = 0x0000;

// Create a GRADIENT_TRIANGLE structure that
// references the TRIVERTEX vertices.
GRADIENT_TRIANGLE gTriangle;
gTriangle.Vertex1 = 0;
gTriangle.Vertex2 = 1;
gTriangle.Vertex3 = 2;

// Draw a shaded triangle.
GradientFill(hdc, vertex, 3, &gTriangle, 1, GRADIENT_FILL_TRIANGLE);
    EndPaint(hWnd, &ps);
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

The following image shows the drawing output of the preceding code example.



Related topics

[Bitmaps Overview](#)

[Bitmap Functions](#)

[Drawing a Shaded Rectangle](#)

[EMRGRADIENTFILL](#)

GRADIENT_TRIANGLE

GradientFill

TRIVERTEX

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Testing a Printer for JPEG or PNG Support

Article • 01/07/2021

The [SetDIBitsToDevice](#) function uses color data from a DIB to set the pixels in the specified rectangle on the device that is associated with the destination device context.

[SetDIBitsToDevice](#) is extended to allow a JPEG or PNG image to be passed as the source image.

For example:

C++

```
//  
// pvJpgImage points to a buffer containing the JPEG image  
// nJpgImageSize is the size of the buffer  
// ulJpgWidth is the width of the JPEG image  
// ulJpgHeight is the height of the JPEG image  
  
//  
// Check if CHECKJPEGFORMAT is supported (device has JPEG support)  
// and use it to verify that device can handle the JPEG image.  
  
ul = CHECKJPEGFORMAT;  
  
if (  
    // Check if CHECKJPEGFORMAT exists:  
  
    (ExtEscape(hdc, QUERYESCSUPPORT,  
               sizeof(ul), &ul, 0, 0) > 0) &&  
  
    // Check if CHECKJPEGFORMAT executed without error:  
  
    (ExtEscape(hdc, CHECKJPEGFORMAT,  
               pvJpgImage, nJpgImageSize, sizeof(ul), &ul) > 0) &&  
  
    // Check status code returned by CHECKJPEGFORMAT:  
  
    (ul == 1)  
)  
{  
    //  
    // Initialize the BITMAPINFO.  
    //  
  
    memset(&bmi, 0, sizeof(bmi));
```

```
bmi.bmiHeader.biSize          = sizeof(BITMAPINFOHEADER);
bmi.bmiHeader.biWidth         = ulJpgWidth;
bmi.bmiHeader.biHeight        = -ulJpgHeight; // top-down image
bmi.bmiHeader.biPlanes        = 1;
bmi.bmiHeader.biBitCount      = 0;
bmi.bmiHeader.biCompression   = BI_JPEG;
bmi.bmiHeader.biSizeImage     = nJpgImageSize;

//
// Do the SetDIBitsToDevice.
//

iRet = SetDIBitsToDevice(hdc,
                         ulDstX, ulDstY,
                         ulDstWidth, ulDstHeight,
                         0, 0,
                         0, ulJpgHeight,
                         pvJpgImage,
                         &bmi,
                         DIB_RGB_COLORS);

if (iRet == GDI_ERROR)
    return FALSE;
}

else
{
    //
    // Decompress image into a DIB and call SetDIBitsToDevice
    // with the DIB instead.
    //
}
```

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Sizing a JPEG or PNG Image

Article • 01/07/2021

The [StretchDIBits](#) function copies the color data for a rectangle of pixels in a DIB 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, [StretchDIBits](#) compresses the rows and columns by using the specified raster operation.

[StretchDIBits](#) is extended to allow a JPEG or PNG image to be passed as the source image.

For example:

C++

```
// pvJpgImage points to a buffer containing the JPEG image
// nJpgImageSize is the size of the buffer
// ulJpgWidth is the width of the JPEG image
// ulJpgHeight is the height of the JPEG image
//

//
// Check if CHECKJPEGFORMAT is supported (device has JPEG support)
// and use it to verify that device can handle the JPEG image.
//

ul = CHECKJPEGFORMAT;

if (
    // Check if CHECKJPEGFORMAT exists:

    (ExtEscape(hdc, QUERYESCSUPPORT,
               sizeof(ul), &ul, 0, 0) > 0) &&

    // Check if CHECKJPEGFORMAT executed without error:

    (ExtEscape(hdc, CHECKJPEGFORMAT,
               nJpgImageSize, pvJpgImage, sizeof(ul), &ul) > 0) &&

    // Check status code returned by CHECKJPEGFORMAT:

    (ul == 1)
)
{
    //
    // Initialize the BITMAPINFO.
    //

    memset(&bmi, 0, sizeof(bmi));
}
```

```

bmi.bmiHeader.biSize          = sizeof(BITMAPINFOHEADER);
bmi.bmiHeader.biWidth         = ulJpgWidth;
bmi.bmiHeader.biHeight        = -ulJpgHeight; // top-down image
bmi.bmiHeader.biPlanes        = 1;
bmi.bmiHeader.biBitCount      = 0;
bmi.bmiHeader.biCompression   = BI_JPEG;
bmi.bmiHeader.biSizeImage     = nJpgImageSize;

//
// Do the StretchDIBits.
//

iRet = StretchDIBits(hdc,
                      // destination rectangle
                      ulDstX, ulDstY, ulDstWidth, ulDstHeight,
                      // source rectangle
                      0, 0, ulJpgWidth, ulJpgHeight,
                      pvJpgImage,
                      &bmi,
                      DIB_RGB_COLORS,
                      SRCCOPY);

if (iRet == GDI_ERROR)
    return FALSE;
}

else
{
    //
    // Decompress image into a DIB and call StretchDIBits
    // with the DIB instead.
    //
}

```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Reference

Article • 01/07/2021

The following elements are used with bitmaps:

- [Bitmap Functions](#)
- [Bitmap Structures](#)
- [Bitmap Macros](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Bitmap Functions (Windows GDI)

Article • 01/07/2021

The following functions are used with bitmaps.

[+] Expand table

Function	Description
AlphaBlend	Displays a bitmap with transparent or semitransparent pixels.
BitBlt	Performs a bit-block transfer.
CreateBitmap	Creates a bitmap.
CreateBitmapIndirect	Creates a bitmap.
CreateCompatibleBitmap	Creates a bitmap compatible with a device.
CreateDIBitmap	Creates a device-dependent bitmap (DDB) from a DIB.
CreateDIBSection	Creates a DIB that applications can write to directly.
ExtFloodFill	Fills an area of the display surface with the current brush.
GetBitmapDimensionEx	Gets the dimensions of a bitmap.
GetDIBColorTable	Retrieves RGB color values from a DIB section bitmap.
GetDIBits	Copies a bitmap into a buffer.
GetPixel	Gets the RGB color value of the pixel at a given coordinate.
GetStretchBltMode	Gets the current stretching mode.
GradientFill	Fills rectangle and triangle structures.
LoadBitmap	Loads a bitmap from a module's executable file.
MaskBlt	Combines the color data in the source and destination bitmaps.
PlgBlt	Performs a bit-block transfer.
SetBitmapDimensionEx	Sets the preferred dimensions to a bitmap.
SetDIBColorTable	Sets RGB values in a DIB.
SetDIBits	Sets the pixels in a bitmap using color data from a DIB.
SetDIBitsToDevice	Sets the pixels in a rectangle using color data from a DIB.

Function	Description
SetPixel	Sets the color for a pixel.
SetPixelV	Sets a pixel to the best approximation of a color.
SetStretchBltMode	Sets the bitmap stretching mode.
StretchBlt	Copies a bitmap and stretches or compresses it.
StretchDIBits	Copies the color data in a DIB.
TransparentBlt	Performs a bit-block transfer of color data.

Obsolete Functions

The following functions are provided only for compatibility with 16-bit versions of Microsoft Windows:

- [CreateDiscardableBitmap](#)
- [FloodFill](#)
- [GetBitmapBits](#)
- [SetBitmapBits](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

AlphaBlend function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 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](#)

BitBlt function (wingdi.h)

Article 10/13/2021

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

C++

```
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.

 Expand table

Value	Meaning
<code>BLACKNESS</code>	Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.)
<code>CAPTUREBLT</code>	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.
<code>DSTINVERT</code>	Inverts the destination rectangle.
<code>MERGECOPY</code>	Merges the colors of the source rectangle with the brush currently selected in <code>hdcDest</code> , by using the Boolean AND operator.
<code>MERGEPAINT</code>	Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator.
<code>NOMIRRORBITMAP</code>	Prevents the bitmap from being mirrored.
<code>NOTSRCCOPY</code>	Copies the inverted source rectangle to the destination.
<code>NOTSRCERASE</code>	Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color.
<code>PATCOPY</code>	Copies the brush currently selected in <code>hdcDest</code> , 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.
SRCCAND	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](#), [PlgBlt](#), 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**.

C++

```
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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](#)

CreateBitmap function (wingdi.h)

Article 12/06/2022

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

Syntax

C++

```
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). The buffer size expected, cj , can be calculated using the formula:

C++

```
    cj = (((nWidth * nPlanes * nBitCount + 15) >> 4) << 1) * nHeight;
```

If this parameter is **NULL**, then the contents of the new bitmap are 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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateCompatibleBitmap function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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:

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[CreateDIBSection](#)

[DeleteObject](#)

[SelectObject](#)

CreateDIBitmap function (wingdi.h)

Article 11/19/2022

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

Syntax

C++

```
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.

 Expand table

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 *lpblInit* 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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateDIBSection function (wingdi.h)

Article 11/19/2022

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

C++

```
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.

 Expand table

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:

[+] [Expand table](#)

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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ExtFloodFill function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
-------	---------

FLOODFILLBORDER	The fill area is bounded by the color specified by the <i>color</i> parameter. This style is identical to the filling performed by the FloodFill function.
FLOODFILLSURFACE	The fill area is defined by the color that is specified by <i>color</i> . 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

FloodFill function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GdiAlphaBlend function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

GdiGradientFill function (wingdi.h)

Article 10/13/2021

The **GdiGradientFill** function fills rectangle and triangle structures.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Bitmap Functions](#)

[Bitmaps Overview](#)

[EMRGRADIENTFILL](#)

[GRADIENT_RECT](#)

[GRADIENT_TRIANGLE](#)

[TRIVERTEX](#)

GdiTransparentBlt function (wingdi.h)

Article 10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

GetBitmapBits function (wingdi.h)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

The **GetBitmapDimensionEx** function retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the [SetBitmapDimensionEx](#) function.

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetDIBColorTable function (wingdi.h)

Article11/19/2022

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 12/06/2022

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

C++

```
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.

[+] Expand table

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 *lpvBits* parameter is non-NULL and the function succeeds, the return value is the number of scan lines copied from the bitmap.

If the *lpvBits* 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.

[+] Expand table

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.

[+] Expand table

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 *lpvBits* 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 *lpvBits* is **NULL**, [GetDIBits](#) examines the first member of the first structure pointed to by *lpbi*. 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 *lpvBits* 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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

GetPixel function (wingdi.h)

Article11/05/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetStretchBltMode function (wingdi.h)

Article02/22/2024

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

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GradientFill function (wingdi.h)

Article10/13/2021

The **GradientFill** function fills rectangle and triangle structures.

Syntax

C++

```
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
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

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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 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](#)

LoadBitmapA function (winuser.h)

Article11/20/2024

[[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

C++

```
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.

[+] [Expand table](#)

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 that 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

[] [Expand table](#)

Requirement	Value
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](#)

MaskBlt function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PlgBlt function (wingdi.h)

Article10/13/2021

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

C++

```
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 **PlgBlt** 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, **PigBlt** converts the source rectangle to match the destination rectangle.

Not all devices support the **PigBlt** 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, **PigBlt** 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetBitmapBits function (wingdi.h)

Article02/22/2024

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

C++

```
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 *lpBits* 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 *lpBits* must be WORD aligned.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetDIBColorTable function (wingdi.h)

Article 02/22/2024

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

C++

```
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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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.

 [Expand table](#)

Value	Meaning
<code>DIB_PAL_COLORS</code>	The color table consists of an array of 16-bit indexes into the logical palette of the device context identified by the <code>hdc</code> parameter.
<code>DIB_RGB_COLORS</code>	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.

 [Expand table](#)

Return code	Description
<code>ERROR_INVALID_PARAMETER</code>	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 *iEnableICM* 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/19/2022

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

C++

```
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.

 Expand table

Value	Meaning
<code>DIB_PAL_COLORS</code>	The color table consists of an array of 16-bit indexes into the currently selected logical palette.
<code>DIB_RGB_COLORS</code>	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 *iEnableICM* 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetPixel function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[Expand table](#)

Return code	Description
ERROR_INVALID_PARAMETER	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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetStretchBltMode function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>BLACKONWHITE</code>	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.
<code>COLORONCOLOR</code>	Deletes the pixels. This mode deletes all eliminated lines of pixels without trying to preserve their information.
<code>HALFTONE</code>	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 <code>HALFTONE</code> 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.

 Expand table

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_DELETESCANS) 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

StretchBlt function (wingdi.h)

Article 10/13/2021

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

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article11/19/2022

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

C++

```
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.

 [Expand table](#)

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 *iEnableCM* 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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

TransparentBlt function (wingdi.h)

Article 10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 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](#)

Bitmap Structures

Article • 11/19/2022

The following structures are used with bitmaps:

- [BITMAP](#)
- [BITMAPCOREHEADER](#)
- [BITMAPCOREINFO](#)
- [BITMAPFILEHEADER](#)
- [BITMAPINFO](#)
- [BITMAPINFOHEADER](#)
- [BITMAPV4HEADER](#)
- [BITMAPV5HEADER](#)
- [BLENDFUNCTION](#)
- [COLORADJUSTMENT](#)
- [DIBSECTION](#)
- [GRADIENT_RECT](#)
- [GRADIENT_TRIANGLE](#)
- [RGBQUAD](#)
- [RGBTRIPLE](#)
- [SIZE](#)
- [TRIVERTEX](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

BITMAP structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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 :

syntax

```
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateBitmapIndirect](#)

[GetDIBits](#)

[GetDeviceCaps](#)

[GetObject](#)

[SetDIBits](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPCOREHEADER structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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:

C++

```
pColor = ((LPBYTE) pBitmapCoreInfo +  
          (WORD) (pBitmapCoreInfo -> bcSize))
```

Requirements

 [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPCOREINFO structure (wingdi.h)

Article09/01/2022

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

Syntax

C++

```
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[1]`

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.

[+] Expand table

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. |
| 24 | 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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[BITMAPCOREHEADER](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

[CreateDIBPatternBrush](#)

[CreateDIBSection](#)

[CreateDIBitmap](#)

[RGBTRIPLE](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPFILEHEADER structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER, *LPBITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

Members

bfType

The file type; must be `0x4d42` (the ASCII string "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](#).

Examples

For an example, see [Storing an image](#).

Requirements

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPINFO structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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[1]

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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPINFOHEADER structure (wingdi.h)

Article01/26/2024

[The feature associated with this page, [DirectShow](#), is a legacy feature. It has been superseded by [MediaPlayer](#), [IMFMediaEngine](#), and [Audio/Video Capture in Media Foundation](#). Those features have been optimized for Windows 10 and Windows 11. Microsoft strongly recommends that new code use [MediaPlayer](#), [IMFMediaEngine](#) and [Audio/Video Capture in Media Foundation](#) instead of [DirectShow](#), when possible. Microsoft suggests that existing code that uses the legacy APIs be rewritten to use the new APIs if possible.]

The **BITMAPINFOHEADER** structure contains information about the dimensions and color format of a device-independent bitmap (DIB).

Note This structure is also described in the GDI documentation. However, the semantics for video data are slightly different than the semantics used for GDI. If you are using this structure to describe video data, use the information given here.

Syntax

C++

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

Members

biSize

Specifies the number of bytes required by the structure. This value does not include the size of the color table or the size of the color masks, if they are appended to the end of structure. See Remarks.

biWidth

Specifies the width of the bitmap, in pixels. For information about calculating the stride of the bitmap, see Remarks.

biHeight

Specifies the height of the bitmap, in pixels.

- For uncompressed RGB bitmaps, if **biHeight** is positive, the bitmap is a bottom-up DIB with the origin at the lower left corner. If **biHeight** is negative, the bitmap is a top-down DIB with the origin at the upper left corner.
- For YUV bitmaps, the bitmap is always top-down, regardless of the sign of **biHeight**. Decoders should offer YUV formats with positive **biHeight**, but for backward compatibility they should accept YUV formats with either positive or negative **biHeight**.
- For compressed formats, **biHeight** must be positive, regardless of image orientation.

biPlanes

Specifies the number of planes for the target device. This value must be set to 1.

biBitCount

Specifies the number of bits per pixel (bpp). For uncompressed formats, this value is the average number of bits per pixel. For compressed formats, this value is the implied bit depth of the uncompressed image, after the image has been decoded.

biCompression

For compressed video and YUV formats, this member is a FOURCC code, specified as a **DWORD** in little-endian order. For example, YUYV video has the FOURCC 'VYUY' or 0x56595559. For more information, see [FOURCC Codes](#).

For uncompressed RGB formats, the following values are possible:

[+] Expand table

Value	Meaning
BI_RGB	Uncompressed RGB.
BI_BITFIELDS	Uncompressed RGB with color masks. Valid for 16-bpp and 32-bpp bitmaps.

See Remarks for more information. Note that **BI_JPG** and **BI_PNG** are not valid video formats.

For 16-bpp bitmaps, if **biCompression** equals **BI_RGB**, the format is always RGB 555. If **biCompression** equals **BI_BITFIELDS**, the format is either RGB 555 or RGB 565. Use the subtype GUID in the [AM_MEDIA_TYPE](#) structure to determine the specific RGB type.

biSizeImage

Specifies the size, in bytes, of the image. This can be set to 0 for uncompressed RGB bitmaps.

biXPelsPerMeter

Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap.

biYPelsPerMeter

Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.

biClrUsed

Specifies the number of color indices in the color table that are actually used by the bitmap. See Remarks for more information.

biClrImportant

Specifies the number of color indices that are considered important for displaying the bitmap. If this value is zero, all colors are important.

Remarks

Color Tables

The **BITMAPINFOHEADER** structure may be followed by an array of palette entries or color masks. The rules depend on the value of **biCompression**.

- If **biCompression** equals **BI_RGB** and the bitmap uses 8 bpp or less, the bitmap has a color table immediately following the **BITMAPINFOHEADER** structure. The color table consists of an array of **RGBQUAD** values. The size of the array is given by the **biClrUsed** member. If **biClrUsed** is zero, the array contains the maximum number of colors for the given bitdepth; that is, $2^{\text{biBitCount}}$ colors.
- If **biCompression** equals **BI_BITFIELDS**, the bitmap uses three **DWORD** color masks (red, green, and blue, respectively), which specify the byte layout of the pixels. The 1 bits in each mask indicate the bits for that color within the pixel.
- If **biCompression** is a video FOURCC, the presence of a color table is implied by the video format. You should not assume that a color table exists when the bit depth is 8 bpp or less. However, some legacy components might assume that a color table is present. Therefore, if you are allocating a **BITMAPINFOHEADER** structure, it is recommended to allocate space for a color table when the bit depth is 8 bpp or less, even if the color table is not used.

When the **BITMAPINFOHEADER** is followed by a color table or a set of color masks, you can use the **BITMAPINFO** structure to reference the color table of the color masks. The **BITMAPINFO** structure is defined as follows:

syntax

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD          bmiColors[1];
} BITMAPINFO;
```

If you cast the **BITMAPINFOHEADER** to a **BITMAPINFO**, the **bmiHeader** member refers to the **BITMAPINFOHEADER** and the **bmiColors** member refers to the first entry in the color table, or the first color mask.

Be aware that if the bitmap uses a color table or color masks, then the size of the entire format structure (the **BITMAPINFOHEADER** plus the color information) is not equal to `sizeof(BITMAPINFOHEADER)` or `sizeof(BITMAPINFO)`. You must calculate the actual size for each instance.

Calculating Surface Stride

In an uncompressed bitmap, the stride is the number of bytes needed to go from the start of one row of pixels to the start of the next row. The image format defines a

minimum stride for an image. In addition, the graphics hardware might require a larger stride for the surface that contains the image.

For uncompressed RGB formats, the minimum stride is always the image width in bytes, rounded up to the nearest **DWORD**. To calculate the stride and image size, you can use the **GDI_DIBWIDTHBYTES** and/or **GDI_DIBSIZE** macros, or the following formula:

C++

```
stride = (((biWidth * biBitCount) + 31) & ~31) >> 3;
biSizeImage = abs(biHeight) * stride;
```

For YUV formats, there is no general rule for calculating the minimum stride. You must understand the rules for the particular YUV format. For a description of the most common YUV formats, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Decoders and video sources should propose formats where **biWidth** is the width of the image in pixels. If the video renderer requires a surface stride that is larger than the default image stride, it modifies the proposed media type by setting the following values:

- It sets **biWidth** equal to the surface stride in pixels.
- It sets the **rcTarget** member of the [VIDEOINFOHEADER](#) or [VIDEOINFOHEADER2](#) structure equal to the image width, in pixels.

Then the video renderer proposes the modified format by calling [IPin::QueryAccept](#) on the upstream pin. For more information about this mechanism, see [Dynamic Format Changes](#).

If there is padding in the image buffer, never dereference a pointer into the memory that has been reserved for the padding. If the image buffer has been allocated in video memory, the padding might not be readable memory.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	wingdi.h

See also

[DirectShow Structures](#)

[VIDEOINFOHEADER Structure](#)

[VIDEOINFOHEADER2 Structure](#)

[Working with Video Frames](#)

Feedback

Was this page helpful?

 Yes

 No

BITMAPV4HEADER structure (wingdi.h)

Article03/13/2023

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

C++

```
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.

 Expand table

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**.
- 32 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.

[+] [Expand table](#)

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**.

 [Expand table](#)

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

 [Expand table](#)

Requirement	Value
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)

Article03/13/2023

The **BITMAPV5HEADER** structure is the bitmap information header file. It is an extended version of the [BITMAPINFOHEADER](#) structure.

Syntax

C++

```
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.

 Expand table

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	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 position.

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.

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.

- | | |
|----|--|
| 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.

[Expand table](#)

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**.

 Expand table

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.

 [Expand table](#)

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

Requirement	Value
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)

Article04/02/2021

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

Syntax

C++

```
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.

[+] Expand table

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.

[\[+\] Expand table](#)

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.

[\[+\] Expand table](#)

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.

[\[+\] Expand 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.

[\[+\] Expand table](#)

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.

[\[+\] Expand table](#)

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

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[AlphaBlend](#)

[Bitmap Structures](#)

[Bitmaps Overview](#)

Feedback

Was this page helpful?

 Yes

 No

COLORADJUSTMENT structure (wingdi.h)

Article 02/22/2024

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

C++

```
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.

[+] Expand table

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.

[\[\] Expand table](#)

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

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

DIBSECTION structure (wingdi.h)

Article11/19/2022

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

C++

```
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[3]

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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

GRADIENT_RECT structure (wingdi.h)

Article02/22/2024

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

C++

```
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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

👍 Yes

👎 No

GRADIENT_TRIANGLE structure (wingdi.h)

Article04/02/2021

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

C++

```
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

[\[+\] Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

Yes

No

RGBQUAD structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

RGBTRIPLE structure (wingdi.h)

Article02/22/2024

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

C++

```
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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

SIZE structure (windef.h)

Article02/22/2024

The SIZE structure defines the width and height of a rectangle.

Syntax

C++

```
typedef struct tagSIZE {
    LONG cx;
    LONG cy;
} SIZE, *PSIZE, *LPSIZE;
```

Members

`cx`

Specifies the rectangle's width. The units depend on which function uses this structure.

`cy`

Specifies the rectangle's height. The units depend on which function uses this structure.

Remarks

The rectangle dimensions stored in this structure can correspond to viewport extents, window extents, text extents, bitmap dimensions, or the aspect-ratio filter for some extended functions.

Requirements

[+] Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

RECT

RECTL

Feedback

Was this page helpful?

 Yes

 No

TRIVERTEX structure (wingdi.h)

Article 02/22/2024

The **TRIVERTEX** structure contains color information and position information.

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

Bitmap Macros

Article • 03/11/2025

The following macro is used with bitmaps.

MAKEROP4

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

MAKEROP4 macro (wingdi.h)

07/09/2025

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

C++

```
DWORD MAKEROP4(  
    DWORD fore,  
    DWORD back  
)
```

Parameters

`fore`

The foreground ternary raster operation code.

`back`

The background ternary raster operation code.

Return value

Type: [DWORD](#)

The return value is a quaternary raster operation code for use with the [MaskBlt](#) function.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
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](#)

Brushes (Windows GDI)

Article • 01/07/2021

A *brush* is a graphics tool that applications use to paint the interior of polygons, ellipses, and paths. Drawing applications use brushes to paint shapes; word processing applications use brushes to paint rules; computer-aided design (CAD) applications use brushes to paint the interiors of cross-section views; and spreadsheet applications use brushes to paint the sections of pie charts and the bars in bar graphs.

- [About Brushes](#)
- [Using Brushes](#)
- [Brush Reference](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Brushes

Article • 01/07/2021

There are two types of brushes: logical and physical. A *logical brush* is a description of the ideal bitmap that an application uses to paint shapes. A *physical brush* is the actual bitmap that a device driver creates based on an application's logical-brush definition. For more information about bitmaps, see [Bitmaps](#).

When an application calls one of the functions that creates a brush, it retrieves a handle that identifies a logical brush. When the application passes this handle to the [SelectObject](#) function, the device driver for the corresponding display or printer creates the physical brush.

The following topics describe brushes:

- [Brush Origin](#)
- [Logical Brush Types](#)
- [Pattern Block Transfer](#)
- [ICM-Enabled Brush Functions](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

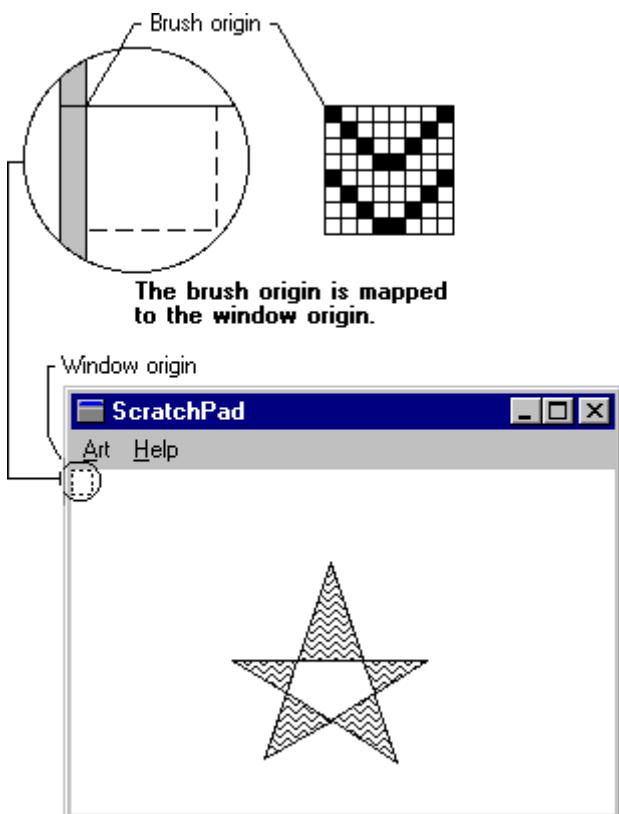
Brush Origin

Article • 01/07/2021

When an application calls a drawing function to paint a shape, the system positions a brush at the start of the paint operation and maps a pixel in the brush bitmap to the client area at the *window origin*, which is the upper-left corner of the window. The coordinates of the pixel that the system maps are called the *brush origin*. The default brush origin is located in the upper-left corner of the brush bitmap, at the coordinates (0,0). The system then copies the brush across the client area, forming a pattern that is as tall as the bitmap. The copy operation continues, row by row, until the entire client area is filled. However, the brush pattern is visible only within the boundaries of the specified shape.

There are instances when the default brush origin should not be used. For example, it may be necessary for an application to use the same brush to paint the backgrounds of its parent and child windows and blend a child window's background with that of the parent window. To do this, the application should reset the brush origin by calling the [SetBrushOrgEx](#) function and shifting the origin the required number of pixels. (An application can retrieve the current brush origin by calling the [GetBrushOrgEx](#) function.)

The following illustration shows a five-pointed star filled by using an application-defined brush. The illustration shows a zoomed image of the brush, as well as the location to which it was mapped at the beginning of the paint operation.



Feedback

Was this page helpful?

 Yes

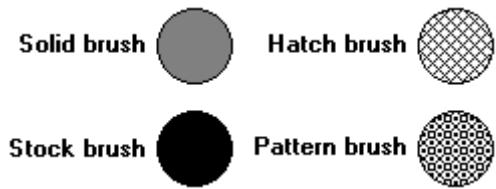
 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Logical Brush Types

Article • 01/07/2021

There are four types of logical brushes: [solid](#), [stock](#), [hatch](#), and [pattern](#). These brushes are shown in the following illustration.



The stock and hatch types each have several predefined brushes.

The [CreateBrushIndirect](#) function creates a logical brush with a specified style, color, and pattern.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Solid Brush

Article • 01/07/2021

A *solid brush* is a logical brush that contains 64 pixels of the same color. An application can create a solid logical brush by calling the [CreateSolidBrush](#) function, specifying the color of the brush required. After creating the solid brush, the application can select it into its device context and use it to paint filled shapes.

Feedback

Was this page helpful?

 Yes

 No

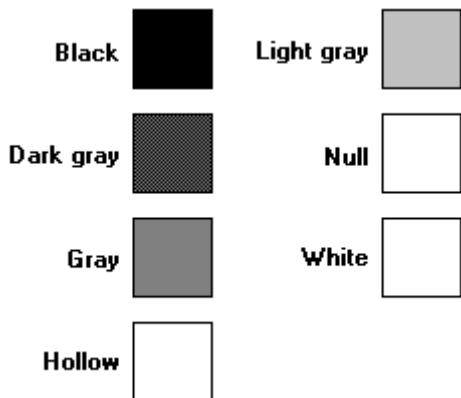
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Stock Brush

Article • 01/07/2021

There are seven predefined logical stock brushes maintained by the graphics device interface (GDI). There are also 21 predefined logical stock brushes maintained by the window management interface (USER).

The following illustration shows rectangles painted by using the seven predefined stock brushes.



An application can retrieve a handle identifying one of the seven stock brushes by calling the [GetStockObject](#) function, specifying the brush type.

The 21 stock brushes maintained by the window management interface correspond to the colors of window elements such as menus, scroll bars, and buttons. An application can obtain a handle identifying one of these brushes by calling the [GetSysColorBrush](#) function and specifying a system-color value. An application can retrieve the color corresponding to a particular window element by calling the [GetSysColor](#) function. An application can set the color corresponding to a window element by calling the [SetSysColors](#) function.

Feedback

Was this page helpful?

Yes

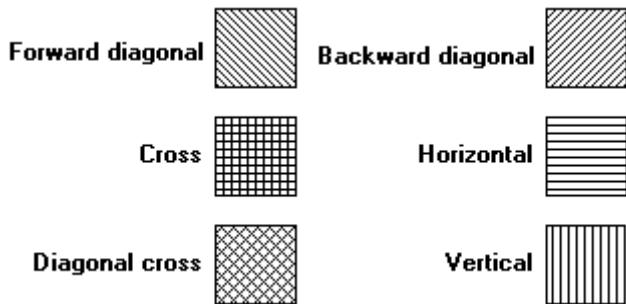
No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Hatch Brush

Article • 01/07/2021

There are six predefined logical hatch brushes maintained by GDI. The following rectangles were painted by using the six predefined hatch brushes.



An application can create a hatch brush by calling the [CreateHatchBrush](#) function, specifying one of the six hatch styles.

Feedback

Was this page helpful?

Yes

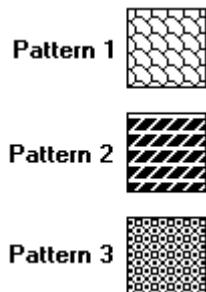
No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pattern Brush

Article • 01/07/2021

A pattern (or custom) brush is created from an application-defined bitmap or device-independent bitmap (DIB). The following rectangles were painted by using different pattern brushes.



To create a logical pattern brush, an application must first create a bitmap. After creating the bitmap, the application can create the logical pattern brush by calling the [CreatePatternBrush](#) or [CreateDIBPatternBrushPt](#) function, supplying a handle that identifies the bitmap (or DIB). The brushes that appear in the preceding illustration were created from monochrome bitmaps. For a description of bitmaps, DIBs, and the functions that create them, see [Bitmaps](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pattern Block Transfer

Article • 01/07/2021

The name of the [PatBlt](#) function (an abbreviation for pattern block transfer) implies that this function simply replicates the brush (or pattern) until it fills a specified rectangle. However, the function is actually much more powerful. Before replicating the brush, it combines the color data for the pattern with the color data for the existing pixels on the video display by using a raster operation (ROP). An ROP is a bitwise operation that is applied to the bits of color data for the replicated brush and the bits of color data for the target rectangle on the display device. There are 256 ROPs; however, the [PatBlt](#) function recognizes only those that require a pattern and a destination (not those that require a source). The following table identifies the most common ROPs.

[Expand table](#)

ROP	Description
PATCOPY	Copies the pattern to the destination bitmap.
PATINVERT	Combines the destination bitmap with the pattern by using the Boolean XOR operator.
DSTINVERT	Inverts the destination bitmap.
BLACKNESS	Turns all output to binary zeros.
WHITENESS	Turns all output to binary ones.

For more information, see [Raster Operation Codes](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ICM-Enabled Brush Functions

Article • 01/07/2021

Microsoft Image Color Management (ICM) ensures that a color image, graphic, or text object is rendered as close as possible to its original intent on any device, despite differences in imaging technologies and color capabilities between devices. Whether you are scanning an image or other graphic on a color scanner, downloading it over the Internet, viewing or editing it on the screen, or outputting it to paper, film, or other media, ICM 2.0 helps you keep its colors consistent and accurate. For more information about ICM, see [Windows Color System](#).

The following brush functions are enabled for use with ICM:

- [CreateBrushIndirect](#)
- [CreateDIBPatternBrush](#)
- [CreateDIBPatternBrushPt](#)
- [CreateHatchBrush](#)
- [CreatePatternBrush](#)
- [CreateSolidBrush](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

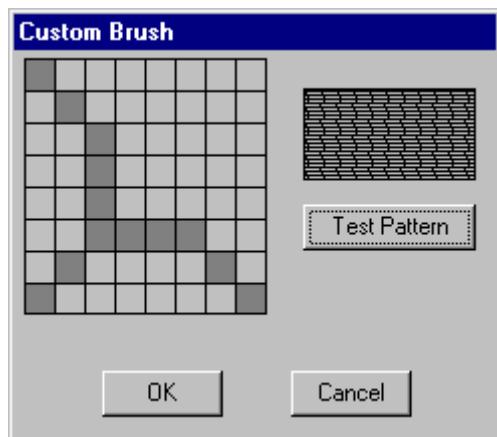
Using Brushes

Article • 01/07/2021

You can use a brush to paint the interior of virtually any shape by using a graphics device interface (GDI) function. This includes the interiors of rectangles, ellipses, polygons, and paths. Depending on the requirements of your application, you can use a solid brush of a specified color, a stock brush, a hatch brush, or a pattern brush.

This section contains code samples that demonstrate the creation of a custom brush dialog box. The dialog box contains a grid that represents the bitmap the system uses as a brush. A user can use this grid to create a pattern-brush bitmap and then view the custom pattern by clicking the **Test Pattern** button.

The following illustration shows a pattern created by using the **Custom Brush** dialog box.



To display a dialog box, you must first create a dialog box template. The following dialog box template defines the **Custom Brush** dialog box.

C++

```
CustBrush DIALOG 6, 18, 160, 118
STYLE WS_DLGFREAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Custom Brush"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL      "", IDD_GRID, "Static", SS_BLACKFRAME |
                  WS_CHILD, 3, 2, 83, 79
    CONTROL      "", IDD_RECT, "Static", SS_BLACKFRAME |
                  WS_CHILD, 96, 11, 57, 28
    PUSHBUTTON   "Test Pattern", IDD_PAINTRECT, 96, 47, 57, 14
    PUSHBUTTON   "OK", IDD_OK, 29, 98, 40, 14
    PUSHBUTTON   "Cancel", IDD_CANCEL, 92, 98, 40, 14
END
```

The **Custom Brush** dialog box contains five controls: a bitmap-grid window, a pattern-viewing window, and three push buttons, labeled **Test Pattern**, **OK**, and **Cancel**. The **Test Pattern** push button enables the user to view the pattern. The dialog box template specifies the overall dimensions of the dialog box window, assigns a value to each control, specifies the location of each control, and so forth. For more information, see [Dialog Boxes](#).

The control values in the dialog box template are constants that have been defined as follows in the application's header file.

C++

```
#define IDD_GRID      120
#define IDD_RECT      121
#define IDD_PAINTRECT 122
#define IDD_OK        123
#define IDD_CANCEL    124
```

After you create a dialog box template and include it in the application's resource-definition file, you must write a dialog procedure. This procedure processes messages that the system sends to the dialog box. The following excerpt from an application's source code shows the dialog box procedure for the **Custom Brush** dialog box and the two application-defined functions it calls.

C++

```
BOOL CALLBACK BrushDlgProc( HWND hdlg, UINT message, LONG wParam,
                           LONG lParam)
{
    static HWND hwndGrid;           // grid-window control
    static HWND hwndBrush;          // pattern-brush control
    static RECT rctGrid;            // grid-window rectangle
    static RECT rctBrush;            // pattern-brush rectangle
    static UINT bBrushBits[8];       // bitmap bits
    static RECT rect[64];           // grid-cell array
    static HBITMAP hbm;             // bitmap handle
    HBRUSH hbrush;                 // current brush
    HBRUSH hbrushOld;              // default brush
    Hrgn hrgnCell;                // test-region handle
    HDC hdc;                       // device context (DC) handle
    int x, y, deltaX, deltaY;       // drawing coordinates
    POINTS ptlHit;                 // mouse coordinates
    int i;                          // count variable

    switch (message)
    {
        case WM_INITDIALOG:
            // Retrieve a window handle for the grid-window and
```

```

// pattern-brush controls.

hwndGrid = GetDlgItem(hdlg, IDD_GRID);
hwndBrush = GetDlgItem(hdlg, IDD_RECT);

// Initialize the array of bits that defines the
// custom brush pattern with the value 1 to produce a
// solid white brush.

for (i=0; i<8; i++)
    bBrushBits[i] = 0xFF;

// Retrieve dimensions for the grid-window and pattern-
// brush controls.

GetClientRect(hwndGrid, &rctGrid);
GetClientRect(hwndBrush, &rctBrush);

// Determine the width and height of a single cell.

deltaX = (rctGrid.right - rctGrid.left)/8;
deltaY = (rctGrid.bottom - rctGrid.top)/8;

// Initialize the array of cell rectangles.

for (y=rctGrid.top, i=0; y < rctGrid.bottom; y += deltaY)
{
    for(x=rctGrid.left; x < (rctGrid.right - 8) && i < 64;
        x += deltaX, i++)
    {
        rect[i].left = x; rect[i].top = y;
        rect[i].right = x + deltaX;
        rect[i].bottom = y + deltaY;
    }
}
return FALSE;

case WM_PAINT:

// Draw the grid.

hdc = GetDC(hwndGrid);

for (i=rctGrid.left; i<rctGrid.right;
     i+=(rctGrid.right - rctGrid.left)/8)
{
    MoveToEx(hdc, i, rctGrid.top, NULL);
    LineTo(hdc, i, rctGrid.bottom);
}
for (i=rctGrid.top; i<rctGrid.bottom;
     i+=(rctGrid.bottom - rctGrid.top)/8)
{
    MoveToEx(hdc, rctGrid.left, i, NULL);
    LineTo(hdc, rctGrid.right, i);
}

```

```
        }

    ReleaseDC(hwndGrid, hdc);
    return FALSE;

case WM_LBUTTONDOWN:

    // Store the mouse coordinates in a POINT structure.

    ptlHit = MAKEPOINTS((POINTS FAR *)lParam);

    // Create a rectangular region with dimensions and
    // coordinates that correspond to those of the grid
    // window.

    hrgnCell = CreateRectRgn(rctGrid.left, rctGrid.top,
                             rctGrid.right, rctGrid.bottom);

    // Retrieve a window DC for the grid window.

    hdc = GetDC(hwndGrid);

    // Select the region into the DC.

    SelectObject(hdc, hrgnCell);

    // Test for a button click in the grid-window rectangle.

    if (PtInRegion(hrgnCell, ptlHit.x, ptlHit.y))
    {
        // A button click occurred in the grid-window
        // rectangle; isolate the cell in which it occurred.

        for(i=0; i<64; i++)
        {
            DeleteObject(hrgnCell);

            hrgnCell = CreateRectRgn(rect[i].left,
                                    rect[i].top,
                                    rect[i].right, rect[i].bottom);

            if (PtInRegion(hrgnCell, ptlHit.x, ptlHit.y))
            {
                InvertRgn(hdc, hrgnCell);

                // Set the appropriate brush bits.

                if (i % 8 == 0)
                    bBrushBits[i/8] = bBrushBits[i/8] ^ 0x80;
                else if (i % 8 == 1)
                    bBrushBits[i/8] = bBrushBits[i/8] ^ 0x40;
                else if (i % 8 == 2)
                    bBrushBits[i/8] = bBrushBits[i/8] ^ 0x20;
                else if (i % 8 == 3)
                    bBrushBits[i/8] = bBrushBits[i/8] ^ 0x10;
            }
        }
    }
}
```

```

        else if (i % 8 == 4)
            bBrushBits[i/8] = bBrushBits[i/8] ^ 0x08;
        else if (i % 8 == 5)
            bBrushBits[i/8] = bBrushBits[i/8] ^ 0x04;
        else if (i % 8 == 6)
            bBrushBits[i/8] = bBrushBits[i/8] ^ 0x02;
        else if (i % 8 == 7)
            bBrushBits[i/8] = bBrushBits[i/8] ^ 0x01;

        // Exit the "for" loop after the bit is set.

        break;
    } // end if

} // end for

} // end if

// Release the DC for the control.

ReleaseDC(hwndGrid, hdc);
return TRUE;

case WM_COMMAND:
switch (wParam)
{
    case IDD_PAINTRECT:

        hdc = GetDC(hwndBrush);

        // Create a monochrome bitmap.

        hbm = CreateBitmap(8, 8, 1, 1,
                           (LPBYTE)bBrushBits);

        // Select the custom brush into the DC.

        hbrush = CreatePatternBrush(hbm);

        hbrushOld = SelectObject(hdc, hbrush);

        // Use the custom brush to fill the rectangle.

        Rectangle(hdc, rctBrush.left, rctBrush.top,
                  rctBrush.right, rctBrush.bottom);

        // Clean up memory.
        SelectObject(hdc, hbrushOld);
        DeleteObject(hbrush);
        DeleteObject(hbm);

        ReleaseDC(hwndBrush, hdc);
        return TRUE;
}

```

```

        case IDD_OK:

        case IDD_CANCEL:
            EndDialog(hdlg, TRUE);
            return TRUE;

    } // end switch
    break;
default:
    return FALSE;
}
}

int GetStrLength(LPTSTR cArray)
{
    int i = 0;

    while (cArray[i++] != 0);
    return i-1;

}

DWORD RetrieveWidth(LPTSTR cArray, int iLength)
{
    int i, iTmp;
    double dVal, dCount;

    dVal = 0.0;
    dCount = (double)(iLength-1);
    for (i=0; i<iLength; i++)
    {
        iTmp = cArray[i] - 0x30;
        dVal = dVal + (((double)iTmp) * pow(10.0, dCount--));
    }

    return (DWORD)dVal;
}

```

The dialog box procedure for the **Custom Brush** dialog box processes four messages, as described in the following table.

[\[+\] Expand table](#)

Message	Action
WM_INITDIALOG	Retrieves a window handle and dimensions for the grid-window and pattern-brush controls, computes the dimensions of a single cell in the grid-window control, and initializes an array of grid-cell coordinates.
WM_PAINT	Draws the grid pattern in the grid-window control.

Message	Action
WM_LBUTTONDOWN	Determines whether the cursor is within the grid-window control when the user presses the left mouse button. If so, the dialog box procedure inverts the appropriate grid cell and records the state of that cell in an array of bits that is used to create the bitmap for the custom brush.
WM_COMMAND	Processes input for the three push button controls. If the user clicks the Test Pattern button, the dialog box procedure paints the Test Pattern control with the new custom brush pattern. If the user clicks the OK or Cancel button, the dialog box procedure performs actions accordingly.

For more information about messages and message processing, see [Messages and Message Queues](#).

After you write the dialog box procedure, include the function definition for the procedure in the application's header file and then call the dialog box procedure at the appropriate point in the application.

The following excerpt from the application's header file shows the function definition for the dialog box procedure and the two functions it calls.

C++

```
BOOL CALLBACK BrushDlgProc(HWND, UINT, WPARAM, LPARAM);
int GetStrLength(LPTSTR);
DWORD RetrieveWidth(LPTSTR, int);
```

Finally, the following code shows how the dialog box procedure is called from the application's source-code file.

C++

```
DialogBox((HANDLE)GetModuleHandle(NULL),
(LPTSTR)"CustBrush",
hWnd,
(DLGPROC) BrushDlgProc);
```

This call is usually made in response to the user choosing an option from the application's menu.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Brush Reference

Article • 01/07/2021

The following elements are used with brushes:

- [Brush Functions](#)
- [Brush Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Brush Functions (Windows GDI)

Article • 01/07/2021

The following functions are used with brushes.

[+] Expand table

Function	Description
CreateBrushIndirect	Creates a brush with a specified style, color, and pattern
CreateDIBPatternBrushPt	Creates a brush with the pattern from a DIB
CreateHatchBrush	Creates a brush with a hatch pattern and color
CreatePatternBrush	Creates a brush with a bitmap pattern
CreateSolidBrush	Creates a brush with a solid color
GetBrushOrgEx	Gets the brush origin for a device context
GetSysColorBrush	Gets a handle to a brush that corresponds to a color index
PatBlt	Paints a rectangle
SetBrushOrgEx	Sets the brush origin for a device context
SetDCBrushColor	Sets the current device context brush color.

Obsolete Functions

The following functions are provided only for compatibility with 16-bit versions of Windows.

[CreateDIBPatternBrush](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

CreateBrushIndirect function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateDIBPatternBrush function (wingdi.h)

Article 02/22/2024

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

C++

```
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.

 Expand table

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

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

Syntax

C++

```
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.

 Expand table

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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateHatchBrush function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

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.

C++

```
#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:  
LRESULT 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  
//  
//  
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreatePatternBrush function (wingdi.h)

Article10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateSolidBrush function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetBrushOrgEx function (wingdi.h)

Article11/19/2022

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetSysColorBrush function (winuser.h)

02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

PatBlt function (wingdi.h)

Article 10/13/2021

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

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetBrushOrgEx function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Brush Structures

Article • 01/07/2021

The following structures are used with brushes:

- [LOGBRUSH](#)
- [LOGBRUSH32](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

LOGBRUSH structure (wingdi.h)

Article04/02/2021

The **LOGBRUSH** structure defines the style, color, and pattern of a physical brush. It is used by the [CreateBrushIndirect](#) and [ExtCreatePen](#) functions.

Syntax

C++

```
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.

[+] Expand table

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.
----------	--------------

1bColor

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.

[\[+\] Expand table](#)

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.

[+] Expand table

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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

LOGBRUSH32 structure (wingdi.h)

Article04/02/2021

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

C++

```
typedef struct tagLOGBRUSH32 {
    UINT     lbStyle;
    COLORREF lbColor;
    ULONG    lbHatch;
} LOGBRUSH32, *PLOGBRUSH32, *NPLOGBRUSH32, *LPLOGBRUSH32;
```

Members

lbStyle

The brush style. The **lbStyle** member must be one of the following styles.

[+] Expand table

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	Same as 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	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, **lbColor** is ignored.

If **IbStyle** is BS_DIBPATTERN or BS_DIBPATTERNPT, 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 **lbColor** member must be one of the following values.

[\[+\] Expand table](#)

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, **lbColor** is a [COLORREF](#) color value. To create a [COLORREF](#) color value, use the [RGB](#) macro.

lbHatch

A hatch style. The meaning depends on the brush style defined by **IbStyle**.

If **IbStyle** is BS_DIBPATTERN, the **lbHatch** 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 **lbHatch** 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.

[+] [Expand table](#)

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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

Clipping (Windows GDI)

Article • 01/07/2021

Clipping is the process of limiting output to a region or path within the client area of an application window. The following sections discuss clipping.

- [About Clipping](#)
- [Using Clipping](#)
- [Clipping Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Clipping

Article • 01/07/2021

Clipping is used by applications in a variety of ways. Word processing and spreadsheet applications clip keyboard input to keep it from appearing in the margins of a page or spreadsheet. Computer-aided design (CAD) and drawing applications clip graphics output to keep it from overwriting the edges of a drawing or picture.

A *clipping region* is a region with edges that are either straight lines or curves. A *clip path* is a region with edges that are straight lines, Bézier curves, or combinations of both. For more information about regions, see [Regions](#). For more information about paths, see [Paths](#).

This overview covers the following topics:

- [Clipping Regions](#)
- [Clip Paths](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Clipping Regions

Article • 01/07/2021

A clipping region is one of the graphic objects that an application can select into a device context (DC). It is typically rectangular. Some device contexts provide a predefined or default clipping region while others do not. For example, if you obtain a device context handle from the [BeginPaint](#) function, the DC contains a predefined rectangular clipping region that corresponds to the invalid rectangle that requires repainting. However, when you obtain a device context handle by calling the [GetDC](#) function with a **NULLhWnd** parameter, or by calling the [CreateDC](#) function, the DC does not contain a default clipping region. For more information about device contexts returned by the [BeginPaint](#) function, see [Painting and Drawing](#). For more information about device contexts returned by the [CreateDC](#) and [GetDC](#) functions, see [Device Contexts](#).

Applications can perform a variety of operations on clipping regions. Some of these operations require a handle identifying the region and some do not. For example, an application can perform the following operations directly on a device context's clipping region.

- Determine whether graphics output appears within the region's borders by passing coordinates of the corresponding line, arc, bitmap, text, or filled shape to the [PtVisible](#) function.
- Determine whether part of the client area intersects a region by calling the [RectVisible](#) function.
- Move the existing region by a specified offset by calling the [OffsetClipRgn](#) function.
- Exclude a rectangular part of the client area from the current clipping region by calling the [ExcludeClipRect](#) function.
- Combine a rectangular part of the client area with the current clipping region by calling the [IntersectClipRect](#) function.

After obtaining a handle identifying the clipping region, an application can perform any operation that is common with regions, such as:

- Combining a copy of the current clipping region with a second region by calling the [CombineRgn](#) function.
- Compare a copy of the current clipping region to a second region by calling the [EqualRgn](#) function.
- Determine whether a point lies within the interior of a copy of the current clipping region by calling the [PtInRegion](#) function.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Clip Paths

Article • 01/07/2021

Like a clipping region, a clip path is another graphics object that an application can select into a device context. Unlike a clipping region, a clip path is always created by an application and it is used for clipping to one or more irregular shapes. For example, an application can use the lines and curves that form the outlines of characters in a string of text to define a clip path.

To create a clip path, it's first necessary to create a path that describes the required irregular shape. Paths are created by calling the appropriate graphics device interface (GDI) drawing functions after calling the [BeginPath](#) function and before calling the [EndPath](#) function. This collection of functions is called a path bracket. For more information about paths and path brackets, see [Paths](#).

After the path is created, it can be converted to a clip path by calling the [SelectClipPath](#) function, identifying a device context, and specifying a usage mode. The usage mode determines how the system combines the new clip path with the device context's original clipping region. The following table describes the usage modes.

[+] Expand table

Mode	Description
RGN_AND	The clip path includes the intersection (overlapping areas) of the device context's clipping region and the current path.
RGN_COPY	The clip path is the current path.
RGN_DIFF	The clip path includes the device context's clipping region with any intersecting parts of the current path excluded.
RGN_OR	The clip path includes the union (combined areas) of the device context's clipping region and the current path.
RGN_XOR	The clip path includes the union of the device context's clipping region and the current path but excludes the intersection.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Clipping

Article • 01/07/2021

This section contains example code that shows how to generate a clip path consisting of a character string. The example creates a logical font and uses it to draw a string within a clip path, then fills the path by drawing horizontal and vertical lines.

C++

```
// DoClipPat - Draws a clip path using the specified string
// Return value - TRUE if successful; FALSE otherwise
// lplf - address of a LOGFONT structure that defines the font to
//         use to draw the clip path
// lpsz - address of a string to use for the clip path

BOOL DoClipPath(LPLOGFONT lplf, LPSTR lpsz)
{
    LOGFONT lf;           // logical font structure
    HFONT hfont;          // new logical font handle
    HFONT hfontOld;        // original logical font handle
    HDC hdc;              // display DC handle
    int nXStart, nYStart; // drawing coordinates
    RECT rc;              // rectangle structure for painting window
    SIZE sz;              // size structure that receives text extents
    int nStrLen;          // length of the string
    int i;                // loop counter
    HRESULT hr;
    size_t * pcch;

    // Retrieve a cached DC for the window.

    hdc = GetDC(hwnd);

    // Erase the current window contents.

    GetClientRect(hwnd, &rc);
    FillRect(hdc, &rc, GetStockObject(WHITE_BRUSH));

    // Use the specified font to create a logical font and select it
    // into the DC.

    hfont = CreateFontIndirect(lplf);
    if (hfont == NULL)
        return FALSE;
    hfontOld = SelectObject(hdc, hfont);

    // Create a clip path.

    hr = StringCchLength(lpsz, STRSAFE_MAX_CCH, pcch);
    if (FAILED(hr))
    {
```

```

        // TODO: write error handler
    }
    nStrLen = *pcch
    BeginPath(hdc);
    TextOut(hdc, nXStart, nYStart, lpsz, nStrLen);
    EndPath(hdc);
    SelectClipPath(hdc, RGN_DIFF);

    // Retrieve the dimensions of the rectangle surrounding
    // the text.

    GetTextExtentPoint32(hdc, lpsz, nStrLen, &sz);

    // Draw horizontal lines through the clip path.

    for (i = nYStart + 1; i < (nYStart + sz.cy); i += 3)
    {
        MoveToEx(hdc, nXStart, i, (LPPOINT) NULL);
        LineTo(hdc, (nXStart + sz.cx), i);
    }

    // Draw vertical lines through the clip path.

    for (i = nXStart + 1; i < (nXStart + sz.cx); i += 3)
    {
        MoveToEx(hdc, i, nYStart, (LPPOINT) NULL);
        LineTo(hdc, i, (nYStart + sz.cy));
    }

    // Select the original font into the DC and release the DC.

    SelectObject(hdc, hfontOld);
    DeleteObject(hfont);
    ReleaseDC(hwnd, hdc);

    return TRUE;
}

```

For an example that demonstrates how an application creates a rectangular clipping region, see [Regions](#).

Feedback

Was this page helpful?

 Yes

 No

Clipping Reference

Article • 01/07/2021

The following elements are used with clipping.

- Clipping Functions

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Clipping Functions

Article • 01/07/2021

The following functions are used with clipping.

[+] Expand table

Function	Description
ExcludeClipRect	Creates a new clipping region that consists of the existing clipping region minus the specified rectangle.
ExtSelectClipRgn	Combines the specified region with the current clipping region using the specified mode.
GetClipBox	Retrieves the dimensions of the tightest bounding rectangle that can be drawn around the current visible area on the device.
GetClipRgn	Retrieves a handle identifying the current application-defined clipping region for the specified device context.
GetMetaRgn	Retrieves the current metaregion for the specified device context.
GetRandomRgn	Copies the system clipping region of a specified device context to a specific region.
IntersectClipRect	Creates a new clipping region from the intersection of the current clipping region and the specified rectangle.
OffsetClipRgn	Moves the clipping region of a device context by the specified offsets.
PtVisible	Determines whether the specified point is within the clipping region of a device context.
RectVisible	Determines whether any part of the specified rectangle lies within the clipping region of a device context.
SelectClipPath	Selects the current path as a clipping region for a device context, combining the new region with any existing clipping region by using the specified mode.
SelectClipRgn	Selects a region as the current clipping region for the specified device context.
SetMetaRgn	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.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ExcludeClipRect function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[Expand table](#)

Return code	Description
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ExtSelectClipRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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.

 Expand table

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.

[Expand table](#)

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

GetClipBox function (wingdi.h)

Article02/22/2024

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

C++

```
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.

 Expand table

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetMetaRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetRandomRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

IntersectClipRect function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[+] Expand table

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.)

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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

OffsetClipRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PtVisible function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

RectVisible function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SelectClipPath function (wingdi.h)

Article 10/13/2021

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

C++

```
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.

 Expand table

Value	Meaning
<code>RGN_AND</code>	The new clipping region includes the intersection (overlapping areas) of the current clipping region and the current path.
<code>RGN_COPY</code>	The new clipping region is the current path.
<code>RGN_DIFF</code>	The new clipping region includes the areas of the current clipping region with those of the current path excluded.
<code>RGN_OR</code>	The new clipping region includes the union (combined areas) of the current clipping region and the current path.
<code>RGN_XOR</code>	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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

Syntax

C++

```
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.

 Expand table

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 **NUL**L region handle.

Examples

For an example, see [Clipping Output](#).

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetMetaRgn function (wingdi.h)

Article02/22/2024

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

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Colors

Article • 01/07/2021

Color is an important element in the pictures and images generated by applications. This overview describes how applications can manage and use colors with pens, brushes, text, or bitmaps.

- [About Colors](#)
 - [Using Color](#)
 - [Color Reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Colors

Article • 01/07/2021

Color can be used to communicate ideas, show relationships between items, and improve the appeal and quality of output. Windows enables applications to discover the color capabilities of given devices and to choose from the available colors those that best suit their needs.

This overview provides information on the following topics:

- [Color Basics](#)
- [Color Palettes](#)

Although not described in this overview, image color matching is an important feature of color management that helps ensure that color images look the same whether displayed on screen or printed on paper. For more information, see [Windows Color System](#).

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Color Basics

Article • 01/07/2021

The color capabilities of devices, such as displays and printers, can range from monochrome to thousands of colors. Because an application may need to generate output for devices throughout this range, it should be prepared to handle varying color capabilities.

An application can discover the number of colors available for a given device by using the [GetDeviceCaps](#) function to retrieve the NUMCOLORS value. This value specifies the count of colors available for use by the application. Usually, this count corresponds to a physical property of the output device, such as the number of inks in the printer or the number of distinct color signals the display adapter can transmit to the monitor.

Although the NUMCOLORS value specifies the count of colors, it does not identify what the available colors are. An application can discover what colors are available by enumerating all pens having the PS_SOLID type. Because the device driver that supports a given device usually has a full range of solid pens and because the system requires that solid pens have only colors that the device can generate, enumerating these pens is often equivalent to enumerating the colors. An application can enumerate the pens by using the [EnumObjects](#) function. For a code example, see [Enumerating Colors](#).

For more information, see the following topics:

- [Color Values](#)
- [Color Approximations and Dithering](#)
- [Color in Bitmaps](#)
- [Color Mixing](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Values

Article • 01/07/2021

Color is defined as a combination of three primary colors red, green, and blue. The system identifies a color by giving it a color value (sometimes called an RGB triplet), which consists of three 8-bit values specifying the intensities of its color components. Black has the minimum intensity for red, green, and blue, so the color value for black is (0, 0, 0). White has the maximum intensity for red, green, and blue, so its color value is (255, 255, 255).

ⓘ Note

If image color matching is enabled, the definition of color and the meaning of a color value depends on the type of color space that is currently set for the device context.

The system and applications use parameters and variables having the [COLORREF](#) type to pass and store color values. For example, the [EnumObjects](#) function identifies the color of each pen by setting the `lpenColor` member in a [LOGPEN](#) structure to a color value. Applications can extract the individual values of the red, green, and blue components from a color value by using the [GetRValue](#), [GetGValue](#), and [GetBValue](#) macros, respectively. Applications can create a color value from individual component values by using the [RGB](#) macro. When creating or examining a logical palette, an application uses the [RGBQUAD](#) structure to define color values and to examine individual component values.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Approximations and Dithering

Article • 01/07/2021

Although an application can use color without regard to the color capabilities of the device, the resulting output may not be as informative and pleasing as output for which color is carefully chosen. Few, if any, devices guarantee an exact match for every possible color value; therefore, if an application requests a color that the device cannot generate, the system approximates that color by using a color that the device can generate. For example, if an application attempts to create a red pen for a black and white printer, it will receive a black pen instead the system uses black as the approximation for red.

An application can discover whether the system will approximate a given color by using the [GetNearestColor](#) function. The function takes a color value and returns the color value of the closest matching color the device can generate. The method the system uses to determine this approximation depends on the device driver and its color capabilities. In most cases, the approximated color's overall intensity is closest to that of the requested color.

When an application creates a pen or sets the color for text, the system always approximates a color if no exact match exists. When an application creates a solid brush, the system may attempt to simulate the requested color by dithering. *Dithering* simulates a color by alternating two or more colors in a pattern. For example, different shades of pink can be simulated by alternating different combinations of red and white. Depending on the colors and the pattern, dithering can produce reasonable simulations. It is most useful for monochrome devices, because it expands the number of available "colors" well beyond simple black and white.

The method used to create dithered colors depends on the device driver. Most device drivers use a standard dithering algorithm, which generates a pattern based on the intensity values of the requested red, green, and blue colors. In general, any requested color that cannot be generated by the device is subject to simulation, but an application is not notified when the system simulates a color. Furthermore, an application cannot modify or change the dithering algorithm of the device driver. An application, however, can bypass the algorithm by creating and using pattern brushes. In this way, the application creates its own dithered colors by combining solid colors in the bitmap that it uses to create the brush.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color in Bitmaps

Article • 01/07/2021

The system handles colors in bitmaps differently than colors in pens, brushes, and text. Compatible bitmaps, created by using the [CreateBitmap](#) or [CreateCompatibleBitmap](#) function, are device specific and retain color information in a device-dependent format. No color values are used, and the colors are not subject to approximations and dithering.

Device-independent bitmaps (DIBs) retain color information either as color values or color palette indexes. If color values are used, the colors are subject to approximation, but not dithering. Color palette indexes can only be used with devices that support color palettes. Although the system does not approximate or dither colors identified by indexes, the resulting color may be different than that intended, because the indexes yield valid results only in the context of the color palette that was current at the time the bitmap was created. If the palette changes, so do the colors in the bitmap. For more information on palette indexes, see [Default Palette](#) and [PALETTEINDEX](#).

In addition to referencing the logical palette, an application can also reference a value in a DIB color table. To select a color in a DIB color table, call [DIBINDEX](#). Note that this is only possible for a device context that has a DIB selected into it.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Mixing

Article • 01/07/2021

Color mixing lets an application create new colors by combining the pen or brush color with colors in the existing image. The application can choose either to draw the pen or brush color as is (effectively drawing over any existing image) or to mix the color with the colors already present.

The foreground mix mode, sometimes called the binary raster operation, determines how these colors are mixed. An application can merge colors, preserving all components of both colors; mask colors, removing or moderating components that are not common; or exclusively mask colors, removing or moderating components that are common. There are several variations on these basic mixing operations.

Color mixing is subject to color approximation. If the result of color mixing is a color that the device cannot generate, the system approximates the result, using a color it can generate. If an application mixes dithered colors, the individual colors used to create the dithered color are mixed, and the results are subject to color approximation.

An application sets the foreground mix mode by using the [SetROP2](#) function and retrieves the current mode by using the [GetROP2](#) function.

Although there is a background mix mode, that mode does not control the mixing of colors. Instead, it specifies whether a background color is used when drawing styled lines, hatched brushes, and text.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Palettes (Windows GDI)

Article • 01/07/2021

A color palette is an array that contains color values identifying the colors that can currently be displayed or drawn on the output device. Color palettes are used by devices that are capable of generating many colors but that can only display or draw a subset of these at any given time. For such devices, the system maintains a *system palette* to track and manage the current colors of the device. Applications do not have direct access to the system palette. Instead, the system associates a default palette with each device context. Applications can use the colors in the default palette or define their own colors by creating *logical palettes* and associating them with individual device contexts.

An application can determine whether a device supports color palettes by checking for the RC_PALETTE bit in the RASTERCAPS value returned by the [GetDeviceCaps](#) function.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Default Palette

Article • 01/07/2021

The *default palette* is an array of color values identifying the colors that can be used with a device context by default. The system associates the default palette with a context whenever an application creates a context for a device that supports color palettes. The default palette ensures that colors are available for use by an application without any further action.

The default palette typically has 20 entries (colors), but the exact number of entries may vary from device to device. This number is equal to the NUMCOLORS value returned by the [GetDeviceCaps](#) function. An application can retrieve the color values for colors in the default palette by enumerating solid pens, the same technique used to discover the colors available on nonpalette devices. The colors in the default palette depend on the device. Display devices, for example, often use the 16 standard colors of the VGA display and 4 other colors defined by Windows. Print devices may use other default colors.

When using the default palette, applications use color values to specify pen and text colors. If the requested color is not in the palette, the system approximates the color by using the closest color in the palette. If an application requests a solid brush color that is not in the palette, the system simulates the color by dithering with colors that are in the palette.

To avoid approximations and dithering, applications can also specify pen, brush, and text colors by using color palette indexes rather than color values. A color palette index is an integer value that identifies a specific palette entry. Applications can use color palette indexes in place of color values but must use the [PALETTEINDEX](#) macro to create the indexes.

Color palette indexes are only useful for devices that support color palettes. To avoid this device dependence, applications that use the same code to draw to both palette and nonpalette devices should use palette-relative color values to specify pen, brush, and text colors. These values are identical to color values except when creating solid brushes. (On palette devices, a solid brush color specified by a palette-relative color value is subject to color approximation instead of dithering.) Applications must use the [PALETTERGB](#) macro to create palette-relative color values.

The system does not allow an application to change the entries in the default palette. To use colors other than those in the default palette, an application must create its own logical palette and select the palette into the device context.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Logical Palette

Article • 01/07/2021

A *logical palette* is a color palette that an application creates and associates with a given device context. Logical palettes let applications define and use colors that meet their specific needs. Applications can create any number of logical palettes, using them for separate device contexts or switching between them for a single device context. The maximum number of palettes that an application can create depends on the resources of the system.

An application creates a logical palette by using the [CreatePalette](#) function. The application fills a **LOGPALETTE** structure, which specifies the number of entries and the color values for each entry, and then the application passes the structure to [CreatePalette](#). The function returns a palette handle that the application uses in all subsequent operations to identify the palette. To use colors in the logical palette, the application selects the palette into a device context by using the [SelectPalette](#) function and then realizes the palette by using the [RealizePalette](#) function. The colors in the palette are available as soon as the logical palette is realized.

An application should limit the size of its logical palettes to just enough entries to represent the colors needed. Applications cannot create logical palettes larger than the maximum palette size, a device-dependent value. Applications can obtain the maximum size by using the [GetDeviceCaps](#) function to retrieve the SIZEPALETTE value.

Although an application can specify any color value for a given entry in a logical palette, not all colors can be generated by the given device. The system does not provide a way to discover which colors are supported, but the application can discover the total number of these colors by retrieving the color resolution of the device. The color resolution, specified in color bits per pixel, is equal to the COLORRES value returned by the [GetDeviceCaps](#) function. A device that has a color resolution of 18 has 262,144 possible colors. If an application requests a color that is not supported, the system chooses an appropriate approximation.

Once a logical palette is created, an application can change colors in the palette by using the [SetPaletteEntries](#) function. If the logical palette has been selected and realized, changing the palette does not immediately affect the colors being displayed. The application must use the [UnrealizeObject](#) and [RealizePalette](#) functions to update the colors. In some cases, the application may need to deselect, unrealize, select, and realize the logical palette to ensure that the colors are updated exactly as requested. If an application selects a logical palette into more than one device context, changes to the logical palette affect all device contexts for which it is selected.

An application can change the number of entries in a logical palette by using the [ResizePalette](#) function. If the application reduces the size, the remaining entries are unchanged. If the application extends the size, the system sets the color for each new entry to black (0, 0, 0) and the flag to zero.

An application can retrieve the color and flag values for entries in a given logical palette by using the [GetPaletteEntries](#) function. An application can retrieve the index for the entry in a given logical palette that most closely matches a specified color value by using the [GetNearestPaletteIndex](#) function.

When an application no longer needs a logical palette, it can delete it by using the [DeleteObject](#) function. The application must make sure the logical palette is no longer selected into a device context before deleting the palette.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Palette Animation

Article • 01/07/2021

Palette animation is a technique to simulate motion by rapidly changing the colors of selected entries in a color palette. An application can carry out palette animation by creating a logical palette that contains "reserved" entries and then using the [AnimatePalette](#) function to change colors in those reserved entries.

An application creates a reserved entry in a logical palette by setting the **peFlags** member of the [PALETTEENTRY](#) structure to the PC_RESERVED flag. Once this logical palette is selected and realized, the application can call the [AnimatePalette](#) function to change one or more reserved entries. If the given palette is associated with the active window, the system updates the colors on the screen immediately.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

System Palette

Article • 01/07/2021

The system maintains a *system palette* for each device that uses palettes. The system palette contains the color values for all colors that can currently be displayed or drawn by the device. Other than viewing the contents of the system palette, applications cannot access the system palette directly. Instead, the system has complete control of the system palette and permits access only through the use of logical palettes.

An application can view the contents of the system palette by using the [GetSystemPaletteEntries](#) function. This function retrieves the contents of one or more entries, up to the total number of entries in the system palette. The total is always equal to the number returned for the SIZEPALETTE value by the [GetDeviceCaps](#) function and is the same as the maximum size for any given logical palette.

Although applications cannot change colors in the system palette directly, they may cause changes when realizing logical palettes. To realize a palette, the system examines each requested color and attempts to find an entry in the system palette that contains an exact match. If the system finds a matching color, it maps the logical palette index to the corresponding system palette index. If the system does not find an exact match, it copies the requested color to an unused system palette entry before mapping the indexes. If all system palette entries are in use, the system maps the logical palette index to the system palette entry whose color most closely matches the requested color. After this mapping is set, applications cannot override it. For example, applications cannot use system palette indexes to specify colors; only logical palette indexes are permitted.

Applications can modify the way indexes are mapped by setting the **peFlags** member of the [PALETTEENTRY](#) structure to selected values when creating the logical palette. For example, the PC_NOCOLLAPSE flag directs the system to immediately copy the requested color to an unused system palette entry regardless of whether a system palette entry already contains that color. Also, the PC_EXPLICIT flag directs the system to map the logical palette index to an explicitly given system palette index. (The application gives the system palette index in the low-order word of the **PALETTEENTRY** structure.)

Palettes can be realized as either a background palette or a foreground palette by specifying **TRUE** or **FALSE** respectively for the *bForceBackground* parameter in the [SelectPalette](#) function. There can be only one foreground palette in the system at a time. If the window is the currently active window or a descendent of the currently active window, it can realize a foreground palette. Otherwise the palette is realized as a background palette regardless of the value of the *bForceBackground* parameter. The critical property of a foreground palette is that, when realized, it can overwrite all entries

(except for the static entries) in the system palette. The system accomplishes this by marking all of the entries that are not static in the system palette as unused before the realization of a foreground palette, thereby eliminating all of the used entries. No preprocessing occurs on the system palette for a background palette realization. The foreground palette sets all of the possible nonstatic colors. Background palettes can set only what remains open and are prioritized in a first-come, first-serve manner. Typically, applications use background palettes for child windows which realize their own individual palettes. This helps minimize the number of changes that occur to the system palette.

An unused system palette entry is any entry that is not reserved and does not contain a static color. Reserved entries are explicitly marked with the PC_RESERVED value. These entries are created when an application realizes a logical palette for palette animation. Static color entries are created by the system and correspond to the colors in the default palette. The [GetDeviceCaps](#) function can be used to retrieve the NUMRESERVED value, which specifies the number of system palette entries reserved for static colors.

Because the system palette has a limited number of entries, selecting and realizing a logical palette for a given device may affect the colors associated with other logical palettes for the same device. These color changes are especially dramatic when they occur on the display. An application can make sure that reasonable colors are used for its currently selected logical palette by resetting the palette before each use. An application resets the palette by calling the [UnrealizeObject](#) and [RealizePalette](#) functions. Using these functions causes the system to remap the colors in the logical palette to reasonable colors in the system palette.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

System Palette and Static Colors

Article • 01/07/2021

Ordinarily, the system palette entries that the system reserves for static colors cannot be changed. An application can override this default behavior by using the [SetSystemPaletteUse](#) function to reduce the number of static color entries and, thereby, increase the number of unused system palette entries. However, because changing the static colors can have an immediate and dramatic effect on all windows on the display, an application should not call [SetSystemPaletteUse](#), unless it has a maximized window and the input focus.

When an application calls [SetSystemPaletteUse](#) with the SYSPAL_NOSTATIC value, the system frees all but two of the reserved entries, allowing those entries to receive new color values when the application subsequently realizes its logical palette. The remaining two static color entries remain reserved and are set to white and black. An application can restore the reserved entries by calling [SetSystemPaletteUse](#) with the SYSPAL_STATIC value. It can discover the current system palette usage by using the [GetSystemPaletteUse](#) function.

Furthermore, after setting the system palette usage to SYSPAL_NOSTATIC, the application must immediately realize its logical palette, call the [GetSysColor](#) function to save the current system color settings, call the [SetSysColors](#) function to set the system colors to reasonable values using black and white, and finally send the [WM_SYSCOLORCHANGE](#) message to other top-level windows to allow them to be redrawn with the new system colors. When setting system colors using black and white, the application should make sure adjacent or overlapping items, such as window frames and borders, are set to black and white, respectively.

Before the application loses the input focus, closes its window, or terminates, it must immediately call [SetSystemPaletteUse](#) with the SYSPAL_STATIC value, realize its logical palette, restore the system colors to their previous values, and send the [WM_SYSCOLORCHANGE](#) message. The system sends a [WM_PAINT](#) message to any window that is affected by a system color change. Applications that have brushes using the existing system colors should delete those brushes and re-create them using the new system colors.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Palette Messages

Article • 01/07/2021

Changes to the system palette for the display device can have dramatic and sometimes undesirable effects on the colors used in windows on the desktop. To minimize the impact of these changes, the system provides a set of messages that help applications manage their logical palettes while ensuring that colors in the active window are as close as possible to the colors intended.

The system sends a [WM_QUERYNEWPALETTE](#) message to a top-level or overlapped window just before activating the window. This message gives an application the opportunity to select and realize its logical palette so that it receives the best possible mapping of colors for its logical palette. When the application receives the message, it should use the [SelectPalette](#), [UnrealizeObject](#), and [RealizePalette](#) functions to select and realize the logical palette. Doing so directs the system to update colors in the system palette so that its colors match as many colors in the logical palette as possible.

When an application causes changes to the system palette as a result of realizing its logical palette, the system sends a [WM_PALETTECHANGED](#) message to all top-level and overlapped windows. This message gives applications the opportunity to update the colors in the client areas of their windows, replacing colors that have changed with colors that more closely match the intended colors. An application that receives the [WM_PALETTECHANGED](#) message should use [UnrealizeObject](#) and [RealizePalette](#) to reset the logical palettes associated with all inactive windows and then update the colors in the client area for each inactive window by using the [UpdateColors](#) function. This technique does not guarantee the greatest number of exact color matches; however, it does ensure that colors in the logical palette are mapped to reasonable colors in the system palette.

ⓘ Note

To avoid creating an infinite loop, an application should never realize the palette for the window whose handle matches the handle passed in the *wParam* parameter of the [WM_PALETTECHANGED](#) message.

The [UpdateColors](#) function typically updates a client area of an inactive window faster than redrawing the area. However, because [UpdateColors](#) performs 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 means **UpdateColors** cannot be used to update colors when the window becomes active. In such cases, the application should redraw the client area.

The system may send the **WM_QUERYNEWPALETTE** message when changes to the logical palette are made. Also, the system may send the **WM_PALETTEISCHANGING** message to all top-level and overlapped windows when the system palette is about to change.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Halftone Palette and Color Adjustment

Article • 01/07/2021

Halftone palettes are intended to be used whenever the stretching mode of a device context is set to HALFTONE. An application creates a halftone palette by using the [CreateHalftonePalette](#) function. The application must select and realize this palette into the device context before calling the [StretchBlt](#) or [StretchDIBits](#) function.

The system automatically adjusts the input color of source bitmaps whenever applications call the [StretchBlt](#) and [StretchDIBits](#) functions and the stretching mode of a device context is set to HALFTONE. These color adjustments affect certain attributes of the image, such as contrast and brightness. An application can set the color adjustment values by using the [SetColorAdjustment](#) function. The application can retrieve the color adjustment values for the specified device context by using the [GetColorAdjustment](#) function.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Using Color

Article • 01/07/2021

- [Enumerating Colors](#)
- [Creating Colored Pens and Brushes](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Enumerating Colors

Article • 01/07/2021

You can determine how many colors a device supports and what those colors are by retrieving the count of colors for the device and enumerating the colors of the solid pens. To retrieve the number of colors, use the [GetDeviceCaps](#) function with the NUMCOLORS value. To enumerate solid pens, use the [EnumObjects](#) function and a corresponding callback function that receives information about each pen.

C++

```
// GetTheColors - returns the count and color values of solid colors
// Returns a pointer to the array containing colors
// hdc - handle to device context

COLORREF *GetTheColors(HDC hdc)
{
    int cColors;
    COLORREF *aColors;

    // Get the number of colors.
    cColors = GetDeviceCaps(hdc, NUMCOLORS);

    // Allocate space for the array.
    aColors = (COLORREF *)LocalAlloc(LPTR, sizeof(COLORREF) *
        (cColors+1));

    // Save the count of colors in first element.
    aColors[0] = (LONG)cColors;

    // Enumerate all pens and save solid colors in the array.
    EnumObjects(hdc, OBJ_PEN, (GOBJENUMPROC)MyEnumProc, (LONG)aColors);

    // Refresh the count of colors.
    aColors[0] = (LONG)cColors;

    return aColors;
}

int MyEnumProc(LPVOID lp, LPBYTE lpb)
{
    LPLOGPEN lopn;
    COLORREF *aColors;
    int iColor;

    lopn = (LPLOGPEN)lp;
    aColors = (COLORREF *)lpb;

    if (lopn->lopnStyle==PS_SOLID)
    {
```

```
// Check for too many colors.  
if ((iColor = (int)aColors[0]) <= 0)  
    return 0;  
aColors[iColor] = lopn->lopnColor;  
aColors[0]--;  
}  
return 1;  
}
```

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Creating Colored Pens and Brushes

Article • 01/07/2021

Although you can specify any color for a pen when creating it, the system uses only colors that are available on the device. This means the system uses the closest matching color when it realizes the pen for drawing. When creating brushes, the system generates a dithered color if you specify a color that the device does not support. In either case, you can use the [RGB](#) macro to specify a color when creating a pen or brush.

C++

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    RECT clientRect;
    RECT textRect;
    HRGN bgRgn;
    HBRUSH hBrush;
    HPEN hPen;

    switch (message)
    {

        case WM_PAINT:
        {
            hdc = BeginPaint(hWnd, &ps);

            // Fill the client area with a brush
            GetClientRect(hWnd, &clientRect);
            bgRgn = CreateRectRgnIndirect(&clientRect);
            hBrush = CreateSolidBrush(RGB(200,200,200));
            FillRgn(hdc, bgRgn, hBrush);

            hPen = CreatePen(PS_DOT,1,RGB(0,255,0));
            SelectObject(hdc, hPen);
            SetBkColor(hdc, RGB(0,0,0));
            Rectangle(hdc, 10,10,200,200);

            // Text caption
            SetBkColor(hdc, RGB(255,255,255));
            SetRect(&textRect, 10, 210, 200,200);
            DrawText(hdc,TEXT("PS_DOT"),-1,&textRect, DT_CENTER | DT_NOCLIP);

            hPen = CreatePen(PS_DASHDOTDOT,1,RGB(0,255,255));
        }
    }
}
```

```

SelectObject(hdc, hPen);
SetBkColor(hdc, RGB(255,0,0));
SelectObject(hdc,CreateSolidBrush(RGB(0,0,0)));
Rectangle(hdc, 210,10,400,200);

// Text caption
SetBkColor(hdc, RGB(255,200,200));
SetRect(&textRect, 210, 210, 400,200);
DrawText(hdc,TEXT("PS_DASHDOTDOT"),-1,&textRect, DT_CENTER | DT_NOCLIP);

hPen = CreatePen(PS_DASHDOT,1,RGB(255,0,0));
SelectObject(hdc, hPen);
SetBkColor(hdc, RGB(255,255,0));
SelectObject(hdc,CreateSolidBrush(RGB(100,200,255)));
Rectangle(hdc, 410,10,600,200);

// Text caption
SetBkColor(hdc, RGB(200,255,200));
SetRect(&textRect, 410, 210, 600,200);
DrawText(hdc,TEXT("PS_DASHDOT"),-1,&textRect, DT_CENTER | DT_NOCLIP);

// When fnPenStyle is PS_SOLID, nWidth may be more than 1.
// Also, if you set the width of any pen to be greater than 1,
// then it will draw a solid line, even if you try to select another
style.
hPen = CreatePen(PS_SOLID,5,RGB(255,0,0));
SelectObject(hdc, hPen);
// Setting the background color doesn't matter
// when the style is PS_SOLID
SetBkColor(hdc, RGB(255,255,255));
SelectObject(hdc,CreateSolidBrush(RGB(200,100,50)));
Rectangle(hdc, 10,300,200,500);

// Text caption
SetBkColor(hdc, RGB(200,200,255));
SetRect(&textRect, 10, 510, 200,500);
DrawText(hdc,TEXT("PS_SOLID"),-1,&textRect, DT_CENTER | DT_NOCLIP);

hPen = CreatePen(PS_DASH,1,RGB(0,255,0));
SelectObject(hdc, hPen);
SetBkColor(hdc, RGB(0,0,0));
SelectObject(hdc,CreateSolidBrush(RGB(200,200,255)));
Rectangle(hdc, 210,300,400,500);

// Text caption
SetBkColor(hdc, RGB(255,255,200));
SetRect(&textRect, 210, 510, 400,200);
DrawText(hdc,TEXT("PS_DASH"),-1,&textRect, DT_CENTER | DT_NOCLIP);

hPen = CreatePen(PS_NULL,1,RGB(0,255,0));
SelectObject(hdc, hPen);

```

```
// Setting the background color doesn't matter
// when the style is PS_NULL
SetBkColor(hdc, RGB(0,0,0));
SelectObject(hdc,CreateSolidBrush(RGB(255,255,255)));
Rectangle(hdc, 410,300,600,500);

// Text caption
SetBkColor(hdc, RGB(200,255,255));
SetRect(&textRect, 410, 510, 600,500);
DrawText(hdc,TEXT("PS_NULL"),-1,&textRect, DT_CENTER | DT_NOCLIP);

// Clean up
DeleteObject(bgRgn);
DeleteObject(hBrush);
DeleteObject(hPen);

GetStockObject(WHITE_BRUSH);
GetStockObject(DC_PEN);

EndPaint(hWnd, &ps);
break;
}

case WM_DESTROY:
PostQuitMessage(0);
break;
default:
return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Reference (Windows GDI)

Article • 01/07/2021

The following elements are associated with color.

- [Color Functions](#)
- [Color Structures](#)
- [Color Macros](#)
- [Color Messages](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Color Functions

Article • 01/07/2021

The following functions are used with color.

[+] Expand table

Function	Description
AnimatePalette	Replaces entries in the specified logical palette.
CreateHalftonePalette	Creates a halftone palette for the specified device context (DC).
CreatePalette	Creates a logical palette.
GetColorAdjustment	Retrieves the color adjustment values for the specified DC.
GetNearestColor	Retrieves a color value identifying a color from the system palette that will be displayed when the specified color value is used.
GetNearestPaletteIndex	Retrieves the index for the entry in the specified logical palette most closely matching a specified color value.
GetPaletteEntries	Retrieves a specified range of palette entries from the given logical palette.
GetSystemPaletteEntries	Retrieves a range of palette entries from the system palette that is associated with the specified DC.
GetSystemPaletteUse	Retrieves the current state of the system (physical) palette for the specified DC.
RealizePalette	Maps palette entries from the current logical palette to the system palette.
ResizePalette	Increases or decreases the size of a logical palette based on the specified value.
SelectPalette	Selects the specified logical palette into a device context.
SetColorAdjustment	Sets the color adjustment values for a DC using the specified values.
SetPaletteEntries	Sets RGB (red, green, blue) color values and flags in a range of entries in a logical palette.
SetSystemPaletteUse	Allows an application to specify whether the system palette contains 2 or 20 static colors.
UnrealizeObject	Resets the origin of a brush or resets a logical palette.

Function	Description
UpdateColors	Updates the client area of the specified device context by remapping the current colors in the client area to the currently realized logical palette.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

AnimatePalette function (wingdi.h)

Article02/22/2024

The **AnimatePalette** function replaces entries in the specified logical palette.

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateHalftonePalette function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreatePalette function (wingdi.h)

Article02/22/2024

The [CreatePalette](#) function creates a logical palette.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetColorAdjustment function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[COLORADJUSTMENT](#)

[Color Functions](#)

[Colors Overview](#)

[SetColorAdjustment](#)

GetNearestColor function (wingdi.h)

Article10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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)

Article 10/13/2021

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetPaletteEntries function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetSystemPaletteEntries function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>SYSPAL_NOSTATIC</code>	The system palette contains no static colors except black and white.
<code>SYSPAL_STATIC</code>	The system palette contains static colors that will not change when an application realizes its logical palette.
<code>SYSPAL_ERROR</code>	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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

RealizePalette function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ResizePalette function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SelectPalette function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

SetColorAdjustment function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetPaletteEntries function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetSystemPaletteUse function (wingdi.h)

Article 10/13/2021

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

C++

```
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.

 Expand table

Value	Meaning
<code>SYSPAL_NOSTATIC</code>	The system palette contains two static colors (black and white).
<code>SYSPAL_NOSTATIC256</code>	The system palette contains no static colors.
<code>SYSPAL_STATIC</code>	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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

UnrealizeObject function (wingdi.h)

Article10/13/2022

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

C++

```
BOOL UnrealizeObject(  
    HGDIOBJ h  
)
```

Parameters

h

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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Color Structures

Article • 01/07/2021

The following structures are used with color:

- [COLORREF](#)
- [LOGPALETTE](#)
- [PALETTEENTRY ↗](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

COLORREF

Article • 04/07/2022

The COLORREF value is used to specify an [RGB](#) color.

C++

```
typedef DWORD COLORREF;
typedef DWORD* LPCOLORREF;
```

Remarks

When specifying an explicit [RGB](#) color, the COLORREF value has the following hexadecimal form:

0x00bbggrr

The low-order byte contains a value for the relative intensity of red; the second byte contains a value for green; and the third byte contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is 0xFF.

To create a COLORREF color value, use the [RGB](#) macro. To extract the individual values for the red, green, and blue components of a color value, use the [GetRValue](#), [GetGValue](#), and [GetBValue](#) macros, respectively.

Example

C++

```
// Color constants.
const COLORREF rgbRed    = 0x000000FF;
const COLORREF rgbGreen  = 0x0000FF00;
const COLORREF rgbBlue   = 0x00FF0000;
const COLORREF rgbBlack  = 0x00000000;
const COLORREF rgbWhite  = 0x00FFFFFF;
```

Example from [Windows Classic Samples](#) on GitHub.

Requirements

Requirement	Value
-------------	-------

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Windef.h (include Windows.h)

See also

[Colors Overview](#)

[Color Structures](#)

[GetBValue](#)

[GetGValue](#)

[GetRValue](#)

[RGB](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

LOGPALETTE structure (wingdi.h)

Article02/22/2024

The **LOGPALETTE** structure defines a logical palette.

Syntax

C++

```
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[1]`

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

Color Macros

Article • 01/07/2021

The following macros are used with color:

- [DIBINDEX](#)
- [GetBValue](#)
- [GetGValue](#)
- [GetRValue](#)
- [PALETTEINDEX](#)
- [PALETERGB](#)
- [RGB](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

DIBINDEX macro (mmsystem.h)

07/09/2025

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

C++

```
LONG DIBINDEX(  
    WORD n  
)
```

Parameters

n

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

Return value

Type: **LONG**

The return value is a color table index specifier in the form of a 32-bit [COLORREF](#) value.

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

Requirement	Value
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](#)

GetBValue macro (wingdi.h)

07/09/2025

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

Syntax

C++

```
BYTE GetBValue(  
    DWORD rgb  
);
```

Parameters

rgb

Specifies an RGB color value.

Return value

Type: **BYTE**

The return value is the intensity of the blue component of the specified RGB color.

Remarks

The intensity value is in the range 0 through 255.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetGValue](#)

[GetRValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

[RGB](#)

GetGValue macro (wingdi.h)

07/09/2025

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

Syntax

C++

```
BYTE GetGValue(  
    DWORD rgb  
);
```

Parameters

rgb

Specifies an RGB color value.

Return value

Type: **BYTE**

The return value is the intensity of the green component of the specified RGB color.

Remarks

The intensity value is in the range 0 through 255.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetBValue](#)

[GetRValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

[RGB](#)

GetRValue macro (wingdi.h)

07/09/2025

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

Syntax

C++

```
BYTE GetRValue(  
    DWORD rgb  
);
```

Parameters

rgb

Specifies an RGB color value.

Return value

Type: **BYTE**

The return value is the intensity of the red component of the specified RGB color.

Remarks

The intensity value is in the range 0 through 255.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)

See also

[Color Macros](#)

[Colors Overview](#)

[GetBValue](#)

[GetGValue](#)

[PALETTEINDEX](#)

[PALETTERGB](#)

[RGB](#)

PALETTEINDEX macro (wingdi.h)

07/09/2025

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

C++

```
COLORREF PALETTEINDEX(  
    WORD i  
)
```

Parameters

i

An index to the palette entry containing the color to be used for a graphics operation.

Return value

Type: [COLORREF](#)

The return value is a logical-palette index specifier.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[COLORREF](#)

[Color Macros](#)

[Colors Overview](#)

[PALETTERGB](#)

[RGB](#)

PALETERGB macro (wingdi.h)

07/09/2025

The **PALETERGB** 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

C++

```
COLORREF PALETERGB(
    BYTE r,
    BYTE g,
    BYTE 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

Type: [COLORREF](#)

The return value is a palette-relative RGB specifier. For output devices that support logical palettes, the system matches a palette-relative RGB value to the nearest color in the logical palette of the device context as though the application had specified an index to that palette entry. If an output device does not support a system palette, the system uses the palette-relative RGB as though it were a conventional RGB value returned by the **RGB** macro.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	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](#)

RGB macro (wingdi.h)

07/09/2025

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

Syntax

C++

```
COLORREF RGB(  
    BYTE r,  
    BYTE g,  
    BYTE 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

Type: **COLORREF**

The return value is the resultant RGB color as a **COLORREF** value.

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

 [Expand table](#)

Requirement	Value
Minimum supported client	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](#)

Color Messages

Article • 01/07/2021

The following messages are used with color:

- [WM_PALETTECHANGED](#)
- [WM_PALETTEISCHANGING](#)
- [WM_QUERYNEWPALETTE](#)
- [WM_SYSCOLORCHANGE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

WM_PALETTECHANGED message

Article • 01/07/2021

The **WM_PALETTECHANGED** message is sent to all top-level and overlapped windows after the window with the keyboard focus has realized its logical palette, thereby changing the system palette. This message enables a window that uses a color palette but does not have the keyboard focus to realize its logical palette and update its client area.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam
);
```

Parameters

wParam

A handle to the window that caused the system palette to change.

lParam

This parameter is not used.

Remarks

This message must be sent to all top-level and overlapped windows, including the one that changed the system palette. If any child windows use a color palette, this message must be passed on to them as well.

To avoid creating an infinite loop, a window that receives this message must not realize its palette, unless it determines that *wParam* does not contain its own window handle.

Requirements

Requirement	Value
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

[Colors Overview](#)

[Color Messages](#)

[WM_PALETTEISCHANGING](#)

[WM_QUERYNEWPALETTE](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_PALETTEISCHANGING message

Article • 01/07/2021

The **WM_PALETTEISCHANGING** message informs applications that an application is going to realize its logical palette.

A window receives this message through its **WindowProc** function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

A handle to the window that is going to realize its logical palette.

lParam

This parameter is not used.

Return value

If an application processes this message, it should return zero.

Remarks

The application changing its palette does not wait for acknowledgment of this message before changing the palette and sending the **WM_PALETTECHANGED** message. As a result, the palette may already be changed by the time an application receives this message.

If the application either ignores or fails to process this message and a second application realizes its palette while the first is using palette indexes, there is a strong possibility that the user will see unexpected colors during subsequent drawing operations.

Requirements

Requirement	Value
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

[Colors Overview](#)

[Color Messages](#)

[WM_PALETTECHANGED](#)

[WM_QUERYNEWPALETTE](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_QUERYNEWPALETTE message

Article • 01/07/2021

The **WM_QUERYNEWPALETTE** message informs a window that it is about to receive the keyboard focus, giving the window the opportunity to realize its logical palette when it receives the focus.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

This parameter is not used.

lParam

This parameter is not used.

Return value

If the window realizes its logical palette, it must return **TRUE**; otherwise, it must return **FALSE**.

Requirements

Requirement	Value
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

[Colors Overview](#)

[Color Messages](#)

[WM_PALETTECHANGED](#)

[WM_PALETTEISCHANGING](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_SYSCOLORCHANGE message

Article • 01/07/2021

The **WM_SYSCOLORCHANGE** message is sent to all top-level windows when a change is made to a system color setting.

A window receives this message through its **WindowProc** function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

This parameter is not used.

lParam

This parameter is not used.

Remarks

The system sends a **WM_PAINT** message to any window that is affected by a system color change.

Applications that have brushes using the existing system colors should delete those brushes and re-create them using the new system colors.

Top level windows that use common controls must forward the **WM_SYSCOLORCHANGE** message to the controls; otherwise, the controls will not be notified of the color change. This ensures that the colors used by your common controls are consistent with those used by other user interface objects. For example, a toolbar control uses the "3D Objects" color to draw its buttons. If the user changes the 3D Objects color but the **WM_SYSCOLORCHANGE** message is not forwarded to the toolbar,

the toolbar buttons will remain in their original color while the color of other buttons in the system changes.

Requirements

Requirement	Value
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

[Colors Overview](#)

[Color Messages](#)

[WM_PAINT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Coordinate Spaces and Transformations

Article • 01/07/2021

Applications use coordinate spaces and transformations to scale, rotate, translate, shear, and reflect graphics output. A *coordinate space* is a planar space that locates two-dimensional objects by using two reference axes that are perpendicular to each other. There are four coordinate spaces: world, page, device, and physical device (client area, desktop, or page of printer paper).

A *transformation* is an algorithm that alters ("transforms") the size, orientation, and shape of objects. Transformations also transfer a graphics object from one coordinate space to another. Ultimately, the object appears on the physical device, which is usually a screen or printer.

This chapter is divided into three topics:

- [About Coordinate Spaces and Transformations](#)
- [Using Coordinate Spaces and Transformations](#)
- [Coordinate Space and Transformation Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

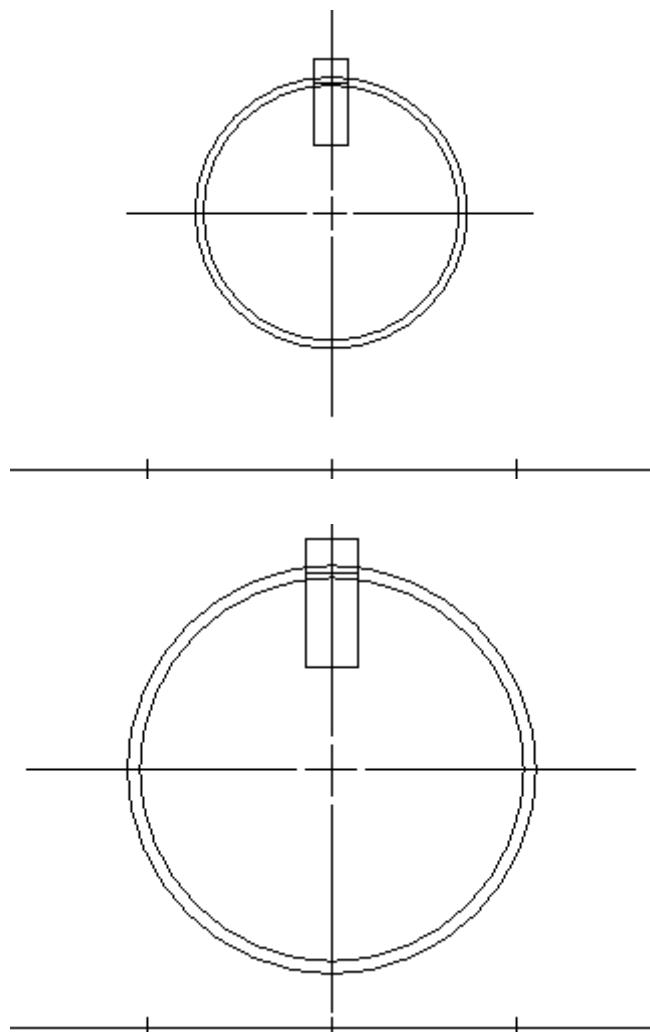
About Coordinate Spaces and Transformations

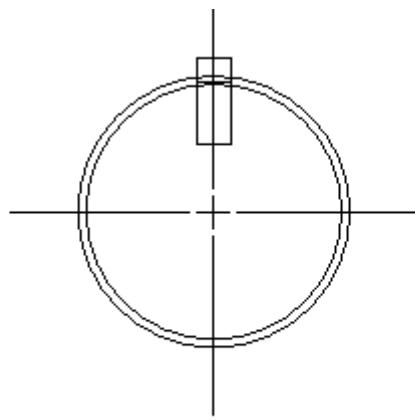
Article • 01/07/2021

Coordinate spaces and transformations are used by the following types of applications:

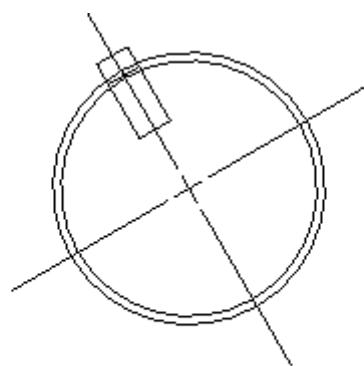
- Desktop publishing applications (to "zoom" parts of a page or to display adjacent pages in a window).
- Computer-aided design (CAD) applications (to rotate objects, scale drawings, or create perspective views).
- Spreadsheet applications (to move and size graphs).

The following illustrations show successive views of an object created in a drawing application. The first illustration shows the object as it appears in the original drawing; the succeeding five illustrations show the effects of applying various transformations.

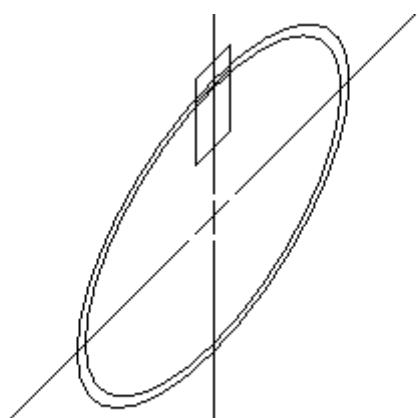




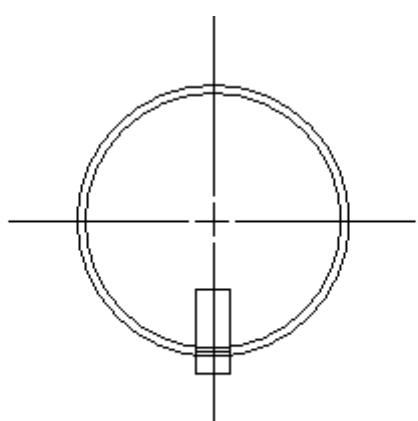
— + + —



— + + —



— + + —



— + + —

The following subtopics describe various aspects of coordinate spaces and transformations.

- Transformation of Coordinate Spaces
- World-Space to Page-Space Transformations
- Page-Space to Device-Space Transformations
- Device-Space to Physical-Device Transformation
- Default Transformations

Feedback

Was this page helpful?

 Yes

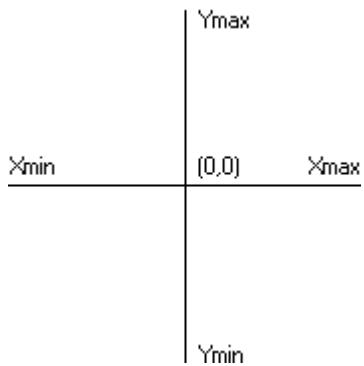
 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Transformation of Coordinate Spaces

Article • 01/07/2021

A *coordinate space* is a planar space based on the Cartesian coordinate system. This system provides a means of specifying the location of each point on a plane. It requires two axes that are perpendicular and equal in length. The following illustration shows a coordinate space.



The system supports four coordinate spaces, as described in the following table.

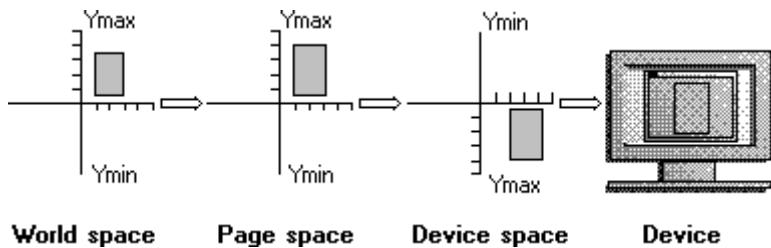
[] [Expand table](#)

Coordinate space	Description
world	Used optionally as the starting coordinate space for graphics transformations. It allows scaling, translation, rotation, shearing, and reflection. World space measures 2^{32} units high by 2^{32} units wide.
page	Used either as the next space after world space or as the starting space for graphics transformations. It sets the mapping mode. Page space also measures 2^{32} units high by 2^{32} units wide.
device	Used as the next space after page space. It only allows translation, which ensures the origin of the device space maps to the proper location in physical device space. Device space measures 2^{27} units high by 2^{27} units wide.
physical device	The final (output) space for graphics transformations. It usually refers to the client area of the application window; however, it can also include the entire desktop, a complete window (including the frame, title bar, and menu bar), or a page of printer or plotter paper, depending on the function that obtained the handle to the device context. Physical device dimensions vary according to the dimensions set by the display, printer, or plotter technology.

Page space works with device space to provide applications with device-independent units, such as millimeters and inches. This overview refers to both world space and page space as logical space.

To depict output on a physical device, the system copies (or maps) a rectangular region from one coordinate space into the next using a transformation until the output appears in its entirety on the physical device. Mapping begins in the application's world space if the application has called the [SetWorldTransform](#) function; otherwise, mapping occurs in page space. As the system copies each point within the rectangular region from one space into another, it applies an algorithm called a transformation. A *transformation* alters (or transforms) the size, orientation, and shape of objects that are copied from one coordinate space into another. Although a transformation affects an object as a whole, it is applied to each point, or to each line, in the object.

The following illustration shows a typical transformation performed by using the [SetWorldTransform](#) function.



Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

World-Space to Page-Space Transformations

Article • 01/07/2021

World-space to page-space transformations support translation and scaling. In addition, they support rotation, shear, and reflection capabilities. The following sections describe these transformations, illustrate their effects, and provide the algorithms used to achieve them.

- [Translation](#)
- [Scaling](#)
- [Rotation](#)
- [Shear](#)
- [Reflection](#)
- [Combined World-to-Page Space Transformations](#)

Feedback

Was this page helpful?



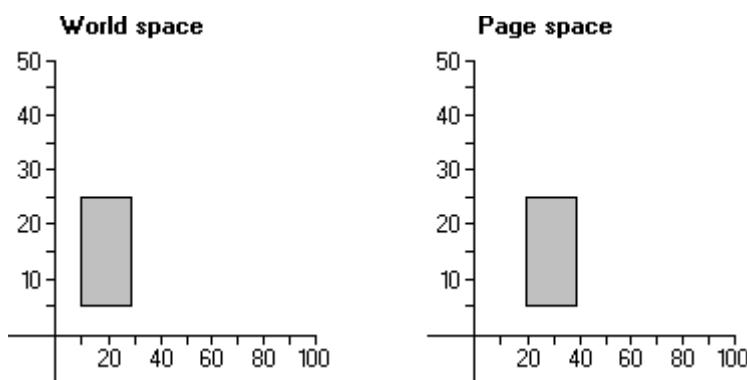
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Translation

Article • 01/07/2021

Some applications translate (or shift) objects drawn in the client area by calling the **SetWorldTransform** function to set the appropriate world-space to page-space transformation. The SetWorldTransform function receives a pointer to an **XFORM** structure containing the appropriate values. The eDx and eDy members of XFORM specify the horizontal and vertical translation components, respectively.

When *translation* occurs, each point in an object is shifted vertically, horizontally, or both, by a specified amount. The following illustration shows a 20- by 20-unit rectangle that was translated to the right by 10 units when copied from world-coordinate space to page-coordinate space.



In the preceding illustration, the x-coordinate of each point in the rectangle is 10 units greater than the original x-coordinate.

Horizontal translation can be represented by the following algorithm.

syntax

```
x' = x + Dx
```

Where x' is the new x-coordinate, x is the original x-coordinate, and Dx is the horizontal distance moved.

Vertical translation can be represented by the following algorithm.

syntax

```
y' = y + Dy
```

Where y' is the new y-coordinate, y is the original y-coordinate, and Dy is the vertical distance moved.

The horizontal and vertical translation transformations can be combined into a single operation by using a 3-by-3 matrix.

syntax

$$\begin{vmatrix} x' & y' & 1 \end{vmatrix} = \begin{vmatrix} x & y & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{vmatrix}$$

(The rules of matrix multiplication state that the number of rows in one matrix must equal the number of columns in the other. The integer 1 in the matrix $|x\ y\ 1|$ is a placeholder that was added to meet this requirement.)

The 3-by-3 matrix that produced the illustrated translation transformation contains the following values.

syntax

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 10 & 0 & 1 \end{vmatrix}$$

Feedback

Was this page helpful?

 Yes

 No

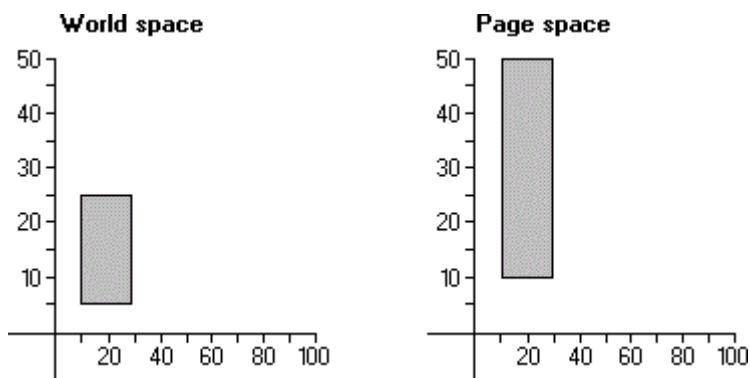
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Scaling

Article • 01/07/2021

Most CAD and drawing applications provide features that scale output created by the user. Applications that include scaling (or zoom) capabilities call the [SetWorldTransform](#) function to set the appropriate world-space to page-space transformation. This function receives a pointer to an [XFORM](#) structure containing the appropriate values. The eM11 and eM22 members of XFORM specify the horizontal and vertical scaling components, respectively.

When *scaling* occurs, the vertical and horizontal lines (or vectors), that constitute an object, are stretched or compressed with respect to the x- or y-axis. The following illustration shows a 20-by-20-unit rectangle scaled vertically to twice its original height when copied from world-coordinate space to page-coordinate space.



In the preceding illustration, the vertical lines that define the original rectangle's side measure 20 units, while the vertical lines that define the scaled rectangle's sides measure 40 units.

Vertical scaling can be represented by the following algorithm.

syntax

```
y' = y * Dy
```

Where y' is the new length, y is the original length, and Dy is the vertical scaling factor.

Horizontal scaling can be represented by the following algorithm.

syntax

```
x' = x * Dx
```

Where x' is the new length, x is the original length, and Dx is the horizontal scaling factor.

The vertical and horizontal scaling transformations can be combined into a single operation by using a 2-by-2 matrix.

syntax

$$\begin{vmatrix} x' & y' \end{vmatrix} = \begin{vmatrix} Dx & 0 \\ 0 & Dy \end{vmatrix} * \begin{vmatrix} x & y \end{vmatrix}$$

The 2-by-2 matrix that produced the scaling transformation contains the following values.

syntax

$$\begin{vmatrix} 1 & 0 \\ 0 & 2 \end{vmatrix}$$

Feedback

Was this page helpful?

 Yes

 No

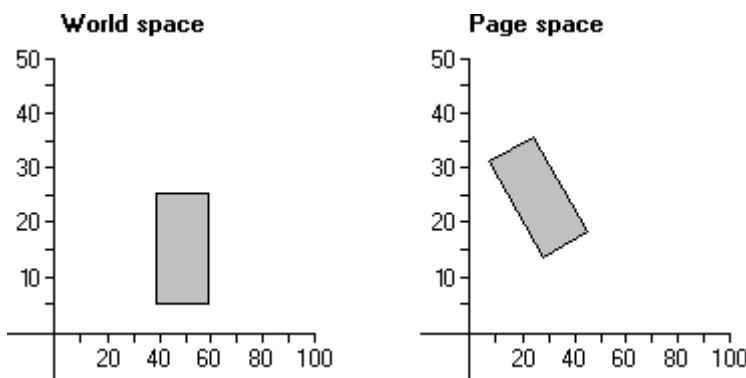
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Rotation

Article • 01/07/2021

Many CAD applications provide features that rotate objects drawn in the client area. Applications that include rotation capabilities use the [SetWorldTransform](#) function to set the appropriate world-space to page-space transformation. This function receives a pointer to an [XFORM](#) structure containing the appropriate values. The eM11, eM12, eM21, and eM22 members of XFORM specify respectively, the cosine, sine, negative sine, and cosine of the angle of rotation.

When *rotation* occurs, the points that constitute an object are rotated with respect to the coordinate-space origin. The following illustration shows a 20-by-20-unit rectangle rotated 30 degrees when copied from world-coordinate space to page-coordinate space.



In the preceding illustration, each point in the rectangle was rotated 30 degrees with respect to the coordinate-space origin.

The following algorithm computes the new x-coordinate (x') for a point (x,y) that is rotated by angle A with respect to the coordinate-space origin.

syntax

```
x' = (x * cos A) - (y * sin A)
```

The following algorithm computes the y-coordinate (y') for a point (x,y) that is rotated by the angle A with respect to the origin.

syntax

```
y' = (x * sin A) + (y * cos A)
```

The two rotation transformations can be combined in a 2-by-2 matrix as follows.

syntax

$$|x' \ y'| = |x \ y| * \begin{vmatrix} \cos A & \sin A \\ -\sin A & \cos A \end{vmatrix}$$

The 2-by-2 matrix that produced the rotation contains the following values.

syntax

.8660	.5000
-.5000	.8660

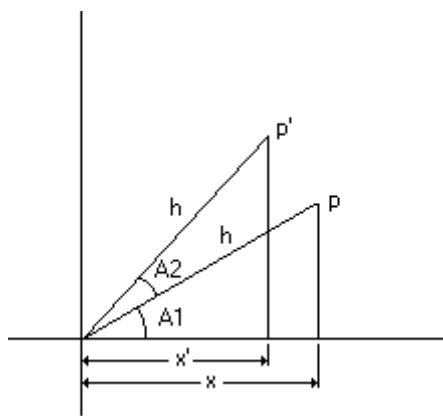
Rotation Algorithm Derivation

Rotation algorithms are based on trigonometry's addition theorem stating that the trigonometric function of a sum of two angles (A_1 and A_2) can be expressed in terms of the trigonometric functions of the two angles.

syntax

$$\begin{aligned}\sin(A_1 + A_2) &= (\sin A_1 * \cos A_2) + (\cos A_1 * \sin A_2) \\ \cos(A_1 + A_2) &= (\cos A_1 * \cos A_2) - (\sin A_1 * \sin A_2)\end{aligned}$$

The following illustration shows a point p rotated counterclockwise to a new position p' . In addition, it shows two triangles formed by a line drawn from the coordinate-space origin to each point and a line drawn from each point through the x -axis.



Using trigonometry, the x-coordinate of point p can be obtained by multiplying the length of the hypotenuse h by the cosine of A1.

syntax

$$x = h * \cos A1$$

The y-coordinate of point p can be obtained by multiplying the length of the hypotenuse h by the sine of A1.

syntax

```
y = h * sin A1
```

Likewise, the x-coordinate of point p' can be obtained by multiplying the length of the hypotenuse h by the cosine of (A1 + A2).

syntax

```
x' = h * cos (A1 + A2)
```

Finally, the y-coordinate of point p' can be obtained by multiplying the length of the hypotenuse h by the sine of (A1 + A2).

syntax

```
y' = h * sin (A1 + A2)
```

Using the addition theorem, the preceding algorithms become the following:

syntax

```
x' = (h * cos A1 * cos A2) - (h * sin A1 * sin A2)  
y' = (h * cos A1 * sin A2) + (h * sin A1 * cos A2)
```

The rotation algorithms for a given point rotated by angle A2 can be obtained by substituting x for each occurrence of ($h * \cos A1$) and by substituting y for each occurrence of ($h * \sin A1$).

syntax

```
x' = (x * cos A2) - (y * sin A2)  
y' = (x * sin A2) + (y * cos A2)
```

Feedback

Was this page helpful?

 Yes

 No

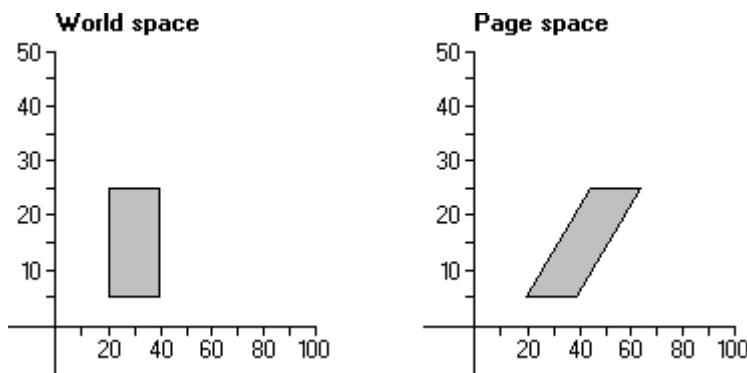
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Shear

Article • 01/07/2021

Some applications provide features that shear objects drawn in the client area. Applications that use shear capabilities use the [SetWorldTransform](#) function to set appropriate values in the world-space to page-space transformation. This function receives a pointer to an [XFORM](#) structure containing the appropriate values. The eM12 and eM21 members of XFORM specify the horizontal and vertical proportionality constants, respectively.

There are two components of the *shear transformation*. The first alters the vertical lines in an object; the second alters the horizontal lines. The following illustration shows a 20-by-20-unit rectangle sheared horizontally when copied from world space to page space.



A horizontal shear can be represented by the following algorithm:

syntax

```
x' = x + (Sx * y)
```

where x is the original x-coordinate, Sx is the proportionality constant, and x' is the result of the shear transformation.

A vertical shear can be represented by the following algorithm:

syntax

```
y' = y + (Sy * x)
```

where y is the original y-coordinate, Sy is the proportionality constant, and y' is the result of the shear transformation.

The horizontal-shear and vertical-shear transformations can be combined into a single operation using a 2-by-2 matrix.

syntax

```
|x' y'| == |x y| * | 1 Sx|
           | Sy   1|
```

The 2-by-2 matrix that produced the shear contains the following values:

syntax

```
|1 1|
 |0 1|
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Reflection

Article • 07/28/2022

Some applications provide features that reflect (or mirror) objects drawn in the client area. Applications that contain reflection capabilities use the [SetWorldTransform](#) function to set the appropriate values in the world-space to page-space transformation. This function receives a pointer to an [XFORM](#) structure containing the appropriate values. The eM11 and eM22 members of XFORM specify the horizontal and vertical reflection components, respectively.

The *reflection transformation* creates a mirror image of an object with respect to either the x- or y-axis. In short, reflection is just negative scaling. To produce a horizontal reflection, x-coordinates are multiplied by -1. To produce a vertical reflection, y-coordinates are multiplied by -1.

Horizontal reflection can be represented by the following algorithm:

syntax

```
x' = -x
```

where x is the x-coordinate and x' is the result of the reflection.

The 2-by-2 matrix that produced horizontal reflection contains the following values:

syntax

$$\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$$

Vertical reflection can be represented by the following algorithm:

syntax

```
y' = -y
```

where y is the y-coordinate and y' is the result of the reflection.

The 2-by-2 matrix that produced vertical reflection contains the following values:

syntax

$$\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$$

The horizontal-reflection and vertical-reflection operations can be combined into a single operation by using the following 2-by-2 matrix:

syntax

$$\begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$$

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Combined World-to-Page Space Transformations

Article • 01/07/2021

The five world-to-page transformations can be combined into a single 3-by-3 matrix. The [CombineTransform](#) function can be used to combine two world-space to page-space transformations. The combined transformations can be used to alter output associated with a particular device context (DC) by calling the [SetWorldTransform](#) function and supplying the elements for this matrix. When an application calls [SetWorldTransform](#), it stores the elements of the 3-by-3 matrix in an [XFORM](#) structure. The members of this structure correspond to the first two columns of a 3-by-3 matrix; the last column of the matrix is not required because its values are constant.

The elements of the current world transformation matrix can be revived by calling the [GetWorldTransform](#) function and supplying a pointer to an [XFORM](#) structure.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Page-Space to Device-Space Transformations

Article • 01/07/2021

The page-space to device-space transformation determines the mapping mode for all graphics output associated with a particular DC. A *mapping mode* is a scaling transformation that specifies the size of the units used for drawing operations. The mapping mode may also perform translation. In some cases, the mapping mode alters the orientation of the x-axis and y-axis in device space. The following topics describe the mapping modes.

- [Mapping Modes and Translations](#)
- [Predefined Mapping Modes](#)
- [Application-Defined Mapping Modes](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Mapping Modes and Translations

Article • 01/07/2021

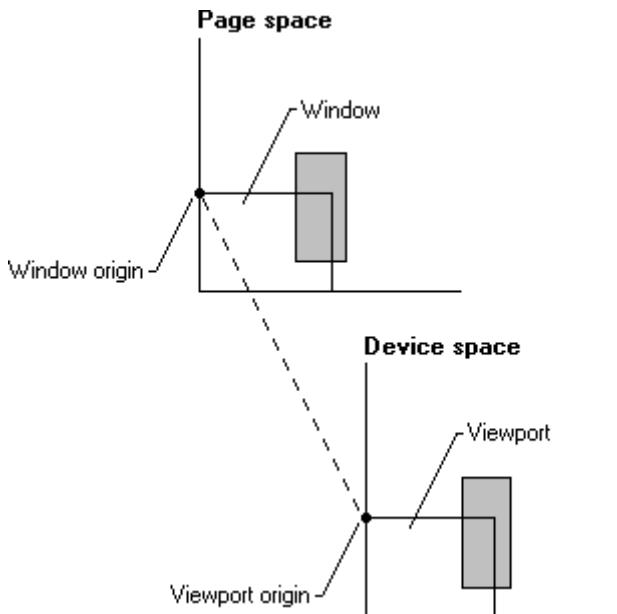
The mapping modes are described in the following table.

[+] Expand table

Mapping mode	Description
MM_ANISOTROPIC	Each unit in page space is mapped to an application-specified unit in device space. The axis may or may not be equally scaled (for example, a circle drawn in world space may appear to be an ellipse when depicted on a given device). The orientation of the axis is also specified by the application.
MM_HIENGLISH	Each unit in page space is mapped to 0.001 inch in device space. The value of x increases from left to right. The value of y increases from bottom to top.
MM HIMETRIC	Each unit in page space is mapped to 0.01 millimeter in device space. The value of x increases from left to right. The value of y increases from bottom to top.
MM_ISOTROPIC	Each unit in page space is mapped to an application-defined unit in device space. The axes are always equally scaled. The orientation of the axes may be specified by the application.
MM_LOENGLISH	Each unit in page space is mapped to 0.01 inch in device space. The value of x increases from left to right. The value of y increases from bottom to top.
MM_LOMETRIC	Each unit in page space is mapped to 0.1 millimeter in device space. The value of x increases from left to right. The value of y increases from bottom to top.
MM_TEXT	Each unit in page space is mapped to one pixel; that is, no scaling is performed at all. When no translation is in effect (this is the default), page space in the MM_TEXT mapping mode is equivalent to physical device space. The value of x increases from left to right. The value of y increases from top to bottom.
MM_TWIPS	Each unit in page space is mapped to one twentieth of a printer's point (1/1440 inch). The value of x increases from left to right. The value of y increases from bottom to top.

To set a mapping mode, call the [SetMapMode](#) function. Retrieve the current mapping mode for a DC by calling the [GetMapMode](#) function.

The page-space to device-space transformations consist of values calculated from the points given by the window and viewport. In this context, the window refers to the logical coordinate system of the page space, while the viewport refers to the device coordinate system of the device space. The window and viewport each consist of an origin, a horizontal ("x") extent, and a vertical ("y") extent. The window parameters are in logical coordinates; the viewport in device coordinates (pixels). The system combines the origins and extents from both the window and viewport to create the transformation. This means that the window and viewport each specify half of the factors needed to define the transformation used to map points in page space to device space. Thus, the system maps the window origin to the viewport origin and the window extents to the viewport extents, as shown in the following illustration.



The window and viewport extents establish a ratio or scaling factor used in the page-space to device-space transformations. For the six predefined mapping modes (MM_HIENGLISH, MM_LOENGLISH, MM_HIMETRIC, MM_LOMETRIC, MM_TEXT, and MM_TWIPS), the extents are set by the system when [SetMapMode](#) is called. They cannot be changed. The other two mapping modes (MM_ISOTROPIC and MM_ANISOTROPIC) require that the extents are specified. This is done by calling [SetMapMode](#) to set the appropriate mode and then calling the [SetWindowExtEx](#) and [SetViewportExtEx](#) functions to specify the extents. In the MM_ISOTROPIC mapping mode, it is important to call [SetWindowExtEx](#) before calling [SetViewportExtEx](#).

The window and viewport origins establish the translation used in the page-space to device-space transformations. Set the window and viewport origins by using the [SetWindowOrgEx](#) and [SetViewportOrgEx](#) functions. The origins are independent of the extents, and an application can set them regardless of the current mapping mode. Changing a mapping mode does not affect the currently set origins (although it can affect the extents). Origins are specified in absolute units that the current mapping

mode does not affect. To alter the origins, use the [OffsetWindowOrgEx](#) and [OffsetViewportOrgEx](#) functions.

The following formula shows the math involved in converting a point from page space to device space.

syntax

```
Dx = ((Lx - W0x) * VEx / WEx) + V0x
```

The following variables are involved.

syntax

Dx	x value in device units
Lx	x value in logical units (also known as page space units)
W0x	window x origin
V0x	viewport x origin
WEx	window x-extent
VEx	viewport x-extent

The same equation with y replacing x transforms the y component of a point.

The formula first offsets the point from its coordinate origin. This value, no longer biased by the origin, is then scaled into the destination coordinate system by the ratio of the extents. Finally, the scaled value is offset by the destination origin to its final mapping.

The [LPtoDP](#) and [DPtoLP](#) functions may be used to convert from logical points to device points and from device points to logical points, respectively.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Predefined Mapping Modes

Article • 01/07/2021

Of the six predefined mapping modes, one is device dependent (MM_TEXT) and the remaining five (MM_HIENGLISH, MM_LOENGLISH, MM_HIMETRIC, MM_LOMETRIC, and MM_TWIPS) are device independent.

The default mapping mode is MM_TEXT. One logical unit equals one pixel. Positive x is to the right, and positive y is down. This mode maps directly to the device's coordinate system. The logical-to-physical mapping involves only an offset in x and y that is defined by the application-controlled window and viewport origins. The viewport and window extents are all set to 1, creating a one-to-one mapping.

Applications that display geometric shapes (circles, squares, polygons, and so on) make use of one of the device-independent mapping modes. For example, if you are writing an application to provide charting capabilities for a spreadsheet program and want to guarantee that the diameter of each pie chart is 2 inches, use the MM_LOENGLISH mapping mode and call the appropriate functions to draw and fill the chart. Specifying MM_LOENGLISH, guarantees that the diameter of the chart is consistent on any display or printer. If MM_TEXT is used instead of MM_LOENGLISH, a chart that appears circular on a VGA display would appear elliptical on an EGA display and would appear very small on a 300 dpi (dots per inch) laser printer.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Application-Defined Mapping Modes

Article • 01/07/2021

The two application-defined mapping modes (MM_ISOTROPIC and MM_ANISOTROPIC) are provided for application-specific mapping modes. The MM_ISOTROPIC mode guarantees that logical units in the x-direction and in the y-direction are equal, while the MM_ANISOTROPIC mode allows the units to differ. A CAD or drawing application can benefit from the MM_ISOTROPIC mapping mode but may need to specify logical units that correspond to the increments on an engineer's scale (1/64 inch). These units would be difficult to obtain with the predefined mapping modes (MM_HIENGLISH or MM_HIMETRIC); however, they can easily be obtained by selecting the MM_ISOTROPIC (or MM_ANISOTROPIC) mode. The following example shows how to set logical units to 1/64 inch:

C++

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 64, 64, NULL);
SetViewportExtEx(hDC, GetDeviceCaps(hdc, LOGPIXELSX),
                 GetDeviceCaps(hdc, LOGPIXELSY), NULL);
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device-Space to Physical-Device Transformation

Article • 01/07/2021

The device-space to physical-device transformation is unique in several respects. For example, it is limited to translation and is controlled by the system. The sole purpose of this transformation is to ensure that the origin of device space is mapped to the proper point on the physical device. There are no functions to set this transformation, nor are there any functions to retrieve related data.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Default Transformations

Article • 01/07/2021

Whenever an application creates a DC and immediately begins calling GDI drawing or output functions, it takes advantage of the default page-space to device-space, and device-space to client-area transformations. A world-to-page space transformation cannot happen until the application first calls the [SetGraphicsMode](#) function to set the mode to GM_ADVANCED and then calls the [SetWorldTransform](#) function.

Use of MM_TEXT (the default page-space to device-space transformation) results in a one-to-one mapping; that is, a given point in page space maps to the same point in device space. As previously mentioned, this transformation is not specified by a matrix. Instead, it is obtained by dividing the width of the viewport by the width of the window and the height of the viewport by the height of the window. In the default case, the viewport dimensions are 1-pixel by 1-pixel and the window dimensions are 1-page unit by 1-page unit.

The device-space to physical-device (client area, desktop, or printer paper) transformation always results in a one-to-one mapping; that is, one unit in device space is always equivalent to one unit in the client area, on the desktop, or on a page. The sole purpose of this transformation is translation; it ensures that output appears correctly in an application's window no matter where that window is moved on the desktop.

The one unique aspect of MM_TEXT is the orientation of the y-axis in page space. In MM_TEXT, the positive y-axis extends downward and the negative y-axis extends upward.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Coordinate Spaces and Transformations

Article • 01/07/2021

This section contains an example that demonstrates the following tasks:

- Drawing graphics with predefined units.
- Centering graphics in the application's client area.
- Scaling graphics output to half its original size.
- Translating graphics output 3/4 of an inch to the right.
- Rotating graphics 30 degrees.
- Shearing graphics output along the x-axis.
- Reflecting graphics output about an imaginary horizontal axis drawn through its midpoint.

The following example was used to create the illustrations that appear earlier in this overview.

C++

```
void TransformAndDraw(int iTransform, HWND hWnd)
{
    HDC hDC;
    XFORM xForm;
    RECT rect;

    // Retrieve a DC handle for the application's window.

    hDC = GetDC(hWnd);

    // Set the mapping mode to LOENGLISH. This moves the
    // client area origin from the upper left corner of the
    // window to the lower left corner (this also reorients
    // the y-axis so that drawing operations occur in a true
    // Cartesian space). It guarantees portability so that
    // the object drawn retains its dimensions on any display.

    SetGraphicsMode(hDC, GM_ADVANCED);
    SetMapMode(hDC, MM_LOENGLISH);

    // Set the appropriate world transformation (based on the
    // user's menu selection).

    switch (iTransform)
    {
        case SCALE:           // Scale to 1/2 of the original size.
            xForm.eM11 = (FLOAT) 0.5;
```

```

xForm.eM12 = (FLOAT) 0.0;
xForm.eM21 = (FLOAT) 0.0;
xForm.eM22 = (FLOAT) 0.5;
xForm.eDx = (FLOAT) 0.0;
xForm.eDy = (FLOAT) 0.0;
SetWorldTransform(hDC, &xForm);
break;

case TRANSLATE: // Translate right by 3/4 inch.
xForm.eM11 = (FLOAT) 1.0;
xForm.eM12 = (FLOAT) 0.0;
xForm.eM21 = (FLOAT) 0.0;
xForm.eM22 = (FLOAT) 1.0;
xForm.eDx = (FLOAT) 75.0;
xForm.eDy = (FLOAT) 0.0;
SetWorldTransform(hDC, &xForm);
break;

case ROTATE: // Rotate 30 degrees counterclockwise.
xForm.eM11 = (FLOAT) 0.8660;
xForm.eM12 = (FLOAT) 0.5000;
xForm.eM21 = (FLOAT) -0.5000;
xForm.eM22 = (FLOAT) 0.8660;
xForm.eDx = (FLOAT) 0.0;
xForm.eDy = (FLOAT) 0.0;
SetWorldTransform(hDC, &xForm);
break;

case SHEAR: // Shear along the x-axis with a
// proportionality constant of 1.0.
xForm.eM11 = (FLOAT) 1.0;
xForm.eM12 = (FLOAT) 1.0;
xForm.eM21 = (FLOAT) 0.0;
xForm.eM22 = (FLOAT) 1.0;
xForm.eDx = (FLOAT) 0.0;
xForm.eDy = (FLOAT) 0.0;
SetWorldTransform(hDC, &xForm);
break;

case REFLECT: // Reflect about a horizontal axis.
xForm.eM11 = (FLOAT) 1.0;
xForm.eM12 = (FLOAT) 0.0;
xForm.eM21 = (FLOAT) 0.0;
xForm.eM22 = (FLOAT) -1.0;
xForm.eDx = (FLOAT) 0.0;
xForm.eDy = (FLOAT) 0.0;
SetWorldTransform(hDC, &xForm);
break;

case NORMAL: // Set the unity transformation.
xForm.eM11 = (FLOAT) 1.0;
xForm.eM12 = (FLOAT) 0.0;
xForm.eM21 = (FLOAT) 0.0;
xForm.eM22 = (FLOAT) 1.0;
xForm.eDx = (FLOAT) 0.0;

```

```
    xForm.eDy = (FLOAT) 0.0;
    SetWorldTransform(hDC, &xForm);
    break;

}

// Find the midpoint of the client area.

GetClientRect(hWnd, (LPRECT) &rect);
DPtoLP(hDC, (LPPOINT) &rect, 2);

// Select a hollow brush.

SelectObject(hDC, GetStockObject(HOLLOW_BRUSH));

// Draw the exterior circle.

Ellipse(hDC, (rect.right / 2 - 100), (rect.bottom / 2 + 100),
        (rect.right / 2 + 100), (rect.bottom / 2 - 100));

// Draw the interior circle.

Ellipse(hDC, (rect.right / 2 - 94), (rect.bottom / 2 + 94),
        (rect.right / 2 + 94), (rect.bottom / 2 - 94));

// Draw the key.

Rectangle(hDC, (rect.right / 2 - 13), (rect.bottom / 2 + 113),
          (rect.right / 2 + 13), (rect.bottom / 2 + 50));
Rectangle(hDC, (rect.right / 2 - 13), (rect.bottom / 2 + 96),
          (rect.right / 2 + 13), (rect.bottom / 2 + 50));

// Draw the horizontal lines.

MoveToEx(hDC, (rect.right/2 - 150), (rect.bottom / 2 + 0), NULL);
LineTo(hDC, (rect.right / 2 - 16), (rect.bottom / 2 + 0));

MoveToEx(hDC, (rect.right / 2 - 13), (rect.bottom / 2 + 0), NULL);
LineTo(hDC, (rect.right / 2 + 13), (rect.bottom / 2 + 0));

MoveToEx(hDC, (rect.right / 2 + 16), (rect.bottom / 2 + 0), NULL);
LineTo(hDC, (rect.right / 2 + 150), (rect.bottom / 2 + 0));

// Draw the vertical lines.

MoveToEx(hDC, (rect.right/2 + 0), (rect.bottom / 2 - 150), NULL);
LineTo(hDC, (rect.right / 2 + 0), (rect.bottom / 2 - 16));

MoveToEx(hDC, (rect.right / 2 + 0), (rect.bottom / 2 - 13), NULL);
LineTo(hDC, (rect.right / 2 + 0), (rect.bottom / 2 + 13));

MoveToEx(hDC, (rect.right / 2 + 0), (rect.bottom / 2 + 16), NULL);
LineTo(hDC, (rect.right / 2 + 0), (rect.bottom / 2 + 150));
```

```
    ReleaseDC(hWnd, hDC);  
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Coordinate Space and Transformation Reference

Article • 01/07/2021

The following elements are used with coordinate spaces and transformations.

- [Coordinate Space and Transformation Functions](#)
- [Coordinate Space and Transformation Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Coordinate Space and Transformation Functions

Article • 01/07/2021

The following functions are used with coordinate spaces and transformations.

[+] Expand table

Function	Description
ClientToScreen	Converts the client-area coordinates of a specified point to screen coordinates.
CombineTransform	Concatenates two world-space to page-space transformations.
DPtoLP	Converts device coordinates into logical coordinates.
GetCurrentPositionEx	Retrieves the current position in logical coordinates.
GetDisplayAutoRotationPreferences	Gets the orientation preferences of the display.
GetGraphicsMode	Retrieves the current graphics mode for the specified device context.
GetMapMode	Retrieves the current mapping mode.
GetViewportExtEx	Retrieves the x-extent and y-extent of the current viewport for the specified device context.
GetViewportOrgEx	Retrieves the x-coordinates and y-coordinates of the viewport origin for the specified device context.
GetWindowExtEx	Retrieves the x-extent and y-extent of the window for the specified device context.
GetWindowOrgEx	Retrieves the x-coordinates and y-coordinates of the window origin for the specified device context.
GetWorldTransform	Retrieves the current world-space to page-space transformation.
LPtoDP	Converts logical coordinates into device coordinates.
MapWindowPoints	Converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.

Function	Description
ModifyWorldTransform	Changes the world transformation for a device context using the specified mode.
OffsetViewportOrgEx	Modifies the viewport origin for a device context using the specified horizontal and vertical offsets.
OffsetWindowOrgEx	Modifies the window origin for a device context using the specified horizontal and vertical offsets.
ScaleViewportExtEx	Modifies the viewport for a device context using the ratios formed by the specified multiplicands and divisors.
ScaleWindowExtEx	Modifies the window for a device context using the ratios formed by the specified multiplicands and divisors.
ScreenToClient	Converts the screen coordinates of a specified point on the screen to client coordinates.
SetDisplayAutoRotationPreferences ↴	Sets the orientation preferences of the display.
SetGraphicsMode	Sets the graphics mode for the specified device context.
SetMapMode	Sets the mapping mode of the specified device context.
SetViewportExtEx	Sets the horizontal and vertical extents of the viewport for a device context by using the specified values.
SetViewportOrgEx	Specifies which device point maps to the window origin (0,0).
SetWindowExtEx	Sets the horizontal and vertical extents of the window for a device context by using the specified values.
SetWindowOrgEx	Specifies which window point maps to the viewport origin (0,0).
SetWorldTransform	Sets a two-dimensional linear transformation between world space and page space for the specified device context.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ClientToScreen function (winuser.h)

02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

CombineTransform function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

DPtoLP function (wingdi.h)

Article 02/22/2024

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

C++

```
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCurrentPositionEx function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[MoveToEx](#)

[POINT](#)

GetGraphicsMode function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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.

[Expand table](#)

Value	Meaning
<code>GM_COMPATIBLE</code>	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.
<code>GM_ADVANCED</code>	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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetMapMode function (wingdi.h)

Article10/13/2021

The **GetMapMode** function retrieves the current mapping mode.

Syntax

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetViewportExtEx function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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](#)

GetWorldTransform function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

LPtoDP function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

MapWindowPoints function (winuser.h)

11/19/2022

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

C++

```
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:

C++

```
RECT      rc[10];

for(int i = 0; i < (sizeof(rc)/sizeof(rc[0])); i++)
{
    MapWindowPoints(hWnd1, hWnd2, (LPOINT)&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:

C++

```
POINT pt[2];

MapWindowPoints(hWnd1, hWnd2, &pt[0], 1);
```

```
MapWindowPoints(hwnd1, hwnd2, &pt[1], 1);
```

Requirements

 [Expand table](#)

Requirement	Value
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](#)

ModifyWorldTransform function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
MWT_IDENTITY	Resets the current world transformation by using the identity matrix. If this mode is specified, the XFORM structure pointed to by <i>lpxform</i> is ignored.
MWT_LEFTMULTIPLY	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.)

MWT_RIGHTMULTIPLY	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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

OffsetViewportOrgEx function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ScaleViewportExtEx function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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 *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 viewport extents are modified as follows:

C++

```
xNewVE = (xOldVE * Xnum) / Xdenom  
yNewVE = (yOldVE * Ynum) / Ydenom
```

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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 *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 extents are modified as follows:

C++

```
xNewWE = (xOldWE * Xnum) / Xdenom  
yNewWE = (yOldWE * Ynum) / Ydenom
```

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ScreenToClient function (winuser.h)

11/19/2022

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

SetGraphicsMode function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

[] Expand table

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetMapMode function (wingdi.h)

Article10/13/2021

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

C++

```
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.

[+] Expand table

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).
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.

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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetViewportExtEx function (wingdi.h)

Article 11/19/2022

The **SetViewportExtEx** function sets the horizontal and vertical extents of the viewport for a device context by using the specified values.

Syntax

C++

```
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 *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 *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 [Invalidate the Client Area](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[Coordinate Space and Transformation Functions](#)

[Coordinate Spaces and Transformations Overview](#)

[GetViewportExtEx](#)

[SIZE](#)

[SetWindowExtEx](#)

SetViewportOrgEx function (wingdi.h)

Article 11/19/2022

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

Syntax

C++

```
BOOL SetViewportOrgEx(
    [in]  HDC      hdc,
    [in]  int       x,
    [in]  int       y,
    [out] LPPOINT  lpp
);
```

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] `lpp`

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.

C++

```
//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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/19/2022

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

Syntax

C++

```
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 [InvalidateRect](#).

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article11/19/2022

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

Syntax

C++

```
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.

C++

```
//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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetWorldTransform function (wingdi.h)

Article02/22/2024

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

C++

```
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).

syntax

```
| 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.

syntax

```
| 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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](#)

Coordinate Space and Transformation Structures

Article • 01/07/2021

The following structure is used with coordinate spaces and transformations.

XFORM

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

XFORM structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagXFORM {  
    FLOAT eM11;  
    FLOAT eM12;  
    FLOAT eM21;  
    FLOAT eM22;  
    FLOAT eDx;  
    FLOAT eDy;  
} XFORM, *PXFORM, *LPXFORM;
```

Members

eM11

The following.

[+] Expand table

Operation	Meaning
Scaling	Horizontal scaling component
Rotation	Cosine of rotation angle
Reflection	Horizontal component

eM12

The following.

[+] Expand table

Operation	Meaning
Shear	Horizontal proportionality constant

Rotation

Sine of the rotation angle

eM21

The following.

 Expand table

Operation	Meaning
Shear	Vertical proportionality constant
Rotation	Negative sine of the rotation angle

eM22

The following.

 Expand table

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.

 Expand table

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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

[Yes](#)

[No](#)

Device Contexts

Article • 01/07/2021

A *device context* is a structure that defines a set of graphic objects and their associated attributes, as well as the graphic modes that affect output. The *graphic objects* include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations. The remainder of this section is divided into the following three areas.

- [About Device Contexts](#)
- [Using the Device Context Functions](#)
- [Device Context Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Device Contexts

Article • 01/07/2021

Device independence is one of the chief features of Microsoft Windows. Applications can draw and print output on a variety of devices. The software that supports this device independence is contained in two dynamic-link libraries. The first, Gdi.dll, is referred to as the graphics device interface (GDI); the second is referred to as a device driver. The name of the second depends on the device where the application draws output. For example, if the application draws output in the client area of its window on a VGA display, this library is Vga.dll; if the application prints output on an Epson FX-80 printer, this library is Epson9.dll.

An application must inform GDI to load a particular device driver and, once the driver is loaded, to prepare the device for drawing operations (such as selecting a line color and width, a brush pattern and color, a font typeface, a clipping region, and so on). These tasks are accomplished by creating and maintaining a device context (DC). A DC is a structure that defines a set of graphic objects and their associated attributes, and the graphic modes that affect output. The graphic objects include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations. Unlike most of the structures, an application never has direct access to the DC; instead, it operates on the structure indirectly by calling various functions.

This overview provides information on the following topics:

- [Graphic Objects](#)
- [Graphic Modes](#)
- [Device Context Types](#)
- [Device Context Operations](#)
- [ICM-Enabled Device Context Functions](#)

An important concept is the layout of a DC or a window, which describes the order in which GDI objects and text are revealed (either left-to-right or right-to-left). For more information, see "Window Layout and Mirroring" in [Window Features](#) and the [GetLayout](#) and [SetLayout](#) functions.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Graphic Objects

Article • 01/07/2021

The pen, brush, bitmap, palette, region, and path associated with a DC are referred to as its graphic objects. The following attributes are associated with each of these objects.

[+] Expand table

Graphic object	Associated attributes
Bitmap	Size, in bytes; dimensions, in pixels; color-format; compression scheme; and so on.
Brush	Style, color, pattern, and origin.
Palette	Colors and size (or number of colors).
Font	Typeface name, width, height, weight, character set, and so on.
Path	Shape.
Pen	Style, width, and color.
Region	Location and dimensions.

When an application creates a DC, the system automatically stores a set of default objects in it (there is no default bitmap or path). An application can examine the attributes of the default objects by calling the [GetCurrentObject](#) and [GetObject](#) functions. The application can change these defaults by creating a new object and selecting it into the DC. An object is selected into a DC by calling the [SelectObject](#) function.

An application can set the current brush color to a specified color value with [SetDCBrushColor](#).

The [GetDCBrushColor](#) function returns the DC brush color. The [SetDCPenColor](#) function sets the pen color to a specified color value. The [GetDCPenColor](#) function returns the DC pen color.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Graphic Modes

Article • 01/07/2021

Windows supports five graphic modes that allow an application to specify how colors are mixed, where output appears, how the output is scaled, and so on. These modes, which are stored in a DC, are described in the following table.

[+] Expand table

Graphics mode	Description
Background	Defines how background colors are mixed with existing window or screen colors for bitmap and text operations.
Drawing	Defines how foreground colors are mixed with existing window or screen colors for pen, brush, bitmap, and text operations.
Mapping	Defines how graphics output is mapped from logical (or world) space onto the window, screen, or printer paper.
Polygon-fill	Defines how the brush pattern is used to fill the interior of complex regions.
Stretching	Defines how bitmap colors are mixed with existing window or screen colors when the bitmap is compressed (or scaled down).

As it does with graphic objects, the system initializes a DC with default graphic modes. An application can retrieve and examine these default modes by calling the following functions.

[+] Expand table

Graphics mode	Function
Background	GetBkMode
Drawing	GetROP2
Mapping	GetMapMode
Polygon-fill	GetPolyFillMode
Stretching	GetStretchBltMode

An application can change the default modes by calling one of the following functions.

[+] Expand table

Graphics mode	Function
Background	SetBkMode
Drawing	SetROP2
Mapping	SetMapMode
Polygon-fill	SetPolyFillMode
Stretching	SetStretchBltMode

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device Context Types

Article • 01/07/2021

There are four types of DCs: display, printer, memory (or compatible), and information. Each type serves a specific purpose, as described in the following table.

[\[+\] Expand table](#)

Device context	Description
Display	Supports drawing operations on a video display.
Printer	Supports drawing operations on a printer or plotter.
Memory	Supports drawing operations on a bitmap.
Information	Supports the retrieval of device data.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Display Device Contexts

Article • 01/07/2021

An application obtains a display DC by calling the [BeginPaint](#), [GetDC](#), or [GetDCEx](#) function and identifying the window in which the corresponding output will appear. Typically, an application obtains a display DC only when it must draw in the client area. However, one may obtain a [window device context](#) by calling the [GetWindowDC](#) function. When the application is finished drawing, it must release the DC by calling the [EndPaint](#) or [ReleaseDC](#) function.

There are five types of DCs for video displays:

- Class
- Common
- Private
- Window
- Parent

Class Device Contexts

Class device contexts are supported strictly for compatibility with 16-bit versions of Windows. When writing your application, avoid using the class device context; use a private device context instead.

Common Device Contexts

Common device contexts are display DCs maintained in a special cache by the system. Common device contexts are used in applications that perform infrequent drawing operations. Before the system returns the DC handle, it initializes the common device context with default objects, attributes, and modes. Any drawing operations performed by the application use these defaults unless one of the GDI functions is called to select a new object, change the attributes of an existing object, or select a new mode.

Because only a limited number of common device contexts exist, an application should release them after it has finished drawing. When the application releases a common device context, any changes to the default data are lost.

Private Device Contexts

Private device contexts are display DCs that, unlike common device contexts, retain any changes to the default data even after an application releases them. Private device contexts are used in applications that perform numerous drawing operations such as computer-aided design (CAD) applications, desktop-publishing applications, drawing and painting applications, and so on. Private device contexts are not part of the system cache and therefore need not be released after use. The system automatically removes a private device context after the last window of that class has been destroyed.

An application creates a private device context by first specifying the CS_OWNDC window-class style when it initializes the **style** member of the **WNDCLASS** structure and calls the [RegisterClass](#) function. (For more information about window classes, see [Window Classes](#).)

After creating a window with the CS_OWNDC style, an application can call the [GetDC](#), [GetDCEx](#), or [BeginPaint](#) function once to obtain a handle identifying a private device context. The application can continue using this handle (and the associated DC) until it deletes the window created with this class. Any changes to graphic objects and their attributes, or graphic modes are retained by the system until the window is deleted.

Window Device Contexts

A *window device context* enables an application to draw anywhere in a window, including the nonclient area. Window device contexts are typically used by applications that process the [WM_NCPAINT](#) and [WM_NCACTIVATE](#) messages for windows with custom nonclient areas. Using a window device context is not recommended for any other purpose. For more information; see [GetWindowDC](#).

Parent Device Contexts

A *parent device context* enables an application to minimize the time necessary to set up the clipping region for a window. An application typically uses parent device contexts to speed up drawing for control windows without requiring a private or class device context. For example, the system uses parent device contexts for push button and edit controls. Parent device contexts are intended for use with child windows only, never with top-level or pop-up windows. For more information; see [Parent Display Device Contexts](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Printer Device Contexts (Windows GDI)

Article • 01/07/2021

The printer DC can be used when printing on a dot-matrix printer, ink-jet printer, laser printer, or plotter. An application creates a printer DC by calling the [CreateDC](#) function and supplying the appropriate arguments (the name of the printer driver, the name of the printer, the file or device name for the physical output medium, and other initialization data). When an application has finished printing, it deletes the printer DC by calling the [DeleteDC](#) function. An application must delete (rather than release) a printer DC; the [ReleaseDC](#) function fails when an application attempts to use it to release a printer DC.

For more information, see [Printer Output](#).

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Memory Device Contexts

Article • 01/07/2021

To enable applications to place output in memory rather than sending it to an actual device, use a special device context for bitmap operations called a *memory device context*. A memory DC enables the system to treat a portion of memory as a virtual device. It is an array of bits in memory that an application can use temporarily to store the color data for bitmaps created on a normal drawing surface. Because the bitmap is compatible with the device, a memory DC is also sometimes referred to as a *compatible device context*.

The memory DC stores bitmap images for a particular device. An application can create a memory DC by calling the [CreateCompatibleDC](#) function.

The original bitmap in a memory DC is simply a placeholder. Its dimensions are one pixel by one pixel. Before an application can begin drawing, it must select a bitmap with the appropriate width and height into the DC by calling the [SelectObject](#) function. To create a bitmap of the appropriate dimensions, use the [CreateBitmap](#), [CreateBitmapIndirect](#), or [CreateCompatibleBitmap](#) function. After the bitmap is selected into the memory DC, the system replaces the single-bit array with an array large enough to store color information for the specified rectangle of pixels.

When an application passes the handle returned by [CreateCompatibleDC](#) to one of the drawing functions, the requested output does not appear on a device's drawing surface. Instead, the system stores the color information for the resultant line, curve, text, or region in the array of bits. The application can copy the image stored in memory back onto a drawing surface by calling the [BitBlt](#) function, identifying the memory DC as the source device context and a window or screen DC as the target device context.

When displaying a DIB or a DDB created from a DIB on a palette device, you can improve the speed at which the image is drawn by arranging the logical palette to match the layout of the system palette. To do this, call [GetDeviceCaps](#) with the NUMRESERVED value to get the number of reserved colors in the system. Then call [GetSystemPaletteEntries](#) and fill in the first and last NUMRESERVED/2 entries of the logical palette with the corresponding system colors. For example, if NUMRESERVED is 20, you would fill in the first and last 10 entries of the logical palette with the system colors. Then fill in the remaining 256-NUMRESERVED colors of the logical palette (in our example, the remaining 236 colors) with colors from the DIB and set the PC_NOCOLLAPSE flag on each of these colors.

For more information about color and palettes, see [Colors](#). For more information about bitmaps and bitmap operations, see [Bitmaps](#).

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Information Device Contexts

Article • 01/07/2021

The information DC is used to retrieve default device data. For example, an application can call the [CreateIC](#) function to create an information DC for a particular model of printer and then call the [GetCurrentObject](#) and [GetObject](#) functions to retrieve the default pen or brush attributes. Because the system can retrieve device information without creating the structures normally associated with the other types of device contexts, an information DC involves far less overhead and is created significantly faster than any of the other types. After an application finishes retrieving data by using an information DC, it must call the [DeleteDC](#) function.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device Context Operations

Article • 01/07/2021

An application can perform the following operations on a device context:

- Enumerate existing graphic objects.
- Select new graphic objects.
- Delete existing graphic objects.
- Save the current graphic objects, their attributes, and the graphic modes.
- Restore previously saved graphic objects, their attributes, and the graphic modes.

In addition, an application can use a device context to:

- Determine how graphics output is translated.
- Cancel lengthy drawing operations (begun by a thread in a multithreaded application).
- Reset a printer to a particular state.

This subsection provides information on the following topics.

- [Operations on Graphic Objects](#)
- [Cancellation of Drawing Operations](#)
- [Retrieving Device Data](#)
- [Saving, Restoring, and Resetting a Device Context](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Operations on Graphic Objects

Article • 01/07/2021

After an application creates a display or printer DC, it can begin drawing on the associated device or, in the case of the memory DC, it can begin drawing on the bitmap stored in memory. However, before drawing begins, and sometimes while drawing is in progress, it is often necessary to replace the default objects with new objects.

An application can examine a default object's attributes by calling the [GetCurrentObject](#) and [GetObject](#) functions. The [GetCurrentObject](#) function returns a handle identifying the current pen, brush, palette, bitmap, or font, and the [GetObject](#) function initializes a structure containing that object's attributes.

Some printers provide resident pens, brushes, and fonts that can be used to improve drawing speed in an application. Two functions can be used to enumerate these objects: [EnumObjects](#) and [EnumFontFamilies](#). If the application must enumerate resident pens or brushes, it can call the [EnumObjects](#) function to examine the corresponding attributes. If the application must enumerate resident fonts, it can call the [EnumFontFamilies](#) function (which can also enumerate GDI fonts).

Once an application determines that a default object needs replacing, it creates a new object by calling one of the following creation functions.

[+] Expand table

Graphic object	Function
Bitmap	CreateBitmap , CreateBitmapIndirect , CreateCompatibleBitmap , CreateDiscardableBitmap , CreateDIBitmap
Brush	CreateBrushIndirect , CreateDIBPatternBrush , CreateDIBPatternBrushPt , CreateHatchBrush , CreatePatternBrush , CreateSolidBrush
Color Palette	CreatePalette
Font	CreateFont , CreateFontIndirect
Pen	CreatePen , CreatePenIndirect , ExtCreatePen
Region	CreateEllipticRgn , CreateEllipticRgnIndirect , CreatePolygonRgn , CreatePolyPolygonRgn , CreateRectRgn , CreateRectRgnIndirect , CreateRoundRectRgn

Each of these functions returns a handle identifying a new object. After an application retrieves a handle, it must call the [SelectObject](#) function to replace the default object. However, the application should save the handle identifying the default object and use this handle to replace the new object when it is no longer needed. When the application finishes drawing with the new object, it must restore the default object by calling the [SelectObject](#) function and then delete the new object by calling the [DeleteObject](#) function. Failing to delete objects causes serious performance problems.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Cancellation of Drawing Operations

Article • 01/07/2021

When complex drawing applications perform lengthy graphics operations, they consume valuable system resources. By taking advantage of the system's multitasking features, an application can use threads and the [CancelDC](#) function to manage these operations. For example, if the graphics operation performed by thread A is consuming needed resources, thread B can call the CancelDC function to halt that operation.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Retrieving Device Data

Article • 01/07/2021

Applications can use the following functions to retrieve device data using a device context: [GetDeviceCaps](#) and [DeviceCapabilities](#).

[GetDeviceCaps](#) retrieves general device data for the following devices:

- Raster displays
- Dot-matrix printers
- Ink-jet printers
- Laser printers
- Vector plotters
- Raster cameras

The data includes the supported capabilities of the device, including device resolution (for video displays), color format (for video displays and color printers), number of graphic objects, raster capabilities, curve drawing, line drawing, polygon drawing, and text drawing. An application retrieves this data by supplying a handle identifying the appropriate device context, as well as an index specifying the type of data the function is to retrieve.

The [DeviceCapabilities](#) function retrieves data specific to printers, including the number of available paper bins, the duplex capabilities of the printer, the resolutions supported by the printer, the maximum and minimum supported paper size, and so on. An application retrieves this data by supplying strings specifying a printer device and port, as well as an index specifying the type of data that the function is to retrieve.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Saving, Restoring, and Resetting a Device Context

Article • 01/07/2021

The following functions enable an application to save, restore, and reset a device context: [SaveDC](#), [RestoreDC](#), and [ResetDC](#). The SaveDC function records on a special GDI stack the current DC's graphic objects and their attributes, and graphic modes. A drawing application can call this function before a user begins drawing and save the application's original state providing a clean slate for the user. To return to this original state, the application calls the RestoreDC function.

[ResetDC](#) is provided to reset printer DC data. An application calls this function to reset the paper orientation, paper size, output scaling factor, number of copies to be printed, paper source (or bin), duplex mode, and so on. Typically, an application calls this function after a user has changed one of the printer options and the system has issued a [WM_DEVMODECHANGE](#) message.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ICM-Enabled Device Context Functions

Article • 01/07/2021

Microsoft Image Color Management (ICM) ensures that a color image, graphic, or text object is rendered as closely as possible to its original intent on any device, despite differences in imaging technologies and color capabilities between devices. (For more information, see [Windows Color System](#).)

There are various functions in the graphics device interface (GDI) that use or operate on color data. The following device context functions are enabled for use with ICM:

- [CreateCompatibleDC](#)
- [CreateDC](#)
- [GetDCBrushColor](#)
- [GetDCPenColor](#)
- [ResetDC](#)
- [SelectObject](#)
- [SetDCBrushColor](#)
- [SetDCPenColor](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Using the Device Context Functions

Article • 01/07/2021

- [Obtaining a Private Display Device Context](#)
- [Retrieving the Capabilities of a Printer](#)
- [Retrieving Graphic-Object Attributes and Selecting New Graphic Objects](#)
- [Setting and Retrieving the Device Context Brush Color Value](#)
- [Setting the Pen or Brush Color](#)
- [Getting Information on a Display Monitor](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Obtaining a Private Display Device Context

Article • 01/07/2021

An application performing numerous drawing operations in the client area of its window must use a private display DC. To create this type of DC, the application must specify the **CS_OWNDC** constant for the style member of the **WNDCLASS** structure when registering the window class. After registering the window class, the application obtains a handle identifying a private display DC by calling the **GetDC** function.

The following example shows how to create a private display DC.

C++

```

wc.hbrBackground = GetStockObject(WHITE_BRUSH);
wc.lpszMenuName = "GenericMenu";
wc.lpszClassName = "GenericWClass";

// Register the window class and return the resulting code.

return RegisterClass(&wc);
}

LRESULT APIENTRY MainWndProc(
    HWND hwnd,           // window handle
    UINT message,        // type of message
    WPARAM wParam,       // additional information
    LPARAM lParam)      // additional information
{
    PAINTSTRUCT ps;           // paint structure

    // Retrieve a handle identifying the private DC.

    hdc = GetDC(hwnd);

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);

            // Draw and paint using private DC.

            EndPaint(hwnd, &ps);

        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}

```

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Retrieving the Capabilities of a Printer

Article • 01/07/2021

Not every output device supports the entire set of graphics functions. For example, because of hardware limitations, most vector plotters do not support bit-block transfers. An application can determine whether a device supports a particular graphics function by calling the [GetDeviceCaps](#) function, specifying the appropriate index, and examining the return value.

The following example shows how an application tests a printer to determine whether it supports bit-block transfers.

C++

```
// Examine the raster capabilities of the device
// identified by hdcPrint to verify that it supports
// the BitBlt function.

if ((GetDeviceCaps(hdcPrint, RASTERCAPS)
    & RC_BITBLT) == 0)
{
    DeleteDC(hdcPrint);
    break;
}

else
{
    // Print the bitmap using the printer DC.
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Retrieve object attributes, select new objects

Article • 01/07/2021

An application can retrieve the attributes for a pen, brush, palette, font, or bitmap by calling the [GetCurrentObject](#) and [GetObject](#) functions. The [GetCurrentObject](#) function returns a handle identifying the object currently selected into the DC; the [GetObject](#) function returns a structure that describes the attributes of the object.

The following example shows how an application can retrieve the current brush attributes and use the retrieved data to determine whether it is necessary to select a new brush.

C++

```
HDC hdc;           // display DC handle
HBRUSH hbrushNew, hbrushOld; // brush handles
HBRUSH hbrush;       // brush handle
LOGBRUSH lb;         // logical-brush structure

// Retrieve a handle identifying the current brush.

hbrush = GetCurrentObject(hdc, OBJ_BRUSH);

// Retrieve a LOGBRUSH structure that contains the
// current brush attributes.

GetObject(hbrush, sizeof(LOGBRUSH), &lb);

// If the current brush is not a solid-black brush,
// replace it with the solid-black stock brush.

if ((lb.lbStyle != BS_SOLID)
    || (lb.lbColor != 0x000000))
{
    hbrushNew = GetStockObject(BLACK_BRUSH);
    hbrushOld = SelectObject(hdc, hbrushNew);
}

// Perform painting operations with the solid-black brush.

// After completing the last painting operation with the new
// brush, the application should select the original brush back
// into the device context and delete the new brush.
// In this example, hbrushNew contains a handle to a stock object.
// It is not necessary (but it is not harmful) to call
// DeleteObject on a stock object. If hbrushNew contained a handle
```

```
// to a brush created by a function such as CreateBrushIndirect,  
// it would be necessary to call DeleteObject.  
  
SelectObject(hdc, hbrushOld);  
DeleteObject(hbrushNew);
```

ⓘ Note

The application saved the original brush handle when calling the [SelectObject](#) function the first time. This handle is saved so that the original brush can be selected back into the DC after the last painting operation has been completed with the new brush. After the original brush is selected back into the DC, the new brush is deleted, freeing memory in the GDI heap.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Setting and Retrieving the Device Context Brush Color Value

Article • 06/11/2021

The following example shows how an application can retrieve the current DC brush color by using the [SetDCBrushColor](#) and the [GetDCBrushColor](#) functions.

C++

```
SelectObject(hdc,GetStockObject(DC_BRUSH));
SetDCBrushColor(hdc,RGB(00,0xff,00));
PatBlt(hdc,0,0,200,200,PATCOPY);
SetDCBrushColor(hdc,RGB(00,00,0xff));
PatBlt(hdc,0,0,200,200,PATCOPY);
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Setting the Pen or Brush Color

Article • 01/07/2021

The following example shows how an application can change the DC pen color by using the [GetStockObject](#) function or [SetDCPenColor](#) and the [SetDCBrushColor](#) functions.

C++

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:

            {
                hdc = BeginPaint(hWnd, &ps);
                // Initializing original object
                HGDIOBJ original = NULL;

                // Saving the original object
                original = SelectObject(hdc,GetStockObject(DC_PEN));

                // Rectangle function is defined as...
                // BOOL Rectangle(hdc, xLeft, yTop, yRight, yBottom);

                // Drawing a rectangle with just a black pen
                // The black pen object is selected and sent to the current
                device context
                // The default brush is WHITE_BRUSH
                SelectObject(hdc, GetStockObject(BLACK_PEN));
            }
    }
}
```

```

        Rectangle(hdc,0,0,200,200);

    //      Select DC_PEN so you can change the color of the pen with
    //      COLORREF SetDCPenColor(HDC hdc, COLORREF color)
    SelectObject(hdc, GetStockObject(DC_PEN));

    //      Select DC_BRUSH so you can change the brush color from the
    //      default WHITE_BRUSH to any other color
    SelectObject(hdc, GetStockObject(DC_BRUSH));

    //      Set the DC Brush to Red
    //      The RGB macro is declared in "Windowsx.h"
    SetDCBrushColor(hdc, RGB(255,0,0));

    //      Set the Pen to Blue
    SetDCPenColor(hdc, RGB(0,0,255));

    //      Drawing a rectangle with the current Device Context
    Rectangle(hdc,100,300,200,400);

    //      Changing the color of the brush to Green
    SetDCBrushColor(hdc, RGB(0,255,0));
    Rectangle(hdc,300,150,500,300);

    //      Restoring the original object
    SelectObject(hdc,original);

    // It is not necessary to call DeleteObject to delete stock objects.
}

break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

Feedback

Was this page helpful?

 Yes

 No

Getting Information on a Display Monitor

Article • 01/07/2021

The following code sample shows how to use [EnumDisplayDevices](#) to get information on a display monitor.

C++

```
BOOL GetDisplayMonitorInfo(int nDeviceIndex, LPSTR lpszMonitorInfo)
{
    FARPROC EnumDisplayDevices;
    HINSTANCE hInstUser32;
    DISPLAY_DEVICE DispDev;
    char szSaveDeviceName[33]; // 32 + 1 for the null-terminator
    BOOL bRet = TRUE;
    HRESULT hr;

    hInstUser32 = LoadLibrary("c:\\windows\\User32.DLL");
    if (!hInstUser32) return FALSE;

    // Get the address of the EnumDisplayDevices function
    EnumDisplayDevices =
        (FARPROC)GetProcAddress(hInstUser32, "EnumDisplayDevicesA");
    if (!EnumDisplayDevices) {
        FreeLibrary(hInstUser32);
        return FALSE;
    }

    ZeroMemory(&DispDev, sizeof(DispDev));
    DispDev.cb = sizeof(DispDev);

    // After the first call to EnumDisplayDevices,
    // DispDev.DeviceString is the adapter name
    if (EnumDisplayDevices(NULL, nDeviceIndex, &DispDev, 0))
    {
        hr = StringCchCopy(szSaveDeviceName, 33,
DispDev.DeviceName);
        if (FAILED(hr))
        {
            // TODO: write error handler
        }

        // After second call, DispDev.DeviceString is the
        // monitor name for that device
        EnumDisplayDevices(szSaveDeviceName, 0, &DispDev, 0);

        // In the following, lpszMonitorInfo must be 128 + 1 for
        // the null-terminator.
        hr = StringCchCopy(lpszMonitorInfo, 129,
```

```
DispDev.DeviceString);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }

} else {
    bRet = FALSE;
}

FreeLibrary(hInstUser32);

return bRet;
}
```

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device Context Reference

Article • 01/07/2021

The following elements are associated with device contexts:

- [Device Context Functions](#)
- [Device Context Structures](#)
- [Device Context Messages](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device Context Functions

Article • 01/07/2021

The following functions are used with device contexts.

[+] Expand table

Function	Description
CancelDC	Cancels any pending operation on the specified device context.
ChangeDisplaySettings	Changes the settings of the default display device to the specified graphics mode.
ChangeDisplaySettingsEx	Changes the settings of the specified display device to the specified graphics mode.
CreateCompatibleDC	Creates a memory device context compatible with the specified device.
CreateDC	Creates a device context for a device using the specified name.
CreateIC	Creates an information context for the specified device.
DeleteDC	Deletes the specified device context.
DeleteObject	Deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object.
DeviceCapabilities	Retrieves the capabilities of a printer device driver.
DrawEscape	Provides drawing capabilities of the specified video display that are not directly available through the graphics device interface.
EnumDisplayDevices	Retrieves information about the display devices in a system.
EnumDisplaySettings	Retrieves information about one of the graphics modes for a display device.
EnumDisplaySettingsEx	Retrieves information about one of the graphics modes for a display device.
EnumObjects	Enumerates the pens or brushes available for the specified device context.
EnumObjectsProc	An application-defined callback function used with the EnumObjects function.
GetCurrentObject	Retrieves a handle to an object of the specified type that has been selected into the specified device context.

Function	Description
GetDC	Retrieves a handle to a display device context for the client area of a specified window or for the entire screen.
GetDCBrushColor	Retrieves the current brush color for the specified device context.
GetDCEx	Retrieves a handle to a display device context for the client area of a specified window or for the entire screen.
GetDCOrgEx	Retrieves the final translation origin for a specified device context.
GetDCPenColor	Retrieves the current pen color for the specified device context.
GetDeviceCaps	Retrieves device-specific information for the specified device.
GetLayout	Retrieves the layout of a device context.
GetObject	Retrieves information for the specified graphics object.
GetObjectType	Retrieves the type of the specified object.
GetStockObject	Retrieves a handle to one of the stock pens, brushes, fonts, or palettes.
ReleaseDC	Releases a device context, freeing it for use by other applications.
ResetDC	Updates the specified printer or plotter device context using the specified information.
RestoreDC	Restores a device context to the specified state.
SaveDC	Saves the current state of the specified device context by copying data describing selected objects and graphic modes to a context stack.
SelectObject	Selects an object into the specified device context.
SetDCBrushColor	Sets the current device context brush color to the specified color value.
SetDCPenColor	Sets the current device context pen color to the specified color value.
SetLayout	Sets the layout for a device context.
WindowFromDC	Returns a handle to the window associated with a device context.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

CancelDC function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ChangeDisplaySettingsA function (winuser.h)

Article 02/09/2023

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

C++

```
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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

 Expand table

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.

[Expand table](#)

Parameters	Meaning
wParam	New bits per pixel
LOWORD(lParam)	New pixel width
HIWORD(lParam)	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 that 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](#).

[+] Expand table

Requirement	Value
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)

Article 02/09/2023

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

C++

```
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.

[+] [Expand table](#)

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.

[+] [Expand table](#)

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.

[Expand table](#)

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.

 Expand table

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.

[+] Expand table

Parameters	Meaning
wParam	New bits per pixel
LOWORD(lParam)	New pixel width
HIWORD(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:

C++

```
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 that 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](#).

[+] Expand table

Requirement	Value
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](#)

CreateCompatibleDC function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/09/2023

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

Syntax

C++

```
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 that 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](#).

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

CreateICA function (wingdi.h)

Article 02/22/2024

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

C++

```
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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

DeleteDC function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

DeleteObject function (wingdi.h)

Article02/22/2024

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

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

DrawEscape function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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 `lpszInData` 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 *lpszInData* is passed directly to the specified display driver.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

EnumDisplayDevicesA function (winuser.h)

Article 02/09/2023

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

Syntax

C++

```
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 that 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

[Expand table](#)

Requirement	Value
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](#)

EnumDisplaySettingsA function (winuser.h)

02/09/2023

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

C++

```
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 that 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

[+] [Expand table](#)

Requirement	Value
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)

Article 02/09/2023

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

C++

```
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.

 Expand table

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.

 Expand table

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 that 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

 [Expand table](#)

Requirement	Value
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](#)

EnumObjects function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCurrentObject function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>OBJ_BITMAP</code>	Returns the current selected bitmap.
<code>OBJ_BRUSH</code>	Returns the current selected brush.
<code>OBJ_COLORSPACE</code>	Returns the current color space.
<code>OBJ_FONT</code>	Returns the current selected font.
<code>OBJ_PAL</code>	Returns the current selected palette.
<code>OBJ_PEN</code>	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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetDC function (winuser.h)

Article 10/13/2021

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 [GetDCEEx](#) function is an extension to **GetDC**, which gives an application more control over how and whether clipping occurs in the client area.

Syntax

C++

```
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

 Expand table

Requirement	Value
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

GetDCBrushColor function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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](#)

GetDCEx function (winuser.h)

Article 10/13/2021

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

C++

```
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.

 Expand table

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 OWNDCC 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 [GetDCEx](#). 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

 [Expand table](#)

Requirement	Value
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](#)

GetDCOrgEx function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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)

Article03/11/2023

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

Syntax

C++

```
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.

 Expand table

Index	Meaning
DRIVERVERSION	The device driver version.
TECHNOLOGY	Device technology. It can be any one of the following values.

 Expand 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
		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.
HORZSIZE		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	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.
---------------------	---

SHADEBLENDCAPS	Value that indicates the shading and blending capabilities of the device. See Remarks for further comments.
-----------------------	---

[Expand table](#)

SB_CONST_ALPHA	Handles the SourceConstantAlpha member of the BLENDFUNCTION structure, which is referenced by the blendFunction parameter of the AlphaBlend function.
SB_GRAD_RECT	Capable of doing GradientFill rectangles.
SB_GRAD_TRI	Capable of doing GradientFill triangles.
SB_NONE	Device does not support any of these capabilities.
SB_PIXEL_ALPHA	Capable of handling per-pixel alpha in AlphaBlend .
SB_PREMULT_ALPHA	Capable of handling premultiplied alpha in AlphaBlend .

RASTERCAPS	Value that indicates the raster capabilities of the device, as shown in the following table.
-------------------	--

[Expand 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 Value that indicates the curve capabilities of the device, as shown in the following table.

[+] [Expand 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.

LINECAPS Value that indicates the line capabilities of the device, as shown in the following table:

[+] [Expand 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.

POLYGONALCAPS

Value that indicates the polygon capabilities of the device, as shown in the following table.

[\[+\] Expand 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.

[\[+\] Expand 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.

[Expand table](#)

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.

[Expand table](#)

Index	Printer escape replaced
PHYSICALWIDTH	GETPHYSpagesize
PHYSICALHEIGHT	GETPHYSpagesize

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetLayout function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

GetObject function (wingdi.h)

The **GetObject** function retrieves information for the specified graphics object.

Syntax

C++

```
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 *hgdiobj*.

 Expand table

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
	EXTLOGOPEN

HPEN returned from a call to

[ExtCreatePen](#)

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Last updated on 02/27/2023

GetObjectType function (wingdi.h)

Article 10/13/2021

The `GetObjectType` retrieves the type of the specified object.

Syntax

C++

```
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.

 [Expand table](#)

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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetStockObject function (wingdi.h)

Article10/12/2022

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

Syntax

C++

```
HGDIOBJ GetStockObject(  
    [in] int i  
);
```

Parameters

[in] i

The type of stock object. This parameter can be one of the following values.

[] [Expand table](#)

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 black. 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ReleaseDC function (winuser.h)

Article10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

ResetDCA function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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 **IpInitData** member of the **DEVMODE** structure.

! Note

The wingdi.h header defines **ResetDC** as an alias that 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](#).

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

RestoreDC function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SaveDC function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SelectObject function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

[] Expand table

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.

[+] [Expand table](#)

Value	Meaning
SIMPLEREGION	Region consists of a single rectangle.
COMPLEXREGION	Region consists of more than one rectangle.
NULLRREGION	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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetDCBrushColor function (wingdi.h)

Article10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetLayout function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>LAYOUT_BITMAPORIENTATIONPRESERVED</code>	Disables any reflection during BitBlt and StretchBlt operations.
<code>LAYOUTRTL</code>	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.

 Expand 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

WindowFromDC function (winuser.h)

02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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

Requirement	Value
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](#)

Device Context Structures

Article • 01/07/2021

The following structures are used with device contexts.

[DISPLAY_DEVICE](#)

[VIDEOPARAMETERS](#) ↗

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) ↗ | [Get help at Microsoft Q&A](#)

DISPLAY_DEVICEA structure (wingdi.h)

Article 11/20/2024

The **DISPLAY_DEVICE** structure receives information about the display device specified by the *iDevNum* parameter of the [EnumDisplayDevices](#) function.

Syntax

C++

```
typedef struct _DISPLAY_DEVICEA {
    DWORD cb;
    CHAR DeviceName[32];
    CHAR DeviceString[128];
    DWORD StateFlags;
    CHAR DeviceID[128];
    CHAR DeviceKey[128];
} DISPLAY_DEVICEA, *PDISPLAY_DEVICEA, *LPDISPLAY_DEVICEA;
```

Members

cb

Size, in bytes, of the **DISPLAY_DEVICE** structure. This must be initialized prior to calling [EnumDisplayDevices](#).

DeviceName[32]

An array of characters identifying the device name. This is either the adapter device or the monitor device.

DeviceString[128]

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.

[+] Expand table

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 (SM_MONITORS) 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[128]`

Not used.

`DeviceKey[128]`

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.

① Note

The wingdi.h header defines `DISPLAY_DEVICE` as an alias that 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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device Context Messages

Article • 01/07/2021

The following message is used with device contexts.

WM_DEVMODECHANGE

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

WM_DEVMODECHANGE message

Article • 01/07/2021

The **WM_DEVMODECHANGE** message is sent to all top-level windows whenever the user changes device-mode settings.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

hwnd

A handle to a window.

uMsg

WM_DEVMODECHANGE

wParam

This parameter is not used.

lParam

A pointer to a string that specifies the device name.

Return value

An application should return zero if it processes this message.

Remarks

This message cannot be sent directly to a window. To send the **WM_DEVMODECHANGE** message to all top-level windows, use the [SendMessageTimeout](#) function with the

hWnd parameter set to `HWND_BROADCAST`.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	<code>Winuser.h</code> (include <code>Windows.h</code>)

See also

[Device Contexts Overview](#)

[Device Context Messages](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Filled Shapes

Article • 01/07/2021

Filled shapes are geometric forms that are outlined by using the current pen and filled by using the current brush. There are five filled shapes:

- Ellipse
- Chord
- Pie
- Polygon
- Rectangle

This overview describes the filled shapes.

- [About Filled Shapes](#)
- [Using Filled Shapes](#)
- [Filled Shape Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Filled Shapes

Article • 01/07/2021

Applications use filled shapes for a variety of tasks. Spreadsheet applications, for example, use filled shapes to construct charts and graphs, and drawing and painting applications use filled shapes to allow the user to draw figures and illustrations.

- [Ellipse](#)
- [Chord](#)
- [Pie](#)
- [Polygon](#)
- [Rectangle](#)

Feedback

Was this page helpful?



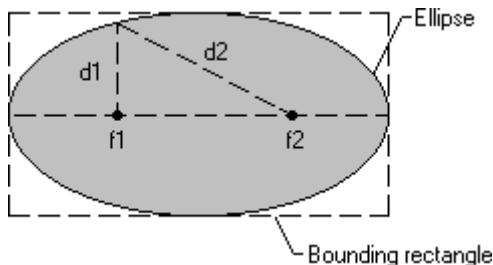
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Ellipses

Article • 01/07/2021

An ellipse is a closed curve defined by two fixed points (f_1 and f_2) such that the sum of the distances ($d_1 + d_2$) from any point on the curve to the two fixed points is constant.

The following illustration shows an ellipse drawn by using the [Ellipse](#) function.



When calling [Ellipse](#), an application supplies the coordinates of the upper-left and lower-right corners of the ellipse's bounding rectangle. A *bounding rectangle* is the smallest rectangle completely surrounding the ellipse. When the system draws the ellipse, it excludes the right and lower sides if no world transformations are set. Therefore, for any rectangle measuring x units wide by y units high, the associated ellipse measures x_1 units wide by y_1 units high. If the application sets a world transformation by calling the [SetWorldTransform](#) or [ModifyWorldTransform](#) function, the system includes the right and lower sides.

Feedback

Was this page helpful?

Yes

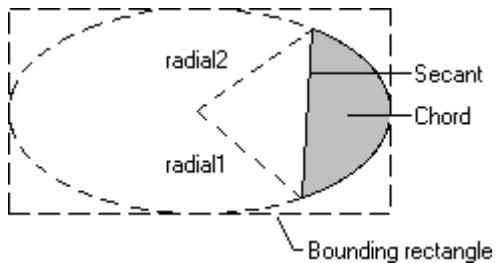
No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Chords

Article • 01/07/2021

A *chord* is a region bounded by the intersection of an ellipse and a line segment called a *secant*. The following illustration shows a chord drawn by using the **Chord** function.



When calling **Chord**, an application supplies the coordinates of the upper-left and lower-right corners of the ellipse's bounding rectangle, as well as the coordinates of two points defining two radials. A radial is a line drawn from the center of an ellipse's bounding rectangle to a point on the ellipse.

When the system draws the curved part of the chord, it does so by using the current arc direction for the specified device context. The default arc direction is counterclockwise. You can have your application reset the arc direction by calling the **SetArcDirection** function.

Feedback

Was this page helpful?

Yes

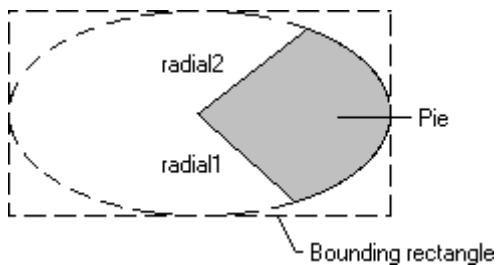
No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Pies

Article • 01/07/2021

A *pie* is a region bounded by the intersection of an ellipse curve and two radials. The following illustration shows a pie drawn by using the [Pie](#) function.



When calling [Pie](#), an application supplies the coordinates of the upper-left and lower-right corners of the ellipse's bounding rectangle, as well as the coordinates of two points defining two radials.

When the system draws the curved part of the pie, it uses the current arc direction for the given device context. The default arc direction is counterclockwise. An application can reset the arc direction by calling the [SetArcDirection](#) function.

Feedback

Was this page helpful?

Yes	No
-----	----

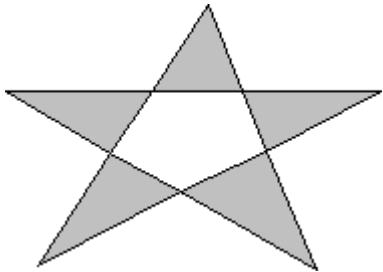
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Polygons

Article • 01/07/2021

A *polygon* is a filled shape with straight sides. The sides of a polygon are drawn by using the current pen. When the system fills a polygon, it uses the current brush and the current polygon fill mode. The two fill modes, alternate (the default) and winding, determine whether regions within a complex polygon are filled or left unpainted. An application can select either mode by calling the [SetPolyFillMode](#) function. For more information about polygon fill modes, see [Regions](#).

The following illustration shows a polygon drawn by using [Polygon](#).



In addition to drawing a single polygon by using [Polygon](#), an application can draw multiple polygons by using the [PolyPolygon](#) function.

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing Rectangles

Article • 01/07/2021

A rectangle is a four-sided polygon whose opposing sides are parallel and equal in length. Although an application can draw a rectangle by calling the [Polygon](#) function, supplying the coordinates of each corner, the [Rectangle](#) function provides a simpler method. This function requires only the coordinates for the upper-left and the lower-right corners. When an application calls the [Rectangle](#) function, the system draws the rectangle, excluding the right and lower sides if no world transformation is set for the given device context.

If a world transformation has been set by using the [SetWorldTransform](#) or [ModifyWorldTransform](#) function, the system includes the right and lower edges.

In addition to drawing a traditional rectangle, you can draw rectangles with rounded corners. The [RoundRect](#) function requires that the application supply the coordinates of the lower-left and upper-right corners, as well as the width and height of the ellipse used to round each corner.

Applications can use the following functions to manipulate rectangles.

[+] [Expand table](#)

Function	Description
FillRect	Repaints the interior of a rectangle.
FrameRect	Redraws the sides of a rectangle.
InvertRect	Inverts the colors that appear within the interior of a rectangle.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

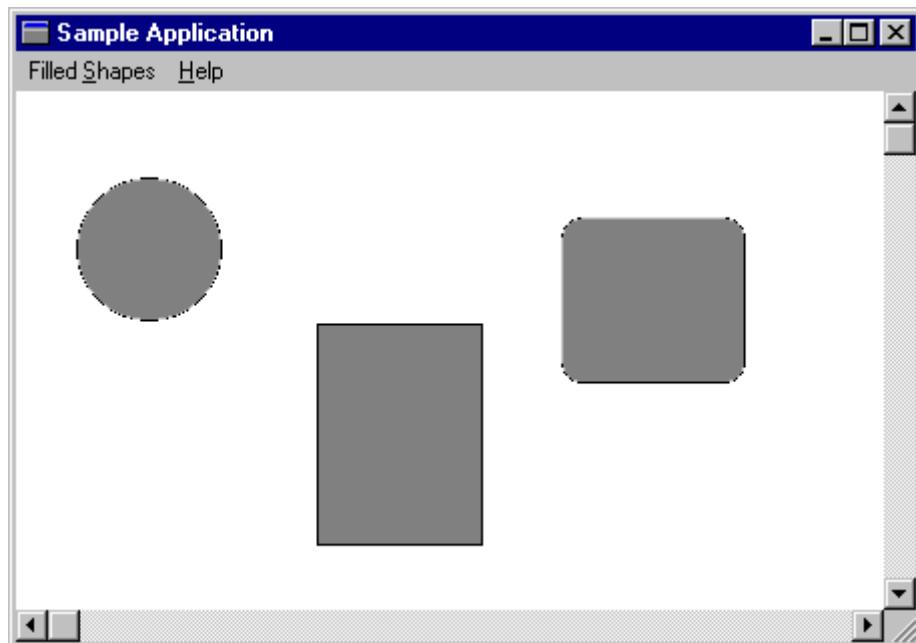
Using Filled Shapes

Article • 01/07/2021

This section illustrates how to use filled shape functions. The example uses the main window procedure from an application that enables the user to draw ellipses, rectangles, and rectangles with rounded corners.

The user draws a filled shape by selecting a particular shape from the menu, positioning the cursor at the upper-left corner of the shape (or the shape's bounding rectangle in the case of an ellipse), and then dragging the mouse until the desired dimensions are obtained.

The following illustration shows three filled shapes drawn using the sample code in this section.



To enable the user to draw filled shapes, include the following popup menu in your application.

C++

```
POPUP "Filled &Shapes"
{
    MENUITEM "&Ellipse", IDM_ELLIPSE
    MENUITEM "&Rectangle", IDM_RECTANGLE
    MENUITEM "R&oundRect", IDM_ROUNDRRECT
}
```

The menu item values in the menu template are constants that you must define as follows in your application's header file.

C++

```
#define IDM_ELLIPSE    1100
#define IDM_RECTANGLE   1200
#define IDM_ROUNDRECT   1300
```

Finally, include the following window procedure in your application.

C++

```
BOOL CALLBACK MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam,
                           LPARAM lParam)
{
    HDC hdc;           // handle to device context (DC)
    PAINTSTRUCT ps;   // paint data for Begin/EndPaint
    POINT ptClientUL; // client area upper left corner
    POINT ptClientLR; // client area lower right corner
    static HDC hdcCompat; // handle to DC for bitmap
    static POINT pt;     // x- and y-coordinates of cursor
    static RECT rcTarget; // rect to receive filled shape
    static RECT rcClient; // client area rectangle
    static BOOL fSizeEllipse; // TRUE if ellipse is sized
    static BOOL fDrawEllipse; // TRUE if ellipse is drawn
    static BOOL fDrawRectangle; // TRUE if rectangle is drawn
    static BOOL fSizeRectangle; // TRUE if rectangle is sized
    static BOOL fSizeRoundRect; // TRUE if rounded rect is sized
    static BOOL fDrawRoundRect; // TRUE if rounded rect is drawn
    static int nEllipseWidth; // width for round corners
    static int nEllipseHeight; // height for round corners

    switch (uMsg)
    {
        case WM_COMMAND:
            switch (wParam)
            {

                // Set the appropriate flag to indicate which
                // filled shape the user is drawing.

                case IDM_ELLIPSE:
                    fSizeEllipse = TRUE;
                    break;

                case IDM_RECTANGLE:
                    fSizeRectangle = TRUE;
                    break;

                case IDM_ROUNDRECT:
                    fSizeRoundRect = TRUE;
                    break;

                default:
                    return DefWindowProc(hwnd, uMsg, wParam,
```

```
    lParam);  
}  
break;  
  
  
case WM_CREATE:  
    nEllipseWidth = 20;  
    nEllipseHeight = 20;  
  
    return 0;  
  
case WM_PAINT:  
  
    BeginPaint(hwnd, &ps);  
  
    // Because the default brush is white, select  
    // a different brush into the device context  
    // to demonstrate the painting of filled shapes.  
  
    SelectObject(ps.hdc, GetStockObject(GRAY_BRUSH));  
  
    // If one of the filled shape "draw" flags is TRUE,  
    // draw the corresponding shape.  
  
    if (fDrawEllipse)  
    {  
        Ellipse(ps.hdc, rcTarget.left, rcTarget.top,  
                rcTarget.right, rcTarget.bottom);  
        fDrawEllipse = FALSE;  
        rcTarget.left = rcTarget.right = 0;  
        rcTarget.top = rcTarget.bottom = 0;  
    }  
  
    if (fDrawRectangle)  
    {  
        Rectangle(ps.hdc, rcTarget.left, rcTarget.top,  
                  rcTarget.right, rcTarget.bottom);  
        fDrawRectangle = FALSE;  
        rcTarget.left = rcTarget.right = 0;  
        rcTarget.top = rcTarget.bottom = 0;  
    }  
  
    if (fDrawRoundRect)  
    {  
        RoundRect(ps.hdc, rcTarget.left, rcTarget.top,  
                  rcTarget.right, rcTarget.bottom,  
                  nEllipseWidth, nEllipseHeight);  
        fDrawRoundRect = FALSE;  
        rcTarget.left = rcTarget.right = 0;  
        rcTarget.top = rcTarget.bottom = 0;  
    }  
  
    EndPaint(hwnd, &ps);  
    break;
```

```

case WM_SIZE:

    // Convert the client coordinates of the client area
    // rectangle to screen coordinates and save them in a
    // rectangle. The rectangle is passed to the ClipCursor
    // function during WM_LBUTTONDOWN processing.

    GetClientRect(hwnd, &rcClient);
    ptClientUL.x = rcClient.left;
    ptClientUL.y = rcClient.top;
    ptClientLR.x = rcClient.right;
    ptClientLR.y = rcClient.bottom;
    ClientToScreen(hwnd, &ptClientUL);
    ClientToScreen(hwnd, &ptClientLR);
    SetRect(&rcClient, ptClientUL.x, ptClientUL.y,
            ptClientLR.x, ptClientLR.y);
    return 0;

case WM_LBUTTONDOWN:

    // Restrict the cursor to the client area.
    // This ensures that the window receives a matching
    // WM_BUTTONUP message.

    ClipCursor(&rcClient);

    // Save the coordinates of the cursor.

    pt.x = (LONG) LOWORD(lParam);
    pt.y = (LONG) HIWORD(lParam);

    // If the user chooses one of the filled shapes,
    // set the appropriate flag to indicate that the
    // shape is being sized.

    if (fDrawEllipse)
        fSizeEllipse = TRUE;

    return 0;

case WM_MOUSEMOVE:

    // If one of the "size" flags is set, draw
    // the target rectangle as the user drags
    // the mouse.

    if ((wParam && MK_LBUTTON)
        && (fSizeEllipse || fSizeRectangle
        || fSizeRoundRect))
    {

        // Set the mixing mode so that the pen color is the
        // inverse of the background color. The previous
        // rectangle can then be erased by drawing

```

```

// another rectangle on top of it.

hdc = GetDC(hwnd);
SetROP2(hdc, R2_NOTXORPEN);

// If a previous target rectangle exists, erase
// it by drawing another rectangle on top.

if (!IsRectEmpty(&rcTarget))
{
    Rectangle(hdc, rcTarget.left, rcTarget.top,
               rcTarget.right, rcTarget.bottom);
}

// Save the coordinates of the target rectangle.
// Avoid invalid rectangles by ensuring that the
// value of the left coordinate is greater than
// that of the right, and that the value of the
// bottom coordinate is greater than that of
// the top.

if ((pt.x < (LONG) LOWORD(lParam)) &&
    (pt.y > (LONG) HIWORD(lParam)))
{
    SetRect(&rcTarget, pt.x, HIWORD(lParam),
            LOWORD(lParam), pt.y);
}

else if ((pt.x > (LONG) LOWORD(lParam)) &&
          (pt.y > (LONG) HIWORD(lParam)))
{
    SetRect(&rcTarget, LOWORD(lParam),
            HIWORD(lParam), pt.x, pt.y);
}

else if ((pt.x > (LONG) LOWORD(lParam)) &&
          (pt.y < (LONG) HIWORD(lParam)))
{
    SetRect(&rcTarget, LOWORD(lParam), pt.y,
            pt.x, HIWORD(lParam));
}
else
{
    SetRect(&rcTarget, pt.x, pt.y, LOWORD(lParam),
            HIWORD(lParam));
}

// Draw the new target rectangle.

Rectangle(hdc, rcTarget.left, rcTarget.top,
           rcTarget.right, rcTarget.bottom);
ReleaseDC(hwnd, hdc);
}

return 0;

```

```
case WM_LBUTTONDOWN:
    // If one of the "size" flags is TRUE, reset it to FALSE,
    // and then set the corresponding "draw" flag. Invalidate
    // the appropriate rectangle and issue a WM_PAINT message.

    if (fSizeEllipse)
    {
        fSizeEllipse = FALSE;
        fDrawEllipse = TRUE;
    }

    if (fSizeRectangle)
    {
        fSizeRectangle = FALSE;
        fDrawRectangle = TRUE;
    }

    if (fSizeRoundRect)
    {
        fSizeRoundRect = FALSE;
        fDrawRoundRect = TRUE;
    }

    if (fDrawEllipse || fDrawRectangle || fDrawRoundRect)
    {
        InvalidateRect(hwnd, &rcTarget, TRUE);
        UpdateWindow(hwnd);
    }

    // Release the cursor.

    ClipCursor((LPRECT) NULL);
    return 0;

case WM_DESTROY:
    // Destroy the background brush, compatible bitmap,
    // and bitmap.

    DeleteDC(hdcCompat);
    PostQuitMessage(0);
    break;

default:
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
return (LRESULT) NULL;
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Filled Shape Reference

Article • 01/07/2021

The following elements are used with filled shapes:

- Filled Shape Functions

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Filled Shape Functions

Article • 01/07/2021

The following functions are used with filled shapes.

 Expand table

Function	Description
Chord	Draws an area bounded by an ellipse and a line segment.
Ellipse	Draws an ellipse.
FillRect	Fills a rectangle using a brush.
FrameRect	Draws a border around a rectangle using a brush.
InvertRect	Inverts the color values of the pixels in a rectangle.
Pie	Draws a pie-shaped wedge bounded by an ellipse and two radials.
Polygon	Draws a polygon.
PolyPolygon	draws a series of closed polygons.
Rectangle	Draws a rectangle.
RoundRect	Draws a rectangle with rounded corners.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Chord function (wingdi.h)

Article 02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Ellipse function (wingdi.h)

Article 02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

FillRect function (winuser.h)

02/22/2024

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

C++

```
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:

C++

```
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

 Expand table

Requirement	Value
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)

02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

InvertRect function (winuser.h)

02/22/2024

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

C++

```
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

Requirement	Value
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](#)

Pie function (wingdi.h)

Article 02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Polygon function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PolyPolygon function (wingdi.h)

Article11/19/2022

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Rectangle function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

RoundRect function (wingdi.h)

Article 10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Fonts and Text (Windows GDI)

Article • 01/07/2021

Fonts are used to draw text on video displays and other output devices. The font and text functions enable you to install, select, and query different fonts.

- [About Fonts](#)
- [About Text Output](#)
- [Using the Font and Text-Output Functions](#)
- [Font and Text Reference](#)
- [Font Embedding Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Fonts

Article • 01/07/2021

To a person trained in the mechanics of manuscript composition or familiar with standard typography, some of the typographic terms used in this overview may be unusual. Most of the differences between standard typography and text on Microsoft Windows reflect changes in technology. The original typographic terms were based on hot-metal composition, whereas the terms used here, which appear as member names for the font and text output structures, reflect a new technology based on laser-printer output and composition performed on a personal computer using desktop publishing software.

The following sections discuss fonts:

- [Font Elements](#)
- [Font Families](#)
- [Raster, Vector, TrueType, and OpenType Fonts](#)
- [Character Sets Used by Fonts](#)
- [Font Installation and Deletion](#)
- [Fonts from Multiple Resource Files](#)
- [Font Creation and Selection](#)
- [Embedded Fonts](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Font Elements

Article • 01/07/2021

A *font* is a collection of characters and symbols that share a common design. The three major elements of this design are referred to as typeface, style, and size.

Typeface

The term typeface refers to specific characteristics of characters and symbols in the font, such as the width of the thick and thin strokes that compose the characters and the presence or absence of serifs. A serif is the short cross line at the ends of an unconnected stroke. A font or typeface without serifs is usually called a sans-serif font.

Style

The term style refers to the weight and slant of a font. Font weights can range from thin to black. The following is a list of possible weights for fonts (from lightest to heaviest):
Thin Extralight Light Normal Medium Semibold Bold Extrabold Heavy

Three terms categorize the slant of a font: roman, oblique, and italic.

The characters in a roman font are upright. The characters in an oblique font are artificially slanted. The slant is achieved by performing a shear transformation on the characters from a roman font. The characters in an italic font are truly slanted and appear as they were designed. For more information on shearing, see [Coordinate Spaces and Transformations](#).

Size

The *font size* is an imprecise value. It can generally be determined by measuring the distance from the bottom of a lowercase g to the top of an adjacent uppercase M, as shown in the following illustration.



A font's size is specified in units called points. A point is .013837 of an inch. Following the point system devised by Pierre Simon Fournier, it is common practice to

approximate a point as 1/72 inch.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Font Families

Article • 01/07/2021

A *font family* is a set of fonts having common stroke width and serif characteristics. There are five font families. A sixth family allows an application to use the default font. The following table describes the font-families.

 [Expand table](#)

Font family	Description
Decorative	Specifies a novelty font. An example is Old English.
Dontcare	Specifies a generic family name. This name is used when information about a font does not exist or does not matter. The default font is used.
Modern	Specifies a monospace font with or without serifs. Monospace fonts are usually modern; examples include Pica, Elite, and Courier New.
Roman	Specifies a proportional font with serifs. An example is Times New Roman.
Script	Specifies a font that is designed to look like handwriting; examples include Script and Cursive.
Swiss	Specifies a proportional font without serifs. An example is Arial.

These font families correspond to constants found in the Wingdi.h file: FF_DECORATIVE, FF_DONTCARE, FF_MODERN, FF_ROMAN, FF_SCRIPT, and FF_SWISS. An application uses these constants when it creates a font, selects a font, or retrieves information about a font.

Fonts within a family are distinguished by size (10 point, 24 point, and so on) and style (regular, italic, and so on).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Raster, Vector, TrueType, and OpenType Fonts

Article • 01/07/2021

Applications can use four different kinds of font technologies to display and print text:

- Raster
- Vector
- TrueType
- Microsoft OpenType

The differences between these fonts reflect the way that the *glyph* for each character or symbol is stored in the respective font-resource file:

- In raster fonts, a glyph is a bitmap that the system uses to draw a single character or symbol in the font.
- In vector fonts, a glyph is a collection of line endpoints that define the line segments that the system uses to draw a character or symbol in the font.
- In TrueType and OpenType fonts, a glyph is a collection of line and curve commands as well as a collection of hints.

The system uses the line and curve commands to define the outline of the bitmap for a character or symbol in the TrueType or Microsoft OpenType font. The system uses the hints to adjust the length of the lines and shapes of the curves used to draw the character or symbol. These hints and the respective adjustments are based on the amount of scaling used to reduce or increase the size of the bitmap. An OpenType font is equivalent to a TrueType font except that an OpenType font allows PostScript glyph definitions in addition to TrueType glyph definitions.

Because the bitmaps for each glyph in a raster font are designed for a specific resolution of device, raster fonts are generally considered to be device dependent. Vector fonts, on the other hand, are not device dependent, because each glyph is stored as a collection of scalable lines. However, vector fonts are generally drawn more slowly than raster or TrueType and OpenType fonts. TrueType and OpenType fonts provide both relatively fast drawing speed and true device independence. By using the hints associated with a glyph, a developer can scale the characters from a TrueType or OpenType font up or down and still maintain their original shape.

As previously mentioned, the glyphs for a font are stored in a font-resource file. A font-resource file is actually a DLL that contains only data, there is no code. For raster and vector fonts, this data is divided into two parts: a header describing the font's metrics

and the glyph data. A font-resource file for a raster or vector font is identified by the .fon file name extension. For TrueType and OpenType fonts, there are two files for each font: the first file contains a relatively short header and the second contains the actual font data. The first file is identified by an .fot extension and the second is identified by a .ttf extension.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Character Sets Used by Fonts

Article • 01/07/2021

All fonts use a character set. A character set contains punctuation marks, numerals, uppercase and lowercase letters, and all other printable characters. Each element of a character set is identified by a number.

Most character sets in use are supersets of the U.S. ASCII character set, which defines characters for the 96 numeric values from 32 through 127. There are five major groups of character sets:

- Windows
- Unicode
- OEM (original equipment manufacturer)
- Symbol
- Vendor-specific

Windows Character Set

The Windows character set is the most commonly used character set. It is essentially equivalent to the ANSI character set. The blank character is the first character in the Windows character set. It has a hexadecimal value of 0x20 (decimal 32). The last character in the Windows character set has a hexadecimal value of 0xFF (decimal 255).

Many fonts specify a default character. Whenever a request is made for a character that is not in the font, the system provides this default character. Many fonts using the Windows character set specify the period (.) as the default character. TrueType and OpenType fonts typically use an open box as the default character.

Fonts use a break character called a quad to separate words and justify text. Most fonts using the Windows character set specify that the blank character will serve as the break character.

Unicode Character Set

The Windows character set uses 8 bits to represent each character; therefore, the maximum number of characters that can be expressed using 8 bits is 256 (2^8). This is usually sufficient for Western languages, including the diacritical marks used in French, German, Spanish, and other languages. However, Eastern languages employ thousands of separate characters, which cannot be encoded by using a single-byte coding scheme.

With the proliferation of computer commerce, double-byte coding schemes were developed so that characters could be represented in 8-bit, 16-bit, 24-bit, or 32-bit sequences. This requires complicated passing algorithms; even so, using different code sets could yield entirely different results on two different computers.

To address the problem of multiple coding schemes, the Unicode standard for data representation was developed. A 16-bit character coding scheme, Unicode can represent 65,536 (2^{16}) characters, which is enough to include all languages in computer commerce today, as well as punctuation marks, mathematical symbols, and room for expansion. Unicode establishes a unique code for every character to ensure that character translation is always accurate.

OEM Character Set

The OEM character set is typically used in full-screen MS-DOS sessions for screen display. Characters 32 through 127 are usually the same in the OEM, U.S. ASCII, and Windows character sets. The other characters in the OEM character set (0 through 31 and 128 through 255) correspond to the characters that can be displayed in a full-screen MS-DOS session. These characters are generally different from the Windows characters.

Symbol Character Set

The Symbol character set contains special characters typically used to represent mathematical and scientific formulas.

Vendor-Specific Character Sets

Many printers and other output devices provide fonts based on character sets that differ from the Windows and OEM sets for example, the Extended Binary Coded Decimal Interchange Code (EBCDIC) character set. To use one of these character sets, the printer driver translates from the Windows character set to the vendor-specific character set.

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Font Installation and Deletion

Article • 01/07/2021

An application can use a font to draw text only if that font is either resident on a specified device or installed in the system font table. The font table is an internal array that identifies all nondevice fonts that are available to an application. An application can retrieve the names of fonts currently installed on a device or stored in the internal font table by calling the [EnumFontFamilies](#) or [ChooseFont](#) functions.

To temporarily install a font, call [AddFontResource](#) or [AddFontResourceEx](#). These functions load a font that is stored in a font-resource file. However, this is a temporary installation because after a reboot the font will not be present.

To install a font that will remain after the system is rebooted, use one of the following methods:

- Go to the Control Panel, click the **Fonts** icon, and select **Install New Fonts** from the **File** menu. The font is available to an application even before the reboot. However, in a terminal server situation the font is available for the current session but is not available for other sessions until after a reboot.
- Copy the font into the %windir%\fonts folder. Then, either go to the Control Panel and click the **Fonts** icon, or call [AddFontResource](#) or [AddFontResourceEx](#). The font is available to an application even before the reboot. However, in a terminal server situation the font is available for the current session but is not available for other sessions until after a reboot. If you only copy the font into the %windir%\fonts folder, the font will be available only after the system is rebooted.

When an application finishes using an installed font, it must remove that font by calling the [RemoveFontResource](#) function.

A font installed from a location other than the %windir%\fonts folder cannot be modified when loaded in any active session, including session 0. Any attempt to change, replace, or delete will, therefore, be blocked. If modification to a font is necessary:

- *Temporary fonts* get loaded only in the current session. Before attempting any font modifications, call [RemoveFontResource](#) to force the current session to unload the font.
- *Permanent fonts* remain installed after reboot and are loaded by all created sessions. Call [RemoveFontResource](#) to force the current session to unload the font. Then, in the font registry key
`(HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts)` find and remove the registry value associated with the

font. Finally, reboot the machine to ensure the font isn't loaded in any session. After reboot, proceed with your font modification/deletion.

Whenever an application calls the functions that add and delete font resources, it should also call the [SendMessage](#) function and send a [WM_FONTCHANGE](#) message to all top-level windows in the system. This message notifies other applications that the internal font table has been altered by an application that added or removed a font.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Fonts from Multiple Resource Files

Article • 03/17/2021

Typically, a font is contained in a single font resource file. However, the information for some fonts is spread among several files. For example, Type 1 multiple master fonts require two files:

- .pfm for the font metrics
- .pfb for the font bits

To add a font from multiple files to the system, use the [AddFontResource](#) or [AddFontResourceEx](#) functions. The *lpszFilename* parameter in these functions must point to a string that contains the file names separated by the vertical bar or pipe (|). For example, to specify abcxxxxx.pfm and abcxxxxx.pfb for a Type 1 font, use the string "abcxxxxx.pfm | abcxxxxx.pfb."

[AddFontResourceEx](#) differs from [AddFontResource](#) in that the application calling [AddFontResourceEx](#) can specify the font as private to itself or non-enumerable.

To add a font from a memory image, use [AddFontMemResourceEx](#). This allows an application to use a font that is embedded in a document or a webpage.

To remove a font that came from multiple resource files, call [RemoveFontResource](#) or [RemoveFontResourceEx](#), depending on the function used to add the font. You must specify the same flags that were used to add the font. To remove a font that was added from a memory image, use [RemoveFontMemResourceEx](#).

Using a font that comes from multiple font-resource files is identical to using a font from a single resource file.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Font Creation and Selection

Article • 01/07/2021

The **Font** common dialog box simplifies the process of creating and selecting fonts. By initializing the **CHOOSEFONT** structure and calling the **CHOOSEFONT** function, an application can support the same font-selection interface that previously required many lines of custom code. (For more information about the **Font** common dialog box, see [Common Dialog Box Library](#).)

Selection by the User

Most font creation and selection operations involve the user. For example, word processing applications let the user select unique fonts for headings, footnotes, and body text. After the user selects a font by using the **Font** dialog box and presses the **OK** button, the **CHOOSEFONT** function initializes the members of a **LOGFONT** structure with the attributes of the requested font. To use this font for text-output operations, an application must first create a logical font and then select that font into its device context. A *logical font* is an application-supplied description of an ideal font. A developer can create a logical font by calling the **CreateFont** or the **CreateFontIndirect** functions. In this case, the application would call **CreateFontIndirect** and supply a pointer to the **LOGFONT** structure initialized by **CHOOSEFONT**. In general, it is more efficient to call **CreateFontIndirect** because **CreateFont** requires several parameters while **CreateFontIndirect** requires only one pointer to **LOGFONT**.

Before an application can actually begin drawing text with a logical font, it must find the closest match from the fonts stored internally on the device and the fonts whose resources have been loaded into the operating system. The fonts stored on the device or in the operating system are called *physical fonts*. The process of finding the physical font that most closely matches a specified logical font is called font mapping. This process occurs when an application calls the **SelectObject** function and supplies a handle identifying a logical font. Font mapping is performed by using an internal algorithm that compares the attributes of the requested logical font against the attributes of available physical fonts. When the font mapper algorithm completes its search and determines the closest possible match, the **SelectObject** function returns and the application can begin drawing text with the new font.

The **SetMapperFlags** function specifies whether the font mapper algorithm searches only for physical fonts with aspect ratios that match the physical device. The aspect ratio for a device is the ratio formed by the width and the height of a pixel on that device.

The system font (also known as the shell or default font) is the font used for text in the title bars, menus, and dialog boxes.

Special Font Selection Considerations

Although most font selection operations involve the user, there are some instances where this is not true. For example, a developer may want to use a unique font in an application to draw text in a control window. To select an appropriate font, the application must be able to determine what fonts are available, create a logical font that describes one of these available fonts, and then select that font into the appropriate device context.

An application can enumerate the available fonts by using the [EnumFonts](#) or [EnumFontFamilies](#) functions. [EnumFontFamilies](#) is recommended because it enumerates all the styles associated with a family name. This can be useful for fonts with many or unusual styles and for fonts that cross international borders.

Once an application has enumerated the available fonts and located an appropriate match, it should use the values returned by the font enumeration function to initialize the members of a [LOGFONT](#) structure. Then it can call the [CreateFontIndirect](#) function, passing to it a pointer to the initialized [LOGFONT](#) structure. If the [CreateFontIndirect](#) function is successful, the application can then select the logical font by calling the [SelectObject](#) function. When initializing the members of the [LOGFONT](#) structure, be sure to specify a specific character set in the [lfCharSet](#) member. This member is important in the font mapping process and the results will be inconsistent if this member is not initialized correctly. If you specify a typeface name in the [lfFaceName](#) member of the [LOGFONT](#) structure, make sure that the [lfCharSet](#) value matches the character set of the typeface specified in [lfFaceName](#). For example, if you want to select a font such as MS Mincho, [lfCharSet](#) must be set to the predefined value SHIFTJIS_CHARSET.

The fonts for many East Asian languages have two typeface names: an English name and a localized name. [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) take the localized typeface name for a system locale that matches the language, but they take the English typeface name for 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. Starting with Windows 2000, this is no longer a problem because the font mapper for [CreateFont](#), [CreateFontIndirect](#), and [CreateFontIndirectEx](#) recognizes either typeface name, regardless of locale.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Embedded Fonts

Article • 01/07/2021

Embedding a font is the technique of bundling a document and the fonts it contains into a file for transmission to another computer. Embedding a font guarantees that a font specified in a transmitted file will be present on the computer receiving the file. Not all fonts can be moved from computer to computer, however, since most fonts are licensed to only one computer at a time. Only TrueType and OpenType fonts can be embedded.

Applications should embed a font in a document only when requested by a user. An application cannot be distributed along with documents that contain embedded fonts, nor can an application itself contain an embedded font. Whenever an application distributes a font, in any format, the proprietary rights of the owner of the font must be acknowledged.

It may be a violation of a font vendor's proprietary rights or user license agreement to embed any fonts where embedding is not permitted or to fail to observe the following guidelines on embedding fonts. A font's license may give only read/write permission for a font to be installed and used on the destination computer. Or the license may give read-only permission. Read-only permission allows a document to be viewed and printed (but not modified) by the destination computer; documents with read-only embedded fonts are themselves read-only. Read-only embedded fonts may not be unbundled from the document and installed on the destination computer.

An application can determine the license status by calling the [GetOutlineTextMetrics](#) function and examining 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.

To embed a TrueType font, an application can use the [GetFontData](#) function to read the font file. Setting the *dwTable* and *dwOffset* parameters of [GetFontData](#) to 0L and the *cbData* parameter to 1L ensures that the application reads the entire font file from the beginning.

Several functions are available to embed OpenType fonts depending on the character width and where the font data resides. To embed an OpenType Unicode font that resides in a device context, an application can use [TTEmbedFont](#). To embed an OpenType UCS-4 font that resides in a device context, an application can use [TTEmbedFontEx](#). To embed an OpenType Unicode font that resides in a font file, an

application can use **TTEmbedFontFromFile**. For additional information on OpenType font embedding, see the [Font Embedding Reference](#).

After an application retrieves the font data, it can store the data with the document by using any applicable format. Most applications build a font directory in the document, listing the embedded fonts and whether the embedding is read/write or read-only. An application can use the **otmpStyleName** and **otmFamilyName** members of the **OUTLINETEXTMETRIC** structure to identify the font.

If the read-only bit is set for the embedded font, applications must encrypt the font data before storing it with the document. The encryption method need not be complicated; for example, using the XOR operator to combine the font data with an application-defined constant is adequate and fast.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Text Output

Article • 01/07/2021

Text output is the most common type of graphic output found within the client area; it is used by applications in different ways. Word processing and desktop publishing applications create documents with formatted text; spreadsheet applications use text, numbers, and symbols to specify formulas, label columns, and list values; database applications create records and display queries with text, and CAD applications use text to label objects and display dimensions.

There are functions to format and draw text in an application's client area and on a page of printer paper. These functions can be divided into two categories: those that format the text (or prepare it for output) and those that actually draw the text. The formatting functions align text, set the intercharacter spacing, set the text and text-background colors, and justify text. The drawing functions draw individual characters (or symbols) or entire strings of text.

When working in Microsoft Windows, hard line breaks are specified with the carriage-return/line feed pair (\r\n).

For more information, see the following topics:

- [Formatting Text](#)
- [Drawing Text](#)
- [Complex Scripts](#)
- [ClearType Antialiasing](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Formatting Text (Windows GDI)

Article • 01/07/2021

The formatting functions can be divided into three categories: those that retrieve or set the [text-formatting attributes](#) for a device context, those that retrieve [character widths](#), and those that retrieve [string widths and heights](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Text-Formatting Attributes

Article • 01/07/2021

An application can use six functions to set the text-formatting attributes for a device context: [SetBkColor](#), [SetBkMode](#), [SetTextAlign](#), [SetTextCharacterExtra](#), [SetTextColor](#), and [SetTextJustification](#). These functions affect the text alignment, the intercharacter spacing, the text justification, and text and background colors. In addition, six other functions can be used to retrieve the current text formatting attributes for any device context: [GetBkColor](#), [GetBkMode](#), [GetTextAlign](#), [GetTextCharacterExtra](#), [GetTextColor](#), and [GetTextExtentPoint32](#).

Text Alignment

Applications can use the [SetTextAlign](#) function to specify how the system should position the characters in a string of text when they call one of the drawing functions. This function can be used to position headings, page numbers, callouts, and so on. The system positions a string of text by aligning a reference point on an imaginary rectangle that surrounds the string, with the current cursor position or with a point passed as an argument to one of the text drawing functions. The [SetTextAlign](#) function lets the application specify the location of this reference point. The following is a list of the possible reference point locations.

[+] Expand table

Location	Description
left/bottom	The reference point is located at the lower-left corner of the rectangle.
left/base line	The reference point is located at the intersection of the character-cell base line and the left edge of the rectangle.
left/top	The reference point is located at the top-left corner of the rectangle.
center/bottom	The reference point is located at the center of the bottom of the rectangle.
center/base line	The reference point is located at the intersection of the character-cell base line and the center of the rectangle.
center/top	The reference point is located at the center of the top of the rectangle.
right/bottom	The reference point is located at the lower-right corner of the rectangle.
right/base line	The reference point is located at the intersection of the character-cell base line and the right edge of the rectangle.

Location	Description
right/top	The reference point is located at the top-right corner of the rectangle.

The following illustration shows a string of text drawn by calling the [TextOut](#) function. Before drawing the text, the [SetTextAlign](#) function was called to relocate the reference point at each one of the nine possible locations.

```

Text-alignment demonstration.
+Text-alignment demonstration.
+Text-alignment demonstration.

Text-alignment demonstration.
+Text-alignment demonstration.
+Text-alignment demonstration.

Text-alignment demonstration.+ 
Text-alignment demonstration.+ 
Text-alignment demonstration.+ 

```

The default text alignment for a device context is the upper-left corner of the imaginary rectangle that surrounds the text. An application can retrieve the current text-alignment setting for any device context by calling the [GetTextAlign](#) function.

Intercharacter Spacing

Applications can use the [SetTextCharacterExtra](#) function to alter the intercharacter spacing for all text output operations in a specified device context. The following illustration shows a string of text drawn twice by calling the [TextOut](#) function. Before drawing the text the second time, the [SetTextCharacterExtra](#) function was called to increment the intercharacter spacing.

Experiments in Typography

Experiments in Typography

The default intercharacter spacing value for any device context is zero. An application can retrieve the current intercharacter spacing value for a device context by calling the [GetTextCharacterExtra](#) function.

Text Justification

Applications can use the [GetTextExtentPoint32](#) and [SetTextJustification](#) functions to justify a line of text. Text justification is a common operation in any desktop publishing and in most word processing applications. The [GetTextExtentPoint32](#) function computes the width and height of a string of text. After the width is computed, the application can call the [SetTextJustification](#) function to distribute extra spacing between each of the words in a line of text. The following illustration shows a paragraph of text printed twice: in the first paragraph, the text was not justified; in the second paragraph, the text was justified by calling the [GetTextExtentPoint32](#) and [SetTextJustification](#) functions.

GDI transforms the width, height, and depth parameters, once by using the destination display context and once by using the source display context. If the resulting extents do not match, GDI uses the `StretchBlt` function to compress or stretch the source bitmap as necessary.

GDI transforms the width, height, and depth parameters, once by using the destination display context and once by using the source display context. If the resulting extents do not match, GDI uses the `StretchBlt` function to compress or stretch the source bitmap as necessary.



Text and Background Color

Applications can use the [SetTextColor](#) function to set the color of text drawn in the client-area of their windows, as well as the color of text drawn on a color printer. An application can use the [SetBkColor](#) function to set the color that appears behind each character and the [SetBkMode](#) function to specify how the system should blend the selected background color with the current color or colors on the video display.

The default text color for a display device context is black; the default background color is white; and the default background mode is OPAQUE. An application can retrieve the current text color for a device context by calling the [GetTextColor](#) function. An application can retrieve the current background color for a device context by calling the [GetBkColor](#) function and the current background mode by calling the [GetBkMode](#) function.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Character Widths

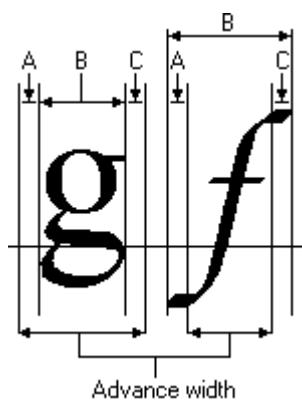
Article • 01/07/2021

Applications need to retrieve character-width data when they perform such tasks as fitting strings of text to page or column widths. There are four functions that an application can use to retrieve character-width data. Two of these functions retrieve the character-advance width and two of these functions retrieve actual character-width data.

An application can use the [GetCharWidth32](#) and [GetCharWidthFloat](#) functions to retrieve the advance width for individual characters or symbols in a string of text. The advance width is the distance that the cursor on a video display or the print-head on a printer must advance before printing the next character in a string of text. The [GetCharWidth32](#) function returns the advance width as an integer value. If greater precision is required, an application can use the [GetCharWidthFloat](#) function to retrieve fractional advance-width values.

An application can retrieve actual character-width data by using the [GetCharABCWidths](#) and [GetCharABCWidthsFloat](#) functions. The [GetCharABCWidthsFloat](#) function works with all fonts. The [GetCharABCWidths](#) function only works with TrueType and OpenType fonts. For more information about TrueType and OpenType fonts, see [Raster, Vector, TrueType, and OpenType Fonts](#).

The following illustration shows the three components of a character width:



The A spacing is the width to add to the current position before placing the character. The B spacing is the width of the character itself. The C spacing is the white space to the right of the character. The total advance width is determined by calculating the sum of A+B+C. The character cell is an imaginary rectangle that surrounds each character or symbol in a font. Because characters can overhang or underhang the character cell, either or both of the A and C increments can be a negative number.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

String Widths and Heights

Article • 01/07/2021

In addition to retrieving character-width data for individual characters, applications also need to compute the width and height of entire strings. Two functions retrieve string-width and height measurements: [GetTextExtentPoint32](#), and [GetTabbedTextExtent](#). If the string does not contain tab characters, an application can use the [GetTextExtentPoint32](#) function to retrieve the width and height of a specified string. If the string contains tab characters, an application should call the [GetTabbedTextExtent](#) function.

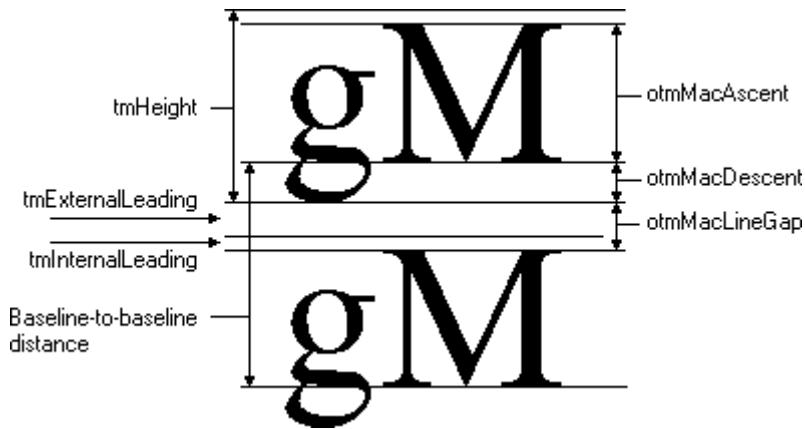
Applications can use the [GetTextExtentExPoint](#) function for word-wrapping operations. This function returns the number of characters from a specified string that fit within a specified space.

Font Ascenders and Descenders

Some applications determine the line spacing between text lines of different sizes by using a font's maximum ascender and descender. An application can retrieve these values by calling the [GetTextMetrics](#) function and then checking the **tmAscent** and **tmDescent** members of the [TEXTMETRIC](#).

The maximum ascent and descent are different from the typographic ascent and descent. In TrueType and OpenType fonts, the typographic ascent and descent are typically the top of the f glyph and bottom of the g glyph. An application can retrieve the typographic ascender and descender for a TrueType or OpenType font by calling the [GetOutlineTextMetrics](#) function and checking the values in the **otmMacAscent** and **otmMacDescent** members of the [OUTLINETEXTMETRIC](#) structure.

The following figure shows the difference between the vertical text metric values returned in the [NEWTEXTMETRIC](#) and [OUTLINETEXTMETRIC](#) structures. (The names beginning with otm are members of the [OUTLINETEXTMETRIC](#) structure.)



Font Dimensions

An application can retrieve the physical dimensions of a TrueType or OpenType font by calling the [GetOutlineTextMetrics](#) function. An application can retrieve the physical dimensions of any other font by calling the [GetTextMetrics](#) function. To determine the dimensions of an output device, an application can call the [GetDeviceCaps](#) function. [GetDeviceCaps](#) returns both physical and logical dimensions.

A logical inch is a measure the system uses to present legible fonts on the screen and is approximately 30 to 40 percent larger than a physical inch. The use of logical inches precludes an exact match between the output of the screen and printer. Developers should be aware that the text on a screen is not simply a scaled version of the text that will appear on the page, particularly if graphics are incorporated into the text.

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing Text (Windows GDI)

Article • 01/07/2021

After an application selects the appropriate font, sets the required text-formatting options, and computes the necessary character width and height values for a string of text, it can begin drawing characters and symbols by calling any of the text-output functions:

- [DrawText](#)
- [DrawTextEx](#)
- [ExtTextOut](#)
- [PolyTextOut](#)
- [TabbedTextOut](#)
- [TextOut](#)

When an application calls one of these functions, the operating system passes the call to the graphics engine, which in turn passes the call to the appropriate device driver. At the device driver level, all of these calls are supported by one or more calls to the driver's own [ExtTextOut](#) or [TextOut](#) function. An application will achieve the fastest execution by calling [ExtTextOut](#), which is quickly converted into an [ExtTextOut](#) call for the device. However, there are instances when an application should call one of the other three functions; for example, to draw multiple lines of text within the borders of a specified rectangular region, it is more efficient to call [DrawText](#). To create a multicolumn table with justified columns of text, it is more efficient to call [TabbedTextOut](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Complex Scripts

Article • 01/07/2021

While the functions discussed in the preceding work well for many languages, they may not deal with the needs of complex scripts. *Complex scripts* are languages whose printed form is not rendered in a simple way. For example, a complex script may allow bidirectional rendering, contextual shaping of glyphs, or combining characters. Due to these special requirements, the control of text output must be very flexible.

Functions that display text [TextOut](#), [ExtTextOut](#), [TabbedTextOut](#), [DrawText](#), and [GetTextExtentExPoint](#) have been extended to support complex scripts. In general, this support is transparent to the application. However, applications should save characters in a buffer and display a whole line of text at one time, so that the complex script shaping modules can use context to reorder and generate glyphs correctly. In addition, because the width of a glyph can vary by context, applications should use [GetTextExtentExPoint](#) to determine line length rather than using cached character widths.

In addition, complex script-aware applications should consider adding support for right-to-left reading order and right alignment to their applications. You can toggle the reading order or alignment between left and right with the following code:

C++

```
// Save lAlign (this example uses static variables)
static LONG lAlign = TA_LEFT;

// When user toggles alignment (assuming TA_CENTER is not supported).

lAlign = TA_RIGHT;

// When the user toggles reading order.

lAlign = TA_RTLREADING;

// Before calling ExtTextOut, for example, when processing WM_PAINT

SetTextAlign (hDc, lAlign);
```

To toggle both attributes at once, execute the following statement and then call [SetTextAlign](#) and [ExtTextOut](#), as shown previously:

C++

```
lAlign = TA_RIGHT^TA_RTLREADING; //pre-inline !
```

You can also process complex scripts with Uniscribe. Uniscribe is a set of functions that allow a fine degree of control for complex scripts. For more information, see [Uniscribe](#) and [Processing Complex Scripts](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ClearType Antialiasing

Article • 01/07/2021

Microsoft ClearType antialiasing is a smoothing method that improves font display resolution over traditional antialiasing. It dramatically improves readability on color LCD monitors with a digital interface, such as those in laptops and high-quality flat desktop displays. Readability on CRT screens is also somewhat improved.

However, ClearType is dependent on the orientation and ordering of the LCD stripes. Currently, ClearType is implemented only for LCDs with vertical stripes that are ordered RGB. In particular, this affects tablet PCs, where the display can be oriented in any direction, and those screens that can be turned from landscape to portrait.

ClearType antialiasing is allowed:

- For 16-, 24-, and 32-bit color (disabled for 256 colors or less)
- For screen DC and memory DC (not for printer DC)
- For TrueType fonts and OpenType fonts with TrueType outlines

ClearType antialiasing is disabled:

- Under terminal server client
- For bitmap fonts, vector fonts, device fonts, Type 1 fonts, or Postscript OpenType fonts without TrueType outlines
- If the font has tuned embedded bitmaps, only for those font sizes that contain the embedded bitmaps

To activate ClearType antialiasing, call [SystemParametersInfo](#) once to turn on font smoothing and then a second time to set the smoothing type to FE_FONTSMOOTHINGCLEARATYPE, as shown in the following code sample:

C++

```
SystemParametersInfo(SPI_SETFONTSMOOTHING,
                     TRUE,
                     0,
                     SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
SystemParametersInfo(SPI_SETFONTSMOOTHINGTYPE,
                     0,
                     (PVOID)FE_FONTSMOOTHINGCLEARATYPE,
                     SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
```

You can adjust the appearance of text by changing the contrast value used in the ClearType algorithm. The default is 1,400, but it can be any value from 1,000 to 2,200.

Depending on the display device and the user's sensitivity to colors, a higher or lower contrast value may improve readability. To change the contrast, call [SystemParametersInfo](#) with SPI_SETFONTSMOOTHINGCONTRAST. The following code sets the contrast value to 1,600.

C++

```
SystemParametersInfo(SPI_SETFONTSMOOTHINGCONTRAST,  
    0,  
    (PVOID)1600,  
    SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
```

You should consider the following details for application compatibility:

- Text rendering with ClearType is slightly slower than with standard antialiasing.
- Applications should not use XOR to display selected text. Applications should set the background color and redisplay the selected text.
- Applications should not paint the same text on top of itself in transparent mode. If this occurs, the edge pixels that are antialiased will color merge with themselves instead of with the background color. This results in darkened and colorful edges.
- Applications should not paint text by painting the characters individually when in opaque mode because the edge of a character may be clipped by the following character. This occurs because a character that is smoothed with ClearType may have a negative A or C width where the regular character has a positive A or C width. Only the B width of the character is guaranteed to be the same. Likewise, applications should be careful if smoothed text is next to unsmoothed text.
- If an application renders text and then manipulates the bitmap, font smoothing should be turned off by setting the **IfQuality** member of the [LOGFONT](#) structure to NONANTIALIASED_QUALITY. For example, a game may add a bitmap shadow effect, or text rendered into a bitmap may be scaled to produce a thumbview.
- If the user is running in portrait mode (that is, monitor striping is horizontal) ClearType antialiasing should be disabled.

The *fdwQuality* parameter in [CreateFont](#) and the **IfQuality** member of [LOGFONT](#) accept the CLEARTYPE_QUALITY flag. Rasterization of fonts created with this flag will use the ClearType rasterizer. This flag has no effect on previous versions of the operating system.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using the Font and Text-Output Functions

Article • 01/07/2021

This section describes how you can use the font and text-output functions to draw normal text, draw text from different fonts on the same line, rotate lines of text, display the font-selection common dialog-box, enumerate fonts, and so on:

- Using a stock font to draw text
- Creating a logical font
- Enumerating the installed fonts
- Checking the text capabilities of a device
- Setting the text alignment
- Drawing text from different fonts on the same line
- Rotating lines of text
- Retrieving character outlines
- Using portable TrueType metrics
- Using PANOSE numbers
- Specifying length of text-output string

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using a Stock Font to Draw Text

Article • 01/07/2021

The system provides six stock fonts. A stock font is a logical font that an application can obtain by calling the [GetStockObject](#) function and specifying the requested font. The following list contains the values that you can specify to obtain a stock font.

[+] [Expand table](#)

Value	Meaning
ANSI_FIXED_FONT	Specifies a monospace font based on the Windows character set. A Courier font is typically used.
ANSI_VAR_FONT	Specifies a proportional font based on the Windows character set. MS Sans Serif is typically used.
DEVICE_DEFAULT_FONT	Specifies the preferred font for the specified device. This is typically the System font for display devices; however, for some dot-matrix printers this is a font that is resident on the device. (Printing with this font is usually faster than printing with a downloaded, bitmap font).
OEM_FIXED_FONT	Specifies a monospace font based on an OEM character set. For IBM computers and compatibles, the OEM font is based on the IBM PC character set.
SYSTEM_FONT	Specifies the System font. This is a proportional font based on the Windows character set, and is used by the operating system to display window titles, menu names, and text in dialog boxes. The System font is always available. Other fonts are available only if they have been installed.
SYSTEM_FIXED_FONT	Specifies a monospace font compatible with the System font in early versions of Windows.

For more information on fonts, see [About Fonts](#).

The following example retrieves a handle to the variable stock font, selects it into a device context, and then writes a string using that font:

C++

```
HFONT hFont, hOldFont;  
  
// Retrieve a handle to the variable stock font.
```

```
hFont = (HFONT)GetStockObject(ANSI_VAR_FONT);

// Select the variable stock font into the specified device context.
if (hOldFont = (HFONT)SelectObject(hdc, hFont))
{
    // Display the text string.
    TextOut(hdc, 10, 50, L"Sample ANSI_VAR_FONT text", 25);

    // Restore the original font.
    SelectObject(hdc, hOldFont);
}
```

If other stock fonts are not available, [GetStockObject](#) returns a handle to the System font (SYSTEM_FONT). You should use stock fonts only if the mapping mode for your application's device context is MM_TEXT.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Creating a Logical Font

Article • 01/07/2021

You can use the **Font** common dialog box to display available fonts. The **ChooseFont** dialog box is displayed after an application initializes the members of a **CHOOSEFONT** structure and calls the **CHOOSEFONT** function. After the user selects one of the available fonts and presses the **OK** button, the **ChooseFont** function initializes a **LOGFONT** structure with the relevant data. Your application can then call the **CreateFontIndirect** function and create a logical font based on the user's request. The following example demonstrates how this is done.

C++

```
HFONT FAR PASCAL MyCreateFont( void )
{
    CHOOSEFONT cf;
    LOGFONT lf;
    HFONT hfont;

    // Initialize members of the CHOOSEFONT structure.

    cf.lStructSize = sizeof(CHOOSEFONT);
    cf.hwndOwner = (HWND)NULL;
    cf.hDC = (HDC)NULL;
    cf.lpLogFont = &lf;
    cf.iPointSize = 0;
    cf.Flags = CF_SCREENFONTS;
    cf.rgbColors = RGB(0,0,0);
    cf.lCustData = 0L;
    cf.lpfnHook = (LPCFHOOKPROC)NULL;
    cf.lpTemplateName = (LPSTR)NULL;
    cf.hInstance = (HINSTANCE) NULL;
    cf.lpszStyle = (LPSTR)NULL;
    cf.nFontType = SCREEN_FONTTYPE;
    cf.nSizeMin = 0;
    cf.nSizeMax = 0;

    // Display the CHOOSEFONT common-dialog box.

    ChooseFont(&cf);

    // Create a logical font based on the user's
    // selection and return a handle identifying
    // that font.

    hfont = CreateFontIndirect(cf.lpLogFont);
    return (hfont);
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enumerating the Installed Fonts

Article • 01/07/2021

In some instances, an application must be able to enumerate the available fonts and select the one most appropriate for a particular operation. An application can enumerate the available fonts by calling the [EnumFonts](#) or [EnumFontFamilies](#) function. These functions send information about the available fonts to a callback function that the application supplies. The callback function receives information in [LOGFONT](#) and [NEWTEXTMETRIC](#) structures. (The [NEWTEXTMETRIC](#) structure contains information about a TrueType font. When the callback function receives information about a non-TrueType font, the information is contained in a [TEXTMETRIC](#) structure.) By using this information, an application can limit the user's choices to only those fonts that are available.

The [EnumFontFamilies](#) function is similar to the [EnumFonts](#) function but includes some extra functionality. [EnumFontFamilies](#) allows an application to take advantage of styles available with TrueType fonts. New and upgraded applications should use [EnumFontFamilies](#) instead of [EnumFonts](#).

TrueType fonts are organized around a typeface name (for example, Courier New) and style names (for example, italic, bold, and extra-bold). The [EnumFontFamilies](#) function enumerates all the styles associated with a specified family name, not simply the bold and italic attributes. For example, when the system includes a TrueType font called Courier New Extra-Bold, [EnumFontFamilies](#) lists it with the other Courier New fonts. The capabilities of [EnumFontFamilies](#) are helpful for fonts with many or unusual styles and for fonts that cross international borders.

If an application does not supply a typeface name, the [EnumFonts](#) and [EnumFontFamilies](#) functions supply information about one font in each available family. To enumerate all the fonts in a device context, the application can specify **NULL** for the typeface name, compile a list of the available typefaces, and then enumerate each font in each typeface.

The following example uses the [EnumFontFamilies](#) function to retrieve the number of available raster, vector, and TrueType font families.

C++

```
UINT uAlignPrev;
int aFontCount[] = { 0, 0, 0 };
char szCount[8];
HRESULT hr;
size_t * pcch;
```

```

EnumFontFamilies(hdc, (LPCTSTR) NULL,
    (FONTENUMPROC) EnumFamCallBack, (LPARAM) aFontCount);

uAlignPrev = SetTextAlign(hdc, TA_UPDATECP);

MoveToEx(hdc, 10, 50, (LPOINT)NULL);
TextOut(hdc, 0, 0, "Number of raster fonts: ", 24);
itoa(aFontCount[0], szCount, 10);

    hr = StringCchLength(szCount, 9, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, 0, 0, szCount, *pcch);

MoveToEx(hdc, 10, 75, (LPOINT)NULL);
TextOut(hdc, 0, 0, "Number of vector fonts: ", 24);
itoa(aFontCount[1], szCount, 10);
    hr = StringCchLength(szCount, 9, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, 0, 0, szCount, *pcch);

MoveToEx(hdc, 10, 100, (LPOINT)NULL);
TextOut(hdc, 0, 0, "Number of TrueType fonts: ", 26);
itoa(aFontCount[2], szCount, 10);
    hr = StringCchLength(szCount, 9, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, 0, 0, szCount, *pcch);

SetTextAlign(hdc, uAlignPrev);

BOOL CALLBACK EnumFamCallBack(LPLOGFONT lplf, LPNEWTEXTMETRIC lpntm, DWORD
FontType, LPVOID aFontCount)
{
    int far * aiFontCount = (int far *) aFontCount;

    // Record the number of raster, TrueType, and vector
    // fonts in the font-count array.

    if (FontType & RASTER_FONTTYPE)
        aiFontCount[0]++;
    else if (FontType & TRUETYPE_FONTTYPE)
        aiFontCount[2]++;
    else
        aiFontCount[1]++;

    if (aiFontCount[0] || aiFontCount[1] || aiFontCount[2])

```

```
        return TRUE;
    else
        return FALSE;

    UNREFERENCED_PARAMETER( lplf );
    UNREFERENCED_PARAMETER( lptnm );
}
```

This example uses two masks, RASTER_FONTTYPE and TRUETYPE_FONTTYPE, to determine the type of font being enumerated. If the RASTER_FONTTYPE bit is set, the font is a raster font. If the TRUETYPE_FONTTYPE bit is set, the font is a TrueType font. If neither bit is set, the font is a vector font. A third mask, DEVICE_FONTTYPE, is set when a device (for example, a laser printer) supports downloading TrueType fonts; it is zero if the device is a display adapter, dot-matrix printer, or other raster device. An application can also use the DEVICE_FONTTYPE mask to distinguish GDI-supplied raster fonts from device-supplied fonts. The system can simulate bold, italic, underline, and strikeout attributes for GDI-supplied raster fonts, but not for device-supplied fonts.

An application can also check bits 1 and 2 in the **tmPitchAndFamily** member of the **NEWTEXTMETRIC** structure to identify a TrueType font. If bit 1 is 0 and bit 2 is 1, the font is a TrueType font.

Vector fonts are categorized as OEM_CHARSET instead of ANSI_CHARSET. Some applications identify vector fonts by using this information, checking the **tmCharSet** member of the **NEWTEXTMETRIC** structure. This categorization usually prevents the font mapper from choosing vector fonts unless they are specifically requested. (Most applications no longer use vector fonts because their strokes are single lines and they take longer to draw than TrueType fonts, which offer many of the same scaling and rotation features that required vector fonts.)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Checking the Text Capabilities of a Device

Article • 01/07/2021

You can use the [EnumFonts](#) and [EnumFontFamilies](#) functions to enumerate the fonts that are available in a printer-compatible memory device context. You can also use the [GetDeviceCaps](#) function to retrieve information about the text capabilities of a device. By calling the [GetDeviceCaps](#) function with the NUMFONTS index, you can determine the minimum number of fonts supported by a printer. (An individual printer may support more fonts than specified in the return value from [GetDeviceCaps](#) with the NUMFONTS index.) By using the TEXTCAPS index, you can identify many of the text capabilities of the specified device.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Setting the Text Alignment

Article • 01/07/2021

You can query and set the text alignment for a device context by using the [GetTextAlign](#) and [SetTextAlign](#) functions. The text-alignment settings determine how text is positioned relative to a specified location. Text can be aligned to the right or left of the position or centered over it; it can also be aligned above or below the point.

The following example shows a method for determining which horizontal alignment flag is set:

C++

```
switch ((TA_LEFT | TA_RIGHT | TA_CENTER) & GetTextAlign(hdc))
{
    case TA_LEFT:
        .
        .
        .

    case TA_RIGHT:
        .
        .
        .

    case TA_CENTER:
        .
        .
        .

}
```

You can also use the [SetTextAlign](#) function to update the current position when a text-output function is called. For instance, the following example uses the [SetTextAlign](#) function to update the current position when the [TextOut](#) function is called. In this example, the *cArial* parameter is an integer that specifies the number of Arial fonts.

C++

```
UINT uAlignPrev;
char szCount[8];
HRESULT hr;
size_t * pcch;

uAlignPrev = SetTextAlign(hdc, TA_UPDATECP);
MoveToEx(hdc, 10, 50, (LPOINT) NULL);
TextOut(hdc, 0, 0, "Number of Arial fonts: ", 23);
itoa(cArial, szCount, 10);

hr = StringCchLength(szCount, 9, pcch);
```

```
if (FAILED(hr))
{
// TODO: write error handler
}

TextOut(hdc, 0, 0, (LPSTR) szCount, *pcch);
SetTextAlign(hdc, uAlignPrev);
```

ⓘ 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.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing Text from Different Fonts on the Same Line

Article • 11/19/2022

Different type styles within a font family can have different widths. For example, bold and italic styles of a family are always wider than the roman style for a specified point size. When you display or print several type styles on a single line, you must keep track of the width of the line to avoid having characters displayed or printed on top of one another.

You can use two functions to retrieve the width (or extent) of text in the current font. 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 a specified array of tab-stop positions. The [GetTextExtentPoint32](#) function computes the width and height of a line of text.

When necessary, the system synthesizes a font by changing the character bitmaps. To synthesize a character in a bold font, the system draws the character twice: at the starting point, and again one pixel to the right of the starting point. To synthesize a character in an italic font, the system draws two rows of pixels at the bottom of the character cell, moves the starting point one pixel to the right, draws the next two rows, and continues until the character has been drawn. By shifting pixels, each character appears to be sheared to the right. The amount of shear is a function of the height of the character.

One way to write a line of text that contains multiple fonts is to use the [GetTextExtentPoint32](#) function after each call to [TextOut](#) and add the length to a current position. The following example writes the line "This is a sample string." using bold characters for "This is a", switches to italic characters for "sample", then returns to bold characters for "string." After printing all the strings, it restores the system default characters.

C++

```
int XIncrement;
int YStart;
TEXTMETRIC tm;
HFONT hfntDefault, hfntItalic, hfntBold;
SIZE sz;
LPSTR lpszString1 = "This is a ";
LPSTR lpszString2 = "sample ";
LPSTR lpszString3 = "string.";
HRESULT hr;
```

```
size_t * pcch;

// Create a bold and an italic logical font.

hfntItalic = MyCreateFont();
hfntBold = MyCreateFont();

// Select the bold font and draw the first string
// beginning at the specified point (XIncrement, YStart).

XIncrement = 10;
YStart = 50;
hfntDefault = SelectObject(hdc, hfntBold);
hr = StringCchLength(lpszString1, 11, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, XIncrement, YStart, lpszString1, *pcch);

// Compute the length of the first string and add
// this value to the x-increment that is used for the
// text-output operation.

hr = StringCchLength(lpszString1, 11, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
GetTextExtentPoint32(hdc, lpszString1, *pcch, &sz);
XIncrement += sz.cx;

// Retrieve the overhang value from the TEXTMETRIC
// structure and subtract it from the x-increment.
// (This is only necessary for non-TrueType raster
// fonts.)

GetTextMetrics(hdc, &tm);
XIncrement -= tm.tmOverhang;

// Select an italic font and draw the second string
// beginning at the point (XIncrement, YStart).

hfntBold = SelectObject(hdc, hfntItalic);
GetTextMetrics(hdc, &tm);
XIncrement -= tm.tmOverhang;
hr = StringCchLength(lpszString2, 8, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, XIncrement, YStart, lpszString2, *pcch);

// Compute the length of the second string and add
```

```

// this value to the x-increment that is used for the
// text-output operation.

hr = StringCchLength(lpszString2, 8, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
GetTextExtentPoint32(hdc, lpszString2, *pcch, &sz);
XIncrement += sz.cx;

// Reselect the bold font and draw the third string
// beginning at the point (XIncrement, YStart).

SelectObject(hdc, hfntBold);
hr = StringCchLength(lpszString3, 8, pcch);
    if (FAILED(hr))
    {
        // TODO: write error handler
    }
TextOut(hdc, XIncrement - tm.tmOverhang, YStart, lpszString3,
        *pcch);

// Reselect the original font.

SelectObject(hdc, hfntDefault);

// Delete the bold and italic fonts.

DeleteObject(hfntItalic);
DeleteObject(hfntBold);

```

In this example, the [GetTextExtentPoint32](#) function initializes the members of a [SIZE](#) structure with the length and height of the specified string. The [GetTextMetrics](#) function retrieves the overhang for the current font. Because the overhang is zero if the font is a TrueType font, the overhang value does not change the string placement. For raster fonts, however, it is important to use the overhang value.

The overhang is subtracted from the bold string once, to bring subsequent characters closer to the end of the string if the font is a raster font. Because overhang affects both the beginning and end of the italic string in a raster font, the glyphs start at the right of the specified location and end at the left of the endpoint of the last character cell. (The [GetTextExtentPoint32](#) function retrieves the extent of the character cells, not the extent of the glyphs.) To account for the overhang in the raster italic string, the example subtracts the overhang before placing the string and subtracts it again before placing subsequent characters.

The [SetTextJustification](#) function adds extra space to the break characters in a line of text. You can use the [GetTextExtentPoint](#) function to determine the extent of a string,

then subtract that extent from the total amount of space the line should occupy, and use the [SetTextJustification](#) function to distribute the extra space among the break characters in the string. The [SetTextCharacterExtra](#) function adds extra space to every character cell in the selected font, including the break character. (You can use the [GetTextCharacterExtra](#) function to determine the current amount of extra space being added to the character cells; the default setting is zero.)

You can place characters with greater precision by using the [GetCharWidth32](#) or [GetCharABCWidths](#) function to retrieve the widths of individual characters in a font. The [GetCharABCWidths](#) function is more accurate than the [GetCharWidth32](#) function, but it can only be used with TrueType fonts.

ABC spacing also allows an application to perform very accurate text alignment. For example, when the application right aligns a raster roman font without using ABC spacing, the advance width is calculated as the character width. This means the white space to the right of the glyph in the bitmap is aligned, not the glyph itself. By using ABC widths, applications have more flexibility in the placement and removal of white space when aligning text, because they have information that allows them to finely control intercharacter spacing.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Rotating Lines of Text

Article • 01/07/2021

You can rotate TrueType fonts at any angle. This is useful for labeling charts and other illustrations. The following example rotates a string in 10-degree increments around the center of the client area by changing the value of the **IfEscapement** and **IfOrientation** members of the [LOGFONT](#) structure used to create the font.

C++

```
#include "strsafe.h"
HRESULT 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.

LocalFree((LOCALHANDLE) plf);
    EndPaint(hWnd, &ps);
    break;
}
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Retrieving Character Outlines

Article • 01/07/2021

You can use the [GetGlyphOutline](#) function to retrieve the outline of a glyph from a TrueType font. The glyph outline returned by the [GetGlyphOutline](#) function is for a grid-fitted glyph. (A grid-fitted glyph has been modified so that its bitmap image conforms as closely as possible to the original design of the glyph.) If your application requires an unmodified glyph outline, request the glyph outline for a character in a font whose size is equal to the font's em units. (To create a font with this size, set the **IfHeight** member of the [LOGFONT](#) structure to the negative of the value of the **ntmSizeEM** member of the [NEWTEXTMETRIC](#) structure.)

[GetGlyphOutline](#) returns the outline as a bitmap or as a series of polylines and splines. When an application retrieves a glyph outline as a series of polylines and splines, the information is returned in a [TTPOLYGONHEADER](#) structure followed by as many [TTPOLYCURVE](#) structures as 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 where the outline for a contour begins. The [TTPOLYCURVE](#) structures that follow can be either polyline records or spline records.

To render a TrueType character outline, you must use both the polyline and the spline records. The system can render both polylines and splines easily. Each polyline and spline record contains as many sequential points as possible, to minimize the number of records returned.

The starting point specified in the [TTPOLYGONHEADER](#) structure is always on the outline of the glyph. The specified point serves as both the starting and ending points for the contour.

This section provides information on the following topics.

- [Polyline Records](#)
- [Spline Records](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Polyline Records

Article • 01/07/2021

[Polyline](#) records are a series of points; lines drawn between the points describe the outline of the character. A polyline record begins with the last point in the previous record (or, for the first record in the contour, the starting point). Each point in the record is on the glyph outline and can be connected simply by using straight lines.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Spline Records

Article • 01/07/2021

Spline records represent the quadratic curves (that is, quadratic b-splines) used by TrueType. A spline record begins with the last point in the previous record (or for the first record in the contour, with the starting point). For the first spline record, the starting point and the last point in the record are on the glyph outline. For all other spline records, only the last point is on the glyph outline. All other points in the spline records are off the glyph outline and must be rendered as the control points of b-splines.

The last spline or polyline record in a contour always ends with the contour's starting point. This arrangement ensures that every contour is closed.

Because b-splines require three points (one point off the glyph outline between two points that are on the outline), you must perform some calculations when a spline record contains more than one off-curve point.

For example, if a spline record contains three points (A, B, and C) and it is not the first record, points A and B are off the glyph outline. To interpret point A, use the current position (which is always on the glyph outline) and the point on the glyph outline between points A and B. To find the midpoint (M) between A and B, you can perform the following calculation.

$$M = A + (B - A)/2$$

The midpoint between consecutive off-outline points in a spline record is a point on the glyph outline, according to the definition of the spline format used in TrueType fonts.

If the current position is designated by P, the two quadratic splines defined by this spline record are (P, A, M) and (M, B, C).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Portable TrueType Metrics

Article • 01/07/2021

Applications that use the TrueType text metrics can achieve a high degree of printer and document portability; they can use TrueType metrics even if they must maintain compatibility with early 16-bit versions of Windows.

Design widths overcome most of the problems of device-dependent text introduced by physical devices. Design widths are a kind of logical width. Independent of any rasterization problems or scaling transformations, each glyph has a logical width and height. Composed to a logical page, each character in a string has a place that is independent of the physical device widths. Although a logical width implies that widths can be scaled linearly at all point sizes, this is not necessarily true for either nonportable or most TrueType fonts. At smaller point sizes, some glyphs are made wider relative to their height for better readability.

The characters in TrueType core fonts are designed against a 2048 by 2048 grid. The design width is the width of a character in these grid units. (TrueType supports any integer grid size up to 16,384 by 16,384; grid sizes that are integer powers of 2 scale faster than other grid sizes.)

The font outline is designed in notional units. The em square is the notional grid against which the font outline is fitted. (You can use the **otmEMSSquare** member of **OUTLINETEXTMETRIC** and the **ntmSizeEM** member of **NEWTEXTMETRIC** to retrieve the size of the em square in notional units.) When a font is created that has a point size (in device units) equal to the size of its em square, the ABC widths for this font are the desired design widths. For example, assume the size of an em square is 1000 and the ABC widths of a character in the font are 150, 400, and 150. A character in this font that is 10 device units high would have ABC widths of 1.5, 4, and 1.5, respectively. Since the MM_TEXT mapping mode is most commonly used with fonts (and MM_TEXT is equivalent to device units), this is a simple calculation.

Because of the high resolution of TrueType design widths, applications that use them must take into account the large numeric values that can be created. For more information, see the following topics:

- [Device vs. Design Units](#)
- [Metrics for Portable Documents](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Device vs. Design Units

Article • 01/07/2021

An application can retrieve font metrics for a physical font only after the font has been selected into a device context. When a font is selected into a device context, it is scaled for the device. The font metrics specific to the device are known as device units.

Portable metrics in fonts are known as design units. To apply to a specified device, design units must be converted to device units. Use the following formula to convert design units to device units.

$$\text{DeviceUnits} = (\text{DesignUnits}/\text{unitsPerEm}) * (\text{PointSize}/72) * \text{DeviceResolution}$$

The variables in this formula have the following meanings.

[+] [Expand table](#)

Variable	Description
<i>DeviceUnits</i>	Specifies the <i>DesignUnits</i> font metric converted to device units. This value is in the same units as the value specified for <i>DeviceResolution</i> .
<i>DesignUnits</i>	Specifies the font metric to be converted to device units. This value can be any font metric, including the width of a character or the ascender value for an entire font.
<i>unitsPerEm</i>	Specifies the em square size for the font.
<i>PointSize</i>	Specifies size of the font, in points. (One point equals 1/72 of an inch.)
<i>DeviceResolution</i>	Specifies number of device units (pixels) per inch. Typical values might be 300 for a laser printer or 96 for a VGA screen.

This formula should not be used to convert device units back to design units. Device units are always rounded to the nearest pixel. The propagated round-off error can become very large, especially when an application is working with screen sizes.

To request design units, create a logical font whose height is specified as *unitsPerEm*. Applications can retrieve the value for *unitsPerEm* by calling the [EnumFontFamilies](#) function and checking the **ntmSizeEM** member of the [NEWTEXTMETRIC](#) structure.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Metrics for Portable Documents

Article • 01/07/2021

The following table specifies the most important font metrics for applications that require portable documents and the functions that allow an application to retrieve them.

[+] Expand table

Function	Metric	Use
EnumFontFamilies	ntmSizeEM	Retrieval of design metrics; conversion to device metrics.
GetCharABCWidths	ABCWidths	Accurate placement of characters at the start and end of margins, picture boundaries, and other text breaks.
GetCharWidth32	AdvanceWidths	Placement of characters on a line.
GetOutlineTextMetrics	otmfsType	Font-embedding bits.
	otmsCharSlopeRise	Y-component for slope of cursor for italic fonts.
	otmsCharSlopeRun	X-component for slope of cursor for italic fonts.
	otmAscent	Line spacing.
	otmDescent	Line spacing.
	otmLineGap	Line spacing.
	otmpFamilyName	Font identification.
	otmpStyleName	Font identification.
	otmpFullName	Font identification (typically, family and style name).

The **otmsCharSlopeRise**, **otmsCharSlopeRun**, **otmAscent**, **otmDescent**, and **otmLineGap** members of the [OUTLINETEXTMETRIC](#) structure are scaled or transformed to correspond to the current device mode and physical height (as specified in the **tmHeight** member of the [NEWTEXTMETRIC](#) structure).

Font identification is important in those instances when an application must select the same font, for example, when a document is reopened or moved to a different operating system. The font mapper always selects the correct font when an application

requests a font by full name. The family and style names provide input to the standard font dialog box, which ensures that the selection bars are properly placed.

The **otmsCharSlopeRise** and **otmsCharSlopeRun** values are used to produce a close approximation of the main italic angle of the font. For typical roman fonts, **otmsCharSlopeRise** is 1 and **otmsCharSlopeRun** is 0. For italic fonts, the values attempt to approximate the sine and cosine of the main italic angle of the font (in counterclockwise degrees past vertical); note that the italic angle for upright fonts is 0. Because these values are not expressed in design units, they should not be converted into device units.

The character placement and line spacing metrics enable an application to compute device-independent line breaks that are portable across screens, printers, typesetters, and even platforms.

To perform device-independent page layout

1. Normalize all design metrics to a common ultra-high resolution (UHR) value (for example, 65,536 DPI); this prevents round-off errors.
2. Compute line breaks based on UHR metrics and physical page width; this yields a starting point and an ending point of a line within the text stream.
3. Compute the device page width in device units (for example, pixels).
4. Fit each line of text into the device page width, using the line breaks computed in step 2.
5. Compute page breaks by using UHR metrics and the physical page length; this yields the number of lines per page.
6. Compute the line heights in device units.
7. Fit the lines of text onto the page, using the lines per page from step 5 and the line heights from step 6.

If all applications adopt these techniques, developers can virtually guarantee that documents moved from one application to another will retain their original appearance and format.

Feedback

Was this page helpful?



Using PANOSE Numbers

Article • 01/07/2021

TrueType font files include PANOSE numbers, which applications can use to choose a font that closely matches their specifications. The PANOSE system classifies faces by 10 different attributes. For more information about these attributes, see [PANOSE](#). A PANOSE structure is part of the [OUTLINETEXTMETRIC](#) structure (whose values are filled in by calling the [GetOutlineTextMetrics](#) function).

The PANOSE attributes are rated individually on a scale. The resulting values are concatenated to produce a number. Given this number for a font and a mathematical metric to measure distances in the PANOSE space, an application can determine the nearest neighbors.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Specifying length of text-output string

Article • 01/07/2021

Several of the font and text-output functions have a parameter that specifies the length of the text-output string. A typical example is the *cchText* parameter of [DrawTextEx](#).

Each of these functions has both an "ANSI" version and a Unicode version (for example, [DrawTextExA](#) and [DrawTextExW](#), respectively). For the "ANSI" version of each function, the length is specified as a BYTE count and for the Unicode function it is specified as a WORD count.

It is traditional to think of this as a "character count". That is generally accurate for many languages, including English, but it is not accurate in general. In "ANSI" strings, characters in [SBCS](#) code pages take one byte each, but most characters in [DBCS](#) code pages take two bytes. Similarly, most currently defined Unicode characters reside in the Basic Multilingual Plane (BMP) and their UTF-16 representations fit in one WORD, but supplementary characters are represented in Unicode by "surrogates", which require two WORDs.

Each of these functions takes a length count. For the "ANSI" version of each function, the length is specified as a BYTE count length of a string not including the **NULL** terminator. For the Unicode function, the length count is the byte count divided by `sizeof(WCHAR)`, which is 2, not including the **NULL** terminator. The character count is the count of the number of characters, which might not equal the length count of the string. In some instances, characters take more than one BYTE for ANSI (for example, [DBCS](#) character) and more than one WORD for Unicode (for example, surrogate characters). Further, the number of glyphs might not equal the number of characters because multiple characters might be composited to make one glyph. Length count is the amount of data. Character count is the number of units that are processed as one entity. Glyphs are what gets rendered. For example, in Unicode, you can have a string with length of 3, which is 2 characters and which results in 1 glyph being rendered. However, typically, most Unicode strings length, character count, and number of rendered glyphs are equal.

You can use `_tcslen()` to get the string length. For ANSI, `_tcslen()` returns the number of bytes. For Unicode, `_tcslen()` returns the number of WCHARs (that is, WORDs).

Special processing characters like tabs and soft hyphens that aren't always drawn can affect the drawn output. They get included in the string length and character counts, but might not be directly represented by a rendered glyph.

Some of these functions allow the caller to specify the length as -1 to indicate that the string is null-terminated; in that case the function will compute the character count automatically. Not all of the functions offer this capability. That is specified on a function-by-function basis; see the individual function documentation.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Font and Text Reference

Article • 01/07/2021

The following elements are used with fonts and text:

- [Font and Text Functions](#)
- [Font and Text Structures](#)
- [Font and Text Macros](#)
- [Font and Text Messages](#)
- [Font-Package Function Error Messages](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Font and Text Functions (Windows GDI)

Article • 01/07/2021

The following functions are used with fonts and text.

[+] Expand table

Function	Description
AddFontMemResourceEx	Adds an embedded font to the system font table.
AddFontResource	Adds a font resource to the system font table.
AddFontResourceEx	Adds a private or non-enumerable font to the system font table.
CreateFont	Creates a logical font.
CreateFontIndirect	Creates a logical font from a structure.
CreateFontIndirectEx	Creates a logical font from a structure.
DrawText	Draws formatted text in a rectangle.
DrawTextEx	Draws formatted text in rectangle.
EnumFontFamExProc	An application definedcallback function used with EnumFontFamiliesEx to process fonts.
EnumFontFamiliesEx	Enumerates all fonts in the system with certain characteristics.
ExtTextOut	Draws a character string.
GetAspectRatioFilterEx	Gets the setting for the aspect-ratio filter.
GetCharABCWidths	Gets the widths of consecutive characters from the TrueType font.
GetCharABCWidthsFloat	Gets the widths of consecutive characters from the current font.
GetCharABCWidthsI	Gets the widths of consecutive glyph indices or from an array of glyph indices from the TrueType font.
GetCharacterPlacement	Gets information about a character string.
GetCharWidth32	Gets the widths of consecutive characters from the current font.
GetCharWidthFloat	Gets the fractional widths of consecutive characters from the current font.
GetCharWidthI	Gets the widths of consecutive glyph indices or an array of glyph indices from the current font.

Function	Description
GetFontData	Gets metric data for a TrueType font.
GetFontLanguageInfo	Returns information about the selected font for a display context.
GetFontUnicodeRanges	Tells which Unicode characters are supported by a font.
GetGlyphIndices	Translates a string into an array of glyph indices.
GetGlyphOutline	Gets the outline or bitmap for a character in the TrueType font.
GetKerningPairs	Gets the character-kerning pairs for a font.
GetOutlineTextMetrics	Gets text metrics for TrueType fonts.
GetRasterizerCaps	Tells whether TrueType fonts are installed.
GetTabbedTextExtent	Computes the width and height of a character string, including tabs.
GetTextAlign	Gets the text-alignment setting for a device context.
GetTextCharacterExtra	Gets the current intercharacter spacing for a device context.
GetTextColor	Gets the text color for a device context.
GetTextExtentExPoint	Gets the number of characters in a string that will fit within a space.
GetTextExtentExPoint1	Gets the number of glyph indices that will fit within a space.
GetTextExtentPoint32	Computes the width and height of a string of text.
GetTextExtentPoint1	Computes the width and height of an array of glyph indices.
GetTextFace	Gets the name of the font that is selected into a device context.
GetTextMetrics	Fills a buffer with the metrics for a font.
PolyTextOut	Draws several strings using the font and text colors in a device context.
RemoveFontMemResourceEx	Removes a font whose source was embedded in a document from the system font table.
RemoveFontResource	Removes the fonts in a file from the system font table.
RemoveFontResourceEx	Removes a private or non-enumerable font from the system font table.
SetMapperFlags	Alters the algorithm used to map logical fonts to physical fonts.

Function	Description
SetTextAlign	Sets the text-alignment flags for a device context.
SetTextCharacterExtra	Sets the intercharacter spacing.
SetTextColor	Sets the text color for a device context.
SetTextJustification	Specifies the amount of space the system should add to the break characters in a string.
TabbedTextOut	Writes a character string at a location, expanding tabs to specified values.
TextOut	Writes a character string at a location.

Obsolete Functions

These functions are provided only for compatibility with 16-bit versions of Windows.

- [CreateScalableFontResource](#)
- [EnumFontFamilies](#)
- [EnumFontFamProc ↗](#)
- [EnumFonts](#)
- [EnumFontsProc ↗](#)
- [GetCharWidth](#)
- [GetTextExtentPoint](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

AddFontMemResourceEx function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/09/2023

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

C++

```
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.

 Expand table

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.
.mmm	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 *lpszFileName* point to a string with the file names separated by a "|" --for example, abcxxxxx.pfm | abcxxxxx.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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/09/2023

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

C++

```
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.

 Expand table

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.
.mmm	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.

 Expand table

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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

CreateFontA function (wingdi.h)

08/15/2025

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

C++

```
HFONT CreateFontA(
    [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.

 Expand table

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:

C++

```
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.

[Expand table](#)

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. For a list of possible values, see *IfCharSet* field of the **LOGFONT** structure.

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 *pszFaceName* parameter, make sure that the *iCharSet* value matches the character set of the typeface specified in *pszFaceName*.

[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.

 Expand table

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.

 Expand table

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_LHANGLES	<p>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.</p> <p>If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system.</p> <p>For more information about the orientation of coordinate systems, see the description of the <i>nOrientation</i> parameter</p>
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	<p>Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated.</p> <p>For compatibility, this value is always returned when enumerating fonts.</p>
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.

 Expand table

Value	Meaning
<code>ANTIALIASED_QUALITY</code>	Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large.
<code>CLEARTYPE_QUALITY</code>	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.
<code>DEFAULT_QUALITY</code>	Appearance of the font does not matter.
<code>DRAFT_QUALITY</code>	Appearance of the font is less important than when the <code>PROOF_QUALITY</code> 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.
<code>NONANTIALIASED_QUALITY</code>	Font is never antialiased, that is, font smoothing is not done.
<code>PROOF_QUALITY</code>	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 <code>PROOF_QUALITY</code> 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.

[Expand table](#)

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 *pszFaceName* 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

C++

```
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;
            HFONT hFontOriginal, hFont1, hFont2, hFont3;
```

```

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.
        hFont1 =
CreateFont(48,0,0,0,FW_DONTCARE, FALSE, TRUE, FALSE,DEFAULT_CHARSET,OUT_OUTLINE_PRECI
S,
            CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY,
VARIABLE_PITCH,TEXT("Impact"));
        hFontOriginal = (HFONT)SelectObject(hdc, hFont1);

        //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)
        hFont2 =
CreateFont(36,20,-300,0,FW_DONTCARE, FALSE, TRUE, FALSE,DEFAULT_CHARSET,OUT_OUTLINE_P
RECIS,
            CLIP_DEFAULT_PRECIS,CLEARTYPE_QUALITY, VARIABLE_PITCH,TEXT("Times
New Roman"));
        SelectObject(hdc,hFont2);

        //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)
        hFont3 =
CreateFont(36,10,250,0,FW_DONTCARE, FALSE, TRUE, FALSE,DEFAULT_CHARSET,OUT_OUTLINE_PR
ECIS,
            CLIP_DEFAULT_PRECIS,ANTIALIASED_QUALITY,
VARIABLE_PITCH,TEXT("Arial"));
        SelectObject(hdc,hFont3);

        //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);

    SelectObject(hdc,hFontOriginal);
    DeleteObject(hFont1);
    DeleteObject(hFont2);
    DeleteObject(hFont3);

    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 that 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

 [Expand table](#)

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

Requirement	Value
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)

Article 02/22/2024

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

C++

```
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 that 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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/09/2023

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

C++

```
HFONT CreateFontIndirectEx(
    [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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

CreateScalableFontResourceA function (wingdi.h)

Article02/09/2023

[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

C++

```
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.

 Expand table

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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

DrawText function (winuser.h)

Article 08/02/2022

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

C++

```
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.

[\[+\] Expand table](#)

Value	Meaning
<code>DT_BOTTOM</code>	Justifies the text to the bottom of the rectangle. This value is used only with the <code>DT_SINGLELINE</code> value.
<code>DT_CALCRECT</code>	Determines the width and height of the rectangle. If there are multiple lines of text, <code>DrawText</code> 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, <code>DrawText</code> modifies the right side of the rectangle so that it bounds the last character in the line. In either case, <code>DrawText</code> returns the height of the formatted text but does not draw the text.
<code>DT_CENTER</code>	Centers text horizontally in the rectangle.
<code>DT_EDITCONTROL</code>	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.
<code>DT_END_ELLIPSIS</code>	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. The string is not modified unless the <code>DT_MODIFYSTRING</code> flag is specified. Compare with <code>DT_PATH_ELLIPSIS</code> and <code>DT_WORD_ELLIPSIS</code> .
<code>DT_EXPANDTABS</code>	Expands tab characters. The default number of characters per tab is eight. The <code>DT_WORD_ELLIPSIS</code> , <code>DT_PATH_ELLIPSIS</code> , and <code>DT_END_ELLIPSIS</code> values cannot be used with the <code>DT_EXPANDTABS</code> value.
<code>DT_EXTERNALLEADING</code>	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.

DT_HIDEPREFIX	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_HIDEPREFIX: "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. 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	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, Example: input string: "A&bc&&d" normal: "Abc&d" DT_NOPREFIX: "A&bc&&d" 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	<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: "Ab<u>c</u>&d"</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	<p>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.</p> <p>If this is not specified, output is on one line.</p>
DT_WORD_ELLIPSIS	<p>Truncates any word that does not fit in the rectangle and adds ellipses.</p> <p>Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.</p>

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

[+] [Expand table](#)

Requirement	Value
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)

02/09/2023

The **DrawTextEx** function draws formatted text in the specified rectangle.

Syntax

C++

```
int DrawTextExA(
    [in]      HDC          hdc,
    [in, out] LPSTR       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.

 Expand table

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_HIDEPREFIX	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_HIDEPREFIX: "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, DrawTextEx 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 <i>lprc</i> 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 winuser.h header defines DrawTextEx as an alias that 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](#).

[+] Expand table

Requirement	Value
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-l1-2-0 (introduced in Windows 8.1)

See also

[DRAWTEXTPARAMS](#)

[DrawText](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

EnableEUDC function

This function enables or disables support for end-user-defined characters (EUDC).

Syntax

C++

```
BOOL EnableEUDC(  
    _In_ HDC BOOL fEnableEUDC  
)
```

Parameters

fEnableEUDC [in]

Boolean that is set to **TRUE** to enable EUDC, and to **FALSE** to disable EUDC.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

If EUDC is disabled, trying to display EUDC characters will result in missing or bad glyphs.

During multi-session, this function affects the current session only.

It is recommended that you use this function with Windows XP SP2 or later.

(!) Note

EnableEUDC is recommended to be called only once, as repeated calls may cause performance degradation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Library	Gdi32.lib
DLL	Gdi32.dll

Last updated on 01/07/2026

EnumFontFamiliesA function (wingdi.h)

Article 02/09/2023

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

C++

```
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_FONTTYPE 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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/09/2023

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

C++

```
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.

[] [Expand table](#)

Member	Description
<code>IfCharSet</code>	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.
<code>IfFaceName</code>	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.
<code>IfPitchAndFamily</code>	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**.

[+] Expand table

Values	Meaning
IfCharSet = DEFAULT_CHARSET	Enumerates all uniquely-named fonts within all character sets. If there are two fonts with the same name, only one is enumerated.
IfFaceName = '\0'	
IfCharSet = DEFAULT_CHARSET	Enumerates all character sets and styles in a specific font.

IfFaceName = a specific font

IfCharSet = a specific character set Enumerates all styles of all fonts in the specific character set.

IfFaceName = '\0'

IfCharSet = a specific character set Enumerates all styles of a font in a specific character set.

IfFaceName = a specific font

The following code sample shows how these values are used.

C++

```
// 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;
```

C++

```
// 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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[EnumFontFamExProc](#)

[EnumFontFamilies](#)

[EnumFonts](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[LOGFONT](#)

EnumFontsA function (wingdi.h)

Article02/09/2023

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

C++

```
int EnumFontsA(
    [in] HDC             hdc,
    [in] LPCSTR          lpLogfont,
    [in] FONTEXNUMPROCA 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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[EnumFontFamilies](#)

[EnumFontFamiliesEx](#)

[EnumFontsProc](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetDeviceCaps](#)

ExtTextOutA function (wingdi.h)

Article 02/09/2023

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, opaquning, or both.

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>ETO_CLIPPED</code>	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.

 Expand table

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 that 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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetAspectRatioFilterEx function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
BOOL GetAspectRatioFilterEx(
    [in]  HDC      hdc,
    [out] LPSIZE  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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCharABCWidthsA function (wingdi.h)

Article 02/09/2023

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

C++

```
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 that 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

 [Expand table](#)

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABC](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharWidth](#)

[GetOutlineTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetCharABCWidthsFloatA function (wingdi.h)

Article 02/09/2023

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

Syntax

C++

```
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 that 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

 Expand table

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[ABCFLOAT](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharABCWidths](#)

[GetCharWidth](#)

[GetCharWidthFloat](#)

GetCharABCWidthsI function (wingdi.h)

Article 10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[ABC](#)

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetCharWidth](#)

[GetOutlineTextMetrics](#)

[OUTLINETEXTMETRIC](#)

GetCharacterPlacementA function (wingdi.h)

Article02/09/2023

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

C++

```
DWORD GetCharacterPlacementA(
    [in]      HDC          hdc,
    [in]      LPCSTR       lpString,
    [in]      int          nCount,
    [in]      int          nMexExtent,
    [in, out] LPGCP_RESULTSA 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] *nMaxExtent*

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.

[] [Expand table](#)

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	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).

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 `lpGlyphs` 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.

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 `lpGcpResults->lpOutString` in one call to [GetCharacterPlacement](#) is the input string of a second call).

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.

<code>GCP_MAXEXTENT</code>	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.
<code>GCP_NEUTRAL OVERRIDE</code>	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 <code>GCP_REORDER</code> flag.
<code>GCP_NUMERIC OVERRIDE</code>	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 <code>GCP_REORDER</code> flag.
<code>GCP_NUMERICSLATIN</code>	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.
<code>GCP_NUMERICSLOCAL</code>	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.
<code>GCP_REORDER</code>	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. If this flag is set for Semitic languages and the <code>lpClass</code> array is used, the first two elements of the array are used to specify the reading order beyond the bounds of the string. <code>GCP_CLASS_PREBOUNDRTL</code> and <code>GCP_CLASS_PREBOUNDLTR</code>

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.

If the GCP_REORDER value is not specified, the *lpString* parameter is taken to be visual ordered for languages where this is used, and the *lpOutString* and *lpOrder* fields are ignored.

Use [GetFontLanguageInfo](#) to determine whether the current font supports reordering.

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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCharWidthA function (wingdi.h)

Article 02/22/2024

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

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCharWidth32A function (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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 that 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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetCharWidthFloatA function (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetFontData function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

The **GetFontUnicodeRanges** function returns information about which Unicode characters are supported by a font. The information is returned as a [GLYPHSET](#) structure.

Syntax

C++

```
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 `lpgs` 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/20/2024

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

C++

```
DWORD GetGlyphIndicesA(
    [in]    HDC      hdc,
    [in]    LPCSTR   lpstr,
    [in]    int       c,
    [out]   LPWORD   pgi,
    [in]    DWORD     f1
);
```

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 *lpstr* 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] `f1`

Specifies how glyphs should be handled if they are not supported. This parameter can be the following value.

Value	Meaning
GGI_MARK_NONEXISTING_GLYPHS	Marks unsupported glyphs with the hexadecimal value 0xffff.

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 that 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

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetFontUnicodeRanges](#)

GetGlyphOutlineA function (wingdi.h)

Article 02/09/2023

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

Syntax

C++

```
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.

[] [Expand table](#)

Value	Meaning
<code>GGO_BEZIER</code>	The function retrieves the curve data as a cubic Bézier spline (not in quadratic spline format).
<code>GGO_BITMAP</code>	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 (**lpvBuffer*) 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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetKerningPairsA function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetOutlineTextMetricsA function (wingdi.h)

Article 11/20/2024

The **GetOutlineTextMetrics** function retrieves text metrics for TrueType fonts.

Syntax

C++

```
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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetRasterizerCaps function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetTabbedTextExtentA function (winuser.h)

02/22/2024

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

C++

```
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 that 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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
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](#)

GetTextAlign function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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.

[] Expand table

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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

Syntax

C++

```
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 0x8000000.

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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
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)

Article 02/09/2023

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

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/19/2022

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

C++

```
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 *pgiln* 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 *alpDx* 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetTextExtentPointA function (wingdi.h)

Article 11/20/2024

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

C++

```
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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[Font and Text Functions](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[SIZE](#)

[SetTextCharacterExtra](#)

GetTextExtentPoint32A function (wingdi.h)

Article 02/09/2023

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

Syntax

C++

```
BOOL GetTextExtentPoint32A(
    [in]  HDC      hdc,
    [in]  LPCSTR  lpString,
    [in]  int       c,
    [out] LPVOID  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 *lpDx* 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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetTextExtentPointI function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetTextFaceA function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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 that 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](#).

 [Expand table](#)

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PolyTextOutA function (wingdi.h)

Article02/09/2023

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

Syntax

C++

```
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 that 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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

RemoveFontMemResourceEx function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/09/2023

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

C++

```
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_FONTCHANGE](#) 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.

syntax

```
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 that 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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article11/20/2024

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

Syntax

C++

```
BOOL RemoveFontResourceExA(
    [in] LPCSTR 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.

syntax

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetMapperFlags function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetTextAlign function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>TA_BASELINE</code>	The reference point will be on the base line of the text.
<code>TA_BOTTOM</code>	The reference point will be on the bottom edge of the bounding rectangle.
<code>TA_TOP</code>	The reference point will be on the top edge of the bounding rectangle.
<code>TA_CENTER</code>	The reference point will be aligned horizontally with the center of the bounding rectangle.
<code>TA_LEFT</code>	The reference point will be on the left edge of the bounding rectangle.
<code>TA_RIGHT</code>	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.

 Expand table

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

C++

```
SetTextAlign (hdc, GetTextAlign(hdc) & (~TA_CENTER))
```

or

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

TabbedTextOutA function (winuser.h)

02/09/2023

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

C++

```
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.

① Note

The `winuser.h` header defines `TabbedTextOut` as an alias that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	<code>winuser.h</code> (include <code>Windows.h</code>)
Library	<code>User32.lib</code>
DLL	<code>User32.dll</code>
API set	<code>ext-ms-win-ntuser-misc-l1-5-1</code> (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

TextOutA function (wingdi.h)

Article 02/09/2023

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

Syntax

C++

```
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.

 Expand table

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.

C++

```
// 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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Font and Text Structures

Article • 11/19/2022

The following structures are used with fonts and text.

[ABC](#)
[ABCFLOAT](#)
[AXESLIST](#)
[AXISINFO](#)
[DESIGNVECTOR](#)
[DRAWTEXTPARAMS](#)
[ENUMLOGFONT](#)
[ENUMLOGFONTEX](#)
[ENUMLOGFONTEXDV](#)
[ENUMTEXTMETRIC](#)
[EXTLOGFONT](#)
[FIXED](#)
[GCP_RESULTS](#)
[GLYPHMETRICS](#)
[GLYPHSET](#)
[KERNINGPAIR](#)
[LOGFONT](#)
[MAT2](#)
[NEWTEXTMETRIC](#)
[NEWTEXTMETRICEX](#)
[OUTLINETEXTMETRIC](#)
[PANOSE](#)
[POINTFX](#)
[POLYTEXT](#)
[RASTERIZER_STATUS](#)
[SIZE](#)
[TEXTMETRIC](#)
[TTPOLYCURVE](#)
[TTPOLYGONHEADER](#)
[WCRANGE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ABC structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

ABCFLOAT structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct _ABCFLOAT {
    FLOAT abcfA;
    FLOAT abcfB;
    FLOAT abcfC;
} 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.

abcfC

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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

AXESLISTA structure (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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[MM_MAX_NUMAXES]`

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 that 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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

AXISINFOA structure (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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[MM_MAX_AXES_NAMELEN]`

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 that 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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DESIGNVECTOR structure (wingdi.h)

Article09/01/2022

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

Syntax

C++

```
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[MM_MAX_NUMAXES]

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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

DRAWTEXTPARAMS structure (winuser.h)

Article 02/22/2024

The **DRAWTEXTPARAMS** structure contains extended formatting options for the [DrawTextEx](#) function.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

ENUMLOGFONTA structure (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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[LF_FULLFACESIZE]`

A unique name for the font. For example, ABCD Font Company TrueType Bold Italic Sans Serif.

`elfStyle[LF_FACESIZE]`

The style of the font. For example, Bold Italic.

Remarks

Note

The wingdi.h header defines **ENUMLOGFONT** as an alias that 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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ENUMLOGFONTEXA structure (wingdi.h)

Article 11/20/2024

The **ENUMLOGFONTEX** structure contains information about an enumerated font.

Syntax

C++

```
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[LF_FULLFACESIZE]`

The unique name of the font. For example, ABC Font Company TrueType Bold Italic Sans Serif.

`elfStyle[LF_FACESIZE]`

The style of the font. For example, Bold Italic.

`elfScript[LF_FACESIZE]`

The script, that is, the character set, of the font. For example, Cyrillic.

Remarks

Note

The wingdi.h header defines **ENUMLOGFONTEX** as an alias that 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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ENUMLOGFONTEXDVA structure (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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 that 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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ENUMTEXTMETRICA structure (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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 that 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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

EXTLOGFONTA structure (wingdi.h)

Article 09/01/2022

The **EXTLOGFONT** structure defines the attributes of a font.

Syntax

C++

```
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[LF_FULLFACESIZE]`

A unique name for the font (for example, ABCD Font Company TrueType Bold Italic Sans Serif).

`elfStyle[LF_FACESIZE]`

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 **IfHeight** 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[ELF_VENDOR_SIZE]

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 that 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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

FIXED structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct _FIXED {
#ifndef ...
    WORD fract;
#endif ...
    short value;
#else
    short value;
#endif
#ifndef ...
    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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

GCP_RESULTS structure (wingdi.h)

Article 08/23/2022

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

C++

```
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:

syntax

```
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:

syntax

```
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.

[+] Expand table

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.

[+] Expand table

Value	Meaning
GCPCLASS_LATINNUMERICSEPARATOR	Input only. Character used to separate Latin digits, such as a comma or decimal point.
GCPCLASS_LATINNUMERICTERMINATOR	Input only. Character used to terminate Latin digits, such as a plus or minus sign.
GCPCLASS_NEUTRAL	Input only. Character has no specific classification.
GCPCLASS_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.

[] Expand table

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.

1pGlyphs

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 *lpInputString* 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 that 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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

GLYPHMETRICS structure (wingdi.h)

Article 04/02/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

GLYPHSET structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[] [Expand table](#)

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[1]`

Array of Unicode ranges that are supported in the font.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

KERNINGPAIR structure (wingdi.h)

Article02/22/2024

The KERNINGPAIR structure defines a kerning pair.

Syntax

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetKerningPairs](#)

Feedback

Was this page helpful?

 Yes

 No

LOGFONTA structure (wingdi.h)

The LOGFONT structure defines the attributes of a font.

Syntax

C++

```
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.

 Expand table

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:

C++

```
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.

 Expand table

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:

 [Expand table](#)

Value	Description
ANSI_CHARSET	This font supports the Windows ANSI character set.
ARABIC_CHARSET	This font supports the Arabic character set.
BALTIC_CHARSET	This font supports the Baltic character set.
CHINESEBIG5_CHARSET	This font supports the traditional Chinese (Big 5) character set.
DEFAULT_CHARSET	This font supports character set value based on the system default Windows ANSI code page. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.
EASTEUROPE_CHARSET	This font supports the Eastern European character set.
GB2312_CHARSET	This font supports the simplified (PRC) Chinese character set.
GREEK_CHARSET	This font supports the Greek character set.
HANGEUL_CHARSET	This font supports the Korean (Hangul) character set.
HEBREW_CHARSET	This font supports the Hebrew character set.
JOHAB_CHARSET	This font supports the Korean (Johab) character set.
MAC_CHARSET	This font supports character set value based on the current system Macintosh code page. This value is used primarily in legacy code and should not generally be needed since modern Macintosh computers use Unicode for encoding.
OEM_CHARSET	This font supports an OEM-specific character set. The OEM character set is system dependent.
RUSSIAN_CHARSET	This font supports the Cyrillic character set.
SHIFTJIS_CHARSET	This font supports the Shift-JIS (Japanese Industry Standard) character set.
SYMBOL_CHARSET	This font supports the Windows symbol character set.
THAI_CHARSET	This font supports the Thai character set.
TURKISH_CHARSET	This font supports the Turkish character set.
VIETNAMESE_CHARSET	This font supports the Vietnamese character set.

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 when creating a font, do not specify **OEM_CHARSET** or **DEFAULT_CHARSET**. 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**.

IfOutPrecision

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.

 Expand table

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.

`lfClipPrecision`

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.

 Expand table

Value	Meaning
<code>CLIP_CHARACTER_PRECIS</code>	Not used.
<code>CLIP_DEFAULT_PRECIS</code>	Specifies default clipping behavior.
<code>CLIP_DFA_DISABLE</code>	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.
<code>CLIP_EMBEDDED</code>	You must specify this flag to use an embedded read-only font.
<code>CLIP_LH_ANGLES</code>	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.
<code>CLIP_MASK</code>	Not used.
<code>CLIP_DFA_OVERRIDE</code>	Turns off font association for the font. This is identical to <code>CLIP_DFA_DISABLE</code> , but it can have problems in some situations; the recommended flag to use is <code>CLIP_DFA_DISABLE</code> .
<code>CLIP_STROKE_PRECIS</code>	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.
<code>CLIP_TT_ALWAYS</code>	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.

IfPitchAndFamily

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.

[+] Expand table

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.

IfFaceName[LF_FACESIZE]

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 that 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

 Expand table

Requirement	Value
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](#)

Last updated on 08/15/2025

MAT2 structure (wingdi.h)

Article 02/22/2024

The **MAT2** structure contains the values for a transformation matrix used by the [GetGlyphOutline](#) function.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

NEWTEXTMETRICA structure (wingdi.h)

Article 07/27/2022

The **NEWTEXTMETRIC** structure contains data that describes a physical font.

Syntax

C++

```
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.

[\[+\] Expand table](#)

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 that 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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

NEWTEXTMETRICEXA structure (wingdi.h)

Article 11/20/2024

The NEWTEXTMETRICEX structure contains information about a physical font.

Syntax

C++

```
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 that 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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

OUTLINETEXTMETRICA structure (wingdi.h)

Article 07/27/2022

The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.

Syntax

C++

```
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.

[\[+\] Expand table](#)

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 [C-Compiler Packing Issues](#).

Note

The wingdi.h header defines **OUTLINETEXTMETRIC** as an alias that 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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PANOSE structure (wingdi.h)

Article 04/02/2021

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

C++

```
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.

[+] Expand table

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.

[Expand table](#)

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.

[Expand table](#)

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.

[\[\] Expand table](#)

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.

[\[+\] Expand table](#)

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.

[\[+\] Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

Value	Meaning
PAN_ANY	Any
PAN_NO_FIT	No fit
PAN_MIDLINE_STANDARD_TRIMMED	Standard/trimmed
PAN_MIDLINE_STANDARD_POINTED	Standard/pointed
PAN_MIDLINE_STANDARD_SERIFED	Standard/serifed
PAN_MIDLINE_HIGH_TRIMMED	High/trimmed
PAN_MIDLINE_HIGH_POINTED	High/pointed
PAN_MIDLINE_HIGH_SERIFED	High/serifed
PAN_MIDLINE_CONSTANT_TRIMMED	Constant/trimmed
PAN_MIDLINE_CONSTANT_POINTED	Constant/pointed
PAN_MIDLINE_CONSTANT_SERIFED	Constant/serifed
PAN_MIDLINE_LOW_TRIMMED	Low/trimmed
PAN_MIDLINE_LOW_POINTED	Low/pointed

PAN_MIDLIN LOW SERIFED

Low/serifed

bXHeight

For Latin fonts, one of the following values.

[+] Expand table

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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

POINTFX structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
typedef struct tagPOINTFX {
    FIXED x;
    FIXED y;
} POINTFX, *LPPOINTFX;
```

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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

POLYTEXTA structure (wingdi.h)

Article 11/20/2024

The **POLYTEXT** structure describes how the [PolyTextOut](#) function should draw a string of text.

Syntax

C++

```
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.

[+] Expand table

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.

pdx

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 that 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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[PolyTextOut](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

RASTERIZER_STATUS structure (wingdi.h)

Article 02/22/2024

The **RASTERIZER_STATUS** structure contains information about whether TrueType is installed. This structure is filled when an application calls the [GetRasterizerCaps](#) function.

Syntax

C++

```
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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetRasterizerCaps](#)

Feedback

Was this page helpful?

 Yes

 No

TEXTMETRICA structure (wingdi.h)

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

C++

```
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.

 [Expand table](#)

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. For a list of possible values, see *lfCharSet* field of the [LOGFONT](#) structure.

Remarks

! Note

The wingdi.h header defines TEXTMETRIC as an alias that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Font and Text Structures](#)

[Fonts and Text Overview](#)

[GetTextExtentPoint32](#)

[GetTextMetrics](#)

[LOGFONT](#)

Last updated on 08/15/2025

TTPOLYCURVE structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[+] Expand table

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[1]

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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

TTPOLYGONHEADER structure (wingdi.h)

Article 04/02/2021

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

Syntax

C++

```
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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

WCRANGE structure (wingdi.h)

Article 02/22/2024

The **WCRANGE** structure specifies a range of Unicode characters.

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

Font and Text Macros

Article • 01/07/2021

The following macros are used with fonts and text.

[\[+\] Expand table](#)

Macro	Description
DeleteFont	Deletes a specified font.
SelectFont	Selects a font into the specified device context (DC).

Feedback

Was this page helpful?

[!\[\]\(d324a726a7e63ca3e5c64be1b394ed4d_img.jpg\) Yes](#)

[!\[\]\(092cf8e01250c2c67a0628cd80ea96b8_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

DeleteFont macro (windowsx.h)

07/09/2025

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

Syntax

C++

```
BOOL DeleteFont(  
    HFONT hfont  
);
```

Parameters

`hfont`

A handle to the font object.

Return value

Type: **BOOL**

If **DeleteFont** succeeds, the return value is nonzero. If the specified handle is not valid or is currently selected into a Device Context, the return value is zero.

Remarks

After the font object is deleted, the specified handle is no longer valid.

The **DeleteFont** macro is equivalent to calling [DeleteObject](#) as follows:

syntax

```
DeleteObject((HGDIOBJ)(HFONT)(hfont))
```

Requirements

Requirement	Value
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)

07/09/2025

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

Syntax

C++

```
HFONT SelectFont(
    HDC hdc,
    HFONT 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

Type: [HFONT](#)

If **SelectFont** succeeds, the return value is a handle to the font object being replaced. If an error occurs, the return value is **NULL**.

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:

syntax

```
((HFONT) SelectObject((hdc), (HGDIOBJ)(HFONT)(hfont)))
```

Requirements

 [Expand table](#)

Requirement	Value
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](#)

Font and Text Messages

Article • 01/07/2021

The following message is used with fonts.

[WM_FONTCHANGE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

WM_FONTCHANGE message

Article • 01/07/2021

An application sends the **WM_FONTCHANGE** message to all top-level windows in the system after changing the pool of font resources.

To send this message, call the [SendMessage](#) function with the following parameters.

C++

```
SendMessage(  
    (HWND) hWnd,  
    WM_FONTCHANGE,  
    (WPARAM) wParam,  
    (LPARAM) lParam  
);
```

Parameters

wParam

This parameter is not used.

lParam

This parameter is not used.

Remarks

An application that adds or removes fonts from the system (for example, by using the [AddFontResource](#) or [RemoveFontResource](#) function) should send this message to all top-level windows.

To send the **WM_FONTCHANGE** message to all top-level windows, an application can call the [SendMessage](#) function with the *hwnd* parameter set to **HWND_BROADCAST**.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

[Fonts and Text Overview](#)

[Font and Text Messages](#)

[AddFontResource](#)

[RemoveFontResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Font Embedding Reference

Article • 01/07/2021

The Font Embedding Services Library provides a mechanism to bundle TrueType and Microsoft OpenType fonts into a document or file. Typically, a document containing embedded fonts needs those fonts for rendering the document on another computer. Embedding a font guarantees that a font specified in a file will be present on the computer receiving the file. Some fonts, however, cannot be moved to other computers due to copyright issues limiting distribution.

The following elements are used with Font Embedding Services.

- [Font Embedding Functions](#)
- [Font Embedding Structures](#)
- [Font Embedding Error Messages](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Font Embedding Functions

Article • 01/07/2021

The following functions are used with embedded Microsoft OpenType fonts.

[+] Expand table

Function	Description
CFP_ALLOCPROC	Application-provided memory allocation function for CreateFontPackage and MergeFontPackage.
CFP_FREEPROC	Application-provided memory deallocation function for CreateFontPackage and MergeFontPackage.
CFP_REALLOCPROC	Application-provided memory reallocation function for CreateFontPackage and MergeFontPackage.
CreateFontPackage	Creates a more compact version of a specified TrueType font, in order to pass it to a printer. The resulting font may be subsetted, compressed, or both.
MergeFontPackage	Merges subset fonts created by CreateFontPackage.
READEMBEDPROC	Client-provided callback function to read stream contents from a buffer.
TTCharToUnicode	Converts an array of 8-bit character code values to 16-bit Unicode values.
TTDeleteEmbeddedFont	Releases memory used by an embedded font.
TTEmbedFont	Creates a font structure containing a subsetted wide-character (16-bit) font, using a device context as the font-embedding information source.
TTEmbedFontEx	Creates a font structure containing the subsetted UCS-4 character (32-bit) font, using a device context as the font-embedding information source.
TTEmbedFontFromFileA	Creates a font structure containing a subsetted wide-character (16-bit) font, using a file as the font-embedding information source.
TTEnableEmbeddingForFacename	Adds or removes facenames from the typeface exclusion list.
TTGetEmbeddedFontInfo	Retrieves information about an embedded font.

Function	Description
TTGetEmbeddingType	Returns embedding privileges of a font.
TTGetNewFontName	Creates a new name for an installed embedded font.
TTIsEmbeddingEnabled	Determines if the typeface exclusion list contains a specified font.
TTIsEmbeddingEnabledForFacename	Determines whether embedding is enabled for a specified font.
TTLoadEmbeddedFont	Reads the 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	UCS-4 version of TTRunValidationTests.
WRITEEMBEDPROC	Client-provided callback function to write stream contents to a buffer.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

CFP_ALLOCPROC callback function (fontsub.h)

Article 02/22/2024

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

Syntax

```
C++  
  
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

 Expand table

Requirement	Value
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)

Article 02/22/2024

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

Syntax

C++

```
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

Requirement	Value
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)

Article 02/22/2024

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

Syntax

```
C++  
  
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

 [Expand table](#)

Requirement	Value
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)

Article 10/13/2021

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

C++

```
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:

 [Expand table](#)

Value	Meaning
<code>TTFCFP_FLAGS_SUBSET</code>	If set, requests subsetting.
<code>TTFCFP_FLAGS_COMPRESS</code>	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.
<code>TTFCFP_FLAGS_TTC</code>	If set, specifies that the font in <code>puchSrcBuffer</code> is a TTC; otherwise, it must be a TTF.
<code>TTFCFP_FLAGS_GLYPHLIST</code>	If set, specifies that [*] <code>pusSubsetKeepList</code> 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.

[Expand table](#)

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:

[Expand table](#)

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 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

 Expand table

Requirement	Value
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)

Article 10/13/2021

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

C++

```
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 `*puchMergeFontBuffer`, 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.

 Expand table

Value	Meaning
<code>TTFMFP_SUBSET</code>	Copies a simple working font; see remarks below. <code>puchMergeFontBuffer</code> will be ignored; <code>puchFontPackageBuffer</code> should contain a working font created by CreateFontPackage with <code>usSubsetFormat</code> set to <code>TTFCFP_SUBSET</code> ; this working font will simply be copied to <code>ppuchDestBuffer</code> .
<code>TTFMFP_SUBSET1</code>	Turns a font package into a mergeable working font; see remarks below. <code>puchMergeFontBuffer</code> will be ignored; <code>puchFontPackageBuffer</code> should contain a mergeable working font created by CreateFontPackage with <code>usSubsetFormat</code> set to <code>TTFCFP_SUBSET1</code> . The result in <code>**ppuchDestBuffer</code> will be a working font that may be merged with later.
<code>TTFMFP_DELTA</code>	Merges a Delta font package into a mergeable working font; see remarks below. * <code>puchFontPackageBuffer</code> should contain a font package created by CreateFontPackage with <code>usSubsetFormat</code> set to

TTFCFP_DELTA and *puchMergeFontBuffer* should contain a font package created by a prior call to [MergeFontPackage](#) with *usMode* set to TTFMFP_SUBSET1 or TTFMFP_DELTA. The resulting merged font in *ppuchDestBuffer* 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:

 Expand table

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 TTFFMFP_SUBSET1 or TTFFMFP_DELTA.	Yes
------------------------	--	-----

[+] Expand table

Requirement	Value
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](#)

TTCharToUnicode function (t2embapi.h)

Article 10/13/2021

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

Syntax

C++

```
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 **pusCharCodes.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

 Expand table

Requirement	Value
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)

Article 02/22/2024

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

C++

```
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:

 [Expand table](#)

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

 Expand table

Requirement	Value
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)

Article10/13/2021

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

C++

```
LONG TTEmbedFont(
    [in]          HDC           hDC,
    [in]          ULONG         ulFlags,
    [in]          ULONG         ulCharSet,
    [out]         ULONG         *pulPrivStatus,
    [out]         ULONG         *pulStatus,
    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.

 Expand table

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] `ulCharSet`

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

 Expand table

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.

 Expand table

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 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 [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

 [Expand table](#)

Requirement	Value
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)

Article 10/13/2021

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

C++

```
LONG TTEmbedFontEx(
    [in]          HDC           hdc,
    [in]          ULONG          ulFlags,
    [in]          ULONG          ulCharSet,
    [out]         ULONG          *pulPrivStatus,
    [out]         ULONG          *pulStatus,
    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.

[] [Expand table](#)

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.

 Expand table

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.

 Expand table

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 *ulFlags* field does not specify TTEMBED_SUBSET.

`[in, optional] pTTEmbbedInfo`

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

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 [TTEmbedFont](#); for information on embedding Unicode characters from a file, see [TTEmbedFontFromFileA](#).

Requirements

 Expand table

Requirement	Value
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](#)

[TTEmbedFont](#)

[TTEmbedFontFromFileA](#)

[TTLoadEmbeddedFont](#)

TTEmbedFontFromFileA function (t2embapi.h)

Article10/13/2021

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

C++

```
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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

Value	Meaning
EMBED_PREVIEWPRINT	Preview and Print Embedding.
EMBED_EDITABLE	Editable Embedding.
EMBED_INSTALLABLE	Installable 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 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 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. *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

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

 [Expand table](#)

Requirement	Value
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](#)

[TTEmbedFont](#)

[TTEmbedFontEx](#)

[TTLoadEmbeddedFont](#)

TTEnableEmbeddingForFacename function (t2embapi.h)

Article10/13/2021

Adds or removes facenames from the typeface exclusion list.

Syntax

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
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

[TTIIsEmbeddingEnabled](#)

[TTIIsEmbeddingEnabledForFacename](#)

TTGetEmbeddedFontInfo function (t2embapi.h)

Article10/13/2021

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

C++

```
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.

 [Expand table](#)

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.

[out] pulPrivStatus

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

 Expand table

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.

 Expand table

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:

 Expand table

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 `*pulPrivStatus` identifies embedding privileges of the font. The location referenced by `*pulStatus` identifies whether a subset of the font is embedded.

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

Requirements

 Expand table

Requirement	Value
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)

Article02/22/2024

Obtains the embedding privileges of a font.

Syntax

C++

```
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.

 Expand table

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 *pulPrivStatus*.

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 *pulPrivStatus* for success or failure of the function.

Requirements

 Expand table

Requirement	Value
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)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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)

Article 02/22/2024

Determines whether the typeface exclusion list contains a specified font.

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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)

Article 02/22/2024

Determines whether embedding is enabled for a specified font.

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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)

Article 10/13/2021

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

Syntax

C++

```
LONG TTLoadEmbeddedFont(
    [out]          HANDLE      *phFontReference,
    [in]           ULONG       ulFlags,
    [out]          ULONG       *pulPrivStatus,
    [in]           ULONG       ulPrivs,
    [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:

 Expand table

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.

[Expand table](#)

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] ulPrivils

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.

[Expand table](#)

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.

 Expand table

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] lpfnReadStream

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_FONTPNAMEALREADYEXISTS), and if that font has the same checksum as the embedded font (E_FONTPALREADYEXISTS). 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

[] [Expand table](#)

Requirement	Value
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

Requirement	Value
DLL	T2embed.dll

See also

[TTDeleteEmbeddedFont](#)

[TTEMBEDINFO](#)

[TTGetEmbeddingType](#)

[TTGetNewFontName](#)

[TTLOADINFO](#)

TTRunValidationTests function (t2embapi.h)

Article 02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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)

Article 02/22/2024

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

C++

```
LONG TTRunValidationTestsEx(
    [in] HDC                  hDC,
    [in] TTVALIDATIONTESTSPARAMSEX *pTestParam
);
```

Parameters

[in] `hDC`

Device context handle.

[in] `pTestParam`

Pointer to a [TTVALIDATIONTESTPARAMSEX](#) 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

[+] Expand table

Requirement	Value
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](#)

Font Embedding Services Structures

Article • 01/07/2021

The following structures are used with embedded Microsoft OpenType fonts.

[TTEMBEDINFO](#)

[TTLOADINFO](#)

[TTVALIDATIONTESTPARAMS](#)

[TTVALIDATIONTESTPARAMSEX](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

TTEMPBEDINFO structure (t2embapi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

TTLOADINFO structure (t2embapi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

TTVALIDATIONTESTSPARAMS structure (t2embapi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[Expand table](#)

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

[Expand table](#)

Requirement	Value
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)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

 Expand table

Requirement	Value
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](#)

Font-Embedding Function Error Messages

Article • 01/07/2021

The following LONG values are returned by the font-embedding functions when errors are encountered. When functions are successful, the value E_NONE is returned.

[+] Expand table

Return value	Description
E_NONE	No error.
E_ADDFONTFAILED	An error occurred when the load functions tried to add the new font using AddFontResource .
E_CHARCODECOUNTINVALID	The count of subsetted characters specified in TTEmbedFont is invalid.
E_CHARCODESETINVALID	The character set specified in TTEmbedFont is invalid.
E_COULDNTCREATETEMPFILE	The load functions could not create a temporary file needed in to install a new font or resource file.
E_DEVICETRUETYPEFONT	The specified TrueType® font is not a system font. The font may exist as a device font in a printer.
E_ERRORACCESSINGEXCLUDELIST	An error occurred while attempting to access the Typeface Exclusion List.
E_ERRORACCESSINGFACENAME	A non-DC-related error was encountered while trying to allocate an OUTLINETEXTMETRIC structure.
E_ERRORACCESSINGFONTPDATA	An error was encountered while attempting to use GetFontData .
E_ERRORCOMPRESSINGFONTPDATA	An error occurred while TTEmbedFont attempted to compress the font data.
E_ERRORCONVERTINGCHARS	An error prevented the conversion of a string of single-byte characters to Unicode characters. This can occur in TTCharToUnicode if either <i>pucCharCodes</i> or <i>pusShortCodes</i> are non-null values, or if the conversion fails while using MultiByteToWideChar .
E_ERRORCREATINGFONTPFILE	An error occurred while attempting to create the font file.

Return value	Description
E_ERRORDECOMPRESSINGFONTDATA	An error occurred while trying to decompress data in a font file.
E_ERROREXPANDINGFONTDATA	An error occurred while the load functions attempted to expand embedded, compressed font data.
E_ERRORGETTINGDC	An error occurred while trying to allocate a DC, halting processing.
E_ERRORREADINGFONTDATA	An error occurred while attempting to read font data.
E_ERRORUNICODECONVERSION	An error occurred while allocating memory to convert a name string to Unicode.
E_ERRORUSINGTEMPFILE	An error occurred while the load functions were using a temporary file to install a new font file or resource file.
E_EXCEPTION	An exception was thrown by an unknown cause.
E_FACENAMEINVALID	A null <i>szFaceName</i> parameter was passed to the function.
E_FLAGSINVALID	The <i>ulFlags</i> parameter in the current function is invalid.
E_FONTALREADYEXISTS	The embedded font has the same name and checksum as a font already installed on the system.
E_FONTDATAINVALID	Font data read from disk is not a valid embedded-font structure.
E_FONTFILECREATEFAILED	The load functions could not create the font file (.ttf)
E_FONTFILENOTFOUND	The font file of the specified file name does not exist.
E_FONTINSTALLFAILED	An attempt to install the embedded font in the system failed.
E_FONTNAMEALREADYEXISTS	The embedded font has the same name but a different checksum as a font already installed.
E_FONTNOTEMBEDDABLE	The specified font cannot be embedded due to restrictions from the font manufacturer. Embedding this font in a document violates copyright laws.
E_FONTREFERENCEINVALID	A null <i>phFontReference</i> was passed to the function.
E_HDCINVALID	The device context specified for the TTEmbedFont function is invalid.
E_NAMECHANGEFAILED	TTLoadEmbeddedFont was unable to change the name of the font being loaded.

Return value	Description
E_NOFREEMEMORY	An internal operation failed while attempting to allocate memory.
E_NOOS2	An OS/2 table was not found in the font.
E_NOTATRUETYPEFONT	The specified font is not a TrueType font.
E_PBENABLEDINVALID	A null <i>pbEnabled</i> parameter was passed to the function.
E_PERMISSIONSINVALID	A null <i>pulPermissions</i> parameter was passed to the function.
E_PRIVSINVALID	The <i>ulPrivs</i> parameter specified in the load functions is invalid.
E_PRIVSTATUSINVALID	A null <i>pulPrivStatus</i> parameter was passed to the function.
E_READFROMSTREAMFAILED	An error occurred while attempting to read the embedded font structure from the stream.
E_RESOURCEFILECREATEFAILED	The load functions could not create the font resource file (.fot).
E_SAVETOSTREAMFAILED	An error occurred while attempting to save the embedded-font structure to a stream.
E_STATUSINVALID	A null <i>pulStatus</i> parameter was passed to the function.
E_STREAMINVALID	The stream specified in TTEmbedFont or the load functions is invalid.
E_SUBSETTINGFAILED	TTEmbedFont failed while attempting to create a subset of a font.
E_T2NOFREEMEMORY	An error occurred while attempting to free memory. The memory in question failed during the free operation.
E_WINDOWSAPI	An internal error occurred when one of the functions called a Windows API, such as GetTextMetrics or GetOutlineTextMetrics .
E_API_NOTIMPL	This API function is not implemented in the version of Windows on which it is running.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Font Package Function Error Messages

Article • 01/07/2021

The following LONG values are returned by the font package functions ([CreateFontPackage](#) and [MergeFontPackage](#)) when errors are encountered. When functions are successful, the value NO_ERROR is returned.

[+] [Expand table](#)

Return value	Value	Description
NO_ERROR	0	No error occurred.
ERR_FORMAT	1006	An input data format error occurred.
ERR_GENERIC	1000	An error occurred in generic code.
ERR_MEM	1005	An error occurred during memory allocation.
ERR_NO_GLYPHS	1009	No glyphs were found.
ERR_INVALID_BASE	1085	The font contained an invalid baseline data (BASE) table. Currently this value is not used.
ERR_INVALID_CMAP	1030	The font contained an invalid character-to-glyph mapping (cmap) table.
ERR_INVALID_DELTA_FORMAT	1013	An invalid delta format was detected while trying to subset a format 1 or 2 font.
ERR_INVALID_EBLC	1086	The font contained an invalid embedded bitmap location data (EBLC) table.
ERR_INVALID_GLYF	1061	The font contained an invalid glyph data (glyf) table.
ERR_INVALID_GDEF	1083	The font contained an invalid glyph definition data (GDEF) table. Currently this value is not used.
ERR_INVALID_GPOS	1082	The font contained an invalid glyph positioning data (GPOS) table. Currently this value is not used.
ERR_INVALID_GSUB	1081	The font contained an invalid glyph substitution data (GSUB) table.
ERR_INVALID_HDMX	1089	The font contained an invalid horizontal device metrics (hdmx) table.

Return value	Value	Description
ERR_INVALID_HEAD	1062	The font contained an invalid font header (head) table.
ERR_INVALID_HHEA	1063	The font contained an invalid horizontal header (hhea) table.
ERR_INVALID_HHEA_OR_VHEA	1072	The font contained an invalid horizontal header (hhea) table or an invalid vertical metrics header (vhea) table.
ERR_INVALID_HMTX	1064	The font contained an invalid horizontal metrics (hmtx) table.
ERR_INVALID_HMTX_OR_VMTX	1073	The font contained an invalid horizontal metrics (hmtx) table or an invalid vertical metrics (vmtx) table.
ERR_INVALID_JSTF	1084	The font contained an invalid justification data (JSTF) table.
ERR_INVALID_LTSH	1087	The font contained an invalid linear threshold data (LTSH) table.
ERR_INVALID_TTO	1080	The font was an invalid TrueType Open font.
ERR_INVALID_VDMX	1088	The font contained an invalid vertical device metrics (VDMX) table.
ERR_INVALID_LOCA	1065	The font contained an invalid index to location (loca) table.
ERR_INVALID_MAXP	1066	The font contained an invalid maximum profile (maxp) table.
ERR_INVALID_MERGE_CHECKSUMS	1011	An attempt to merge checksums for two fonts from a different mother font was unsuccessful.
ERR_INVALID_MERGE_FORMATS	1010	An attempt to merge fonts with the wrong dttf formats was unsuccessful.
ERR_INVALID_MERGE_NUMGLYPHS	1012	An attempt to merge the number of glyphs for two fonts from a different mother font was unsuccessful.
ERR_INVALID_NAME	1067	The font package name or a font name was invalid.
ERR_INVALID_POST	1068	The font contained an invalid PostScript information (post) table.

Return value	Value	Description
ERR_INVALID_OS2	1069	The font contained an invalid OS/2 and Windows-specific metrics (OS/2) table.
ERR_INVALID_VHEA	1070	The font contained an invalid vertical metrics header (vhea) table.
ERR_INVALID_VMTX	1071	The font contained an invalid vertical metrics (vmtx) table.
ERR_INVALID_TTC_INDEX	1015	An invalid zero-based (TTC) index into the font file was passed.
ERR_MISSING_CMAP	1030	The font did not contain a cmap table.
ERR_MISSING_EBDT	1044	The font did not contain an EBDT table.
ERR_MISSING_GLYF	1031	The font did not contain a glyf table.
ERR_MISSING_HEAD	1032	The font did not contain a head table.
ERR_MISSING_HHEA	1033	The font did not contain an hhea table.
ERR_MISSING_HMTX	1034	The font did not contain an hmtx table.
ERR_MISSING_LOCA	1035	The font did not contain a loca table.
ERR_MISSING_MAXP	1036	The font did not contain a maxp table.
ERR_MISSING_NAME	1037	The font did not contain a naming (name) table.
ERR_MISSING_POST	1038	The font did not contain a post table.
ERR_MISSING_OS2	1039	The font did not contain an OS/2 table.
ERR_MISSING_VHEA	1040	The font did not contain a vhea table.
ERR_MISSING_VMTX	1041	The font did not contain a vmtx table.
ERR_MISSING_HHEA_OR_VHEA	1042	The font did not contain an hhea table or a vhea table.
ERR_MISSING_HMTX_OR_VMTX	1043	The font did not contain an hmtx table or a vmtx table.
ERR_NOT_TTC	1014	The provided value was not an index for a TTC file.
ERR_PARAMETER0	1100	Calling function parameter 0 was invalid.
ERR_PARAMETER1	1101	Calling function parameter 1 was invalid.
ERR_PARAMETER2	1102	Calling function parameter 2 was invalid.

Return value	Value	Description
ERR_PARAMETER3	1103	Calling function parameter 3 was invalid.
ERR_PARAMETER4	1104	Calling function parameter 4 was invalid.
ERR_PARAMETER5	1105	Calling function parameter 5 was invalid.
ERR_PARAMETER6	1106	Calling function parameter 6 was invalid.
ERR_PARAMETER7	1107	Calling function parameter 7 was invalid.
ERR_PARAMETER8	1108	Calling function parameter 8 was invalid.
ERR_PARAMETER9	1109	Calling function parameter 9 was invalid.
ERR_PARAMETER10	1110	Calling function parameter 10 was invalid.
ERR_PARAMETER11	1111	Calling function parameter 11 was invalid.
ERR_PARAMETER12	1112	Calling function parameter 12 was invalid.
ERR_PARAMETER13	1113	Calling function parameter 13 was invalid.
ERR_PARAMETER14	1114	Calling function parameter 14 was invalid.
ERR_PARAMETER15	1115	Calling function parameter 15 was invalid.
ERR_PARAMETER16	1116	Calling function parameter 16 was invalid.
ERR_READCONTROL	1003	The read control structure did not match data.
ERR_READOUTOFCOMMANDS	1001	A read from memory was not allowed, possibly because data was out of bounds or corrupt.
ERR_VERSION	1008	Major dttf.version value of the input data was greater than the version the function can read.
ERR_WOULD_GROW	1007	The requested action caused data to grow and the application must use original data.
ERR_WRITECONTROL	1004	The write control structure did not match data.
ERR_WRITEOUTOFCOMMANDS	1002	A write to memory was not allowed, possibly because data was out of bounds.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Lines and Curves

Article • 01/07/2021

Lines and curves are used to draw graphics output on raster devices. As discussed in this overview, a *line* is a set of highlighted pixels on a raster display (or a set of dots on a printed page) identified by two points: a starting point and an ending point. A *regular curve* is a set of highlighted pixels on a raster display (or dots on a printed page) that defines the perimeter (or part of the perimeter) of a conic section. An *irregular curve* is a set of pixels that defines a curve that does not fit the perimeter of a conic section.

- [About Lines and Curves](#)
- [Using Lines and Curves](#)
- [Line and Curve Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Lines and Curves

Article • 01/07/2021

Many types of applications use lines and curves to draw graphics output on raster devices. Computer-aided design (CAD) and drawing applications use lines and curves to outline objects, specify the centers of objects, the dimensions of objects, and so on. Spreadsheet applications use lines and curves to draw grids, charts, and graphs. Word processing applications use lines to create rules and borders on a page of text.

- [Lines](#)
- [Curves](#)
- [Combined Lines and Curves](#)
- [Line and Curve Attributes](#)

Feedback

Was this page helpful?



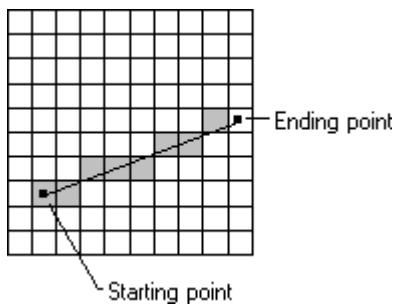
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Lines

Article • 01/07/2021

A line is a set of highlighted pixels on a raster display (or a set of dots on a printed page) identified by two points: a starting point and an ending point. The pixel located at the starting point is always included in the line, and the pixel located at the ending point is always excluded. (This kind of line is sometimes called inclusive-exclusive.)

When an application calls one of the line-drawing functions, graphics device interface (GDI), or in some cases a device driver, determines which pixels should be highlighted. GDI is a dynamic-link library (DLL) that processes graphics function calls from an application and passes those calls to a device driver. A device driver is a DLL that receives input from GDI, converts the input to device commands, and passes those commands to the appropriate device. GDI uses a digital differential analyzer (DDA) to determine the set of pixels that define a line. A DDA determines the set of pixels by examining each point on the line and identifying those pixels on the display surface (or dots on a printed page) that correspond to the points. The following illustration shows a line, its starting point, its ending point, and the pixels highlighted by using a simple DDA.



The simplest and most common DDA is the Bresenham, or incremental, DDA. A modified version of this algorithm draws lines in Windows. The incremental DDA is noted for its simplicity, but it is also noted for its inaccuracy. Because it rounds off to the nearest integer value, it sometimes fails to represent the original line requested by the application. The DDA used by GDI does not round off to the nearest integer. As a result, this new DDA produces output that is sometimes much closer in appearance to the original line requested by the application.

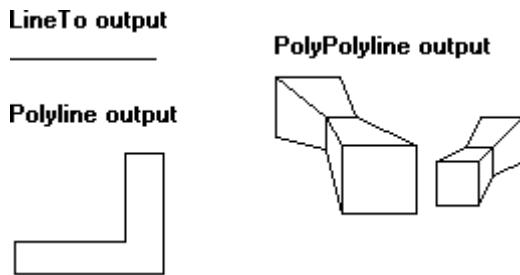
ⓘ Note

If an application requires line output that cannot be achieved with the new DDA, it can draw its own lines by calling the [LineDDA](#) function and supplying a private DDA ([LineDDAProc](#)). However, the [LineDDA](#) function draws lines much slower than the

line-drawing functions. Do not use this function within an application if speed is a primary concern.

An application can use the new DDA to draw single lines and multiple, connected line segments. An application can draw a single line by calling the [LineTo](#) function. This function draws a line from the current position up to, but not including, a specified ending point. An application can draw a series of connected line segments by calling the [Polyline](#) function, supplying an array of points that specify the ending point of each line segment. An application can draw multiple, disjointed series of connected line segments by calling the [PolyPolyline](#) function, supplying the required ending points.

The following illustration shows line output created by calling the [LineTo](#), [Polyline](#), and [PolyPolyline](#) functions.



Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

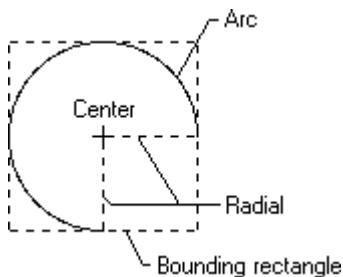
Curves

Article • 01/07/2021

A regular curve is a set of highlighted pixels on a raster display (or dots on a printed page) that define the perimeter (or part of the perimeter) of a conic section. An irregular curve is a set of pixels that define a curve that does not fit the perimeter of a conic section. The ending point is excluded from a curve just as it is excluded from a line.

When an application calls one of the curve-drawing functions, GDI breaks the curve into a number of extremely small, discrete line segments. After determining the endpoints (starting point and ending point) for each of these line segments, GDI determines which pixels (or dots) define each line by applying its DDA.

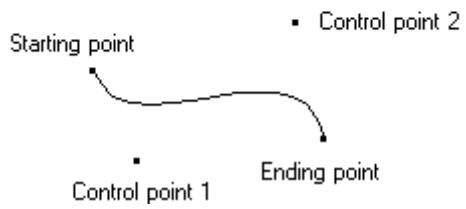
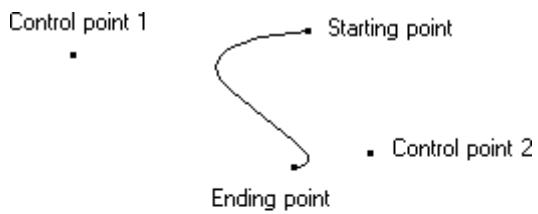
An application can draw an ellipse or part of an ellipse by calling the [Arc](#) function. This function draws the curve within the perimeter of an invisible rectangle called a bounding rectangle. The size of the ellipse is specified by two invisible radials extending from the center of the rectangle to the sides of the rectangle. The following illustration shows an arc (part of an ellipse) drawn by using the [Arc](#) function.



When calling the [Arc](#) function, an application specifies the coordinates of the bounding rectangle and radials. The preceding illustration shows the rectangle and radials with dashed lines while the actual arc was drawn using a solid line.

When drawing the arc of another object, the application can call the [SetArcDirection](#) and [GetArcDirection](#) functions to control the direction (clockwise or counterclockwise) in which the object is drawn. The default direction for drawing arcs and other objects is counterclockwise.

In addition to drawing ellipses or parts of ellipses, applications can draw irregular curves called Bézier curves. A *Bézier curve* is an irregular curve whose curvature is defined by four control points (p_1 , p_2 , p_3 , and p_4). The control points p_1 and p_4 define the starting and ending points of the curve, and the control points p_2 and p_3 define the shape of the curve by marking points where the curve reverses orientation, as shown in the following diagram.



An application can draw irregular curves by calling the [PolyBezier](#) function, supplying the appropriate control points.

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Combined Lines and Curves

Article • 01/07/2021

In addition to drawing lines or curves, applications can draw combinations of line and curve output by calling a single function. For example, an application can draw the outline of a pie chart by calling the [AngleArc](#) function.

The [AngleArc](#) function draws an arc along a circle's perimeter and draws a line connecting the starting point of the arc to the circle's center. In addition to using the [AngleArc](#) function, an application can also combine line and irregular curve output by using the [PolyDraw](#) function.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Line and Curve Attributes

Article • 01/07/2021

A device context (DC) contains attributes that affect line and curve output. The *line and curve attributes* include the current position, brush style, brush color, pen style, pen color, transformation, and so on.

The default current position for any DC is located at the point (0,0) in logical (or world) space. You can set these coordinates to a new position by calling the [MoveToEx](#) function and passing a new set of coordinates.

ⓘ Note

There are two sets of line- and curve-drawing functions. The first set retains the current position in a DC, and the second set alters the position. You can identify the functions that alter the current position by examining the function name. If the function name ends with the preposition "To", the function sets the current position to the ending point of the last line drawn ([LineTo](#), [ArcTo](#), [PolylineTo](#), or [PolyBezierTo](#)). If the function name does not end with this preposition, it leaves the current position intact ([Arc](#), [Polyline](#), or [PolyBezier](#)).

The default brush is a solid white brush. An application can create a new brush by calling the [CreateBrushIndirect](#) function. After creating a brush, the application can select it into its DC by calling the [SelectObject](#) function. Windows provides a complete set of functions to create, select, and alter the brush in an application's DC. For more information about these functions and about brushes in general, see [Brushes](#).

The default pen is a cosmetic, solid black pen that is one pixel wide. An application can create a pen by using the [ExtCreatePen](#) function. After creating a pen, your application can select it into its DC by calling the [SelectObject](#) function. Windows provides a complete set of functions to create, select, and alter the pen in an application's DC. For more information about these functions and about pens in general, see [Pens](#).

The default transformation is the unity transformation (specified by the identity matrix). An application can specify a new transformation by calling the [SetWorldTransform](#) function. Windows provides a complete set of functions to transform lines and curves by altering their width, location, and general appearance. For more information about these functions, see [Coordinate Spaces and Transformations](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Lines and Curves

Article • 01/07/2021

You can use the line and curve functions to draw virtually any shape or object in the client area of an application window. This section illustrates how these functions can be used to draw markers or a pie chart.

- [Drawing markers](#)
- [Drawing a pie chart](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing Markers

Article • 01/07/2021

You can use the line functions to draw markers. A marker is a symbol centered over a point. Drawing applications use markers to designate starting points, ending points, and control points. Spreadsheet applications use markers to designate points of interest on a chart or graph.

In the following code sample, the application-defined Marker function creates a marker by using the [MoveToEx](#) and [LineTo](#) functions. These functions draw two intersecting lines, 20 pixels in length, centered over the cursor coordinates.

C++

```
void Marker(LONG x, LONG y, HWND hwnd)
{
    HDC hdc;

    hdc = GetDC(hwnd);
    MoveToEx(hdc, (int) x - 10, (int) y, (LPPOINT) NULL);
    LineTo(hdc, (int) x + 10, (int) y);
    MoveToEx(hdc, (int) x, (int) y - 10, (LPPOINT) NULL);
    LineTo(hdc, (int) x, (int) y + 10);

    ReleaseDC(hwnd, hdc);
}
```

The system stores the coordinates of the cursor in the *lParam* parameter of the [WM_LBUTTONDOWN](#) message when the user presses the left mouse button. The following code demonstrates how an application gets these coordinates, determines whether they lie within its client area, and passes them to the Marker function to draw the marker.

C++

```
// Line- and arc-drawing variables

static BOOL bCollectPoints;
static POINT ptMouseDown[32];
static int index;
POINTS ptTmp;
RECT rc;

case WM_LBUTTONDOWN:

    if (bCollectPoints && index < 32)
```

```
{  
    // Create the region from the client area.  
  
    GetClientRect(hwnd, &rc);  
    hrgn = CreateRectRgn(rc.left, rc.top,  
        rc.right, rc.bottom);  
  
    ptTmp = MAKEPOINTS((POINTS FAR *) lParam);  
    ptMouseDown[index].x = (LONG) ptTmp.x;  
    ptMouseDown[index].y = (LONG) ptTmp.y;  
  
    // Test for a hit in the client rectangle.  
  
    if (PtInRegion(hrgn, ptMouseDown[index].x,  
        ptMouseDown[index].y))  
    {  
        // If a hit occurs, record the mouse coords.  
  
        Marker(ptMouseDown[index].x, ptMouseDown[index].y,  
            hwnd);  
        index++;  
    }  
}  
break;
```

Feedback

Was this page helpful?

 Yes

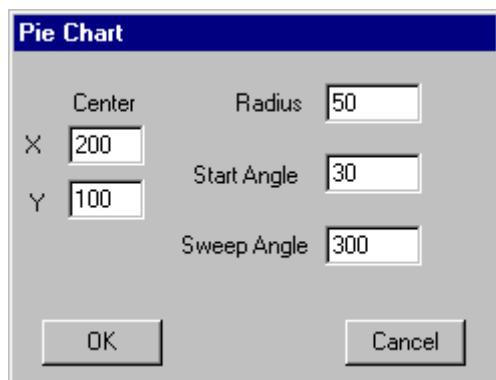
 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

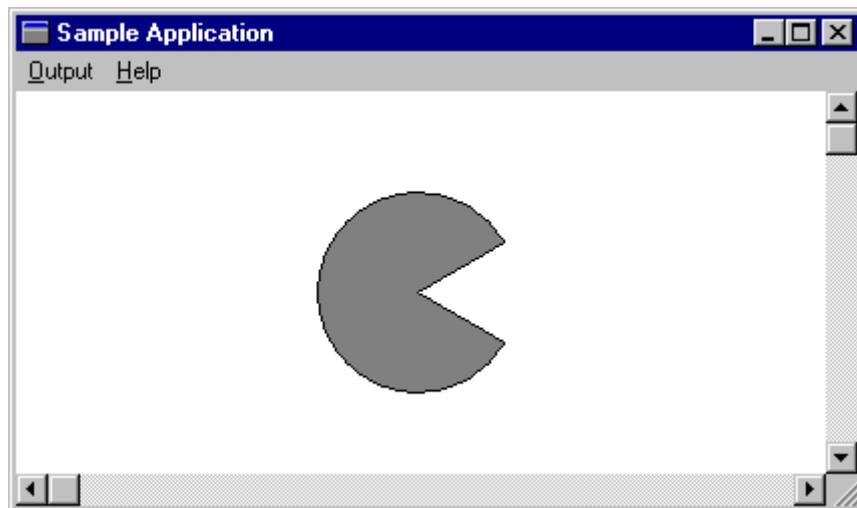
Drawing a Pie Chart

Article • 01/07/2021

You can use the line and curve functions to draw a pie chart. The primary function used to draw pie charts is the [AngleArc](#) function, which requires you to supply the coordinates of the center of the pie, the radius of the pie, a start angle, and a sweep angle. The following screen shot shows a dialog box that the user can use to enter these values.



The values shown above produce the following pie chart.



The dialog box template found in the application's resource script (.RC) file specifies characteristics of the preceding dialog box (its height, the controls it contains, and its style), as follows.

C++

```
AngleArc DIALOG 6, 18, 160, 100
STYLE WS_DLGFREAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Pie Chart"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT    IDD_X, 18, 22, 25, 12, ES_AUTOHSCROLL
    LTEXT      "X", 102, 4, 24, 9, 8
```

```

EDITTEXT    IDD_Y, 18, 39, 25, 12, ES_AUTOHSCROLL
LTEXT      "Y", 104, 5, 42, 12, 8
LTEXT      "Center", 105, 19, 11, 23, 8
EDITTEXT    IDD_RADIUS, 103, 9, 32, 12, ES_AUTOHSCROLL
EDITTEXT    IDD_STARTANGLE, 103, 31, 32, 12, ES_AUTOHSCROLL
EDITTEXT    IDD_SWEEPANGLE, 103, 53, 32, 12, ES_AUTOHSCROLL
LTEXT      "Radius", 109, 73, 11, 25, 8
LTEXT      "Start Angle", 110, 59, 33, 42, 8
LTEXT      "Sweep Angle", 111, 55, 55, 43, 8
PUSHBUTTON "OK", IDD_OK, 9, 82, 40, 14
PUSHBUTTON "Cancel", IDD_CANCEL, 110, 82, 40, 14
END

```

The dialog box procedure, found in the application's source file, retrieves data (center coordinates, arc radius, and start and sweep angles) by following these steps:

1. The application-defined ClearBits function initializes the array that receives the user-input to zero.
2. The application-defined GetStrLength function retrieves the length of the string entered by the user.
3. The application-defined RetrieveInput function retrieves the value entered by the user.

The following sample code shows the dialog box procedure.

C++

```

void ClearBits(LPTSTR, int);
int GetStrLength(LPTSTR);
DWORD RetrieveInput(LPTSTR, int);

BOOL CALLBACK ArcDlgProc(HWND hdlg, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
    CHAR chInput[4];      // receives control-window input
    int cch;              // array-size and count variable

    switch (uMsg)
    {
        case WM_INITDIALOG:
            return FALSE;

        case WM_COMMAND:
            switch (wParam)
            {
                // If the user pressed the OK button, retrieve the
                // data that was entered in the various AngleArc
                // controls.

                case IDD_OK:
                    // Retrieve the x-coordinate of the arc's center.

```

```

        ClearBits(chInput, sizeof(chInput));
        GetDlgItemText(hdlg, IDD_X, chInput,
                      sizeof(chInput));
        cch = GetStrLength(chInput);
        nX = (int)RetrieveInput(chInput, cch);

        // Retrieve the y-coordinate of the arc's center.

        ClearBits(chInput, sizeof(chInput));
        GetDlgItemText(hdlg, IDD_Y, chInput,
                      sizeof(chInput));
        cch = GetStrLength(chInput);
        nY = (int)RetrieveInput(chInput, cch);

        // Retrieve the radius of the arc.

        ClearBits(chInput, sizeof(chInput));
        GetDlgItemText(hdlg, IDD_RADIUS, chInput,
                      sizeof(chInput));
        cch = GetStrLength(chInput);
        dwRadius = (DWORD) RetrieveInput(chInput, cch);

        // Retrieve the start angle.

        ClearBits(chInput, sizeof(chInput));
        GetDlgItemText(hdlg, IDD_STARTANGLE, chInput,
                      sizeof(chInput));
        cch = GetStrLength(chInput);
        xStartAngle = (float) RetrieveInput(chInput, cch);

        // Retrieve the sweep angle.

        ClearBits(chInput, sizeof(chInput));
        GetDlgItemText(hdlg, IDD_SWEEPANGLE, chInput,
                      sizeof(chInput));
        cch = GetStrLength(chInput);
        xSweepAngle = (float) RetrieveInput(chInput, cch);

        EndDialog(hdlg, FALSE);
        return TRUE;

        // If user presses the CANCEL button, close the
        // dialog box.

        case IDD_CANCEL:
            EndDialog(hdlg, FALSE);
            return TRUE;
    } // end switch (wParam)

    break;

    default:
        return FALSE;
} // end switch (message)

```

```

    UNREFERENCED_PARAMETER(lParam);
}

void ClearBits(LPTSTR cArray, int iLength)
{
    int i;

    for (i = 0; i < iLength; i++)
        cArray[i] = 0;
}

int GetStrLength(LPTSTR cArray)
{
    int i = 0;

    while (cArray[i++] != 0);
    return i - 1;
}

DWORD RetrieveInput(LPTSTR cArray, int iLength)
{
    int i, iTmp;
    double dVal, dCount;

    dVal = 0.0;
    dCount = (double) (iLength - 1);

    // Convert ASCII input to a floating-point value.

    for (i = 0; i < iLength; i++)
    {
        iTmp = cArray[i] - 0x30;
        dVal = dVal + (((double)iTmp) * pow(10.0, dCount--));
    }

    return (DWORD) dVal;
}

```

To draw each section of the pie chart, pass the values entered by the user to the [AngleArc](#) function. To fill the pie chart using the current brush, embed the call to [AngleArc](#) in a path bracket. The following code sample shows the defined path bracket and the call to [AngleArc](#).

C++

```

int nX;
int nY;
DWORD dwRadius;
float xStartAngle;
float xSweepAngle;

```

```
hdc = GetDC(hwnd);
BeginPath(hdc);
SelectObject(hdc, GetStockObject(GRAY_BRUSH));
MoveToEx(hdc, nX, nY, (LPPOINT) NULL);
AngleArc(hdc, nX, nY, dwRadius, xStartAngle, xSweepAngle);
LineTo(hdc, nX, nY);
EndPath(hdc);
StrokeAndFillPath(hdc);
ReleaseDC(hwnd, hdc);
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Line and Curve Reference

Article • 01/07/2021

The following elements are used with lines and curves.

- Line and Curve Functions

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Line and Curve Functions

Article • 01/07/2021

The following functions are used with lines and curves.

[+] Expand table

Function	Description
AngleArc	Draws a line segment and an arc.
Arc	Draws an elliptical arc.
ArcTo	Draws an elliptical arc.
GetArcDirection	Retrieves the current arc direction for the specified device context.
LineDDA	Determines which pixels should be highlighted for a line defined by the specified starting and ending points.
LineDDAProc	An application-defined callback function used with the LineDDA function.
LineTo	Draws a line from the current position up to, but not including, the specified point.
MoveToEx	Updates the current position to the specified point and optionally returns the previous position.
PolyBezier	Draws one or more Bézier curves.
PolyBezierTo	Draws one or more Bézier curves.
PolyDraw	Draws a set of line segments and Bézier curves.
Polyline	Draws a series of line segments by connecting the points in the specified array.
PolylineTo	Draws one or more straight lines.
PolyPolyline	Draws multiple series of connected line segments.
SetArcDirection	Sets the drawing direction to be used for arc and rectangle functions.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

AngleArc function (wingdi.h)

Article 02/22/2024

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

C++

```
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

[] [Expand table](#)

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[Arc](#)

[ArcTo](#)

[Line and Curve Functions](#)

[Lines and Curves Overview](#)

[MoveToEx](#)

Arc function (wingdi.h)

Article 02/22/2024

The **Arc** function draws an elliptical arc.

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

The **ArcTo** function draws an elliptical arc.

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetArcDirection function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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:

[] [Expand table](#)

Value	Meaning
AD_COUNTERCLOCKWISE	Arcs and rectangles are drawn counterclockwise.
AD_CLOCKWISE	Arcs and rectangles are drawn clockwise.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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](#)

LineDDA function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

02/22/2024

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

C++

```
LINEDDAPROC Lineddaprof;  
  
VOID Lineddaprof(  
    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

Requirement	Value
Minimum supported client	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)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

MoveToEx function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PolyBezier function (wingdi.h)

Article11/19/2022

The **PolyBezier** function draws one or more Bézier curves.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

The **PolyBezierTo** function draws one or more Bézier curves.

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/19/2022

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

Syntax

C++

```
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 *lppt* array is used. This parameter can be one of the following values.

 Expand table

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.

 Expand table

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 *lppt* 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 *lpp* 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Polyline function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article11/19/2022

The **PolylineTo** function draws one or more straight lines.

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PolyPolyline function (wingdi.h)

Article02/22/2024

The **PolyPolyline** function draws multiple series of connected line segments.

Syntax

C++

```
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 *lppt* 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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetArcDirection function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Metafiles (Windows GDI)

Article • 01/07/2021

A *metafile* is a collection of structures that store a picture in a device-independent format. Device independence is the one feature that sets metafiles apart from bitmaps. Unlike a bitmap, a metafile guarantees device independence. There is a drawback to metafiles however, they are generally drawn more slowly than bitmaps. Therefore, if an application requires fast drawing and device independence is not an issue, it should use bitmaps instead of metafiles.

- [About Metafiles](#)
- [Using Metafiles](#)
- [Metafile Reference](#)

For information about bitmaps, see [Bitmaps](#).

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Metafiles

Article • 01/07/2021

Internally, a metafile is an array of variable-length structures called *metafile records*. The first records in the metafile specify general information such as the resolution of the device on which the picture was created, the dimensions of the picture, and so on. The remaining records, which constitute the bulk of any metafile, correspond to the graphics device interface (GDI) functions required to draw the picture. These records are stored in the metafile after a special metafile device context is created. This metafile device context is then used for all drawing operations required to create the picture. When the system processes a GDI function associated with a metafile DC, it converts the function into the appropriate data and stores this data in a record appended to the metafile.

After a picture is complete and the last record is stored in the metafile, you can pass the metafile to another application by:

- Using the clipboard
- Embedding it within another file
- Storing it on disk
- Playing it repeatedly

A metafile is *played* when its records are converted to device commands and processed by the appropriate device.

There are two types of metafiles:

- Enhanced-format metafiles
- Windows-format metafiles

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enhanced-Format Metafiles

Article • 01/07/2021

The *enhanced-format metafile* consists of the following elements:

- A header
- A table of handles to GDI objects
- A private palette
- An array of metafile records

Enhanced metafiles provide true device independence. You can think of the picture stored in an enhanced metafile as a "snapshot" of the video display taken at a particular moment. This "snapshot" maintains its dimensions no matter where it appears on a printer, a plotter, the desktop, or in the client area of any application.

You can use enhanced metafiles to store a picture created by using the GDI functions (including new path and transformation functions). Because the enhanced metafile format is standardized, pictures that are stored in this format can be copied from one application to another; and, because the pictures are truly device independent, they are guaranteed to maintain their shape and proportion on any output device.

- [Enhanced Metafile Records](#)
- [Enhanced Metafile Creation](#)
- [Enhanced Metafile Operations](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enhanced Metafile Records

Article • 01/07/2021

An enhanced metafile is an array of records. A metafile record is a variable-length [ENHMETARECORD](#) structure. At the beginning of every enhanced metafile record is an [EMR](#) structure, which contains two members. The first member, iType, identifies the record type that is, the GDI function whose parameters are contained in the record. Because the structures are variable in length, the other member, nSize, contains the size of the record. Immediately following the nSize member are the remaining parameters, if any, of the GDI function. The remainder of the structure contains additional data that is dependent on the record type.

The first record in an enhanced metafile is always the [ENHMETAHEADER](#) structure, which is the enhanced-metafile header. The header specifies the following information:

- Size of the metafile, in bytes
- Dimensions of the picture frame, in device units
- Dimensions of the picture frame, in .01-millimeter units
- Number of records in the metafile
- Offset to an optional text description
- Size of the optional palette
- Resolution of the original device, in pixels
- Resolution of the original device, in millimeters

An optional text description can follow the header record. The text description describes the picture and the author's name. The optional palette specifies the colors used to create the enhanced metafile. The remaining records identify the GDI functions used to create the picture. The following hexadecimal output corresponds to a record generated for a call to the [SetMapMode](#) function.

C++

```
00000011 0000000C 00000004
```

The value 0x00000011 specifies the record type (corresponds to the [EMR_SETMAPMODE](#) constant defined in the file Wingdi.h). The value 0x0000000C specifies the length of the record, in bytes. The value 0x00000004 identifies the mapping mode (corresponds to the [MM_LOENGLISH](#) constant defined in the [SetMapMode](#) function).

For a list of additional record types, see [Metafile Structures](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enhanced Metafile Creation

Article • 01/07/2021

You create an enhanced metafile by using the [CreateEnhMetaFile](#) function, supplying the appropriate arguments. The system uses these arguments to maintain picture dimensions, determine whether the metafile should be stored on a disk or in memory, and so on.

To maintain picture dimensions across output devices, [CreateEnhMetaFile](#) requires the resolution of the reference device. This *reference device* is the device on which the picture first appeared, and the *reference DC* is the *device context* associated with the reference device. When calling the [CreateEnhMetaFile](#) function, you must supply a handle that identifies this DC. You can get this handle by calling the [GetDC](#) or [CreateDC](#) function. You can also specify **NULL** as the handle to use the current display device for the reference device.

Most applications store pictures permanently and therefore create an enhanced metafile that is stored on a disk; however, there are some instances when this is not necessary. For example, a word-processing application that provides chart-drawing capabilities could store a user-defined chart in memory as an enhanced metafile and then copy the enhanced metafile bits from memory into the user's document file. An application that requires a metafile that is stored permanently on a disk must supply the file name when it calls [CreateEnhMetaFile](#). If you do not supply a file name, the system automatically treats the metafile as a temporary file and stores it in memory.

You can add an optional text description to a metafile containing information about the picture and the author. An application can display these strings in the File Open dialog box to provide the user with information about metafile content that will help in selecting the appropriate file. If an application includes the text description, it must supply a pointer to the string when it calls [CreateEnhMetaFile](#).

When [CreateEnhMetaFile](#) succeeds, it returns a handle that identifies a special metafile device context. A metafile device context is unique in that it is associated with a file rather than with an output device. When the system processes a GDI function that received a handle to a metafile device context, it converts the GDI function into an enhanced-metafile record and appends the record to the end of the enhanced metafile.

After a picture is complete and the last record is appended to the enhanced metafile, the application can close the file by calling the [CloseEnhMetaFile](#) function. This function closes and deletes the special metafile device context and returns a handle identifying the enhanced metafile.

To delete an enhanced-format metafile or an enhanced-format metafile handle, call the [DeleteEnhMetaFile](#) function.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Enhanced Metafile Operations

Article • 01/07/2021

You can use the handle to an enhanced metafile to accomplish the following tasks:

- Display the picture stored in an enhanced metafile.
- Create copies of an enhanced metafile.
- Edit an enhanced metafile.
- Retrieve the optional description stored in an enhanced metafile.
- Retrieve a copy of an enhanced-metafile header.
- Retrieve a binary version of an enhanced metafile.
- Enumerate the colors in the optional palette.

These tasks are discussed in the sections in the remainder of this topic.

Display the Picture Stored in an Enhanced Metafile

You can display the picture stored in an enhanced metafile using the [PlayEnhMetaFile](#) function. Pass the function a handle to the enhanced metafile, without being concerned with the format of the enhanced metafile records. However, it is sometimes desirable to enumerate the records in the enhanced metafile to search for a particular GDI function and modify the parameters of the function in some manner. To do this, you can use [EnumEnhMetaFile](#) and provide a callback function, [EnhMetaFileProc](#), to process the enhanced metafile records. To modify the parameters for an enhanced metafile record, you must know the format of the parameters within the record.

Create Copies of an Enhanced Metafile

Some applications create temporary backup (or duplicate) copies of a file before enabling the user to alter the original. An application can create a backup copy of an enhanced metafile by calling the [CopyEnhMetaFile](#) function, supplying a handle that identifies the enhanced metafile, and supplying a pointer to the name of the new file.

To create a memory-based enhanced-format metafile, call the [SetEnhMetaFileBits](#) function.

Edit an Enhanced Metafile

Most drawing, illustration, and computer-aided design (CAD) applications require a means of editing a picture stored in an enhanced metafile. Although editing an enhanced metafile is a complex task, you can use the [EnumEnhMetaFile](#) function in combination with other functions to provide this capability in your application. The [EnumEnhMetaFile](#) function and its associated callback function, [EnhMetaFileProc](#), enable the application to process individual records in an enhanced metafile.

Retrieve the Optional Description Stored in an Enhanced Metafile

Some applications display the text description of an enhanced metafile with the corresponding file name in the [Open](#) dialog box. You can determine whether this string exists in an enhanced metafile by retrieving the metafile header with the [GetEnhMetaFileHeader](#) function and examining one of its members. If the string exists, the application retrieves it by calling the [GetEnhMetaFileDescription](#) function.

Retrieve a Binary Version of an Enhanced Metafile

You can retrieve the contents of a metafile by calling the [GetEnhMetaFileBits](#) function; however, before retrieving the contents, you must specify the size of the file. To get the size, you can use the [GetEnhMetaFileHeader](#) function and examine the appropriate member.

Enumerate the Colors in the Optional Palette

To achieve consistent colors when a picture is displayed on various output devices, you can call the [CreatePalette](#) function and store a logical palette in an enhanced metafile. An application that displays the picture stored in the enhanced metafile retrieves this palette and calls the [RealizePalette](#) function before displaying the picture. To determine whether a palette is stored in an enhanced metafile, retrieve the metafile header and examine the appropriate member. If a palette exists, you can call the [GetEnhMetaFilePaletteEntries](#) function to retrieve the logical palette.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Windows-Format Metafiles

Article • 04/26/2022

Microsoft Windows-format metafiles are limited in their capabilities and should rarely be used. The Windows-format functions are supported to maintain backward compatibility with applications that were written to run as 16 bit Windows-based applications. Instead, you should use the enhanced-format functions.

A *Windows-format metafile* is used by 16-bit Windows-based applications. The format consists of a header and an array of metafile records.

The following are the limitations of this format:

- A Windows-format metafile is application and device dependent. Changes in the application's mapping modes or the device resolution affect the appearance of metafiles created in this format.
- A Windows-format metafile does not contain a comprehensive header that describes the original picture dimensions, the resolution of the device on which the picture was created, an optional text description, or an optional palette.
- A Windows-format metafile does not support the new curve, path, and transformation functions. See the list of supported functions in the table that follows.
- Some Windows-format metafile records cannot be scaled.
- The metafile device context associated with a Windows-format metafile cannot be queried (that is, an application cannot retrieve device-resolution data, font metrics, and so on).

The following are the only functions that are supported by Windows-format metafiles.

[+] Expand table

AnimatePaletteArc	LineToMoveToEx	SelectPaletteSetBkColor
BitBlt	OffsetClipRgn	SetBkMode
Chord	OffsetViewportOrgEx	SetDIBitsToDevice
CreateBrushIndirect	OffsetWindowOrgEx	SetMapMode
CreateDIBPatternBrush	PaintRgn	SetMapperFlags
CreateFontIndirect	PatBlt	SetPaletteEntries
CreatePalette	Pie	SetPixel
CreatePatternBrush	Polygon	SetPolyFillMode
CreatePenIndirect	Polyline	SetROP2
DeleteObject	PolyPolygon	SetStretchBltMode
Ellipse	RealizePalette	SetTextAlign

Escape	Rectangle	SetTextCharacterExtra
ExcludeClipRect	ResizePalette	SetTextColor
ExtFloodFill	RestoreDC	SetTextJustification
ExtTextOut	RoundRect	SetViewportOrgEx
FillRgn	SaveDC	SetWindowExtEx
FloodFill	ScaleViewportExtEx	SetWindowOrgEx
FrameRgn	ScaleWindowExtEx	StretchBlt
IntersectClipRect	SelectClipRgn	StretchDIBits
InvertRgn	SelectObject	TextOut

ⓘ Note

To convert a Windows-format metafile to an enhanced-format metafile, call the [GetMetaFileBitsEx](#) function to retrieve the data from the Windows-format metafile and then call the [SetWinMetaFileBits](#) function to convert this data into an enhanced-format metafile. To convert an enhanced-format record into a Windows-format record, call the [GetWinMetaFileBits](#) function.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Metafiles

Article • 01/07/2021

This section explains how to perform the following tasks:

- Creating an enhanced metafile
- Displaying a picture and storing it in an enhanced metafile
- Opening an enhanced metafile and displaying its contents
- Editing an enhanced metafile

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Creating an Enhanced Metafile

Article • 01/07/2021

This section contains an example that demonstrates the creation of an enhanced metafile that is stored on a disk, using a file name specified by the user.

The example uses a device context for the application window as the reference device context. (The system stores the resolution data for this device in the enhanced-metafile's header.) The application retrieves a handle identifying this device context by calling the [GetDC](#) function.

The example uses the dimensions of the application's client area to define the dimensions of the picture frame. Using the rectangle dimensions returned by the [GetClientRect](#) function, the application converts the device units to .01-millimeter units and passes the converted values to the [CreateEnhMetaFile](#) function.

The example displays a **Save As** common dialog box that enables the user to specify the file name of the new enhanced metafile. The system appends the three-character .emf extension to this file name and passes the name to the [CreateEnhMetaFile](#) function.

The example also embeds a text description of the picture in the enhanced-metafile header. This description is specified as a resource in the string table of the application's resource file. However, in a working application, this string would be retrieved from a custom control in a common dialog box or from a separate dialog box displayed solely for this purpose.

C++

```
// Obtain a handle to a reference device context.  
  
hdcRef = GetDC(hWnd);  
  
// Determine the picture frame dimensions.  
// iWidthMM is the display width in millimeters.  
// iHeightMM is the display height in millimeters.  
// iWidthPels is the display width in pixels.  
// iHeightPels is the display height in pixels  
  
iWidthMM = GetDeviceCaps(hdcRef, HORZSIZE);  
iHeightMM = GetDeviceCaps(hdcRef, VERTSIZE);  
iWidthPels = GetDeviceCaps(hdcRef, HORZRES);  
iHeightPels = GetDeviceCaps(hdcRef, VERTRES);  
  
// Retrieve the coordinates of the client  
// rectangle, in pixels.
```

```
GetClientRect(hWnd, &rect);

// Convert client coordinates to .01-mm units.
// Use iWidthMM, iWidthPels, iHeightMM, and
// iHeightPels to determine the number of
// .01-millimeter units per pixel in the x-
// and y-directions.

rect.left = (rect.left * iWidthMM * 100)/iWidthPels;
rect.top = (rect.top * iHeightMM * 100)/iHeightPels;
rect.right = (rect.right * iWidthMM * 100)/iWidthPels;
rect.bottom = (rect.bottom * iHeightMM * 100)/iHeightPels;

// Load the filename filter from the string table.

LoadString(hInst, IDS_FILTERSTRING,
    (LPSTR)szFilter, sizeof(szFilter));

// Replace the '%' separators that are embedded
// between the strings in the string-table entry
// with '\0'.

for (i=0; szFilter[i]!='\0'; i++)
    if (szFilter[i] == '%')
        szFilter[i] = '\0';

// Load the dialog title string from the table.

LoadString(hInst, IDS_TITLESTRING,
    (LPSTR)szTitle, sizeof(szTitle));

// Initialize the OPENFILENAME members.

szFile[0] = '\0';

Ofn.lStructSize = sizeof(OPENFILENAME);
Ofn.hwndOwner = hWnd;
Ofn.lpstrFilter = szFilter;
Ofn.lpstrFile= szFile;
Ofn.nMaxFile = sizeof(szFile)/ sizeof(*szFile);
Ofn.lpstrFileTitle = szFileTitle;
Ofn.nMaxFileTitle = sizeof(szFileTitle);
Ofn.lpstrInitialDir = (LPSTR)NULL;
Ofn.Flags = OFN_SHOWHELP | OFN_OVERWRITEPROMPT;
Ofn.lpstrTitle = szTitle;

// Display the Filename common dialog box. The
// filename specified by the user is passed
// to the CreateEnhMetaFile function and used to
// store the metafile on disk.

GetSaveFileName(&Ofn);

// Load the description from the string table.
```

```
LoadString(hInst, IDS_DESCRIPTIONSTRING,
    (LPSTR)szDescription, sizeof(szDescription));

// Replace the '%' string separators that are
// embedded between strings in the string-table
// entry with '\0'.

for (i=0; szDescription[i]!='\0'; i++)
{
    if (szDescription[i] == '%')
        szDescription[i] = '\0';
}

// Create the metafile device context.

hdcMeta = CreateEnhMetaFile(hdcRef,
    (LPTSTR) Ofn.lpstrFile,
    &rect, (LPSTR)szDescription);

if (!hdcMeta)
    errhandler("CreateEnhMetaFile", hWnd);

// Release the reference device context.

ReleaseDC(hWnd, hdcRef);
```

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Displaying a Picture and Storing It in an Enhanced Metafile

Article • 01/07/2021

This section contains an example demonstrating the creation of a picture and the process of storing the corresponding records in a metafile. The example draws a picture to the display or stores it in a metafile. If a display device context handle is given, it draws a picture to the screen using various GDI functions. If an enhanced metafile device context is given, it stores the same picture in the enhanced metafile.

C++

```
void DrawOrStore(HWND hwnd, HDC hdcMeta, HDC hdcDisplay)
{
    RECT rect;
    HDC hDC;
    int fnMapModeOld;
    HBRUSH hbrOld;

    // Draw it to the display DC or store it in the metafile device context.

    if (hdcMeta)
        hDC = hdcMeta;
    else
        hDC = hdcDisplay;

    // Set the mapping mode in the device context.

    fnMapModeOld = SetMapMode(hDC, MM_LOENGLISH);

    // Find the midpoint of the client area.

    GetClientRect(hwnd, (LPRECT)&rect);
    DPtoLP(hDC, (LPPOINT)&rect, 2);

    // Select a gray brush.

    hbrOld = SelectObject(hDC, GetStockObject(GRAY_BRUSH));

    // Draw a circle with a one inch radius.

    Ellipse(hDC, (rect.right/2 - 100), (rect.bottom/2 + 100),
            (rect.right/2 + 100), (rect.bottom/2 - 100));

    // Perform additional drawing here.
```

```
// Set the device context back to its original state.  
  
SetMapMode(hDC, fnMapModeOld);  
SelectObject(hDC, hbrOld);  
}
```

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Opening an Enhanced Metafile and Displaying Its Contents

Article • 01/07/2021

This section contains an example demonstrating how an application opens an enhanced metafile stored on disk and displays the associated picture in the client area.

The example uses the [Open](#) common dialog box to enable the user to select an enhanced metafile from a list of existing files. It then passes the name of the selected file to the [GetEnhMetaFile](#) function, which returns a handle identifying the file. This handle is passed to the [PlayEnhMetaFile](#) function in order to display the picture.

C++

```
LoadString(hInst, IDS_FILTERSTRING,
    (LPSTR)szFilter, sizeof(szFilter));

// Replace occurrences of '%' string separator
// with '\0'.

for (i=0; szFilter[i]!='\0'; i++)
{
    if (szFilter[i] == '%')
        szFilter[i] = '\0';
}

LoadString(hInst, IDS_DEFEXTSTRING,
    (LPSTR)szDefExt, sizeof(szFilter));

// Use the OpenFilename common dialog box
// to obtain the desired filename.

szFile[0] = '\0';
OPENFILENAME Ofn;
Ofn.lStructSize = sizeof(OPENFILENAME);
Ofn.hwndOwner = hWnd;
Ofn.lpstrFilter = szFilter;
Ofn.lpstrCustomFilter = (LPSTR)NULL;
Ofn.nMaxCustFilter = 0L;
Ofn.nFilterIndex = 1L;
Ofn.lpstrFile = szFile;
Ofn.nMaxFile = sizeof(szFile);
Ofn.lpstrFileTitle = szFileTitle;
Ofn.nMaxFileTitle = sizeof(szFileTitle);
Ofn.lpstrInitialDir = (LPSTR) NULL;
Ofn.lpstrTitle = (LPSTR)NULL;
Ofn.Flags = OFN_SHOWHELP | OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
Ofn.nFileOffset = 0;
```

```
Ofn.nFileExtension = 0;
Ofn.lpstrDefExt = szDefExt;

GetOpenFileName(&Ofn);

// Open the metafile.

HENHMETAFILE hemf = GetEnhMetaFile(Ofn.lpstrFile);

// Retrieve a handle to a window device context.

HDC hDC = GetDC(hWnd);

// Retrieve the client rectangle dimensions.

GetClientRect(hWnd, &rct);

// Draw the picture.

PlayEnhMetaFile(hDC, hemf, &rct);

// Release the metafile handle.

DeleteEnhMetaFile(hemf);

// Release the window DC.

ReleaseDC(hWnd, hDC);
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Editing an Enhanced Metafile

Article • 01/07/2021

To edit a picture stored in an enhanced metafile, an application must perform the tasks described in the following procedure.

To edit a picture stored in an enhanced metafile

1. Use hit-testing to capture the cursor coordinates and retrieve the position of the object (line, arc, rectangle, ellipse, polygon, or irregular shape) that the user wants to alter.
2. Convert these coordinates to logical (or world) units.
3. Call the [EnumEnhMetaFile](#) function and examine each metafile record.
4. Determine whether a given record corresponds to a GDI drawing function.
5. If it does, determine whether the coordinates stored in the record correspond to the line, arc, ellipse, or other graphics element that intersects the coordinates specified by the user.
6. Upon finding the record that corresponds to the output that the user wants to alter, erase the object on the screen that corresponds to the original record.
7. Delete the corresponding record from the metafile, saving a pointer to its location.
8. Permit the user to redraw or replace the object.
9. Convert the GDI functions used to draw the new object into one or more enhanced-metafile records.
10. Store these records in the enhanced metafile.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Metafile Reference

Article • 01/07/2021

The following elements are used with metafiles:

- [Metafile Functions](#)
- [Metafile Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Metafile Functions

Article • 01/07/2021

The following functions are used with enhanced-format metafiles.

[+] Expand table

Function	Description
CloseEnhMetaFile	Closes an enhanced-metafile device context.
CopyEnhMetaFile	Copies the contents of an enhanced-format metafile to a specified file.
CreateEnhMetaFile	Creates a device context for an enhanced-format metafile.
DeleteEnhMetaFile	Deletes an enhanced-format metafile or an enhanced-format metafile handle.
EnhMetaFileProc	An application-defined callback function used with the EnumEnhMetaFile function.
EnumEnhMetaFile	Enumerates the records within an enhanced-format metafile.
GdiComment	Copies a comment from a buffer into a specified enhanced-format metafile.
GetEnhMetaFile	Creates a handle that identifies the enhanced-format metafile stored in the specified file.
GetEnhMetaFileBits	Retrieves the contents of the specified enhanced-format metafile and copies them into a buffer.
GetEnhMetaFileDescription	Retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.
GetEnhMetaFileHeader	Retrieves the record containing the header for the specified enhanced-format metafile.
GetEnhMetaFilePaletteEntries	Retrieves optional palette entries from the specified enhanced metafile.
GetMetaFile	GetMetaFile is no longer available for use as of Windows 2000. Instead, use GetEnhMetaFile .
GetWinMetaFileBits	Converts the enhanced-format records from a metafile into Windows-format records.
PlayEnhMetaFile	Displays the picture stored in the specified enhanced-format

Function	Description
	metafile.
PlayEnhMetaFileRecord	Plays an enhanced-metafile record by executing the graphics device interface (GDI) functions identified by the record.
SetEnhMetaFileBits	Creates a memory-based enhanced-format metafile from the specified data.
SetWinMetaFileBits	Converts a metafile from the older Windows format to the new enhanced format.

Obsolete Functions

The following functions are obsolete. They are provided for compatibility with Windows-format metafiles.

- [CloseMetaFile](#)
- [CopyMetaFile](#)
- [CreateMetaFile](#)
- [DeleteMetaFile](#)
- [EnumMetaFile](#)
- [EnumMetaFileProc](#)
- [GetMetaFileBitsEx](#)
- [PlayMetaFile](#)
- [PlayMetaFileRecord](#)
- [SetMetaFileBitsEx](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

CloseEnhMetaFile function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CloseMetaFile function (wingdi.h)

Article10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CopyEnhMetaFileA function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateEnhMetaFileA function (wingdi.h)

Article 02/09/2023

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

C++

```
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 *lpRect* parameter must be less than the **right** and **bottom** members, respectively. Points along the edges of the rectangle are included in the picture. If *lpRect* is **NULL**, the graphics device interface (GDI) computes the dimensions of the smallest rectangle that surrounds the picture drawn by the application. The *lpRect* parameter should be provided where possible.

The string pointed to by the *lpDescription* 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 *lpDescription* 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 *lpRect* 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 that 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

CreateMetaFileA function (wingdi.h)

Article11/20/2024

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

C++

```
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 that 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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

DeleteEnhMetaFile function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

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

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

EnumEnhMetaFile function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

EnumMetaFile function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GdiComment function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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.

[+] [Expand table](#)

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:
------------------------------------	---

syntax

```
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.
```

[+] [Expand table](#)

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:
------------------------------	--

syntax

```
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.

[+] Expand table

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:
-----------------------------	--

syntax

```
DWORD ident;           // This contains GDICOMMENT_IDENTIFIER.  
DWORD iComment;       // This contains GDICOMMENT_ENDDGROUP.
```

[+] Expand table

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetEnhMetaFileA function (wingdi.h)

Article 11/20/2024

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

Syntax

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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 *lpbBuffer* 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article11/20/2024

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

Syntax

C++

```
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 that 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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

GetMetaFileA function (wingdi.h)

Article11/20/2024

[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

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetEnhMetaFile](#)

GetMetaFileBitsEx function (wingdi.h)

Article 02/22/2024

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

C++

```
UINT GetMetaFileBitsEx(
    [in] HMETAFILE 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 *lpvData* 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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetWinMetaFileBits function (wingdi.h)

Article 02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PlayEnhMetaFile function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetEnhMetaFileBits function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetMetaFileBitsEx function (wingdi.h)

Article10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetWinMetaFileBits function (wingdi.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Metafile Structures

Article • 11/19/2022

The following structures are used with enhanced-format metafiles.

[EMRTEXT](#)
[EMRTRANSPARENTBLT](#)
[EMRWIDENPATH](#)
[ENHMETAHEADER](#)
[ENHMETARECORD](#)
[HANDLETABLE](#)
[POINTL](#)
[RECTL](#)

Note that the [EMR](#) structure is used as the first member of the remaining structures.

[EMR](#)
[EMRABORTPATH](#)
[EMRALPHABLEND](#)
[EMRANGLEARC](#)
[EMRARC](#)
[EMRARCTO](#)
[EMRBEGINPATH](#)
[EMRBITBLT](#)
[EMRCHORD](#)
[EMRCLOSEFIGURE](#)
[EMRCOLORCORRECTPALETTE](#)
[EMRCOLORMATCHTOTARGET](#)
[EMRCREATEBRUSHINDIRECT](#)
[EMRCREATECOLORSPACE](#)
[EMRCREATECOLORSPACEW](#)
[EMRCREATEDIBPATTERNBRUSHPT](#)
[EMRCREATEMONOBRAUSH](#)
[EMRCREATEPALETTE](#)
[EMRCREATEPEN](#)
[EMRDELETECOLORSPACE](#)
[EMRDELETEOBJECT](#)
[EMRELLIPSE](#)
[EMRENDPATH](#)
[EMREOF](#)
[EMREXCLUDECLIPRECT](#)

EMREXTCREATEFONTINDIRECTW

EMREXTCREATEPEN

EMREXTFLOODFILL

EMREXTSELECTCLIPRGN

EMREXTTEXTOUTA

EMREXTTEXTOUTW

EMRFILLPATH

EMRFILLRGN

EMRFLATTENPATH

EMRFORMAT

EMRFRAMERGN

EMRGDICOMMENT

EMRGLSBOUNDEDRECORD

EMRGLSRECORD

EMRGRADIENTFILL

EMRINTERSECTCLIPRECT

EMRINVERTRGN

EMRLINETO

EMRMASKBLT

EMRMODIFYWORLDTRANSFORM

EMRMOVETOEX

EMROFFSETCLIPRGN

EMRPAINTRGN

EMRPIE

EMRPIXELFORMAT

EMRPLGBLT

EMRPOLYBEZIER

EMRPOLYBEZIER16

EMRPOLYBEZIERTO

EMRPOLYBEZIERTO16

EMRPOLYDRAW

EMRPOLYDRAW16

EMRPOLYGON

EMRPOLYGON16

EMRPOLYLINE

EMRPOLYLINE16

EMRPOLYLINETO

EMRPOLYLINETO16

EMRPOLYPOLYGON

EMRPOLYPOLYGON16

EMRPOLYPOLYLINE

EMRPOLYPOLYLINE16
EMRPOLYTEXTOUTA
EMRPOLYTEXTOUTW
EMRREALIZEPALETTE
EMRRECTANGLE
EMRRESIZEPALETTE
EMRRESTOREDC
EMRROUNDDIRECT
EMRSAVEDC
EMRSCALEVIEWPORTEXTEX
EMRSCALEWINDOWEXTEX
EMRSELECTCLIPPATH
EMRSELECTCOLORSPACE
EMRSELECTOBJECT
EMRSELECTPALETTE
EMRSETARCDIRECTION
EMRSETBKCOLOR
EMRSETBKMODE
EMRSETBRUSHORGEX
EMRSETCOLORADJUSTMENT
EMRSETCOLORSPACE
EMRSETDIBITSTODEVICE
EMRSETICMMODE
EMRSETICMPROFILE
EMRSETLAYOUT
EMRSETMAPMODE
EMRSETMAPPERFLAGS
EMRSETMETARGN
EMRSETMITERLIMIT
EMRSETPALETTEENTRIES
EMRSETPIXELV
EMRSETPOLYFILLMODE
EMRSETROP2
EMRSETSTRETCHBLTMODE
EMRSETTEXTALIGN
EMRSETTEXTCOLOR
EMRSETVIEWPORTEXTEX
EMRSETVIEWPORTORGEX
EMRSETWINDOWEXTEX
EMRSETWINDOWORGEX
EMRSETWORLDTRANSFORM

[EMRSTRETCHBLT](#)

[EMRSTRETCHDIBITS](#)

[EMRSTROKEANDFILLPATH](#)

[EMRSTROKEPATH](#)

Obsolete Structures

The following structures are obsolete. They are provided for compatibility with Windows-format metafiles:

[METAHEADERMETARECORD](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

EMR structure (wingdi.h)

Article 02/22/2024

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

C++

```
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_CREATEMONOBRUSH](#) [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_FRAMERGN](#)
[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_SetDIBitsToDevice 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_SetViewportTextEx
EMR_SetViewportOrgEx EMR_SetWindowExTEX EMR_SetWindowOrgEx
EMR_SetWorldTransform EMR.StretchBlt EMR.StretchDibits
EMR_StrokedFillPath EMR_StrokedPath EMR_TransparentBlt
EMR_WidenPath

nSize

The size of the record, in bytes. This member must be a multiple of four.

Requirements

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

Yes

No

EMRALPHABLEND structure (wingdi.h)

Article 02/22/2024

The **EMRALPHABLEND** structure contains members for the [AlphaBlend](#) enhanced metafile record.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRANGLEARC structure (wingdi.h)

Article 02/22/2024

The **EMRANGLEARC** structure contains members for the [AngleArc](#) enhanced metafile record.

Syntax

C++

```
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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRARC structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagEMRARC {
    EMR      emr;
    RECTL   rclBox;
    POINTL ptlStart;
    POINTL ptlEnd;
} EMRARC, *PEMRARC, EMRARCTO, *PEMRARCTO, MRCHORD, *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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRBITBLT structure (wingdi.h)

Article 02/22/2024

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

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCOLORCORRECTPALETTE structure (wingdi.h)

Article 04/02/2021

The **EMRCOLORCORRECTPALETTE** structure contains members for the [ColorCorrectPalette](#) enhanced metafile record.

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCOLORMATCHTOTARGET structure (wingdi.h)

Article 02/22/2024

The **EMRCOLORMATCHTOTARGET** structure contains members for the [ColorMatchToTarget](#) enhanced metafile record.

Syntax

C++

```
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.

[] [Expand table](#)

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.

[+] Expand table

Flag	Meaning
<code>COLORMATCHTOTARGET_EMBEDDED</code>	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[1]`

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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	<code>wingdi.h</code> (include <code>Windows.h</code>)

See also

[ColorMatchToTarget](#)

[Metafile Structures](#)

[Metafiles Overview](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATEBRUSHINDIRECT structure (wingdi.h)

Article 02/22/2024

The **EMRCREATEBRUSHINDIRECT** structure contains members for the [CreateBrushIndirect](#) enhanced metafile record.

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATECOLORSPACE structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[CreateColorSpace](#)

[EMR](#)

[EMRCREATECOLORSPACEW](#)

[LOGCOLORSPACE](#)

[Metafile Structures](#)

[Metafiles Overview](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATECOLORSPACEW structure (wingdi.h)

Article 09/01/2022

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

C++

```
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.

[+] Expand table

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[1]`

An array containing the source profile data. The size of the array is `cbData`.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATEDIBPATTERNBRUSHPT structure (wingdi.h)

Article 02/22/2024

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

C++

```
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

[\[+\] Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

[!\[\]\(1b6eb43e64b367210767d23f12e6ccaa_img.jpg\) Yes](#)

[!\[\]\(05702f660de2f949d189551c418ed23e_img.jpg\) No](#)

EMRCREATEMONOBRUSH structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[\[+\] Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATEPALETTE structure (wingdi.h)

Article 02/22/2024

The **EMRCREATEPALETTE** structure contains members for the [CreatePalette](#) enhanced metafile record.

Syntax

C++

```
typedef struct tagEMRCREATEPALETTE {
    EMR         emr;
    DWORD       ihPal;
    LOGPALETTE lgpl;
} EMRCREATEPALETTE, *PEMRCREATEPALETTE;
```

Members

emr

The base structure for all record types.

ihPal

Index of palette in handle table.

lgpl

A [LOGPALETTE](#) structure that contains information about the palette. Note that **peFlags** members in the [PALETTEENTRY](#) structures do not contain any flags.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[CreatePalette](#)

[LOGPALETTE](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PALETTEENTRY](#)

Feedback

Was this page helpful?

 Yes

 No

EMRCREATEPEN structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRELLIPSE structure (wingdi.h)

Article02/22/2024

The **EMRELLIPSE** and **EMRRECTANGLE** structures contain members for the [Ellipse](#) and [Rectangle](#) enhanced metafile records.

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMREOF structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMREXCLUDECLIPRECT structure (wingdi.h)

Article 02/22/2024

The **EMREXCLUDECLIPRECT** and **EMRINTERSECTCLIPRECT** structures contain members for the [ExcludeClipRect](#) and [IntersectClipRect](#) enhanced metafile records.

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
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

Feedback

Was this page helpful?

 Yes

 No

EMREXTCREATEFONTINDIRECTW structure (wingdi.h)

Article 02/22/2024

The **EMREXTCREATEFONTINDIRECTW** structure contains members for the [CreateFontIndirect](#) enhanced metafile record.

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[CreateFontIndirect](#)

[Metafile Structures](#)

[Metafiles Overview](#)

Feedback

Was this page helpful?

 Yes

 No

EMREXTCREATEPEN structure (wingdi.h)

Article 02/22/2024

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).

Syntax

C++

```
typedef struct tagEMREXTCREATEPEN {
    EMR          emr;
    DWORD        ihPen;
    DWORD        offBmi;
    DWORD        cbBmi;
    DWORD        offBits;
    DWORD        cbBits;
    EXTLOGOPEN32 elp;
} EMREXTCREATEPEN, *PEMREXTCREATEPEN;
```

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

 [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMREXTFLOODFILL structure (wingdi.h)

Article04/02/2021

The **EMREXTFLOODFILL** structure contains members for the [ExtFloodFill](#) enhanced metafile record.

Syntax

C++

```
typedef struct tagEMREXTFLOODFILL {
    EMR      emr;
    POINTL   ptlStart;
    COLORREF crColor;
    DWORD    iMode;
} EMREXTFLOODFILL, *PEMREXTFLOODFILL;
```

Members

`emr`

The base structure for all record types.

`ptlStart`

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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMREXTSELECTCLIPRGN structure (wingdi.h)

Article 02/22/2024

The **EMREXTSELECTCLIPRGN** structure contains members for the [ExtSelectClipRgn](#) enhanced metafile record.

Syntax

C++

```
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[1]`

Buffer containing a [RGNDATA](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMREXTTEXTOUTA structure (wingdi.h)

Article02/22/2024

The **EMREXTTEXTOUTA** and **EMREXTTEXTOUTW** structures contain members for the [ExtTextOut](#), [TextOut](#), or [DrawText](#) enhanced metafile records.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRFILLPATH structure (wingdi.h)

Article02/22/2024

The **EMRFILLPATH**, **EMRSTROKEANDFILLPATH**, and **EMRSTROKEPATH** structures contain members for the [FillPath](#), [StrokeAndFillPath](#), and [StrokePath](#) enhanced metafile records.

Syntax

C++

```
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

 Expand table

Requirement	Value
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

Feedback

Was this page helpful?

 Yes

 No

EMRFILLRGN structure (wingdi.h)

Article 02/22/2024

The **EMRFILLRGN** structure contains members for the [FillRgn](#) enhanced metafile record.

Syntax

C++

```
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[1]`

Buffer containing [RGNDATA](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRFORMAT structure (wingdi.h)

Article02/22/2024

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

C++

```
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.

[+] Expand table

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.

[+] Expand table

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRFRAMERGN structure (wingdi.h)

Article 02/22/2024

The **EMRFRAMERGN** structure contains members for the [FrameRgn](#) enhanced metafile record.

Syntax

C++

```
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[1]`

Buffer containing [RGNDATA](#) structure.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRGDICONMENT structure (wingdi.h)

Article 02/22/2024

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

C++

```
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.

Data[1]

Application-specific data.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

Feedback

Was this page helpful?

 Yes

 No

EMRGLSBOUNDEDRECORD structure (wingdi.h)

Article09/01/2022

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

C++

```
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[1]`

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	<code>wingdi.h</code> (include <code>Windows.h</code>)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

[OpenGL on Windows NT, Windows 2000, and Windows 95/98](#)

Feedback

Was this page helpful?

 Yes

 No

EMRGLSRECORD structure (wingdi.h)

Article02/22/2024

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

C++

```
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[1]`

Array of data representing the OpenGL function to be performed.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRGRADIENTFILL structure (wingdi.h)

Article 02/22/2024

The **EMRGRADIENTFILL** structure contains members for the [GradientFill](#) enhanced metafile record.

Syntax

C++

```
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.

[] Expand table

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[1]

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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRINVERTRGN structure (wingdi.h)

Article02/22/2024

The **EMRINVERTRGN** and **EMRPAINTRGN** structures contain members for the [InvertRgn](#) and [PaintRgn](#) enhanced metafile records.

Syntax

C++

```
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[1]`

Buffer containing an [RGNDATA](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[InvertRgn](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[PaintRgn](#)

Feedback

Was this page helpful?

 Yes

 No

EMRLINETO structure (wingdi.h)

Article02/22/2024

The **EMRLINETO** and **EMRMOVETOEX** structures contains members for the [LineTo](#) and [MoveToEx](#) enhanced metafile records.

Syntax

C++

```
typedef struct tagEMRLINETO {
    EMR      emr;
    POINTL  ptl;
} EMRLINETO, *PEMRLINETO, EMRMOVETOEX, *PEMRMOVETOEX;
```

Members

`emr`

Base structure for all record types.

`ptl`

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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRMASKBLT structure (wingdi.h)

Article 04/02/2021

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

C++

```
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

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRMODIFYWORLDTRANSFORM structure (wingdi.h)

Article02/22/2024

The **EMRMODIFYWORLDTRANSFORM** structure contains members for the [ModifyWorldTransform](#) enhanced metafile record.

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[EMR](#)

[Metafile Structures](#)

[Metafiles Overview](#)

[ModifyWorldTransform](#)

[XFORM](#)

Feedback

Was this page helpful?

 Yes

 No

EMROFFSETCLIPRGN structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPIXELFORMAT structure (wingdi.h)

Article02/22/2024

The **EMRPIXELFORMAT** structure contains the members for the [SetPixelFormat](#) enhanced metafile record. The pixel format information in [ENHMETAHEADER](#) refers to this structure.

Syntax

C++

```
typedef struct tagEMRPIXELFORMAT {
    EMR           emr;
    PIXELFORMATDESCRIPTOR pfd;
} EMRPIXELFORMAT, *PEMRPIXELFORMAT;
```

Members

emr

The base structure for all record types.

pfd

A [PIXELFORMATDESCRIPTOR](#) structure, which describes the pixel format.

Requirements

[+] [Expand table](#)

Requirement	Value
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](#)

[PIXELFORMATDESCRIPTOR](#)

[SetPixelFormat](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPLGBT structure (wingdi.h)

Article09/01/2022

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

C++

```
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[3]`

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

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYDRAW structure (wingdi.h)

Article 02/22/2024

The **EMRPOLYDRAW** structure contains members for the [PolyDraw](#) enhanced metafile record.

Syntax

C++

```
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[1]`

An array of [POINTL](#) structures, representing the data points in logical units.

`abTypes[1]`

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYDRAW16 structure (wingdi.h)

Article11/19/2022

The **EMRPOLYDRAW16** structure contains members for the [PolyDraw](#) enhanced metafile record.

Syntax

C++

```
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[1]`

An array of [POINTS](#) structures, representing the data points in logical units.

`abTypes[1]`

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYLINE structure (wingdi.h)

Article02/22/2024

The **EMRPOLYLINE**, **EMRPOLYBEZIER**, **EMRPOLYGON**, **EMRPOLYBEZIERTO**, and **EMRPOLYLINETO** structures contain members for the [Polyline](#), [PolyBezier](#), [Polygon](#), [PolyBezierTo](#), and [PolylineTo](#) enhanced metafile records.

Syntax

C++

```
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[1]`

Array of 32-bit points, in logical units.

Requirements

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYLINE16 structure (wingdi.h)

Article02/22/2024

The **EMRPOLYLINE16**, **EMRPOLYBEZIER16**, **EMRPOLYGON16**, **EMRPOLYBEZIERTO16**, and **EMRPOLYLINETO16** structures contain members for the [Polyline](#), [PolyBezier](#), [Polygon](#), [PolyBezierTo](#), and [PolylineTo](#) enhanced metafile records.

Syntax

C++

```
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.

`appts[1]`

Array of 16-bit points, in logical units.

Requirements

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYPOLYLINE structure (wingdi.h)

Article 02/22/2024

The **EMRPOLYPOLYLINE** and **EMRPOLYPOLYGON** structures contain members for the [PolyPolyline](#) and [PolyPolygon](#) enhanced metafile records.

Syntax

C++

```
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[1]`

An array of point counts for each poly.

`aptl[1]`

An array of [POINTL](#) structures, representing the points in logical units.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYPOLYLINE16 structure (wingdi.h)

Article 02/22/2024

The **EMRPOLYPOLYLINE16** and **EMRPOLYPOLYGON16** structures contain members for the [PolyPolyline](#) and [PolyPolygon](#) enhanced metafile records.

Syntax

C++

```
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[1]`

An array of point counts for each poly.

```
apts[1]
```

An array of [POINTS](#) structures, representing the points in logical units.

Requirements

[Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRPOLYTEXTOUTA structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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[1]

EMRTEXT structure, which is followed by the string and the intercharacter spacing array.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRRESIZEPALETTE structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[\[\] Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRRESTOREDC structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagEMRRESTOREDC {
    EMR    emr;
    LONG   iRelative;
} EMRRESTOREDC, *PEMRRESTOREDC;
```

Members

`emr`

The base structure for all record types.

`iRelative`

Relative instance to restore.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRROUNDRECT structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSCALEVIEWPORTEXTEX structure (wingdi.h)

Article 02/22/2024

The **EMRSCALEVIEWPORTEXTEX** and **EMRSCALEWINDOWEXTEX** structures contain members for the [ScaleViewportExtEx](#) and [ScaleWindowExtEx](#) enhanced metafile records.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETCOLORSPACE structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h (include Windows.h)

See also

[Metafile Structures](#)

[Metafiles Overview](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSELECTOBJECT structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSELECTPALETTE structure (wingdi.h)

Article02/22/2024

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

C++

```
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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETARCDIRECTION structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETBKCOLOR structure (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETCOLORADJUSTMENT structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagEMRSETCOLORADJUSTMENT {
    EMR           emr;
    COLORADJUSTMENT ColorAdjustment;
} EMRSETCOLORADJUSTMENT, *PEMRSETCOLORADJUSTMENT;
```

Members

emr

The base structure for all record types.

ColorAdjustment

A [COLORADJUSTMENT](#) structure.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETDIBITSTODEVICE structure (wingdi.h)

Article 02/22/2024

The **EMRSETDIBITSTODEVICE** structure contains members for the [SetDIBitsToDevice](#) enhanced metafile record.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETICMPROFILE structure (wingdi.h)

Article 02/22/2024

The **EMRSETICMPROFILE** structure contains members for the [SetICMProfile](#) enhanced metafile record.

Syntax

C++

```
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[1]`

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETMAPPERFLAGS structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagEMRSETMAPPERFLAGS {
    EMR     emr;
    DWORD   dwFlags;
} EMRSETMAPPERFLAGS, *PEMRSETMAPPERFLAGS;
```

Members

emr

The base structure for all record types.

dwFlags

Font mapper flag.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETMITERLIMIT structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagEMRSETMITERLIMIT {
    EMR     emr;
    FLOAT   eMiterLimit;
} EMRSETMITERLIMIT, *PEMRSETMITERLIMIT;
```

Members

`emr`

The base structure for all record types.

`eMiterLimit`

New miter limit.

Requirements

[] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETPALETTEENTRIES structure (wingdi.h)

Article 02/22/2024

The **EMRSETPALETTEENTRIES** structure contains members for the [SetPaletteEntries](#) enhanced metafile record.

Syntax

C++

```
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[1]`

Array of [PALETTEENTRY](#) structures. Note that `peFlags` members in the structures do not contain any flags.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETPIXELV structure (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETVIEWPORTEXTEX structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?



EMRSETVIEWPORTORGEX structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSETWORLDTRANSFORM structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSTRETCHBLT structure (wingdi.h)

Article02/22/2024

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

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSTRETCHDIBITS structure (wingdi.h)

Article 02/22/2024

The **EMRSTRETCHDIBITS** structure contains members for the [StretchDIBits](#) enhanced metafile record.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRTEXT structure (wingdi.h)

Article11/19/2022

The **EMRTEXT** structure contains members for text output.

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRTRANSPARENTBLT structure (wingdi.h)

Article04/02/2021

The **EMRTRANSPARENTBLT** structure contains members for the [TransparentBLT](#) enhanced metafile record.

Syntax

C++

```
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

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRABORTPATH structure (wingdi.h)

Article02/22/2024

Contains data for the [AbortPath](#), [BeginPath](#), [EndPath](#), [CloseFigure](#), [FlattenPath](#), [WidenPath](#), [SetMetaRgn](#), [SaveDC](#), and [RealizePalette](#) enhanced metafile records.

Syntax

C++

```
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

[+] [Expand table](#)

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

EMRSELECTCLIPPATH structure (wingdi.h)

Article04/02/2021

Contains parameters for the [SelectClipPath](#), [SetBkMode](#), [SetMapMode](#), [SetPolyFillMode](#), [SetROP2](#), [SetStretchBltMode](#), [SetTextAlign](#), [SetICMMODE](#), and [SetLayout](#) enhanced metafile records.

Syntax

C++

```
typedef struct tagEMRSELECTCLIPPATH {  
    EMR     emr;  
    DWORD   iMode;  
} EMRSELECTCLIPPATH, *PEMRSELECTCLIPPATH, EMRSETBKMODE, *PEMRSETBKMODE,  
EMRSETRMAPMODE, *PEMRSETRMAPMODE, EMRSETLAYOUT, *PEMRSETLAYOUT,  
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

ENHMETAHEADER structure (wingdi.h)

Article11/19/2022

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

C++

```
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.

rc1Bounds

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.

rc1Frame

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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

ENHMETARECORD structure (wingdi.h)

Article09/01/2022

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

C++

```
typedef struct tagENHMETARECORD {
    DWORD iType;
    DWORD nSize;
    DWORD dParm[1];
} ENHMETARECORD, *PENHMETARECORD, *LPENHMETARECORD;
```

Members

iType

The record type.

nSize

The size of the record, in bytes.

dParm[1]

An array of parameters passed to the GDI function identified by the record.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

HANDLETABLE structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagHANDLETABLE {
    HGDIOBJ objectHandle[1];
} HANDLETABLE, *PHANDLETABLE, *LPHANDLETABLE;
```

Members

objectHandle[1]

An array of handles.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

METAHEADER structure (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

[+] Expand table

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

METARECORD structure (wingdi.h)

Article 02/22/2024

The **METARECORD** structure contains a Windows-format metafile record.

Syntax

C++

```
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[1]`

An array of words containing the function parameters, in reverse of the order they are passed to the function.

Requirements

[] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

POINTL structure (windef.h)

Article 02/22/2024

The POINTL structure defines the x- and y-coordinates of a point.

Syntax

C++

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL, *PPOINTL;
```

Members

x

Specifies the x-coordinate of the point.

y

Specifies the y-coordinate of the point.

Remarks

The POINTL structure is identical to the [POINT](#) structure.

Requirements

 Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

[POINT](#)

POINTS

Feedback

Was this page helpful?

 Yes

 No

RECTL structure (windef.h)

Article02/22/2024

The RECTL structure defines a rectangle by the coordinates of its upper-left and lower-right corners.

Syntax

C++

```
typedef struct _RECTL {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECTL, *PRECTL, *LPRECTL;
```

Members

`left`

Specifies the x-coordinate of the upper-left corner of the rectangle.

`top`

Specifies the y-coordinate of the upper-left corner of the rectangle.

`right`

Specifies the x-coordinate of the lower-right corner of the rectangle.

`bottom`

Specifies the y-coordinate of the lower-right corner of the rectangle.

Remarks

The RECTL structure is identical to the [RECT](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

[RECT](#)

Feedback

Was this page helpful?

 Yes

 No

Multiple Display Monitors

Article • 01/07/2021

Multiple Display Monitors is a set of related features that allow applications to make use of multiple display devices at the same time. There are two ways that multiple monitors may be used: as one large desktop or as a number of independent displays. When used as one large desktop, the monitors create more screen space for applications. This is useful whenever you need to maximize your onscreen workspace, and is especially useful in desktop publishing, Web development, or video editing. When used as independent displays, the monitors can be used for playing games, debugging full-screen applications, teaching, or making presentations.

This chapter covers the following topics:

- [About Multiple Display Monitors](#)
- [Using Multiple Display Monitors](#)
- [Multiple Display Monitors Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About multiple display monitors

Article • 11/06/2024

When multiple monitors are part of the desktop, objects can travel seamlessly between monitors. That is, you can drag windows or shortcuts from one monitor to another, and you can size windows to cover more than one monitor. Also, if one monitor is installed above another, a cursor that leaves the bottom of the upper monitor appears at the top of the lower monitor.

Typically, a user arranges the monitors in the system to reflect the arrangement of the physical display units; for example, side-by-side or one-on-top-of-the-other. The user does this with the Monitors tab, which replaces the **Settings** tab in the **Display Properties** dialog box through Control Panel. The monitors must form one contiguous region, that is, each monitor touches another monitor on at least part of one edge.

When a window is moved or resized, some part of the caption is always visible so the user can move and resize the window using the mouse. Cursor movement is restricted to the area of the monitors, so it is always visible. Shell icons are positioned on the same monitor as the taskbar, and the taskbar can be on any monitor, see [Multiple Monitor Considerations for Older Programs](#).

A multiple monitor system affects certain key combinations.

- CTRL+PRINTSCRN takes a snapshot of the entire virtual screen, see [The Virtual Screen](#).
- ALT+PRINTSCRN takes a snapshot of the foreground window.
- PRINTSCRN takes a snapshot of the entire virtual screen; unless **Use the Print screen to open screen capture** is on in Windows Settings, in which case PRINTSCRN launches your installed snipping tool-type app.

The support for multiple monitors does not affect the performance of applications when running in a single display environment. That is, when running on a single display system, no additional overhead will be present in the high-performance graphics operations code. However, in a multiple monitor system, performance is slightly affected if an application runs only on one of the graphics devices. Also, performance may be greatly affected if an application spans multiple displays, especially for graphics-intensive operations.

Full-screen is a feature provided by the operating system that allows a user to toggle an application into a special state where the application can access VGA graphics hardware directly. This is a key feature for games and other graphics-focused applications that

require high performance. Also, it is often used by developers for text editing since it enables very fast text scrolling.

In a multiple monitor environment, only one graphics device can be VGA compatible. This is a limitation of computer hardware which requires that only one device respond to any hardware address. Because the VGA hardware compatibility standard requires specific hardware addresses, only one VGA graphics device can be present in a machine and only this device can physically respond to VGA addresses. Thus, applications that require going full screen will only run on the particular device that supports VGA hardware compatibility.

This overview provides information on the following topics.

- [The Virtual Screen](#)
- [HMONITOR and the Device Context](#)
- [Enumeration and Display Control](#)
- [Multiple Monitor System Metrics](#)
- [Using Multiple Monitors as Independent Displays](#)
- [Colors on Multiple Display Monitors](#)
- [Positioning Objects on Multiple Display Monitors](#)
- [Multiple Monitor Applications on Different Systems](#)
- [Multiple Monitor Considerations for Older Programs](#)

Feedback

Was this page helpful?

 Yes

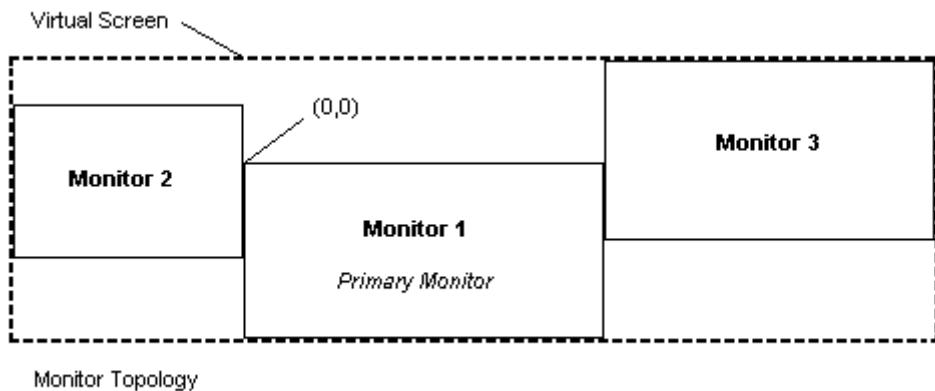
 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

The Virtual Screen

Article • 01/07/2021

The bounding rectangle of all the monitors is the *virtual screen*. The desktop covers the virtual screen instead of a single monitor. The following illustration shows a possible arrangement of three monitors.



The *primary monitor* contains the origin (0,0). This is for compatibility with existing applications that expect a monitor with an origin. However, the primary monitor does not have to be in the upper left of the virtual screen. In Figure 1, it is near the center. When the primary monitor is not in the upper left of the virtual screen, parts of the virtual screen have negative coordinates. Because the arrangement of monitors is set by the user, all applications should be designed to work with negative coordinates. For more information, see [Multiple Monitor Considerations for Older Programs](#).

The coordinates of the virtual screen are represented by a signed 16-bit value because of the 16-bit values contained in many existing messages. Thus, the bounds of the virtual screen are:

syntax

```
SHORT_MIN    <= rcVirtualScreen.left    <= SHORT_MAX - 1  
SHORT_MIN +1 <= rcVirtualScreen.right   <= SHORT_MAX  
SHORT_MIN    <= rcVirtualScreen.top     <= SHORT_MAX - 1  
SHORT_MIN +1 <= rcVirtualScreen.bottom <= SHORT_MAX
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

HMONITOR and the Device Context

Article • 01/07/2021

Each physical display is represented by a monitor handle of type **HMONITOR**. A valid **HMONITOR** is guaranteed to be non-NULL. A physical display has the same **HMONITOR** as long as it is part of the desktop. When a **WM_DISPLAYCHANGE** message is sent, any monitor may be removed from the desktop and thus its **HMONITOR** becomes invalid or has its settings changed. Therefore, an application should check whether all **HMONITORS** are valid when this message is sent.

Any function that returns a display device context (DC) normally returns a DC for the primary monitor. To obtain the DC for another monitor, use the [EnumDisplayMonitors](#) function. Or, you can use the device name from the [GetMonitorInfo](#) function to create a DC with [CreateDC](#). However, if the function, such as [GetWindowDC](#) or [BeginPaint](#), gets a DC for a window that spans more than one display, the DC will also span the two displays.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Enumeration and Display Control

Article • 01/07/2021

To enumerate all the devices on the computer, call the [EnumDisplayDevices](#) function. The information that is returned also indicates which monitor is part of the desktop.

To enumerate the devices on the desktop that intersect a clipping region, call [EnumDisplayMonitors](#). This returns the HMONITOR handle to each monitor, which is used with [GetMonitorInfo](#). To enumerate all the devices in the virtual screen, use [EnumDisplayMonitors](#). as shown in the following code:

C++

```
EnumDisplayMonitors(NULL, NULL, MyInfoEnumProc, 0);
```

To get information about a display device, use [EnumDisplaySettings](#) or [EnumDisplaySettingsEx](#).

The [ChangeDisplaySettingsEx](#) function is used to control the display devices on the computer. It can modify the configuration of the devices, such as specifying the position of a monitor on the virtual desktop and changing the bit depth of any display. Typically, an application does not use this function. To add a display monitor to a multiple-monitor system programmatically, set **DEVMODE.dmFields** to **DM_POSITION** and specify a position (using **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** and set **DEVMODE.dmPelsWidth** and **DEVMODE.dmPelsHeight** to zero.

The following code demonstrates how to detach all secondary display devices from the desktop:

```
void DetachDisplay()
{
    BOOL          FoundSecondaryDisp = FALSE;
    DWORD         DispNum = 0;
    DISPLAY_DEVICE DisplayDevice;
    LONG          Result;
    TCHAR         szTemp[200];
    int           i = 0;
    DEVMODE      defaultMode;

    // initialize DisplayDevice
    ZeroMemory(&DisplayDevice, sizeof(DisplayDevice));
```

```

DisplayDevice.cb = sizeof(DisplayDevice);

// get all display devices
while (EnumDisplayDevices(NULL, DispNum, &DisplayDevice, 0))
{
    ZeroMemory(&defaultMode, sizeof(DEVMODE));
    defaultMode.dmSize = sizeof(DEVMODE);
    if ( !EnumDisplaySettings((LPSTR)DisplayDevice.DeviceName,
ENUM_REGISTRY_SETTINGS, &defaultMode) )
        OutputDebugString("Store default failed\n");

    if ((DisplayDevice.StateFlags & DISPLAY_DEVICE_ATTACHED_TO_DESKTOP)
&&
        !(DisplayDevice.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE))
    {
        DEVMODE      DevMode;
        ZeroMemory(&DevMode, sizeof(DevMode));
        DevMode.dmSize = sizeof(DevMode);
        DevMode.dmFields = DM_PELSWIDTH | DM_PELSHEIGHT | DM_BITSPERPEL
| DM_POSITION
            | DM_DISPLAYFREQUENCY | DM_DISPLAYFLAGS ;
        Result =
ChangeDisplaySettingsEx((LPSTR)DisplayDevice.DeviceName,
                        &DevMode,
                        NULL,
                        CDS_UPDATEREGISTRY,
                        NULL);

        //The code below shows how to re-attach the secondary displays
        to the desktop

        //ChangeDisplaySettingsEx((LPSTR)DisplayDevice.DeviceName,
        //                         &defaultMode,
        //                         NULL,
        //                         CDS_UPDATEREGISTRY,
        //                         NULL);

    }

    // Reinit DisplayDevice just to be extra clean

    ZeroMemory(&DisplayDevice, sizeof(DisplayDevice));
    DisplayDevice.cb = sizeof(DisplayDevice);
    DispNum++;
} // end while for all display devices
}

```

For each display device, the application can save information in the registry that describes the configuration parameters for the device, as well as location parameters. The application can also determine which displays are part of the desktop, and which are not, through the DISPLAY_DEVICE_ATTACHED_TO_DESKTOP flag in the **DISPLAY_DEVICE** structure. Once all the configuration information is stored in the

registry, the application can call [ChangeDisplaySettingsEx](#) again to dynamically change the settings, with no restart required.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Multiple Monitor System Metrics

Article • 11/19/2022

The [GetSystemMetrics](#) function returns values for the primary monitor, except for SM_CXMAXTRACK and SM_CYMAXTRACK, which refer to the entire desktop. The following metrics are the same for all device drivers: SM_CXCURSOR, SM_CYCURSOR, SM_CXICON, SM_CYICON. The following display capabilities are the same for all monitors: LOGPIXELSX, LOGPIXELSY, DESTOPHORZRES, DESKTOPVERTRES.

[GetSystemMetrics](#) also has constants that refer only to a Multiple Monitor system. SM_XVIRTUALSCREEN and SM_YVIRTUALSCREEN identify the upper-left corner of the virtual screen, SM_CXVIRTUALSCREEN and SM_CYVIRTUALSCREEN are the vertical and horizontal measurements of the virtual screen, SM_CMONITORS is the number of monitors attached to the desktop, and SM_SAMEDISPLAYFORMAT indicates whether all the monitors on the desktop have the same color format.

To get information about a single display monitor or all of the display monitors in a desktop, use `EnumDisplayMonitors`. The rectangle of the desktop window returned by [GetWindowRect](#) or [GetClientRect](#) is always equal to the rectangle of the primary monitor, for compatibility with existing applications. Thus, the result of

C++

```
GetWindowRect(GetDesktopWindow(), &rc);
```

will be:

C++

```
rc.left = 0;  
rc.top = 0;  
rc.right = GetSystemMetrics (SM_CXSCREEN);  
rc.bottom = GetSystemMetrics (SM_CYSCREEN);
```

To change the work area of a monitor, call [SystemParametersInfo](#) with `SPI_SETWORKAREA` and `pvParam` pointing to a [RECT](#) structure that is on the desired monitor. If `pvParam` is `NULL`, the work area of the primary monitor is modified. Using `SPI_GETWORKAREA` always returns the work area of the primary monitor. To get the work area of a monitor other than the primary monitor, call [GetMonitorInfo](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Multiple Monitors as Independent Displays

Article • 01/07/2021

When using multiple monitors as independent displays, the desktop contains one display or set of displays. This set of displays always includes the primary monitor and behaves as mentioned in the other sections of this topic. An application can use any other monitor as an independent display.

ⓘ Note

Using other monitors as independent displays isn't supported on drivers that are implemented to the Windows Display Driver Model (WDDM).

The window manager knows nothing about the independent displays. They are completely controlled by the application, and no window manager functions are available to the application (all window manager calls automatically go to the primary display). Each independent display has its own origin and horizontal and vertical coordinates, and is accessed through the GDI functions such as [CreateDC](#) or the DirectX functions such as [DirectDrawCreate](#).

To locate the independent displays, call [EnumDisplayDevices](#) and look for the displays that do not have `DISPLAY_DEVICE_ATTACHED_TO_DESKTOP` flag in the [DISPLAY_DEVICE](#) structure.

An application can open a display by calling

C++

```
hdc = CreateDC(lpszDisplayName, NULL, NULL, lpDevMode);
```

In this call, the `lpszDisplayName` parameter is one of the device names returned by [EnumDisplayDevices](#) and `lpDevMode` is a description of the graphics mode for this device. The resulting `hdc` can be used to perform any graphics operation to the device.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Colors on Multiple Display Monitors

Article • 01/07/2021

Each monitor can have its own color depth. The system automatically adjusts colors as windows move across monitors with different color depths. In general, this produces good results. However, this is not always optimal. To take advantage of the color capabilities of different monitors, see the [Painting on Multiple Display Monitors](#) section that follows.

To determine if all monitors have the same color format, call [GetSystemMetrics](#) with SM_SAMEDISPLAYFORMAT.

If the primary monitor is palettized, [SelectPalette](#) and [RealizePalette](#) work the same as before, but across all monitors. In addition, the palettes of all palettized devices are synchronized. If the primary monitor is not palettized, [SelectPalette](#) and [RealizePalette](#) will select the palette into the background and palettized devices will not be synchronized.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Painting on Multiple Display Monitors

Article • 01/07/2021

The system automatically handles painting into a device context (DC) that spans more than one monitor, even when the monitors have different color depths. Usually this produces good results, but it may not be optimal. For example, a window on two monitors of vastly different color depths could have poor color rendition. Also, monitors with the same color depth may have different color formats for example, colors could be encoded with different numbers of bits, or be located in different places in a pixel's color value.

To get the best results for each of the monitors in a DC that spans more than one display, call [EnumDisplayMonitors](#) to enumerate the monitors that intersect your DC and paint the intersecting area in each of them separately according to the display attributes for that monitor. See the example in [Painting on a DC That Spans Multiple Displays](#).

If you do all of your drawing in your **WM_PAINT** code and if your **WM_PAINT** code handles all of the various video modes, then you should be able to place your **WM_PAINT** code in the **MonitorEnumProc** of [EnumDisplayMonitors](#) with only a few modifications.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Positioning Objects on Multiple Display Monitors

Article • 01/07/2021

A window or menu that is on more than one monitor causes visual disruption for a viewer. To minimize this problem, the system displays menus and new and maximized windows on one monitor. The following table shows how the monitor is chosen.

 Expand table

Object	Location
window	<p>CreateWindow(Ex) displays a window on the monitor that contains the largest part of the window. Maximizes on the monitor that contains the largest part of the window before it was minimized.</p> <p>The ALT-TAB key combination displays a window on the monitor that has the currently active window.</p>
owned window	on the same monitor as its owner.
submenu	Appears on the monitor that contains the largest part of the corresponding menu item.
context menu	Appears on the monitor where the right click occurred.
drop-down list	Appears on the monitor that contains the rectangle of the combo box.
dialog box	Appears on the monitor of the window that owns it. If it is defined with DS_CENTERMOUSE style, it appears on the monitor with the mouse. If it has no owner and the active window and the dialog box are in the same application, the dialog box appears on the monitor of the currently active window. If the dialog box has no owner and the active window is not in the same application as the dialog box, the dialog box appears on the primary monitor.
message box	Appears on the monitor of the window that owns it.

If a window straddles two monitors and one of the monitors is repositioned, the system positions the window on the monitor that contains the largest part of the original window.

An application will also typically need to position objects. For example, a window may need to be created on the same monitor as another window.

To position an object on a multiple monitor system

1. Determine the appropriate monitor.
2. Get the coordinates to the monitor.
3. Position the object using the coordinates.

Typically, you will position an object either on the primary monitor or on a monitor that has an object already on it. To identify the monitor for a given point, rectangle, or window, use [MonitorFromPoint](#), [MonitorFromRect](#), and [MonitorFromWindow](#).

To get the coordinates for the monitor, use [GetMonitorInfo](#), which provides both the work area and the entire monitor rectangle. Note that SM_CXSCREEN and SM_CYSCREEN always refer to the primary monitor, not necessarily the monitor that displays your application. Also, avoid SM_xxVIRTUALSCREEN because this centers your window on the virtual screen, not a monitor.

To center dialog boxes in a window's work area, use the DS_CENTER style. To center the dialog box to an application window, use [GetWindowRect](#). Windows automatically restricts menus and dialog boxes to a monitor. However, there can be a problem for custom menus, custom drop-down boxes, custom tool palettes, and the saved application position.

For an example of how to position objects correctly, see [Positioning Objects on a Multiple Display Setup](#).

Using SM_CXSCREEN and SM_CYSCREEN to determine the location of an application desktop toolbar (also called *appbar*) restricts the appbar to the primary monitor. To allow an appbar to be on any edge of any monitor, use the appropriate system metrics to calculate the edges of the monitors. Also, use the **GET_X_LPARAM** and **GET_Y_LPARAM** macros to extract the coordinates, otherwise the sign of the coordinates may be wrong. These macros are included in Windowsx.h.

The size of a full-screen window needs to change as it moves among monitors with different resolutions. To do this, the application must check what window it is on, using [MonitorFromWindow](#) or [MonitorFromPoint](#), and then use [GetMonitorInfo](#) to get the size of the monitor. As an alternative, you could use the **HMONITOR** from the DirectX **DirectDrawEnumerateEx** function. Then use [SetWindowPos](#) to position and size the window to cover the monitor.

A maximized window does not cover a taskbar that has the "Always on top" property. However, a full-screen window covers the taskbar--for example, in Microsoft PowerPoint

slide shows and games.

To save, and later restore, the position of a window when an application exits, use the [GetWindowPlacement](#) and [SetWindowPlacement](#) functions. However, check that the position is still valid before using it because the monitor could have been moved or removed from the system. The application displays the window on the primary monitor if the **HMONITOR** of a window is invalid.

The system tries to start an application on the monitor that contains its shortcut. So, one way to position an application is to have its shortcut on a desired monitor.

If you use [ShellExecute](#) or [ShellExecuteEx](#), supply an *hWnd* so the system will open any new windows on the same monitor as the calling application.

Note that the values for the [MINMAXINFO](#) structure are slightly altered for a system with multiple monitors.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Multiple Monitor Applications on Different Systems

Article • 01/07/2021

To have your multiple monitor-aware application work both on systems with and without multiple monitor support, link your application with Multimon.h. You must also define COMPILE_MULTIMON_STUBS in exactly one C file. If the system does not support multiple monitors, this returns default values from [GetSystemMetrics](#), and the multiple monitor functions act as if there is only one display. On multiple monitor systems, your application will work normally.

Because negative coordinates can happen easily in a multimonitor system, you should retrieve coordinates that are packed in the IParam by using the **GET_X_LPARAM** and **GET_Y_LPARAM** macros.

Do not use negative coordinates or coordinates larger than SM_CXSCREEN and SM_CYSCREEN to hide a window. Windows that use these limits to hide may appear on another monitor. Likewise, do not use these limits to keep a window visible because this can cause a window to snap to the primary monitor. It is best to reexamine existing applications for these problems. However, you can minimize problems in existing applications by running the application on the primary monitor or by keeping the primary monitor in the upper-left corner of the virtual screen.

Note that SM_CXMAXTRACK and SM_CYMAXTRACK are defined for the desktop, not just one monitor. Windows using these limits may need to be redefined.

A parent or related window might not be on the same monitor as a child window. To locate the monitor of a window, applications should use the [MonitorFromWindow](#) function.

To have a screen saver display on all monitors, link with the latest version of Scrnssave.lib. Otherwise, the screen saver may only appear on the primary monitor and leave the other monitors untouched. Screen savers linked with the latest Scrnssave.lib will work on both single and multiple monitor systems. To have a different screen saver on each monitor, use the multiple monitor functions to handle each monitor separately.

Input devices that deliver coordinates to the system in absolute coordinates, such as tablets, have their cursor input restricted to the primary monitor. To switch tablet input between monitors, see the instructions from the OEM.

To map mouse input that is sent in absolute coordinates to the entire virtual screen, use the [INPUT](#) structure with `MOUSEEVENTF_ABSOLUTE` and `MOUSEEVENTF_VIRTUALDESKTOP`.

The [BitBlt](#) function works well for multiple monitor systems. However, the [MaskBlt](#), [PlgBlt](#), [StretchBlt](#), and [TransparentBlt](#) functions will fail if the source and destination device contexts are different.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Multiple Monitor Considerations for Older Programs

Article • 01/07/2021

Generally, older applications do not need changes to work on multiple monitor systems. For most, the system will appear to have only one display. System metrics reflect the primary display and fullscreen applications show up on the primary. The system handles minimization, maximization, menus, and dialog boxes.

However, some programs will have problems. Some that could have problems are remote control applications, those that have direct hardware access, and those that have patched the GDI or DISPLAY driver. In these cases, the user may be required to disable any monitor beyond the primary monitor.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Multiple Display Monitors

Article • 01/07/2021

The examples in this section deal with different aspects of using Multiple Display Monitors.

- [Painting on a DC That Spans Multiple Displays](#)
- [Positioning Objects on a Multiple Display Setup](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Painting on a DC That Spans Multiple Displays

Article • 01/07/2021

To respond to a **WM_PAINT** message, use code like the following.

C++

```
case WM_PAINT:  
    hdc = BeginPaint(hwnd, &ps);  
    EnumDisplayMonitors(hdc, NULL, MyPaintEnumProc, 0);  
    EndPaint(hwnd, &ps);
```

To paint the top half of a window, use code like the following.

C++

```
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 monitor, use code like the following.

C++

```
hdc = GetDC(NULL);  
EnumDisplayMonitors(hdc, NULL, MyPaintScreenEnumProc, 0);  
ReleaseDC(NULL, hdc);
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Positioning Objects on a Multiple Display Setup

Article • 01/07/2021

The following sample code demonstrates how applications can correctly position objects on multiple displays. Note, do not assume that the RECT is based on the origin (0,0).

C++

```
#include <windows.h>
#include "multimon.h"

#define MONITOR_CENTER      0x0001      // center rect to monitor
#define MONITOR_CLIP        0x0000      // clip rect to monitor
#define MONITOR_WORKAREA   0x0002      // use monitor work area
#define MONITOR_AREA        0x0000      // use monitor entire area

//
// ClipOrCenterRectToMonitor
//
// The most common problem apps have when running on a
// multimonitor system is that they "clip" or "pin" windows
// based on the SM_CXSCREEN and SM_CYSCREEN system metrics.
// Because of app compatibility reasons these system metrics
// return the size of the primary monitor.
//
// This shows how you use the multi-monitor functions
// to do the same thing.
//
void ClipOrCenterRectToMonitor(LPRECT prc, UINT flags)
{
    HMONITOR hMonitor;
    MONITORINFO mi;
    RECT      rc;
    int       w = prc->right - prc->left;
    int       h = prc->bottom - prc->top;

    //
    // get the nearest monitor to the passed rect.
    //
    hMonitor = MonitorFromRect(prc, MONITOR_DEFAULTTONEAREST);

    //
    // get the work area or entire monitor rect.
    //
    mi.cbSize = sizeof(mi);
    GetMonitorInfo(hMonitor, &mi);

    if (flags & MONITOR_WORKAREA)
```

```
    rc = mi.rcWork;
else
    rc = mi.rcMonitor;

//
// center or clip the passed rect to the monitor rect
//
if (flags & MONITOR_CENTER)
{
    prc->left    = rc.left + (rc.right - rc.left - w) / 2;
    prc->top     = rc.top  + (rc.bottom - rc.top - h) / 2;
    prc->right   = prc->left + w;
    prc->bottom  = prc->top + h;
}
else
{
    prc->left    = max(rc.left, min(rc.right-w, prc->left));
    prc->top     = max(rc.top, min(rc.bottom-h, prc->top));
    prc->right   = prc->left + w;
    prc->bottom  = prc->top + h;
}

void ClipOrCenterWindowToMonitor(HWND hwnd, UINT flags)
{
    RECT rc;
    GetWindowRect(hwnd, &rc);
    ClipOrCenterRectToMonitor(&rc, flags);
    SetWindowPos(hwnd, NULL, rc.left, rc.top, 0, 0, SWP_NOSIZE |
    SWP_NOZORDER | SWP_NOACTIVATE);
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Multiple Display Monitors Reference

Article • 01/07/2021

The following elements provide support for multiple monitors:

- [Multiple Display Monitors Functions](#)
- [Multiple Display Monitors Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Multiple Display Monitors Functions

Article • 01/07/2021

The following functions provide support for multiple monitors.

[+] Expand table

Function	Description
EnumDisplayMonitors	Enumerates display monitors that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context.
GetMonitorInfo	Retrieves information about a display monitor.
MonitorEnumProc	An application-defined callback function that is called by the EnumDisplayMonitors function.
MonitorFromPoint	Retrieves a handle to the display monitor that contains a specified point.
MonitorFromRect	Retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.
MonitorFromWindow	Retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

EnumDisplayMonitors function (winuser.h)

Article10/13/2021

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

C++

```
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.

 Expand table

<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

 Expand table

Requirement	Value
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](#)

GetMonitorInfoA function (winuser.h)

Article02/22/2024

The **GetMonitorInfo** function retrieves information about a display monitor.

Syntax

C++

```
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 that 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

 [Expand table](#)

Requirement	Value
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](#)

MONITORENUMPROC callback function (winuser.h)

10/01/2025

An application-defined callback function used with the [EnumDisplayMonitors](#) function. The **MONITORENUMPROC** type defines a pointer to this callback function. *MonitorEnumProc* is a placeholder for the application-defined function name.

Syntax

C++

```
MONITORENUMPROC Monitorenumproc;

BOOL Monitorenumproc(
    HMONITOR unnamedParam1,
    HDC unnamedParam2,
    LPRECT unnamedParam3,
    LPARAM unnamedParam4
)
{...}
```

Parameters

unnamedParam1

Type: **HMONITOR**

A handle to the display monitor. This value will always be non-NULL. This parameter is typically named *hMonitor*.

unnamedParam2

Type: **HDC**

A handle to a device context. This parameter is typically named *hdcMonitor*.

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

Type: **LPRECT**

A pointer to a [RECT](#) structure. This parameter is typically named *lprcMonitor*.

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

Type: **LPARAM**

Application-defined data that [EnumDisplayMonitors](#) passes directly to the enumeration function. This parameter is typically named *dwData*.

Return value

Type: **BOOL**

To continue the enumeration, return **TRUE**.

To stop the enumeration, return **FALSE**.

Remarks

!**Note**

The parameters are defined in the header with no names: `typedef BOOL (CALLBACK* MONITORENUMPROC)(HMONITOR, HDC, LPRECT, LPARAM);`. Therefore, the syntax block lists them as `unnamedParam1 - unnamedParam4`. You can name these parameters anything in your app. However, they are usually named as shown in the parameter descriptions.

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

 [Expand table](#)

Requirement	Value
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)

Article 11/19/2022

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>MONITOR_DEFAULTTONULL</code> 0x00000000	Returns NULL.
<code>MONITOR_DEFAULTTOPRIMARY</code> 0x00000001	Returns a handle to the primary display monitor.
<code>MONITOR_DEFAULTTONEAREST</code> 0x00000002	Returns a handle to the display monitor that is nearest to the point.

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

 Expand table

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>MONITOR_DEFAULTTONEAREST</code>	Returns a handle to the display monitor that is nearest to the rectangle.
<code>MONITOR_DEFAULTTONULL</code>	Returns NULL.
<code>MONITOR_DEFAULTTOPRIMARY</code>	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

 Expand table

Requirement	Value
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)

Article 10/13/2021

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

C++

```
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.

 Expand table

Value	Meaning
<code>MONITOR_DEFAULTTONEAREST</code>	Returns a handle to the display monitor that is nearest to the window.
<code>MONITOR_DEFAULTTONULL</code>	Returns NULL.
<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

 Expand table

Requirement	Value
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](#)

Multiple Display Monitors Structures

Article • 01/07/2021

The following structures are used with the monitor functions:

- [MONITORINFO](#)
- [MONITORINFOEX](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

MONITORINFO structure (winuser.h)

Article 02/22/2024

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

C++

```
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.

[+] Expand table

Value	Meaning
MONITORINFOF_PRIMARY	This is the primary display monitor.

Requirements

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

MONITORINFOEXA structure (winuser.h)

Article02/22/2024

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

C++

```
typedef struct tagMONITORINFOEXA : tagMONITORINFO {
    CHAR szDevice[CCHDEVICENAME];
} MONITORINFOEXA, *LPMONITORINFOEXA;
```

Inheritance

The **MONITORINFOEXA** structure implements [tagMONITORINFO](#).

Members

`szDevice[CCHDEVICENAME]`

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 that 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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Painting and Drawing

Article • 01/07/2021

This overview describes how the system manages output to the screen and explains what applications must do to draw in a window. In particular, this overview describes *display device contexts* (or, more simply, *display DCs*) and how to prepare and use them. The overview does not explain how to use graphics device interface (GDI) functions to generate output, nor does it explain how to print.

- [About Painting and Drawing](#)
- [Using the WM_PAINT Message](#)
- [Using the GetDC Function](#)
- [Painting and Drawing Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Painting and Drawing

Article • 01/07/2021

Nearly all applications use the screen to display the data they manipulate. An application paints images, draws figures, and writes text so that the user can view data as it is created, edited, and printed. Microsoft Windows provides rich support for painting and drawing, but, because of the nature of multitasking operating systems, applications must cooperate with one another when accessing the screen.

To keep all applications functioning smoothly and cooperatively, the system manages all output to the screen. Applications use windows as their primary output device rather than the screen itself. The system supplies display device contexts that uniquely correspond to the windows. Applications use display device contexts to direct their output to the specified windows. Drawing in a window (directing output to it) prevents an application from interfering with the output of other applications and allows applications to coexist with one another while still taking full advantage of the graphics capabilities of the system.

- [When to Draw in a Window](#)
- [The WM_PAINT Message](#)
- [Drawing Without the WM_PAINT Message](#)
- [Window Coordinate System](#)
- [Window Regions](#)
- [Window Background](#)
- [Minimized Windows](#)
- [Resized Windows](#)
- [Nonclient Area](#)
- [Child Window Update Region](#)
- [Display Devices](#)
- [Window Update Lock](#)
- [Accumulated Bounding Rectangle](#)

Feedback

Was this page helpful?



When to Draw in a Window

Article • 01/07/2021

An application draws in a window at a variety of times: when first creating a window, when changing the size of the window, when moving the window from behind another window, when minimizing or maximizing the window, when displaying data from an opened file, and when scrolling, changing, or selecting a portion of the displayed data.

The system manages actions such as moving and sizing a window. If an action affects the content of the window, the system marks the affected portion of the window as ready for updating and, at the next opportunity, sends a **WM_PAINT** message to the window procedure of the window. The message is a signal to the application to determine what must be updated and to carry out the necessary drawing.

Some actions are managed by the application, such as displaying open files and selecting displayed data. For these actions, an application can mark for updating the portion of the window affected by the action, causing a **WM_PAINT** message to be sent at the next opportunity. If an action requires immediate feedback, the application can draw while the action takes place, without waiting for **WM_PAINT**. For example, a typical application highlights the area the user selects rather than waiting for the next **WM_PAINT** message to update the area.

In all cases, an application can draw in a window as soon as it is created. To draw in the window, the application must first retrieve a handle to a display device context for the window. Ideally, an application carries out most of its drawing operations during the processing of **WM_PAINT** messages. In this case, the application retrieves a display device context by calling the **BeginPaint** function. If an application draws at any other time, such as from within **WinMain** or during the processing of keyboard or mouse messages, it calls the **GetDC** or **GetDCEx** function to retrieve the display DC.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

The WM_PAINT Message

Article • 01/07/2021

Typically, an application draws in a window in response to a [WM_PAINT](#) message. The system sends this message to a window procedure when changes to the window have altered the content of the client area. The system sends the message only if there are no other messages in the application message queue.

Upon receiving a [WM_PAINT](#) message, an application can call [BeginPaint](#) to retrieve the display device context for the client area and use it in calls to GDI functions to carry out whatever drawing operations are necessary to update the client area. After completing the drawing operations, the application calls the [EndPaint](#) function to release the display device context.

Before [BeginPaint](#) returns the display device context, the system prepares the device context for the specified window. It first sets the clipping region for the device context to be equal to the intersection of the portion of the window that needs updating and the portion that is visible to the user. Only those portions of the window that have changed are redrawn. Attempts to draw outside this region are clipped and do not appear on the screen.

The system can also send [WM_NCPAINT](#) and [WM_ERASEBKGND](#) messages to the window procedure before [BeginPaint](#) returns. These messages direct the application to draw the nonclient area and window background. The *nonclient area* is the part of a window that is outside of the client area. The area includes features such as the title bar, window menu (also known as the [System](#) menu), and scroll bars. Most applications rely on the default window function, [DefWindowProc](#), to draw this area and therefore pass the [WM_NCPAINT](#) message to this function. The *window background* is the color or pattern that a window is filled with before other drawing operations begin. The background covers any images previously in the window or on the screen under the window. If a window belongs to a window class having a class background brush, the [DefWindowProc](#) function draws the window background automatically.

[BeginPaint](#) fills a [PAINTSTRUCT](#) structure with information such as the dimensions of the portion of the window to be updated and a flag indicating whether the window background has been drawn. The application can use this information to optimize drawing. For example, it can use the dimensions of the update region, specified by the **rcPaint** member, to limit drawing to only those portions of the window that need updating. If an application has very simple output, it can ignore the update region and draw in the entire window, relying on the system to discard (clip) any unneeded output.

Because the system clips drawing that extends outside the clipping region, only drawing that is in the update region is visible.

BeginPaint sets the update region of a window to **NULL**. This clears the region, preventing it from generating subsequent **WM_PAINT** messages. If an application processes a **WM_PAINT** message but does not call **BeginPaint** or otherwise clear the update region, the application continues to receive **WM_PAINT** messages as long as the region is not empty. In all cases, an application must clear the update region before returning from the **WM_PAINT** message.

After the application finishes drawing, it should call **EndPaint**. For most windows, **EndPaint** releases the display device context, making it available to other windows. **EndPaint** also shows the caret, if it was previously hidden by **BeginPaint**. **BeginPaint** hides the caret to prevent drawing operations from corrupting it.

- [The Update Region](#)
- [Invalidating and Validating the Update Region](#)
- [Retrieving the Update Region](#)
- [Excluding the Update Region](#)
- [Synchronous and Asynchronous Drawing](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

The Update Region

Article • 01/07/2021

The *update region* identifies the portion of a window that is out-of-date or invalid and in need of redrawing. The system uses the update region to generate **WM_PAINT** messages for applications and to minimize the time applications spend bringing the contents of their windows up to date. The system adds only the invalid portion of the window to the update region, requiring only that portion to be drawn.

When the system determines that a window needs updating, it sets the dimensions of the update region to the invalid portion of the window. Setting the update region does not immediately cause the application to draw. Instead, the application continues retrieving messages from the application message queue until no messages remain. The system then checks the update region, and if the region is not empty (non-NULL), it sends a **WM_PAINT** message to the window procedure.

An application can use the update region to generate its **WM_PAINT** messages. For example, an application that loads data from open files typically sets the update region while loading so that new data is drawn during processing of the next **WM_PAINT** message. In general, an application should not draw at the time its data changes, but route all drawing operations through the **WM_PAINT** message.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Invalidating and Validating the Update Region

Article • 01/07/2021

An application invalidates a portion of a window and sets the update region by using the [InvalidateRect](#) or [InvalidateRgn](#) function. These functions add the specified rectangle or region (in client coordinates) to the update region, combining the rectangle or region with anything the system or the application may have previously added to the update region.

The [InvalidateRect](#) and [InvalidateRgn](#) functions do not generate [WM_PAINT](#) messages. Instead, the system accumulates the changes made by these functions and its own changes while a window processes other messages in its message queue. By accumulating changes, a window processes all changes at once instead of updating bits and pieces one step at a time.

The [ValidateRect](#) and [ValidateRgn](#) functions validate a portion of the window by removing a specified rectangle or region from the update region. These functions are typically used when the window has updated a specific part of the screen in the update region before receiving the [WM_PAINT](#) message.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Retrieving the Update Region

Article • 01/07/2021

The [GetUpdateRect](#) and [GetUpdateRgn](#) functions retrieve the current update region for the window. GetUpdateRect retrieves the smallest rectangle (in logical coordinates) that encloses the entire update region. GetUpdateRgn retrieves the update region itself. These functions can be used to calculate the current size of the update region to determine where to carry out a drawing operation.

[BeginPaint](#) also retrieves the dimensions of the smallest rectangle enclosing the current update region, copying the dimensions to the **rcPaint** member in the [PAINTSTRUCT](#) structure. Because [BeginPaint](#) validates the update region, any call to [GetUpdateRect](#) and [GetUpdateRgn](#) immediately after a call to [BeginPaint](#) returns an empty update region.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Excluding the Update Region

Article • 01/07/2021

The [ExcludeUpdateRgn](#) function excludes the update region from the clipping region for the display device context. This function is useful when drawing in a window other than when a WM_PAINT message is processing. It prevents drawing in the areas that will be updated during the next WM_PAINT message.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Synchronous and Asynchronous Drawing

Article • 01/07/2021

Most drawing carried out during processing of the [WM_PAINT](#) message is asynchronous; that is, there is a delay between the time a portion of the window is invalidated and the time WM_PAINT is sent. During the delay, the application typically retrieves messages from the queue and carries out other tasks. The reason for the delay is that the system generally treats drawing in a window as a low-priority operation and works as though user-input messages and messages that may affect the position or size of a window will be processed before WM_PAINT .

In some cases, it is necessary for an application to draw synchronously that is, draw in the window immediately after invalidating a portion of the window. A typical application draws its main window immediately after creating the window to signal the user that the application has started successfully. The system draws some control windows synchronously, such as buttons, because such windows serve as the focus for user input. Although any window with a simple drawing routine can be drawn synchronously, all such drawing should be done quickly and should not interfere with the application's ability to respond to user input.

The [UpdateWindow](#) and [RedrawWindow](#) functions allow for synchronous drawing. [UpdateWindow](#) sends a [WM_PAINT](#) message directly to the window if the update region is not empty. [RedrawWindow](#) also sends a [WM_PAINT](#) message, but gives the application greater control over how to draw the window, such as whether to draw the nonclient area and window background or whether to send the message regardless of whether the update region is empty. These functions send the [WM_PAINT](#) message directly to the window, regardless of the number of other messages in the application message queue.

Any window requiring time-consuming drawing operations should be drawn asynchronously to prevent pending messages from being blocked as the window is drawn. Also, any application that frequently invalidates small portions of a window should allow these invalid portions to consolidate into a single asynchronous [WM_PAINT](#) message, rather than a series of synchronous [WM_PAINT](#) messages.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing Without the WM_PAINT Message

Article • 01/07/2021

Although applications carry out most drawing operations while the [WM_PAINT](#) message is processing, it is sometimes more efficient for an application to draw directly in a window without relying on the [WM_PAINT](#) message. This can be useful when the user needs immediate feedback, such as when selecting text and dragging or sizing an object. In such cases, the application usually draws while processing keyboard or mouse messages.

To draw in a window without using a [WM_PAINT](#) message, the application uses the [GetDC](#) or [GetDCEx](#) function to retrieve a display device context for the window. With the display device context, the application can draw in the window and avoid intruding into other windows. When the application has finished drawing, it calls the [ReleaseDC](#) function to release the display device context for use by other applications.

When drawing without using a [WM_PAINT](#) message, the application usually does not invalidate the window. Instead, it draws in such a fashion that it can easily restore the window and remove the drawing. For example, when the user selects text or an object, the application typically draws the selection by inverting whatever is already in the window. The application can remove the selection and restore the original contents of the window by simply inverting again.

The application is responsible for carefully managing any changes it makes to the window. In particular, if an application draws a selection and an intervening [WM_PAINT](#) message occurs, the application must ensure that any drawing done during the message does not corrupt the selection. To avoid this, many applications remove the selection, carry out usual drawing operations, and then restore the selection when drawing is complete.

Feedback

Was this page helpful?

 Yes

 No

Window Coordinate System

Article • 11/19/2022

The coordinate system for a window is based on the coordinate system of the display device. The basic unit of measure is the device unit (typically, the pixel). Points on the screen are described by x- and y-coordinate pairs. The x-coordinates increase to the right; y-coordinates increase from top to bottom. The origin (0,0) for the system depends on the type of coordinates being used.

The system and applications specify the position of a window on the screen in *screen coordinates*. For screen coordinates, the origin is the upper-left corner of the screen. The full position of a window is often described by a [RECT](#) structure containing the screen coordinates of two points that define the upper-left and lower-right corners of the window.

The system and applications specify the position of points in a window by using *client coordinates*. The origin in this case is the upper-left corner of the window or client area. Client coordinates ensure that an application can use consistent coordinate values while drawing in the window, regardless of the position of the window on the screen.

The dimensions of the client area are also described by a [RECT](#) structure that contains client coordinates for the area. In all cases, the upper-left coordinate of the rectangle is included in the window or client area, while the lower-right coordinate is excluded. Graphics operations in a window or client area are excluded from the right and lower edges of the enclosing rectangle.

Occasionally, applications may be required to map coordinates in one window to those of another window. An application can map coordinates by using the [MapWindowPoints](#) function. If one of the windows is the desktop window, the function effectively converts screen coordinates to client coordinates and vice versa; the desktop window is always specified in screen coordinates.

Feedback

Was this page helpful?

 Yes

 No

Window Regions

Article • 01/07/2021

In addition to the update region, every window has a *visible region* that defines the window portion visible to the user. The system changes the visible region for the window whenever the window changes size or whenever another window is moved such that it obscures or exposes a portion of the window. Applications cannot change the visible region directly, but the system automatically uses the visible region to create the clipping region for any display device context retrieved for the window.

The *clipping region* determines where the system permits drawing. When the application retrieves a display device context using the [BeginPaint](#), [GetDC](#), or [GetDCEx](#) function, the system sets the clipping region for the device context to the intersection of the visible region and the update region. Applications can change the clipping region by using functions such as [SetWindowRgn](#), [SelectClipPath](#) and [SelectClipRgn](#), to further limit drawing to a particular portion of the update area.

The `WS_CLIPCHILDREN` and `WS_CLIPSIBLINGS` styles further specify how the system calculates the visible region for a window. If a window has one or both of these styles, the visible region excludes any child window or sibling windows (windows having the same parent window). Therefore, drawing that would otherwise intrude in these windows will always be clipped.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Window Background

Article • 01/07/2021

The window background is the color or pattern used to fill the client area before a window begins drawing. The window background covers whatever was on the screen before the window was moved there, erasing existing images and preventing the application's new output from being mixed with unrelated information.

The system paints the background for a window or gives the window the opportunity to do so by sending it a [WM_ERASEBKGND](#) message when the application calls [BeginPaint](#). If an application does not process the message but passes it to [DefWindowProc](#), the system erases the background by filling it with the pattern in the background brush specified by the window's class. If the brush is not valid or the class has no background brush, the system sets the [fErase](#) member in the [PAINTSTRUCT](#) structure that [BeginPaint](#) returns, but carries out no other action. The application then has a second chance to draw the window background, if necessary.

If it processes [WM_ERASEBKGND](#), the application should use the message's *wParam* parameter to draw the background. This parameter contains a handle to the display device context for the window. After drawing the background, the application should return a nonzero value. This ensures that [BeginPaint](#) does not erroneously set the [fErase](#) member of the [PAINTSTRUCT](#) structure to a nonzero value (indicating the background should be erased) when the application processes the subsequent [WM_PAINT](#) message.

An application can define a class background brush by assigning a brush handle or a system color value to the [hbrBackground](#) member of the [WNDCLASS](#) structure when registering the class with the [RegisterClass](#) function. The [GetStockObject](#) or [CreateSolidBrush](#) function can be used to create a brush handle. A system color value can be one of those defined for the [SetSysColors](#) function. (The value must be increased by one before it is assigned to the member.)

An application can process the [WM_ERASEBKGND](#) message even though a class background brush is defined. This is typical in applications that enable the user to change the window background color or pattern for a specified window without affecting other windows in the class. In such cases, the application must not pass the message to [DefWindowProc](#).

It is not necessary for an application to align brushes, because the system draws the brush using the window origin as the point of reference. Given this, the user can move the window without affecting the alignment of pattern brushes.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Minimized Windows

Article • 01/07/2021

The system reduces an application's main window (overlapping style) to a minimized window when the user clicks Minimize from the window menu or the application calls the [ShowWindow](#) function and specifies a value such as SW_MINIMIZE. Minimizing a window speeds up system performance by reducing the amount of work an application must do when updating its main window.

For a typical application, the system draws an icon, called the class icon, when the window is minimized, labeling the icon with the name of the window. The class icon, a static image that represents the application, is specified by the application when it registers the window class. The application assigns a handle to the class icon to the hIcon member of [WNDCLASS](#) before calling [RegisterClass](#). The application can use the [LoadIcon](#) function to retrieve the icon handle.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Resized Windows

Article • 01/07/2021

The system changes the size of a window when the user chooses window menu commands, such as Size and Maximize, or when the application calls functions, such as the [SetWindowPos](#) function. When a window changes size, the system assumes that the contents of the previously exposed portion of the window are not affected and need not be redrawn. The system invalidates only the newly exposed portion of the window, which saves time when the eventual [WM_PAINT](#) message is processed by the application. In this case, [WM_PAINT](#) is not generated when the size of the window is reduced.

For some windows, any change to the size of the window invalidates the contents. For example, a clock application that adapts the face of the clock to fit neatly within its window must redraw the clock whenever the window changes size. To force the system to invalidate the entire client area of the window when a vertical, horizontal, or both vertical and horizontal change is made, an application must specify the CS_VREDRAW or CS_HREDRAW style, or both, when registering the window class. Any window belonging to a window class having these styles is invalidated each time the user or the application changes the size of the window.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Nonclient Area

Article • 01/07/2021

The system sends a [WM_NCPAINT](#) message to the window whenever a part of the nonclient area of the window, such as the title bar, menu bar, or window frame, must be updated. The system may also send other messages to direct a window to update a portion of its client area; for example, when a window becomes active or inactive, it sends the [WM_NCACTIVATE](#) message to update its title bar. In general, processing these messages for standard windows is not recommended, because the application must be able to draw all the required parts of the nonclient area for the window. For this reason, most applications pass these messages to [DefWindowProc](#) for default processing.

An application that creates custom nonclient areas for its windows must process these messages. When doing so, the application must use a window device context to carry out drawing in the window. The *window device context* enables the application to draw in all portions of the window, including the nonclient area. An application retrieves a window device context by using the [GetWindowDC](#) or [GetDCEx](#) function and, when drawing is complete, must release the window device context by using the [ReleaseDC](#) function.

The system maintains an update region for the nonclient area. When an application receives a [WM_NCPAINT](#) message, the *wParam* parameter contains a handle to a region defining the dimensions of the update region. The application can use the handle to combine the update region with the clipping region for the window device context. The system does not automatically combine the update region when retrieving the window device context unless the application uses [GetDCEx](#) and specifies both the region handle and the DCX_INTERSECTRGN flag. If the application does not combine the update region, only drawing operations that would otherwise extend outside the window are clipped. The application is not responsible for clearing the update region, regardless of whether it uses the region.

If an application processes the [WM_NCACTIVATE](#) message, after processing it must return **TRUE** to direct the system to complete the change of active window. If the window is minimized when the application receives the [WM_NCACTIVATE](#) message, it should pass the message to [DefWindowProc](#). In such cases, the default function redraws the label for the icon.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Child Window Update Region

Article • 01/07/2021

A child window is a window with the WS_CHILD or WS_CHILDWINDOW style. Like other window styles, child windows receive [WM_PAINT](#) messages to prompt updating. Each child window has an update region, which either the system or the application can set to generate eventual [WM_PAINT](#) messages.

A child window's update and visible regions are affected by the child's parent window; this is not true for windows of other styles. The system often sets the child window's update region when it sets the parent window's update region, causing the child window to receive [WM_PAINT](#) messages when the parent window receives them. The system limits the location of the child window's visible region to within the client area of the parent window and clips any portion of the child window moved outside the parent window.

The system sets the update region for a child window whenever part of the parent window's update region includes a portion of the child window. In such cases, the system first sends a [WM_PAINT](#) message to the parent window and then sends a message to the child window, allowing the child to restore any portions of the window that the parent may have drawn over.

The system does not set the parent's update region when the child's is set. An application cannot generate a [WM_PAINT](#) message for the parent window by invalidating the child window. Similarly, an application cannot generate a [WM_PAINT](#) message for the child by invalidating a portion of the parent's client area that lies entirely under the child window. In such cases, neither window receives a [WM_PAINT](#) message.

An application can prevent a child window's update region from being set when the parent window's is set by specifying the WS_CLIPCHILDREN style when creating the parent window. When this style is set, the system excludes the child windows from the parent's visible region and therefore ignores any portion of the update region that may contain the child windows. When the application paints in the parent window, any drawing that would cover the child window is clipped, making a subsequent [WM_PAINT](#) message to the child window unnecessary.

The update and visible regions of a child window are also affected by the child window's siblings. Sibling windows are any windows that have a common parent window. If sibling windows overlap, then setting the update region for one affects the update region of another, causing [WM_PAINT](#) messages to be sent to both windows. If a window in the

parent chain is composited (a window with `WX_EX_COMPOSITED`), sibling windows receive `WM_PAINT` messages in the reverse order of their position in the Z order. Given this, the window highest in the Z order (on the top) receives its `WM_PAINT` message last, and vice versa. If a window in the parent chain is not composited, sibling windows receive `WM_PAINT` messages in Z order.

Sibling windows are not automatically clipped. One sibling can draw over another overlapping sibling even if the window that is drawing has a lower position in the Z order. An application can prevent this by specifying the `WS_CLIPSIBLINGS` style when creating the windows. When this style is set, the system excludes all portions of an overlapping sibling window from a window's visible region if the overlapping sibling window has a higher position in the Z order.

ⓘ Note

The update and visible regions for windows that have the `WS_POPUP` or `WS_POPUPWINDOW` style are not affected by their parent windows.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Display Devices

Article • 01/07/2021

Before painting, the system must prepare the display device for drawing operations. A display device context defines a set of graphic objects and their associated attributes, and the graphic modes that affect output. The system prepares each display device context for output to a window, setting the drawing objects, colors, and modes for the window instead of the display device. When the application supplies the display device context through calls to GDI functions, GDI uses the information in the context to generate output in the specified window without intruding on other windows or other parts of the screen.

The system provides five kinds of display device contexts.

[+] Expand table

Type	Meaning
common	Permits drawing in the client area of a specified window.
class	Permits drawing in the client area of a specified window.
parent	Permits drawing anywhere in the window. Although the parent device context also permits drawing in the parent window, it is not intended to be used in this way.
private	Permits drawing in the client area of a specified window.
window	Permits drawing anywhere in the window.

The system supplies a common, class, parent, or private device context to a window based on the type of display device context specified in that window's class style. The system supplies a window device context only when the application explicitly requests one for example, by calling the [GetWindowDC](#) or [GetDCEx](#) function. In all cases, an application can use the [WindowFromDC](#) function to determine which window a display DC currently represents.

This section provides information on the following topics.

- [Display Device Context Cache](#)
- [Display Device Context Defaults](#)
- [Common Display Device Contexts](#)
- [Private Display Device Contexts](#)

- Parent Display Device Contexts
 - Class Display Device Contexts
 - Window Display Device Contexts
 - Parent Display Device Contexts
-

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Display Device Context Cache

Article • 01/07/2021

The system maintains a cache of display device contexts that it uses for common, parent, and window device contexts. The system retrieves a device context from the cache whenever an application calls the [GetDC](#) or [BeginPaint](#) function; the system returns the DC to the cache when the application subsequently calls the [ReleaseDC](#) or [EndPaint](#) function.

There is no predetermined limit on the amount of device contexts that a cache can hold; the system creates a new display device context for the cache if none is available. Given this, an application can have more than five active device contexts from the cache at a time. However, the application must continue to release these device contexts after use. Because new display device contexts for the cache are allocated in the application's heap space, failing to release the device contexts eventually consumes all available heap space. The system indicates this failure by returning an error when it cannot allocate space for the new device context. Other functions unrelated to the cache may also return errors.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Display Device Context Defaults

Article • 01/07/2021

Upon first creating a display device context, the system assigns default values for the attributes (that is, drawing objects, colors, and modes) that comprise the device context. The following table shows the default values for the attributes of a display device context.

[Expand table](#)

Attribute	Default value
Background color	Background color setting from Control Panel (typically, white).
Background mode	OPAQUE
Bitmap	None
Brush	WHITE_BRUSH
Brush origin	(0,0)
Clipping region	Entire window or client area with the update region clipped, as appropriate. Child and pop-up windows in the client area may also be clipped.
Palette	DEFAULT_PALETTE
Current pen position	(0,0)
Device origin	Upper left corner of the window or the client area.
Drawing mode	R2_COPYPEN
Font	SYSTEM_FONT
Intercharacter spacing	0
Mapping mode	MM_TEXT
Pen	BLACK_PEN
Polygon -fill mode	ALTERNATE
Stretch mode	BLACKONWHITE

Attribute	Default value
Text color	Text color setting from Control Panel (typically, black).
Viewport extent	(1,1)
Viewport origin	(0,0)
Window extent	(1,1)
Window origin	(0,0)

An application can modify the values of the display device context attributes by using selection and attribute functions, such as [SelectObject](#), [SetMapMode](#), and [SetTextColor](#). For example, an application can modify the default units of measure in the coordinate system by using [SetMapMode](#) to change the mapping mode.

Changes to the attribute values of a common, parent, or window device context are not permanent. When an application releases these device contexts, the current selections, such as mapping mode and clipping region, are lost as the context is returned to the cache. Changes to a class or private device context persist indefinitely. To restore them to their original defaults, an application must explicitly set each attribute.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Common Display Device Contexts

Article • 01/07/2021

A *common device context* is used for drawing in the client area of the window. The system provides a common device context by default for any window whose window class does not explicitly specify a display device context style. Common device contexts are typically used with windows that can be drawn without extensive changes to the device context attributes. Common device contexts are convenient because they do not require additional memory or system resources, but they can be inconvenient if the application must set up many attributes before using them.

The system retrieves all common device contexts from the display device context cache. An application can retrieve a common device context immediately after the window is created. Because the common device context is from the cache, the application must always release the device context as soon as possible after drawing. After the common device context is released, it is no longer valid and the application must not attempt to draw with it. To draw again, the application must retrieve a new common device context, and continue to retrieve and release a common device context each time it draws in the window. If the application retrieves the device context handle by using the [GetDC](#) function, it must use the [ReleaseDC](#) function to release the handle. Similarly, for each [BeginPaint](#) function, the application must use a corresponding [EndPaint](#) function.

When the application retrieves the device context, the system adjusts the origin so that it aligns with the upper left corner of the client area. It also sets the clipping region so that output to the device context is clipped to the client area. Any output that would otherwise appear outside the client area is clipped. If the application retrieves the common device context by using [BeginPaint](#), the system also includes the update region in the clipping region to further restrict the output.

When an application releases a common device context, the system restores the default values for the attributes of the device context. An application that modifies attribute values must do so each time it retrieves a common device context. Releasing the device context releases any drawing objects the application may have selected into it, so the application need not release these objects before releasing the device context. In all cases, an application must never assume that the common device context retains nondefault selections after being released.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Private Display Device Contexts

Article • 01/07/2021

A *private device context* enables an application to avoid retrieving and initializing a display device context each time the application must draw in a window. Private device contexts are useful for windows that require many changes to the values of the attributes of the device context to prepare it for drawing. Private device contexts reduce the time required to prepare the device context and therefore the time needed to carry out drawing in the window.

An application directs the system to create a private device context for a window by specifying the CS_OWNDC style in the window class. The system creates a unique private device context each time it creates a new window belonging to the class. Initially, the private device context has the same default values for attributes as a common device context, but the application can modify these at any time. The system preserves changes to the device context for the life of the window or until the application makes additional changes.

An application can retrieve a handle to the private device context by using the [GetDC](#) function any time after the window is created. The application must retrieve the handle only once. Thereafter, it can keep and use the handle any number of times. Because a private device context is not part of the display device context cache, an application need never release the device context by using the [ReleaseDC](#) function.

The system automatically adjusts the device context to reflect changes to the window, such as moving or sizing. This ensures that any overlapping windows are always properly clipped; that is, no action is required by the application to ensure clipping. However, the system does not revise the device context to include the update region. Therefore, when processing a [WM_PAINT](#) message, the application must incorporate the update region either by calling [BeginPaint](#) or by retrieving the update region and intersecting it with the current clipping region. If the application does not call [BeginPaint](#), it must explicitly validate the update region by using the [ValidateRect](#) or [ValidateRgn](#) function. If the application does not validate the update region, the window receives an endless series of [WM_PAINT](#) messages.

Because [BeginPaint](#) hides the caret if a window is showing it, an application that calls [BeginPaint](#) should also call the [EndPaint](#) function to restore the caret. [EndPaint](#) has no other effect on a private device context.

Although a private device context is convenient to use, it is memory-intensive in terms of system resources, requiring 800 or more bytes to store. Private device contexts are

recommended when performance considerations outweigh storage costs.

The system includes the private device context when sending the [WM_ERASEBKGND](#) message to the application. The current selections of the private device context, including mapping mode, are in effect when the application or the system processes these messages. To avoid undesirable effects, the system uses logical coordinates when erasing the background; for example, it uses the [GetClipBox](#) function to retrieve the logical coordinates of the area to erase and passes these coordinates to the [FillRect](#) function. Applications that process these messages can use similar techniques.

An application can use the [GetDCEx](#) function to force the system to return a common device context for the window that has a private device context. This is useful for carrying out quick touch-ups to a window without changing the current values of the attributes of the private device context.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Class Display Device Contexts

Article • 01/07/2021

By using a *class device context*, an application can use a single display device context for every window belonging to a specified class. Class device contexts are often used with control windows that are drawn using the same attribute values. Like private device contexts, class device contexts minimize the time required to prepare a device context for drawing.

The system supplies a class device context for a window if it belongs to a window class having the CS_CLASSDC style. The system creates the device context when creating the first window belonging to the class and then uses the same device context for all subsequently created windows in the class. Initially, the class device context has the same default values for attributes as a common device context, but the application can modify these at any time. The system preserves all changes, except for the clipping region and device origin, until the last window in the class has been destroyed. A change made for one window applies to all windows in that class.

An application can retrieve the handle to the class device context by using the [GetDC](#) function any time after the first window has been created. The application can keep and use the handle without releasing it because the class device context is not part of the display device context cache. If the application creates another window in the same window class, the application must retrieve the class device context again. Retrieving the device context sets the correct device origin and clipping region for the new window. After the application retrieves the class device context for a new window in the class, the device context can no longer be used to draw in the original window without again retrieving it for that window. In general, each time it must draw in a window, an application must explicitly retrieve the class device context for the window.

Applications that use class device contexts should always call [BeginPaint](#) when processing a [WM_PAINT](#) message. The function sets the correct device origin and clipping region for the window, and incorporates the update region. The application should also call [EndPaint](#) to restore the caret if [BeginPaint](#) hid it. [EndPaint](#) has no other effect on a class device context.

The system passes the class device context when sending the [WM_ERASEBKGND](#) message to the application, permitting the current attribute values to affect any drawing carried out by the application or the system when processing this message. As it could with a window having a private device context, an application can use [GetDCEx](#) to force the system to return a common device context for the window that has a class device context.

Using class device contexts is not recommended.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Window Display Device Contexts

Article • 01/07/2021

A *window device context* enables an application to draw anywhere in a window, including the nonclient area. Window device contexts are typically used by applications that process the **WM_NCPAINT** and **WM_NCACTIVATE** messages for windows with custom nonclient areas. Using a window device context is not recommended for any other purpose.

An application can retrieve a window device context by using the **GetWindowDC** or **GetDCEx** function with the DCX_WINDOW option specified. The function retrieves a window device context from the display device context cache. A window that uses a window device context must release it after drawing by using the **ReleaseDC** function as soon as possible. Window device contexts are always from the cache; the CS_OWNDC and CS_CLASSDC class styles do not affect the device context.

When an application retrieves a window device context, the system sets the device origin to the upper left corner of the window instead of the upper left corner of the client area. It also sets the clipping region to include the entire window, not just the client area. The system sets the current attribute values of a window device context to the same default values as a common device context. An application can change the attribute values, but the system does not preserve any changes when the device context is released.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Parent Display Device Contexts

Article • 01/07/2021

A *parent device context* enables an application to minimize the time necessary to set up the clipping region for a window. An application typically uses parent device contexts to speed up drawing for control windows without requiring a private or class device context. For example, the system uses parent device contexts for push button and edit controls. Parent device contexts are intended for use with child windows only, never with top-level or pop-up windows.

An application can specify the CS_PARENTDC style to set the clipping region of the child window to that of the parent window so that the child can draw in the parent. Specifying CS_PARENTDC enhances an application's performance because the system doesn't need to keep recalculating the visible region for each child window.

Attribute values set by the parent window are not preserved for the child window; for example, the parent window cannot set the brush for its child windows. The only property preserved is the clipping region. The window must clip its own output to the limits of the window. Because the clipping region for the parent device context is identical to the parent window, the child window can potentially draw over the entire parent window, but the parent device context must not be used in this way.

The system ignores the CS_PARENTDC style if the parent window uses a private or class device context, if the parent window clips its child windows, or if the child window clips its child windows or sibling windows.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Window Update Lock

Article • 01/07/2021

A *window update lock* is a temporary suspension of drawing in a window. The system uses the lock to prevent other windows from drawing over the tracking rectangle whenever the user moves or sizes a window. Applications can use the lock to prevent drawing if they carry out similar moving or sizing operations with their own windows.

An application uses the [LockWindowUpdate](#) function to set or clear a window update lock, specifying the window to lock. The lock applies to the specified window and all of its child windows. When the lock is set, the [GetDC](#) and [BeginPaint](#) functions return a display device context with a visible region that is empty. Given this, the application can continue to draw in the window, but all output is clipped. The lock persists until the application clears it by calling [LockWindowUpdate](#), specifying **NULL** for the window. Although [LockWindowUpdate](#) forces a window's visible region to be empty, the function does not make the specified window invisible and does not clear the **WS_VISIBLE** style bit.

After the lock is set, the application can use the [GetDCEx](#) function, with the **DCX_LOCKWINDOWUPDATE** value, to retrieve a display device context to draw over the locked window. This allows the application to draw a tracking rectangle when processing keyboard or mouse messages. The system uses this method when the user moves and sizes windows. [GetDCEx](#) retrieves the display device context from the display device context cache, so the application must release the device context as soon as possible after drawing.

While a window update lock is set, the system creates an accumulated bounding rectangle for each locked window. When the lock is cleared, the system uses this bounding rectangle to set the update region for the window and its child windows, forcing an eventual [WM_PAINT](#) message. If the accumulated bounding rectangle is empty (that is, if no drawing has occurred while the lock was set), the update region is not set.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Accumulated Bounding Rectangle

Article • 01/07/2021

The *accumulated bounding rectangle* is the smallest rectangle enclosing the portion of a window or client area affected by recent drawing operations. An application can use this rectangle to conveniently determine the extent of changes caused by drawing operations. It is sometimes used in conjunction with [LockWindowUpdate](#) to determine which portion of the client area must be redrawn after the update lock is cleared.

An application uses the [SetBoundsRect](#) function (specifying DCB_ENABLE) to begin accumulating the bounding rectangle. The system subsequently accumulates points for the bounding rectangle as the application uses the specified display device context. The application can retrieve the current bounding rectangle at any time by using the [GetBoundsRect](#) function. The application stops the accumulation by calling [SetBoundsRect](#) again, specifying the DCB_DISABLE value.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Using the WM_PAINT Message

Article • 01/07/2021

You can use the [WM_PAINT](#) message to carry out the drawing necessary for displaying information. Because the system sends WM_PAINT messages to your application when your window must be updated or when you explicitly request an update, you can consolidate the code for drawing in your application's window procedure. You can then use this code whenever your application must draw either new or existing information.

The following sections show a variety of ways to use the WM_PAINT message to draw in a window:

- [Drawing in the Client Area](#)
- [Redrawing the Entire Client Area](#)
- [Redrawing in the Update Region](#)
- [Invalidating the Client Area](#)
- [Drawing a Minimized Window](#)
- [Drawing a Custom Window Background](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing in the Client Area

Article • 01/07/2021

You use the [BeginPaint](#) and [EndPaint](#) functions to prepare for and complete the drawing in the client area. [BeginPaint](#) returns a handle to the display device context used for drawing in the client area; [EndPaint](#) ends the paint request and releases the device context.

In the following example, the window procedure writes the message "Hello, Windows!" in the client area. To make sure the string is visible when the window is first created, the [WinMain](#) function calls [UpdateWindow](#) immediately after creating and showing the window. This causes a [WM_PAINT](#) message to be sent immediately to the window procedure.

C++

```
LRESULT APIENTRY WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            TextOut(hdc, 0, 0, "Hello, Windows!", 15);
            EndPaint(hwnd, &ps);
            return 0L;

        // Process other messages.
    }
}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    HWND hwnd;

    hwnd = CreateWindowEx(
        // parameters
    );

    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);

    return msg.wParam;
}
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Redrawing the Entire Client Area

Article • 01/07/2021

You can have your application redraw the entire contents of the client area whenever the window changes size by setting the CS_HREDRAW and CS_VREDRAW styles for the window class. Applications that adjust the size of the drawing based on the size of the window use these styles to ensure that they start with a completely empty client area when drawing.

In the following example, the window procedure draws a five-pointed star that fits neatly in the client area. It uses a common device context and must set the mapping mode as well as window and viewport extents each time the **WM_PAINT** message is processed.

C++

```
LRESULT APIENTRY WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    RECT rc;
    POINT aptStar[6] = {50,2, 2,98, 98,33, 2,33, 98,98, 50,2};

    .

    .

    .

    case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        GetClientRect(hwnd, &rc);
        SetMapMode(hdc, MM_ANISOTROPIC);
        SetWindowExtEx(hdc, 100, 100, NULL);
        SetViewportExtEx(hdc, rc.right, rc.bottom, NULL);
        Polyline(hdc, aptStar, 6);
        EndPaint(hwnd, &ps);
        return 0L;

    .

    .

    .

}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASS wc;
```

```
    .  
    .  
  
    wc.style = CS_HREDRAW | CS_VREDRAW;  
    wc.lpfnWndProc = (WNDPROC) WndProc;  
  
    .  
    .  
    .  
  
    RegisterClass(&wc);  
  
    .  
    .  
    .  
  
    return msg.wParam;  
}
```

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Redrawing in the Update Region

Article • 01/07/2021

You can limit the amount of drawing your application carries out when processing the [WM_PAINT](#) message by determining the size and location of the update region.

Because the system uses the update region when creating the clipping region for the window's display device context, you can indirectly determine the update region by examining the clipping region.

In the following example, the window procedure draws a triangle, a rectangle, a pentagon, and a hexagon, but only if all or a portion of each figure lies within the update region. The window procedure uses the [RectVisible](#) function and a 100-by-100 rectangle to determine whether a figure is within the clipping region (and therefore the update region) for the common device context retrieved by [BeginPaint](#).

C++

```
POINT aptTriangle[4] = {50,2, 98,86, 2,86, 50,2},
    aptRectangle[5] = { 2,2, 98,2, 98,98, 2,98, 2,2},
    aptPentagon[6] = {50,2, 98,35, 79,90, 21,90, 2,35, 50,2},
    aptHexagon[7] = {50,2, 93,25, 93,75, 50,98, 7,75, 7,25, 50,2};
.

.

.

case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    SetRect(&rc, 0, 0, 100, 100);

    if (RectVisible(hdc, &rc))
        Polyline(hdc, aptTriangle, 4);

    SetViewportOrgEx(hdc, 100, 0, NULL);
    if (RectVisible(hdc, &rc))
        Polyline(hdc, aptRectangle, 5);

    SetViewportOrgEx(hdc, 0, 100, NULL);
    if (RectVisible(hdc, &rc))
        Polyline(hdc, aptPentagon, 6);

    SetViewportOrgEx(hdc, 100, 100, NULL);
    if (RectVisible(hdc, &rc))
        Polyline(hdc, aptHexagon, 7);
    EndPaint(hwnd, &ps);
    return 0L;

.
```

The coordinates of each figure in this example lie within the same 100-by-100 rectangle. Before drawing a figure, the window procedure sets the viewport origin to a different part of the client area by using the [SetViewportOrgEx](#) function. This prevents figures from being drawn one on top of the other. Changing the viewport origin does not affect the clipping region, but does affect how the coordinates of the rectangle passed to [RectVisible](#) are interpreted. Changing the origin also allows you to use a single rectangle to check the update region rather than individual rectangles for each figure.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Invalidate the Client Area

Article • 01/07/2021

The system is not the only source of **WM_PAINT** messages. The **InvalidateRect** or **InvalidateRgn** function can indirectly generate **WM_PAINT** messages for your windows. These functions mark all or part of a client area as invalid (that must be redrawn).

In the following example, the window procedure invalidates the entire client area when processing **WM_CHAR** messages. This allows the user to change the figure by typing a number and view the results; these results are drawn as soon as there are no other messages in the application's message queue.

C++

```
RECT rc;
POINT aptPentagon[6] = {50,2, 98,35, 79,90, 21,90, 2,35, 50,2},
    aptHexagon[7] = {50,2, 93,25, 93,75, 50,98, 7,75, 7,25, 50,2};
POINT *ppt = aptPentagon;
int cpt = 6;

.

.

.

case WM_CHAR:
    switch (wParam)
    {
        case '5':
            ppt = aptPentagon;
            cpt = 6;
            break;
        case '6':
            ppt = aptHexagon;
            cpt = 7;
            break;
    }
    InvalidateRect(hwnd, NULL, TRUE);
    return 0L;

case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    GetClientRect(hwnd, &rc);
    SetMapMode(hdc, MM_ANISOTROPIC);
    SetWindowExtEx(hdc, 100, 100, NULL);
    SetViewportExtEx(hdc, rc.right, rc.bottom, NULL);
    Polyline(hdc, ppt, cpt);
    EndPaint(hwnd, &ps);
    return 0L;
```

In this example, the **NULL** argument used by **InvalidateRect** specifies the entire client area; the **TRUE** argument causes the background to be erased. If you do not want the application to wait until the application's message queue has no other messages, use the **UpdateWindow** function to force the **WM_PAINT** message to be sent immediately. If there is any invalid part of the client area, **UpdateWindow** sends the **WM_PAINT** message for the specified window directly to the window procedure.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Drawing a Minimized Window

Article • 01/07/2021

You can draw your own minimized windows rather than having the system draw them for you. Most applications define a class icon when registering the window class for the window, and the system draws the icon when the window is minimized. If you set the class icon to **NULL**, however, the system sends a [WM_PAINT](#) message to your window procedure whenever the window is minimized, enabling the window procedure to draw in the minimized window.

In the following example, the window procedure draws a star in the minimized window. The procedure uses the [IsIconic](#) function to determine when the window is minimized. This ensures that the star is drawn only when the window is minimized.

C++

```
POINT aptStar[6] = {50,2, 2,98, 98,33, 2,33, 98,98, 50,2};

.

.

.

case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);

    // Determine whether the window is minimized.

    if (IsIconic(hwnd))
    {
        GetClientRect(hwnd, &rc);
        SetMapMode(hdc, MM_ANISOTROPIC);
        SetWindowExtEx(hdc, 100, 100, NULL);
        SetViewportExtEx(hdc, rc.right, rc.bottom, NULL);
        Polyline(hdc, aptStar, 6);
    }
    else
    {
        TextOut(hdc, 0,0, "Hello, Windows!", 15);
    }
    EndPaint(hwnd, &ps);
    return 0L;
```

You set the class icon to **NULL** by setting the [hIcon](#) member of the [WNDCLASS](#) structure to **NULL** before calling the [RegisterClass](#) function for the window class.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing a Custom Window Background

Article • 01/07/2021

You can draw your own window background rather than having the system draw it for you. Most applications specify a brush handle or system color value for the class background brush when registering the window class; the system uses the brush or color to draw the background. If you set the class background brush to **NULL**, however, the system sends a [WM_ERASEBKGND](#) message to your window procedure whenever the window background must be drawn, letting you draw a custom background.

In the following example, the window procedure draws a large checkerboard pattern that fits neatly in the window. The procedure fills the client area with a white brush and then draws thirteen 20-by-20 rectangles using a gray brush. The display device context to use when drawing the background is specified in the *wParam* parameter for the message.

C++

```
HBRUSH hbrWhite, hbrGray;

.

.

.

case WM_CREATE:
    hbrWhite = GetStockObject(WHITE_BRUSH);
    hbrGray = GetStockObject(GRAY_BRUSH);
    return 0L;

case WM_ERASEBKGND:
    hdc = (HDC) wParam;
    GetClientRect(hwnd, &rc);
    SetMapMode(hdc, MM_ANISOTROPIC);
    SetWindowExtEx(hdc, 100, 100, NULL);
    SetViewportExtEx(hdc, rc.right, rc.bottom, NULL);
    FillRect(hdc, &rc, hbrWhite);

    for (i = 0; i < 13; i++)
    {
        x = (i * 40) % 100;
        y = ((i * 40) / 100) * 20;
        SetRect(&rc, x, y, x + 20, y + 20);
        FillRect(hdc, &rc, hbrGray);
    }
    return 1L;
```

If the application draws its own minimized window, the system also sends the **WM_ERASEBKGND** message to the window procedure to draw the background for the minimized window. You can use the same technique used by **WM_PAINT** to determine whether the window is minimized that is, call the **IsIconic** function and check for the return value **TRUE**.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using the GetDC Function

Article • 01/07/2021

You use the [GetDC](#) function to carry out drawing that must occur instantly rather than when a [WM_PAINT](#) message is processing. Such drawing is usually in response to an action by the user, such as making a selection or drawing with the mouse. In such cases, the user should receive instant feedback and must not be forced to stop selecting or drawing in order for the application to display the result. The following sections show how to use GetDC to draw in a window:

- [Drawing with the Mouse](#)
- [Drawing at Timed Intervals](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Drawing with the Mouse

Article • 01/07/2021

You can permit the user to draw lines with the mouse by having your window procedure draw while processing the [WM_MOUSEMOVE](#) message. The system sends the [WM_MOUSEMOVE](#) message to the window procedure whenever the user moves the cursor within the window. To draw lines, the window procedure can retrieve a display device context and draw a line in the window between the current and previous cursor positions.

In the following example, the window procedure prepares for drawing when the user presses and holds the left mouse button (sending the [WM_LBUTTONDOWN](#) message). As the user moves the cursor within the window, the window procedure receives a series of [WM_MOUSEMOVE](#) messages. For each message, the window procedure draws a line connecting the previous position and the current position. To draw the line, the procedure uses [GetDC](#) to retrieve a display device context; then, as soon as drawing is complete and before returning from the message, the procedure uses the [ReleaseDC](#) function to release the display device context. As soon as the user releases the mouse button, the window procedure clears the flag, and the drawing stops (which sends the [WM_LBUTTONUP](#) message).

C++

```
BOOL fDraw = FALSE;
POINT ptPrevious;

.

.

.

case WM_LBUTTONDOWN:
    fDraw = TRUE;
    ptPrevious.x = LOWORD(lParam);
    ptPrevious.y = HIWORD(lParam);
    return 0L;

case WM_LBUTTONUP:
    if (fDraw)
    {
        hdc = GetDC(hwnd);
        MoveToEx(hdc, ptPrevious.x, ptPrevious.y, NULL);
        LineTo(hdc, LOWORD(lParam), HIWORD(lParam));
        ReleaseDC(hwnd, hdc);
    }
    fDraw = FALSE;
    return 0L;
```

```
case WM_MOUSEMOVE:  
    if (fDraw)  
    {  
        hdc = GetDC(hwnd);  
        MoveToEx(hdc, ptPrevious.x, ptPrevious.y, NULL);  
        LineTo(hdc, ptPrevious.x = LOWORD(lParam),  
                ptPrevious.y = HIWORD(lParam));  
        ReleaseDC(hwnd, hdc);  
    }  
    return 0L;
```

An application that enables drawing, as in this example, typically records either the points or lines so that the lines can be redrawn whenever the window is updated.

Drawing applications often use a memory device context and an associated bitmap to store lines that were drawn by using a mouse.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Drawing at Timed Intervals

Article • 01/07/2021

You can draw at timed intervals by creating a timer with the [SetTimer](#) function. By using a timer to send [WM_TIMER](#) messages to the window procedure at regular intervals, an application can carry out simple animation in the client area while other applications continue running.

In the following example, the application bounces a star from side to side in the client area. Each time the window procedure receives a [WM_TIMER](#) message, the procedure erases the star at the current position, calculates a new position, and draws the star within the new position. The procedure starts the timer by calling [SetTimer](#) while processing the [WM_CREATE](#) message.

C++

```
RECT rcCurrent = {0,0,20,20};
POINT aptStar[6] = {10,1, 1,19, 19,6, 1,6, 19,19, 10,1};
int X = 2, Y = -1, idTimer = -1;
BOOL fVisible = FALSE;
HDC hdc;

HRESULT APIENTRY WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    RECT rc;

    switch (message)
    {
        case WM_CREATE:

            // Calculate the starting point.

            GetClientRect(hwnd, &rc);
            OffsetRect(&rcCurrent, rc.right / 2, rc.bottom / 2);

            // Initialize the private DC.

            hdc = GetDC(hwnd);
            SetViewportOrgEx(hdc, rcCurrent.left,
                rcCurrent.top, NULL);
            SetROP2(hdc, R2_NOT);

            // Start the timer.

            SetTimer(hwnd, idTimer = 1, 10, NULL);
            return 0L;
    }
}
```

```

case WM_DESTROY:
    KillTimer(hwnd, 1);
    PostQuitMessage(0);
    return 0L;

case WM_SIZE:
    switch (wParam)
    {
        case SIZE_MINIMIZED:

            // Stop the timer if the window is minimized.

            KillTimer(hwnd, 1);
            idTimer = -1;
            break;

        case SIZE_RESTORED:

            // Move the star back into the client area
            // if necessary.

            if (rcCurrent.right > (int) LOWORD(lParam))
            {
                rcCurrent.left =
                    (rcCurrent.right =
                        (int) LOWORD(lParam)) - 20;
            }
            if (rcCurrent.bottom > (int) HIWORD(lParam))
            {
                rcCurrent.top =
                    (rcCurrent.bottom =
                        (int) HIWORD(lParam)) - 20;
            }

            // Fall through to the next case.

        case SIZE_MAXIMIZED:

            // Start the timer if it had been stopped.

            if (idTimer == -1)
                SetTimer(hwnd, idTimer = 1, 10, NULL);
            break;
    }
    return 0L;

case WM_TIMER:

    // Hide the star if it is visible.

    if (fVisible)
        Polyline(hdc, aptStar, 6);

    // Bounce the star off a side if necessary.

```

```

GetClientRect(hwnd, &rc);
if (rcCurrent.left + X < rc.left ||
    rcCurrent.right + X > rc.right)
    X = -X;
if (rcCurrent.top + Y < rc.top ||
    rcCurrent.bottom + Y > rc.bottom)
    Y = -Y;

// Show the star in its new position.

OffsetRect(&rcCurrent, X, Y);
SetViewportOrgEx(hdc, rcCurrent.left,
                 rcCurrent.top, NULL);
fVisible = Polyline(hdc, aptStar, 6);

return 0L;

case WM_ERASEBKGND:

// Erase the star.

fVisible = FALSE;
return DefWindowProc(hwnd, message, wParam, lParam);

case WM_PAINT:

// Show the star if it is not visible. Use BeginPaint
// to clear the update region.

BeginPaint(hwnd, &ps);
if (!fVisible)
    fVisible = Polyline(hdc, aptStar, 6);
EndPaint(hwnd, &ps);
return 0L;
}
return DefWindowProc(hwnd, message, wParam, lParam);
}

```

This application uses a private device context to minimize the time required to prepare the device context for drawing. The window procedure retrieves and initializes the private device context when processing the [WM_CREATE](#) message, setting the binary raster operation mode to allow the star to be erased and drawn using the same call to the [Polyline](#) function. The window procedure also sets the viewport origin to allow the star to be drawn using the same set of points regardless of the star's position in the client area.

The application uses the [WM_PAINT](#) message to draw the star whenever the window must be updated. The window procedure draws the star only if it is not visible; that is, only if it has been erased by the [WM_ERASEBKGND](#) message. The window procedure

intercepts the **WM_ERASEBKGND** message to set the *fVisible* variable, but passes the message to **DefWindowProc** so that the system can draw the window background.

The application uses the **WM_SIZE** message to stop the timer when the window is minimized and to restart the timer when the minimized window is restored. The window procedure also uses the message to update the current position of the star if the size of the window has been reduced so that the star is no longer in the client area. The application keeps track of the star's current position by using the structure specified by *rcCurrent*, which defines the bounding rectangle for the star. Keeping all corners of the rectangle in the client area keeps the star in the area. The window procedure initially centers the star in the client area when processing the **WM_CREATE** message.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Painting and Drawing Reference

Article • 01/07/2021

The following elements are associated with painting and drawing:

- [Painting and Drawing Functions](#)
- [Painting and Drawing Structures](#)
- [Painting and Drawing Messages](#)
- [Raster Operation Codes](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Painting and Drawing Functions

Article • 01/07/2021

The following functions are used with painting and drawing.

 Expand table

Function	Description
BeginPaint	Prepares a window for painting.
DrawAnimatedRects	Draws a rectangle and animates it to indicate icon or window activity.
DrawCaption	Draws a window caption.
DrawEdge	Draws one or more edges of rectangle.
DrawFocusRect	Draws a rectangle in the style that indicates the rectangle has the focus.
DrawFrameControl	Draws a frame control.
DrawState	Displays an image and applies a visual effect to indicate a state.
DrawStateProc	A callback function that renders a complex image for DrawState .
EndPaint	Marks the end of painting in a window.
ExcludeUpdateRgn	Prevents drawing within invalid areas of a window.
GdiFlush	Flushes the calling thread's current batch.
GdiGetBatchLimit	Returns the maximum number of function calls that can be accumulated in the calling thread's current batch.
GdiSetBatchLimit	Sets the maximum number of function calls that can be accumulated in the calling thread's current batch.
GetBkColor	Returns the background color for a device context.
GetBkMode	Returns the background mix mode for a device context.
GetBoundsRect	Gets the accumulated bounding rectangle for a device context.
GetROP2	Gets the foreground mix mode of a device context.
GetUpdateRect	Gets the coordinates of the smallest rectangle that encloses the update region of a window.
GetUpdateRgn	Gets the update region of a window.

Function	Description
GetWindowDC	Gets the device context for a window, including title bar, menus, and scroll bars.
GetWindowRgn	Gets a copy of the window region of a window.
GetWindowRgnBox	Gets the dimensions of the tightest bounding rectangle for the window region of a window.
GrayString	Draws gray text at a location.
InvalidateRect	Adds a rectangle to a window's update region.
InvalidateRgn	Invalidates the client area within a region.
LockWindowUpdate	Disables or enables drawing in a window.
OutputProc	A callback function used with the GrayString function. It is used to draw a string.
PaintDesktop	Fills the clipping region in a device context with a pattern.
RedrawWindow	Updates a region in a window's client area.
SetBkColor	Sets the background to a color value.
SetBkMode	Sets the background mix mode of a device context.
SetBoundsRect	Controls the accumulation of bounding rectangle information for a device context.
SetROP2	Sets the foreground mix mode.
SetWindowRgn	Sets the window region of a window.
UpdateWindow	Updates the client area of a window.
ValidateRect	Validates the client area within a rectangle.
ValidateRgn	Validates the client area within a region.
WindowFromDC	Returns a handle to the window associated with a device context.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

BeginPaint function (winuser.h)

02/22/2024

The **BeginPaint** function prepares the specified window for painting and fills a [PAINTSTRUCT](#) structure with information about the painting.

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

DrawAnimatedRects function (winuser.h)

02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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)

Article 02/22/2024

The **DrawCaption** function draws a window caption.

Syntax

C++

```
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.

 Expand table

Value	Meaning
<code>DC_ACTIVE</code>	The function uses the colors that denote an active caption.
<code>DC_BUTTONS</code>	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

 Expand table

Requirement	Value
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)

10/13/2021

The **DrawEdge** function draws one or more edges of rectangle.

Syntax

C++

```
BOOL DrawEdge(
    [in]      HDC      hdc,
    [in, out] LPRECT  qrc,
    [in]      UINT     edge,
    [in]      UINT     grfFlags
);
```

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.

 Expand table

Value	Meaning
<code>BDR_RAISEDINNER</code>	Raised inner edge.
<code>BDR_SUNKENINNER</code>	Sunken inner edge.

The outer-border flags are as follows.

 Expand table

Value	Meaning
BDR_RAISEDOUTER	Raised outer edge.
BDR_SUNKENOUTER	Sunken outer edge.

Alternatively, the *edge* parameter can specify one of the following flags.

[Expand table](#)

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.

[Expand table](#)

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

[Expand table](#)

Requirement	Value
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)

Requirement	Value
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)

10/13/2021

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

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
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)

03/06/2024

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

Syntax

C++

```
BOOL DrawFrameControl(
    [in] HDC      hdc,
    [in] LPRECT  lprc,
    [in] UINT     uType,
    [in] UINT     uState
);
```

Parameters

[in] `hdc`

A handle to the device context of the window in which to draw the control.

[in] `lprc`

A pointer to a [RECT](#) structure that contains the logical coordinates of the bounding rectangle for frame control.

[in] `uType`

The type of frame control to draw. This parameter can be one of the following values.

 Expand table

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] *uState*

The initial state of the frame control. If *uType* is DFC_BUTTON, *uState* can be one of the following values.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
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)

02/09/2023

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

Syntax

C++

```
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.

[Expand table](#)

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 *lData* parameter is a pointer to the string, and the *wData* parameter specifies the length. If *wData* is zero, the string must be null-terminated.

This parameter can also be one of the following state values.

 [Expand table](#)

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 that 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](#).

 Expand table

Requirement	Value
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)

Article10/13/2021

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

EndPaint function (winuser.h)

02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

ExcludeUpdateRgn function (winuser.h)

02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

 Expand table

Requirement	Value
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](#)

GdiFlush function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GdiSetBatchLimit function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetBkColor function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

C++

```
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

 Expand table

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

Requirement	Value
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetBkColor](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[SetBkMode](#)

GetBoundsRect function (wingdi.h)

Article 02/22/2024

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

C++

```
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.

[] [Expand table](#)

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.

 Expand table

Value	Meaning
0	An error occurred. The specified device context handle is invalid.
DCB_DISABLE	Boundary accumulation is off.
DCB_ENABLE	Boundary accumulation is on.
DCB_RESET	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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetROP2 function (wingdi.h)

Article02/22/2024

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

C++

```
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.

 [Expand table](#)

Mix mode	Description
R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTOPEN	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_MERGEOPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEOPENNOT	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_NOTCOPYOPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGEOPEN	Pixel is the inverse of the R2_MERGEOPEN color.
R2_NOTXOROPEN	Pixel is the inverse of the R2_XOROPEN color.
R2_WHITE	Pixel is always 1.
R2_XOROPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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

GetUpdateRect function (winuser.h)

10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
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)

Article 02/22/2024

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

C++

```
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.

[] Expand table

Value	Meaning
COMPLEXREGION	Region consists of more than one rectangle.

ERROR	An error occurred.
NULLREGION	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

 [Expand table](#)

Requirement	Value
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)

02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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)

10/13/2021

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

C++

```
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.

 Expand table

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.

C++

```
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

 [Expand table](#)

Requirement	Value
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

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
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)

02/09/2023

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

C++

```
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 that 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](#).

 [Expand table](#)

Requirement	Value
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](#)

InvalidateRect function (winuser.h)

10/13/2021

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

C++

```
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

 Expand table

Requirement	Value
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)

Article 10/13/2021

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

C++

```
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

Invalidate 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

 [Expand table](#)

Requirement	Value
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](#)

LockWindowUpdate function (winuser.h)

Article10/13/2021

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

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
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

GRAYSTRINGPROC callback function (winuser.h)

10/01/2025

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. *GrayStringProc* (or *OutputProc*) is a placeholder for the application-defined or library-defined function name.

Syntax

C++

```
GRAYSTRINGPROC Graystringproc;

BOOL Graystringproc(
    HDC unnamedParam1,
    LPARAM unnamedParam2,
    int unnamedParam3
)
{...}
```

Parameters

unnamedParam1

Type: **HDC**

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](#). This parameter is typically named *hDc*.

unnamedParam2

Type: **LPARAM**

A pointer to the string to be drawn. This parameter is typically named *lpData*.

unnamedParam3

Type: **int**

The length, in characters, of the string. This parameter is typically named *nCount*.

Return value

Type: **BOOL**

If it succeeds, the callback function should return **TRUE**.

If the function fails, the return value is **FALSE**.

Remarks

! Note

The parameters are defined in the header with no names: `typedef BOOL (CALLBACK* GRAYSTRINGPROC)(HDC, LPARAM, int);`. Therefore, the syntax block lists them as `unnamedParam1 - unnamedParam3`. You can name these parameters anything in your app. However, they are usually named as shown in the parameter descriptions.

The callback function must draw an image relative to the coordinates (0,0).

Requirements

 Expand table

Requirement	Value
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](#)

PaintDesktop function (winuser.h)

02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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

Requirement	Value
DLL	User32.dll

See also

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

RedrawWindow function (winuser.h)

Article 10/13/2021

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

Syntax

C++

```
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.

 Expand table

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.

[] [Expand table](#)

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.

[] [Expand table](#)

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.

[Expand table](#)

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

Requirement	Value
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](#)

SetBkColor function (wingdi.h)

Article02/22/2024

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

C++

```
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

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article05/01/2025

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

C++

```
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.

 Expand table

Value	Meaning
OPAQUE	Background is filled with the current background color before the text, hatched brush, or pen is drawn.
TRANSPARENT	Background is unaffected.

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.

C++

```
#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.

LocalFree((LOCALHANDLE) plf);
    EndPaint(hWnd, &ps);
    break;
}
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 10/13/2021

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

C++

```
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.

 Expand table

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.

DCB_DISABLE	Turns off boundary accumulation.
DCB_ENABLE	Turns on boundary accumulation, which is disabled by default.
DCB_RESET	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.

[Expand table](#)

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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[GetBoundsRect](#)

[Painting and Drawing Functions](#)

[Painting and Drawing Overview](#)

[RECT](#)

SetROP2 function (wingdi.h)

Article 10/13/2021

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

C++

```
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.

 Expand table

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_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_MERGESENSE	Pixel is a combination of the pen color and the screen color.
R2_MERGESENSENNOT	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_NOTCOPYSENSE	Pixel is the inverse of the pen color.
R2_NOTMASKSENSE	Pixel is the inverse of the R2_MASKSENSE color.
R2_NOTMERGESENSE	Pixel is the inverse of the R2_MERGESENSE color.
R2_NOTXORSENSE	Pixel is the inverse of the R2_XORSENSE color.
R2_WHITE	Pixel is always 1.
R2_XORSENSE	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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

SetWindowRgn function (winuser.h)

02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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

WM_WINDOWPOSCHANGED

UpdateWindow function (winuser.h)

02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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)

Requirement	Value
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)

10/13/2021

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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)

Article02/22/2024

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

Syntax

C++

```
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

[Expand table](#)

Requirement	Value
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](#)

Painting and Drawing Structures

Article • 01/07/2021

The following structure is used with painting and drawing.

PAINTSTRUCT

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PAINTSTRUCT structure (winuser.h)

Article09/01/2022

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

C++

```
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[32]

Reserved; used internally by the system.

Requirements

[+] Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

Painting and Drawing Messages

Article • 01/07/2021

The following messages are used with painting and drawing:

- [WM_DISPLAYCHANGE](#)
- [WM_ERASEBKGND](#)
- [WM_NCPAINT](#)
- [WM_PAINT](#)
- [WM_PRINT](#)
- [WM_PRINTCLIENT](#)
- [WM_SETREDRAW](#)
- [WM_SYNCPAINT](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

WM_DISPLAYCHANGE message

Article • 01/07/2021

The **WM_DISPLAYCHANGE** message is sent to all windows when the display resolution has changed.

A window receives this message through its **WindowProc** function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

The new image depth of the display, in bits per pixel.

lParam

The low-order word specifies the horizontal resolution of the screen.

The high-order word specifies the vertical resolution of the screen.

Remarks

This message is only sent to top-level windows. For all other windows it is posted.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	Winuser.h (include Windows.h)

See also

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[HIWORD](#)

[LOWORD](#)

Feedback

Was this page helpful?

 Yes

 No

WM_NCPAINT message

Article • 01/07/2021

The **WM_NCPAINT** message is sent to a window when its frame must be painted.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

A handle to the update region of the window. The update region is clipped to the window frame.

lParam

This parameter is not used.

Return value

An application returns zero if it processes this message.

Remarks

The [DefWindowProc](#) function paints the window frame.

An application can intercept the **WM_NCPAINT** message and paint its own custom window frame. The clipping region for a window is always rectangular, even if the shape of the frame is altered.

The *wParam* value can be passed to [GetDCEx](#) as in the following example.

C++

```
case WM_NCPAINT:  
{  
    HDC hdc;  
    hdc = GetDCEx(hwnd, (HRGN)wParam, DCX_WINDOW|DCX_INTERSECTRGN);  
    // Paint into this DC  
    ReleaseDC(hwnd, hdc);  
}
```

Requirements

Requirement	Value
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

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[DefWindowProc](#)

[GetWindowDC](#)

[WM_PAINT](#)

[GetDCEx](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_PAINT message

Article • 04/07/2022

The **WM_PAINT** message is sent when the system or another application makes a request to paint a portion of an application's window. The message is sent when the [UpdateWindow](#) or [RedrawWindow](#) function is called, or by the [DispatchMessage](#) function when the application obtains a **WM_PAINT** message by using the [GetMessage](#) or [PeekMessage](#) function.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

This parameter is not used.

lParam

This parameter is not used.

Return value

An application returns zero if it processes this message.

Example

C

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)  
{  
    switch (uMsg)  
    {
```

```

case WM_DESTROY:
    PostQuitMessage(0);
    return 0;

case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hwnd, &ps);

    // All painting occurs here, between BeginPaint and EndPaint.
    FillRect(hdc, &ps.rcPaint, (HBRUSH) (COLOR_WINDOW+1));
    EndPaint(hwnd, &ps);
}
return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

```

Example from [Windows Classic Samples](#) on GitHub.

Remarks

The **WM_PAINT** message is generated by the system and should not be sent by an application. To force a window to draw into a specific device context, use the **WM_PRINT** or **WM_PRINTCLIENT** message. Note that this requires the target window to support the **WM_PRINTCLIENT** message. Most common controls support the **WM_PRINTCLIENT** message.

The **DefWindowProc** function validates the update region. The function may also send the **WM_NCPAINT** message to the window procedure if the window frame must be painted and send the **WM_ERASEBKGND** message if the window background must be erased.

The system sends this message when there are no other messages in the application's message queue. **DispatchMessage** determines where to send the message; **GetMessage** determines which message to dispatch. **GetMessage** returns the **WM_PAINT** message when there are no other messages in the application's message queue, and **DispatchMessage** sends the message to the appropriate window procedure.

A window may receive internal paint messages as a result of calling **RedrawWindow** with the **RDW_INTERNALPAINT** flag set. In this case, the window may not have an update region. An application may call the **GetUpdateRect** function to determine whether the window has an update region. If **GetUpdateRect** returns zero, the application need not call the **BeginPaint** and **EndPaint** functions.

An application must check for any necessary internal painting by looking at its internal data structures for each **WM_PAINT** message, because a **WM_PAINT** message may have been caused by both a non-NULL update region and a call to [RedrawWindow](#) with the RDW_INTERNALPAINT flag set.

The system sends an internal **WM_PAINT** message only once. After an internal **WM_PAINT** message is returned from [GetMessage](#) or [PeekMessage](#) or is sent to a window by [UpdateWindow](#), the system does not post or send further **WM_PAINT** messages until the window is invalidated or until [RedrawWindow](#) is called again with the RDW_INTERNALPAINT flag set.

For some common controls, the default **WM_PAINT** message processing checks the *wParam* parameter. If *wParam* is non-NULL, the control assumes that the value is an HDC and paints using that device context.

Requirements

Requirement	Value
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

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[BeginPaint](#)

[DefWindowProc](#)

[DispatchMessage](#)

[EndPaint](#)

[GetMessage](#)

[GetUpdateRect](#)

[PeekMessage](#)

[RedrawWindow](#)

[UpdateWindow](#)

[WM_ERASEBKGND](#)

[WM_NCPAINT](#)

[WM_PRINT](#)

[WM_PRINTCLIENT](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

WM_PRINT message

Article • 01/07/2021

The **WM_PRINT** message is sent to a window to request that it draw itself in the specified device context, most commonly in a printer device context.

A window receives this message through its [WindowProc](#) function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

A handle to the device context to draw in.

lParam

The drawing options. This parameter can be one or more of the following values.

Value	Meaning
PRF_CHECKVISIBLE	Draws the window only if it is visible.
PRF_CHILDREN	Draws all visible children windows.
PRF_CLIENT	Draws the client area of the window.
PRF_ERASEBKGND	Erases the background before drawing the window.
PRF_NONCLIENT	Draws the nonclient area of the window.
PRF_OWNED	Draws all owned windows.

Remarks

The [DefWindowProc](#) function processes this message based on which drawing option is specified: if PRF_CHECKVISIBLE is specified and the window is not visible, do nothing, if PRF_NONCLIENT is specified, draw the nonclient area in the specified device context, if PRF_ERASEBKGND is specified, send the window a [WM_ERASEBKGND](#) message, if PRF_CLIENT is specified, send the window a [WM_PRINTCLIENT](#) message, if PRF_CHILDREN is set, send each visible child window a [WM_PRINT](#) message, if PRF_OWNED is set, send each visible owned window a [WM_PRINT](#) message.

Requirements

Requirement	Value
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

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[DefWindowProc](#)

[WM_ERASEBKGND](#)

[WM_PRINTCLIENT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_PRINTCLIENT message

Article • 01/07/2021

The **WM_PRINTCLIENT** message is sent to a window to request that it draw its client area in the specified device context, most commonly in a printer device context.

Unlike [WM_PRINT](#), **WM_PRINTCLIENT** is not processed by [DefWindowProc](#). A window should process the **WM_PRINTCLIENT** message through an application-defined [WindowProc](#) function for it to be used properly.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

A handle to the device context to draw in.

lParam

The drawing options. This parameter can be one or more of the following values.

Value	Meaning
PRF_CHECKVISIBLE	Draws the window only if it is visible.
PRF_CHILDREN	Draws all visible children windows.
PRF_CLIENT	Draws the client area of the window.
PRF_ERASEBKGND	Erases the background before drawing the window.
PRF_NONCLIENT	Draws the nonclient area of the window.
PRF_OWNED	Draws all owned windows.

Remarks

A window can process this message in much the same manner as [WM_PAINT](#), except that [BeginPaint](#) and [EndPaint](#) need not be called (a device context is provided), and the window should draw its entire client area rather than just the invalid region.

Windows that can be used anywhere in the system, such as controls, should process this message. It is probably worthwhile for other windows to process this message as well because it is relatively easy to implement.

The [AnimateWindow](#) function requires that the window being animated implements the [WM_PRINTCLIENT](#) message.

Requirements

Requirement	Value
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

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[AnimateWindow](#)

[BeginPaint](#)

[EndPaint](#)

[WM_PAINT](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

WM_SETREDRAW message

Article • 07/09/2021

You send the WM_SETREDRAW message to a window to allow changes in that window to be redrawn, or to prevent changes in that window from being redrawn.

To send this message, call the [SendMessage](#) function with the following parameters.

C++

```
SendMessage(  
    (HWND) hWnd,  
    WM_SETREDRAW,  
    (WPARAM) wParam,  
    (LPARAM) lParam  
);
```

Parameters

wParam

The redraw state. If this parameter is **TRUE**, then the content can be redrawn after a change. If this parameter is **FALSE**, then the content can't be redrawn after a change.

lParam

This parameter isn't used.

Return value

Your application should return 0 if it processes this message.

Remarks

This message can be useful if your application must add several items to a list box. Your application can call this message with *wParam* set to **FALSE**, add the items, and then call the message again with *wParam* set to **TRUE**. Finally, your application can call [RedrawWindow](#)(*hWnd*, **NULL**, **NULL**, RDW_ERASE | RDW_FRAME | RDW_INVALIDATE | RDW_ALLCHILDREN) to cause the list box to be repainted.

 Note

You should use **RedrawWindow** with the specified flags, instead of **InvalidateRect**, because the former is necessary for some controls that have nonclient area of their own, or have window styles that cause them to be given a nonclient area (such as **WS_THICKFRAME**, **WS_BORDER**, or **WS_EX_CLIENTEDGE**). If the control does not have a nonclient area, then **RedrawWindow** with these flags will do only as much invalidation as **InvalidateRect** would.

Passing a **WM_SETREDRAW** message to the **DefWindowProc** function removes the **WS_VISIBLE** style from the window when *wParam* is set to **FALSE**. Although the window content remains visible on screen, the **IsWindowVisible** function returns **FALSE** when called on a window in this state.

Passing a **WM_SETREDRAW** message to the **DefWindowProc** function adds the **WS_VISIBLE** style to the window, if not set, when *wParam* is set to **TRUE**. If your application sends the **WM_SETREDRAW** message with *wParam* set to **TRUE** to a hidden window, then the window becomes visible.

Windows 10 and later; Windows Server 2016 and later. The system sets a property named *SysSetRedraw* on a window whose window procedure passes **WM_SETREDRAW** messages to **DefWindowProc**. You can use the **GetProp** function to get the property value when it's available. **GetProp** returns a non-zero value when redraw is disabled. **GetProp** will return zero when redraw is enabled, or when the window property doesn't exist.

Requirements

Requirement	Value
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

- [Painting and drawing overview](#)
- [Painting and drawing messages](#)
- [RedrawWindow](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WM_SYNCPAINT message

Article • 01/07/2021

The **WM_SYNCPAINT** message is used to synchronize painting while avoiding linking independent GUI threads.

A window receives this message through its **WindowProc** function.

C++

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,  
    UINT uMsg,  
    WPARAM wParam,  
    LPARAM lParam  
) ;
```

Parameters

wParam

This parameter is not used.

lParam

This parameter is not used.

Return value

An application returns zero if it processes this message.

Remarks

When a window has been hidden, shown, moved, or sized, the system may determine that it is necessary to send a **WM_SYNCPAINT** message to the top-level windows of other threads. Applications must pass **WM_SYNCPAINT** to **DefWindowProc** for processing. The **DefWindowProc** function will send a **WM_NCPAINT** message to the window procedure if the window frame must be painted and send a **WM_ERASEBKGND** message if the window background must be erased.

Requirements

Requirement	Value
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

[Painting and Drawing Overview](#)

[Painting and Drawing Messages](#)

[DefWindowProc](#)

[GetDCEx](#)

[GetWindowDC](#)

[WM_PAINT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Raster Operation Codes

Article • 01/07/2021

Raster-operation codes define how the graphics device interface (GDI) combines the bits from the selected pen with the bits in the destination bitmap.

This overview lists and describes the binary and ternary raster operations used by GDI. A [binary raster operation](#) involves two operands: a pen and a destination bitmap. A [ternary raster operation](#) involves three operands: a source bitmap, a brush, and a destination bitmap. Both binary and ternary raster operations use Boolean operators.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Binary Raster Operations

Article • 01/07/2021

This section lists the binary raster-operation codes used by the [GetROP2](#) and [SetROP2](#) functions. Raster-operation codes define how GDI combines the bits from the selected pen with the bits in the destination bitmap.

Each raster-operation code represents a Boolean operation in which the values of the pixels in the selected pen and the destination bitmap are combined. The following are the two operands used in these operations.

[+] Expand table

Operand	Meaning
P	Selected pen
D	Destination bitmap

The Boolean operators used in these operations follow.

[+] Expand table

Operator	Meaning
a	Bitwise AND
n	Bitwise NOT (inverse)
o	Bitwise OR
x	Bitwise exclusive OR (XOR)

All Boolean operations are presented in reverse Polish notation. For example, the following operation replaces the values of the pixels in the destination bitmap with a combination of the pixel values of the pen and the selected brush:

C++

DPo

Each raster-operation code is a 32-bit integer whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended 8-bit value that represents all possible outcomes resulting from the Boolean operation on two parameters (in this case, the pen and destination values). For example, the operation indexes for the DPo and DPan operations are shown in the following list.

[\[+\] Expand table](#)

P	D	DPo	Dpan
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	0

The following list outlines the drawing modes and the Boolean operations that they represent.

[\[+\] Expand table](#)

Raster operation	Boolean operation
R2_BLACK	0
R2_COPYPEN	P
R2_MASKNOTPEN	DPna
R2_MASKPEN	DPa
R2_MASKPENNNOT	PDna
R2_MERGENOTPEN	DPno
R2.MergePen	DPo
R2.MergePenNot	PDno
R2_NOP	D
R2_NOT	Dn
R2_NOTCOPYPEN	Pn

Raster operation	Boolean operation
R2_NOTMASKPEN	DPan
R2_NOTMERGESEN	DPon
R2_NOTXORPEN	DPxn
R2_WHITE	1
R2_XORPEN	DPx

For a monochrome device, GDI maps the value zero to black and the value 1 to white. If an application attempts to draw with a black pen on a white destination by using the available binary raster operations, the following results occur.

[\[+\] Expand table](#)

Raster operation	Result
R2_BLACK	Visible black line
R2_COPYPEN	Visible black line
R2_MASKNOTPEN	No visible line
R2_MASKPEN	Visible black line
R2_MASKPENNOD	Visible black line
R2_MERGENOTPEN	No visible line
R2_MERGESEN	Visible black line
R2_MERGESENNO	Visible black line
R2_NOP	No visible line
R2_NOT	Visible black line
R2_NOTCOPYPEN	No visible line
R2_NOTMASKPEN	No visible line
R2_NOTMERGESEN	Visible black line
R2_NOTXORPEN	Visible black line
R2_WHITE	No visible line

Raster operation	Result
R2_XORPEN	No visible line

For a color device, GDI uses RGB values to represent the colors of the pen and the destination. An RGB color value is a long integer that contains a red, a green, and a blue color field, each specifying the intensity of the specified color. Intensities range from 0 through 255. The values are packed in the three low-order bytes of the long integer. The color of a pen is always a solid color, but the color of the destination may be a mixture of any two or three colors. If an application attempts to draw with a white pen on a blue destination by using the available binary raster operations, the following results occur.

[Expand table](#)

Raster operation	Result
R2_BLACK	Visible black line
R2_COPYPEN	Visible white line
R2_MASKNOTPEN	Visible black line
R2_MASKPEN	Invisible blue line
R2_MASKPENNOD	Visible red/green line
R2_MERGENOTPEN	Invisible blue line
R2_MERGEOPEN	Visible white line
R2_MERGEOPENNOD	Visible white line
R2_NOP	Invisible blue line
R2_NOT	Visible red/green line
R2_NOTCOPYPEN	Visible black line
R2_NOTMASKPEN	Visible red/green line
R2_NOTMERGEOPEN	Visible black line
R2_NOTXORPEN	Invisible blue line
R2_WHITE	Visible white line
R2_XORPEN	Visible red/green line

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Ternary Raster Operations

Article • 01/07/2021

This section lists the ternary raster-operation codes used by the [BitBlt](#), [PatBlt](#), and [StretchBlt](#) functions. Ternary raster-operation codes define how GDI combines the bits in a source bitmap with the bits in the destination bitmap.

Each raster-operation code represents a Boolean operation in which the values of the pixels in the source, the selected brush, and the destination are combined. The following are the three operands used in these operations.

[\[+\] Expand table](#)

Operand	Meaning
D	Destination bitmap
P	Selected brush (also called pattern)
S	Source bitmap

Boolean operators used in these operations follow.

[\[+\] Expand table](#)

Operator	Meaning
a	Bitwise AND
n	Bitwise NOT (inverse)
o	Bitwise OR
x	Bitwise exclusive OR (XOR)

All Boolean operations are presented in reverse Polish notation. For example, the following operation replaces the values of the pixels in the destination bitmap with a combination of the pixel values of the source and brush:

C++

PSo

The following operation combines the values of the pixels in the source and brush with the pixel values of the destination bitmap (there are alternative spellings of the same function, so although a particular spelling may not be in the list, an equivalent form would be):

C++

DPSoo

Each raster-operation code is a 32-bit integer whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended, 8-bit value that represents the result of the Boolean operation on predefined brush, source, and destination values. For example, the operation indexes for the PSo and DPSoo operations are shown in the following list.

[] Expand table

P	S	D	PSo	DPSoo
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
Operation index:			00FCh	00FEh

In this case, PSo has the operation index 00FC (read from the bottom up); DPSoo has the operation index 00FE. These values define the location of the corresponding raster-operation codes, as shown in Table A.1, "Raster-Operation Codes." The PSo operation is in line 252 (00FCh) of the table; DPSoo is in line 254 (00FEh).

The most commonly used raster operations have been given special names in the SDK header file, WINDOWS.H. You should use these names whenever possible in your applications.

When the source and destination bitmaps are monochrome, a bit value of zero represents a black pixel and a bit value of 1 represents a white pixel. When the source and the destination bitmaps are color, those colors are represented with RGB values. For more information about RGB values, see [RGB](#).

Raster-Operation Codes

[] Expand table

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
00	00000042	0	BLACKNESS
01	00010289	DPSoon	
02	00020C89	DPSona	
03	000300AA	PSon	
04	00040C88	SDPona	
05	000500A9	DPon	
06	00060865	PDSxnon	
07	000702C5	PDSaon	
08	00080F08	SDPnaa	
09	00090245	PDSxon	
0A	000A0329	DPna	
0B	000B0B2A	PSDnaon	
0C	000C0324	SPna	
0D	000D0B25	PDSnaon	
0E	000E08A5	PDSonon	
0F	000F0001	Pn	
10	00100C85	PDSona	
11	001100A6	DSon	NOTSRCERASE

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
12	00120868	SDPxnon	
13	001302C8	SDPaon	
14	00140869	DPSxnon	
15	001502C9	DPSaon	
16	00165CCA	PSDPSanaxx	
17	00171D54	SSPxDSxaxn	
18	00180D59	SPxPDxa	
19	00191CC8	SDPSanaxn	
1A	001A06C5	PDSPaox	
1B	001B0768	SDPSxaxn	
1C	001C06CA	PSDPAox	
1D	001D0766	DSPDxaxn	
1E	001E01A5	PDSox	
1F	001F0385	PDSaan	
20	00200F09	DPSnaa	
21	00210248	SDPxon	
22	00220326	DSna	
23	00230B24	SPDnaon	
24	00240D55	SPxDSxa	
25	00251CC5	PDSPanaxn	
26	002606C8	SDPSaox	
27	00271868	SDPSxnox	
28	00280369	DPSxa	
29	002916CA	PSDPSaoxxn	
2A	002A0CC9	DPSana	
2B	002B1D58	SSPxPDxaxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
2C	002C0784	SPDSoax	
2D	002D060A	PSDnox	
2E	002E064A	PSDPxox	
2F	002F0E2A	PSDnoan	
30	0030032A	PSna	
31	00310B28	SDPnaon	
32	00320688	SDPSoox	
33	00330008	Sn	NOTSRCCOPY
34	003406C4	SPDSaox	
35	00351864	SPDSxnox	
36	003601A8	SDPox	
37	00370388	SDPoan	
38	0038078A	PSDPoax	
39	00390604	SPDnox	
3A	003A0644	SPDSxox	
3B	003B0E24	SPDnoan	
3C	003C004A	PSx	
3D	003D18A4	SPDSonox	
3E	003E1B24	SPDSnaox	
3F	003F00EA	PSan	
40	00400F0A	PSDnaa	
41	00410249	DPSxon	
42	00420D5D	SDxPDxa	
43	00431CC4	SPDSanaxn	
44	00440328	SDna	SRCERASE
45	00450B29	DPSnaon	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
46	004606C6	DSPDaox	
47	0047076A	PSDPxaxn	
48	00480368	SDPxax	
49	004916C5	PDSPDaoxxn	
4A	004A0789	DPSDoax	
4B	004B0605	PDSnox	
4C	004C0CC8	SDPana	
4D	004D1954	SSPxDSxoxxn	
4E	004E0645	PDSPxox	
4F	004F0E25	PDSnoan	
50	00500325	PDna	
51	00510B26	DSPnaon	
52	005206C9	DPSDaox	
53	00530764	SPDSxaxn	
54	005408A9	DPSonon	
55	00550009	Dn	DSTINVERT
56	005601A9	DPSox	
57	00570389	DPSoan	
58	00580785	PDSPoax	
59	00590609	DPSnox	
5A	005A0049	DPx	PATINVERT
5B	005B18A9	DPSDonox	
5C	005C0649	DPSDxoxx	
5D	005D0E29	DPSnoan	
5E	005E1B29	DPSDnaox	
5F	005F00E9	DPan	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
60	00600365	PDSxa	
61	006116C6	DSPDSaoxxn	
62	00620786	DSPDoax	
63	00630608	SDPnox	
64	00640788	SDPSoax	
65	00650606	DSPnox	
66	00660046	DSx	SRCINVERT
67	006718A8	SDPSonox	
68	006858A6	DSPDSonoxxn	
69	00690145	PDSxxn	
6A	006A01E9	DPSax	
6B	006B178A	PSDPSoaxxn	
6C	006C01E8	SDPax	
6D	006D1785	PDSPDoaxxn	
6E	006E1E28	SDPSnoax	
6F	006F0C65	PDSxnan	
70	00700CC5	PDSana	
71	00711D5C	SSDxPDxaxn	
72	00720648	SDPSxox	
73	00730E28	SDPnoan	
74	00740646	DSPDxox	
75	00750E26	DSPnoan	
76	00761B28	SDPSnaox	
77	007700E6	DSan	
78	007801E5	PDSax	
79	00791786	DSPDSoaxxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
7A	007A1E29	DPSDnoax	
7B	007B0C68	SDPxnan	
7C	007C1E24	SPDSnoax	
7D	007D0C69	DPSxnan	
7E	007E0955	SPxDsxo	
7F	007F03C9	DPSaan	
80	008003E9	DPSaa	
81	00810975	SPxDsxon	
82	00820C49	DPSxna	
83	00831E04	SPDSnoaxn	
84	00840C48	SDPxna	
85	00851E05	PDSPnoaxn	
86	008617A6	DSPDSoaxx	
87	008701C5	PDSaxn	
88	008800C6	DSa	SRCAND
89	00891B08	SDPSnaoxn	
8A	008A0E06	DSPnoa	
8B	008B0666	DSPDxoxn	
8C	008C0E08	SDPnoa	
8D	008D0668	SDPSxoxn	
8E	008E1D7C	SSDxDxax	
8F	008F0CE5	PDSanan	
90	00900C45	PDSxna	
91	00911E08	SDPSnoaxn	
92	009217A9	DPSDPoaxx	
93	009301C4	SPDaxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
94	009417AA	PSDPSoaxx	
95	009501C9	DPSaxn	
96	00960169	DPSxx	
97	0097588A	PSDPSonoxx	
98	00981888	SDPSonoxn	
99	00990066	DSxn	
9A	009A0709	DPSnax	
9B	009B07A8	SDPSoaxn	
9C	009C0704	SPDnax	
9D	009D07A6	DSPDoaxn	
9E	009E16E6	DSPDSaoxx	
9F	009F0345	PDSxan	
A0	00A000C9	DPa	
A1	00A11B05	PDSPnaoxn	
A2	00A20E09	DPSnoa	
A3	00A30669	DPSDxoxn	
A4	00A41885	PDSPonoxn	
A5	00A50065	PDxn	
A6	00A60706	DSPnax	
A7	00A707A5	PDSPoaxn	
A8	00A803A9	DPSoa	
A9	00A90189	DPSoxn	
AA	00AA0029	D	
AB	00AB0889	DPSono	
AC	00AC0744	SPDSxax	
AD	00AD06E9	DPSDaoxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
AE	00AE0B06	DSPnao	
AF	00AF0229	DPno	
B0	00B00E05	PDSnoa	
B1	00B10665	PDSPxoxn	
B2	00B21974	SSPxDSxox	
B3	00B30CE8	SDPanan	
B4	00B4070A	PSDnax	
B5	00B507A9	DPSDoaxn	
B6	00B616E9	DPSDPaoxx	
B7	00B70348	SDPxan	
B8	00B8074A	PSDPxax	
B9	00B906E6	DSPDaoxn	
BA	00BA0B09	DPSnao	
BB	00BB0226	DSno	MERGEPAINT
BC	00BC1CE4	SPDSanax	
BD	00BD0D7D	SDxDxan	
BE	00BE0269	DPSxo	
BF	00BF08C9	DPSano	
C0	00C000CA	PSa	MERGECOPY
C1	00C11B04	SPDSnaoxn	
C2	00C21884	SPDSonoxn	
C3	00C3006A	PSxn	
C4	00C40E04	SPDnoa	
C5	00C50664	SPDSxoxn	
C6	00C60708	SDPnax	
C7	00C707AA	PSDPoaxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
C8	00C803A8	SDPoa	
C9	00C90184	SPDoxn	
CA	00CA0749	DPSDxax	
CB	00CB06E4	SPDSaoxn	
CC	00CC0020	S	SRCCOPY
CD	00CD0888	SDPono	
CE	00CE0B08	SDPnao	
CF	00CF0224	SPno	
D0	00D00E0A	PSDnoa	
D1	00D1066A	PSDPxoxn	
D2	00D20705	PDSnax	
D3	00D307A4	SPDSoaxn	
D4	00D41D78	SSPxPDxax	
D5	00D50CE9	DPSanan	
D6	00D616EA	PSDPSaoxx	
D7	00D70349	DPSxan	
D8	00D80745	PDSPxax	
D9	00D906E8	SDPSaoxn	
DA	00DA1CE9	DPSDanax	
DB	00DB0D75	SPxDxan	
DC	00DC0B04	SPDnao	
DD	00DD0228	SDno	
DE	00DE0268	SDPxo	
DF	00DF08C8	SDPano	
E0	00E003A5	PDSoa	
E1	00E10185	PDSoxn	

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
E2	00E20746	DSPDxax	
E3	00E306EA	PSDPaoxn	
E4	00E40748	SDPSxax	
E5	00E506E5	PDSPaoxn	
E6	00E61CE8	SDPSanax	
E7	00E70D79	SPxPDxan	
E8	00E81D74	SSPxDSxax	
E9	00E95CE6	DSPDSanaxxn	
EA	00EA02E9	DPSao	
EB	00EB0849	DPSxno	
EC	00EC02E8	SDPao	
ED	00ED0848	SDPxno	
EE	00EE0086	DSo	SRCPAINT
EF	00EF0A08	SDPnoo	
F0	00F00021	P	PATCOPY
F1	00F10885	PDSono	
F2	00F20B05	PDSnao	
F3	00F3022A	PSno	
F4	00F40B0A	PSDnao	
F5	00F50225	PDno	
F6	00F60265	PDSxo	
F7	00F708C5	PDSano	
F8	00F802E5	PDSao	
F9	00F90845	PDSxno	
FA	00FA0089	DPo	
FB	00FB0A09	DPSnoo	PATPAINT

Boolean function (hexadecimal)	Raster operation (hexadecimal)	Boolean function in reverse Polish	Common name
FC	00FC008A	PSo	
FD	00FD0A0A	PSDnoo	
FE	00FE02A9	DPSoo	
FF	00FF0062	1	WHITENESS
8000	80000000		NOMIRRORBITMAP

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Paths (Windows GDI)

Article • 01/07/2021

A *path* is one or more figures (or shapes) that are filled, outlined, or both filled and outlined. Paths are used in drawing and painting applications. Computer-aided design (CAD) applications use paths to create unique clipping regions, to draw outlines of irregular shapes, and to fill the interiors of irregular shapes. An irregular shape is a shape composed of Bézier curves and straight lines. (Regular shapes include ellipses, circles, rectangles, and polygons.)

- [About Paths](#)
- [Using Paths](#)
- [Path Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Paths

Article • 01/07/2021

A path is one of the objects associated with a device context (DC). However, unlike the default objects (the pen, the brush, and the font) that are part of any new DC, there is no default path. For more information about DCs, see [Device Contexts](#).

- [Path Creation](#)
- [Outlined and Filled Paths](#)
- [Transformations of Paths](#)
- [Clip Paths and Graphic Effects](#)
- [Conversion of Paths to Regions](#)
- [Curved Paths](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Path Creation

Article • 01/07/2021

To create a path and select it into a DC, it is first necessary to define the points that describe it. This is done by calling the [BeginPath](#) function, specifying the appropriate drawing functions, and then by calling the [EndPath](#) function. This combination of functions ([BeginPath](#), drawing functions, and [EndPath](#)) constitute a *path bracket*. The following is the list of drawing functions that can be used.

- [AngleArc](#)
- [Arc](#)
- [ArcTo](#)
- [Chord](#)
- [CloseFigure](#)
- [Ellipse](#)
- [ExtTextOut](#)
- [LineTo](#)
- [MoveToEx](#)
- [Pie](#)
- [PolyBezier](#)
- [PolyBezierTo](#)
- [PolyDraw](#)
- [Polygon](#)
- [Polyline](#)
- [PolylineTo](#)
- [PolyPolygon](#)
- [PolyPolyline](#)
- [Rectangle](#)
- [RoundRect](#)
- [TextOut](#)

When an application calls [EndPath](#), the system selects the associated path into the specified DC. (If another path had previously been selected into the DC, the system deletes that path without saving it.) After the system selects the path into the DC, an application can operate on the path in one of the following ways:

- Draw the outline of the path (using the current pen).
- Paint the interior of the path (using the current brush).
- Draw the outline and fill the interior of the path.
- Modify the path (converting curves to line segments).
- Convert the path into a clip path.

- Convert the path into a region.
 - Flatten the path by converting each curve in the path into a series of line segments.
 - Retrieve the coordinates of the lines and curves that compose a path.
-

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

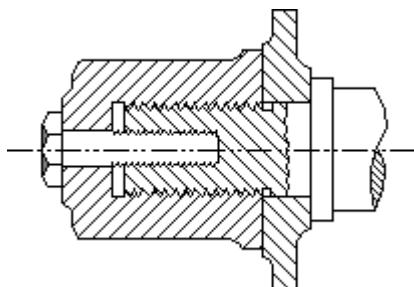
Outlined and Filled Paths

Article • 01/07/2021

An application can draw the outline of a path by calling the [StrokePath](#) function, it can fill the interior of a path by calling the [FillPath](#) function, and it can both outline and fill the path by calling the [StrokeAndFillPath](#) function.

Whenever an application fills a path, the system uses the DC's current fill mode. An application can retrieve this mode by calling the [GetPolyFillMode](#) function, and it can set a new fill mode by calling the [SetPolyFillMode](#) function. For a description of the two fill modes, see [Regions](#).

The following illustration shows the cross-section of an object created by a computer-aided design (CAD) application using paths that were both outlined and filled.



Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Transformations of Paths

Article • 01/07/2021

Paths are defined using logical units and current transformations. (If the [SetWorldTransform](#) function has been called, the logical units are world units; otherwise, the logical units are page units.) An application can use world transformations to scale, rotate, shear, translate, and reflect the lines and Bézier curves that define a path.

ⓘ Note

A world transformation within a path bracket only affects those lines and curves drawn after the transformation was set. It will have no affect on those lines and curves that were drawn before it was set. For a description of the world transformation, see [Coordinate Spaces and Transformations](#).

An application can also use [SetWorldTransform](#) to outline the shape of the pen used to outline a path if the pen is a geometric pen. For a description of geometric pens, see [Pens](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

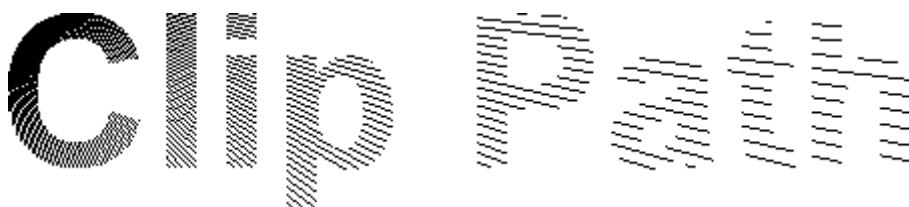
Clip Paths and Graphic Effects

Article • 01/07/2021

An application can use clipping and paths to create special graphic effects. The following illustration shows a string of text drawn with a large Arial font.

Clip Path

The next illustration shows the result of selecting the text as a clip path and drawing radial lines for a circle whose center is located above and left of the string.



ⓘ Note

Before graphics device interface (GDI) adds text created with a bitmapped font to a path, it converts the font to an outline or vector font.

An application creates a clip path by generating a path bracket and then calling the [SelectClipPath](#) function. After a clip path is selected into a DC, output only appears within the borders of the path.

In addition to creating special graphics effects, clip paths are also useful in applications that save images as enhanced metafiles. By using a clip path, an application is able to ensure device independence because the units used to specify a path are logical units.

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Conversion of Paths to Regions

Article • 01/07/2021

An application can convert a path into a region by calling the [PathToRegion](#) function. Like [SelectClipPath](#), [PathToRegion](#) is useful in the creation of special graphics effects. For example, there are no functions that allow an application to offset a path; however, there is a function that enables an application to offset a region ([OffsetRgn](#)). Using [PathToRegion](#), an application can create the effect of animating a complex shape by creating a path that defines the shape, converting the path into a region (by calling [PathToRegion](#)), and then repeatedly painting, moving, and erasing the region (by calling functions in a sequence, such as [FillRgn](#), [OffsetRgn](#), and [FillRgn](#)).

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Curved Paths

Article • 01/07/2021

An application can flatten the curves in a path by calling the [FlattenPath](#) function. This function is especially useful for applications that fit text onto the contour of a path which contains curves. To fit the text, the application must perform the following steps:

1. Create the path where the text appears.
2. Call the [FlattenPath](#) function to convert the curves in a path into line segments.
3. Call the [GetPath](#) function to retrieve those line segments.
4. Calculate the length of each line and the width of each character in the string.
5. Use line-width and character-width data to position each character along the curve.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Paths

Article • 01/07/2021

This section contains a code sample that enables the user to select a font of a particular point size (by using the Choose Font dialog box), select a clip path (by using text drawn with this font), and then view the result of clipping to the text.

This code sample was used to create the illustration that appears in [Clip Paths](#).

C++

```
CHOOSEFONT cf;           // common dialog box font structure
LOGFONT lf;             // logical font structure
HFONT hfont;            // new logical font handle
HFONT hfontOld;          // original logical font handle
HDC hdc;                // display DC handle
int nXStart, nYStart;   // drawing coordinates
RECT rc;                // structure for painting window
SIZE sz;                // structure that receives text extents
double af1Sin[90];      // sine of 0-90 degrees
double af1Cos[90];      // cosine of 0-90 degrees
double flRadius,a;      // radius of circle
int iMode;               // clipping mode
HRGN hrgn;              // clip region handle

LRESULT APIENTRY MainWndProc(
    HWND hwnd,           // window handle
    UINT message,         // type of message
    WPARAM wParam,        // additional information
    LPARAM lParam)        // additional information

{

    PAINTSTRUCT ps;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            EndPaint(hwnd, &ps);
            break;

        case WM_COMMAND:    // command from app's menu
            switch (wParam)
            {
                case IDM_VANISH: // erases client area
                    hdc = GetDC(hwnd);
                    GetClientRect(hwnd, &rc);
                    FillRect(hdc, &rc, GetStockObject(WHITE_BRUSH));
                    ReleaseDC(hwnd, hdc);
                    break;
            }
    }
}
```

```
case IDM_AND: // sets clip mode to RGN_AND
    iMode = RGN_AND;
    break;

case IDM_COPY: // sets clip mode to RGN_COPY
    iMode = RGN_COPY;
    break;

case IDM_DIFF: // sets clip mode to RGN_DIFF
    iMode = RGN_DIFF;
    break;

case IDM_OR: // sets clip mode to RGN_OR
    iMode = RGN_OR;
    break;

case IDM_XOR: // sets clip mode to RGN_XOR
    iMode = RGN_XOR;
    break;

case IDM_CLIP_PATH:
    // Retrieve a cached DC for the window.

    hdc = GetDC(hwnd);

    // Use the font requested by the user in the
    // Choose Font dialog box to create a logical
    // font, then select that font into the DC.

    hfont = CreateFontIndirect(cf.lpData);
    hfontOld = SelectObject(hdc, hfont);

    // Retrieve the dimensions of the rectangle
    // that surrounds the text.

    GetTextExtentPoint32(hdc, "Clip Path", 9, &sz);

    // Set a clipping region using the rect that
    // surrounds the text.

    hrgn = CreateRectRgn(nXStart, nYStart,
        nXStart + sz.cx,
        nYStart + sz.cy);

    SelectClipRgn(hdc, hrgn);

    // Create a clip path using text drawn with
    // the user's requested font.

    BeginPath(hdc);
    TextOut(hdc, nXStart, nYStart, "Clip Path", 9);
    EndPath(hdc);
    SelectClipPath(hdc, iMode);
```

```

// Compute the sine of 0, 1, ... 90 degrees.
for (i = 0; i < 90; i++)
{
    aflSin[i] = sin( (((double)i) / 180.0)
                    * 3.14159);
}

// Compute the cosine of 0, 1,... 90 degrees.
for (i = 0; i < 90; i++)
{
    aflCos[i] = cos( (((double)i) / 180.0)
                    * 3.14159);
}

// Set the radius value.

f1Radius = (double)(2 * sz.cx);

// Draw the 90 rays extending from the
// radius to the edge of the circle.

for (i = 0; i < 90; i++)
{
    MoveToEx(hdc, nXStart, nYStart,
              (LPOINT) NULL);
    LineTo(hdc, nXStart + ((int) (f1Radius
        * aflCos[i])),
            nYStart + ((int) (f1Radius
        * aflSin[i])));
}

// Reselect the original font into the DC.

SelectObject(hdc, hfontOld);

// Delete the user's font.

DeleteObject(hfont);

// Release the DC.

ReleaseDC(hwnd, hdc);

break;

case IDM_FONT:

// Initialize necessary members.

cf.lStructSize = sizeof (CHOOSEFONT);
cf.hwndOwner = hwnd;
cf.lpLogFont = &lf;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;

```

```

        cf.rgbColors = RGB(0, 255, 255);
        cf.nFontType = SCREEN_FONTTYPE;

        // Display the Font dialog box, allow the user
        // to choose a font, and render text in the
        // window with that selection.

        if (ChooseFont(&cf))
        {
            hdc = GetDC(hwnd);
            hfont = CreateFontIndirect(cf.lpLogFont);
            hfontOld = SelectObject(hdc, hfont);
            crOld = SetTextColor(hdc, cf.rgbColors);
            TextOut(hdc, nXStart, nYStart,
                    "Clip Path", 9);
            SetTextColor(hdc, crOld);
            SelectObject(hdc, hfontOld);
            DeleteObject(hfont);
            ReleaseDC(hwnd, hdc);
        }

        break;

    default:
        return DefWindowProc(hwnd, message, wParam,
                             lParam);
    }
    break;

    case WM_DESTROY:    // window is being destroyed
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hwnd, message, wParam, lParam);
}
return 0;
}

```

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Path Reference

Article • 01/07/2021

The following elements are used to create, alter, or draw paths.

- Path Functions

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Path Functions (Windows GDI)

Article • 01/07/2021

The following functions are used to create, alter, or draw paths.

[+] Expand table

Function	Description
AbortPath	Closes and discards any paths in the specified device context.
BeginPath	Opens a path bracket in the specified device context.
CloseFigure	Closes an open figure in a path.
EndPath	Closes a path bracket and selects the path defined by the bracket into the specified device context.
FillPath	Closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.
FlattenPath	Transforms any curves in the path that is selected into the current device context (DC), turning each curve into a sequence of lines.
GetMiterLimit	Retrieves the miter limit for the specified device context.
GetPath	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.
PathToRegion	Creates a region from the path that is selected into the specified device context.
SetMiterLimit	Sets the limit for the length of miter joins for the specified device context.
StrokeAndFillPath	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	Renders the specified path by using the current pen.
WidenPath	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.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

AbortPath function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[EndPath](#)

[Path Functions](#)

[Paths Overview](#)

BeginPath function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CloseFigure function (wingdi.h)

Article02/22/2024

The **CloseFigure** function closes an open figure in a path.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

EndPath function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

FillPath function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[BeginPath](#)

[Path Functions](#)

[Paths Overview](#)

[SetPolyFillMode](#)

[StrokeAndFillPath](#)

[StrokePath](#)

FlattenPath function (wingdi.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetMiterLimit function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

GetPath function (wingdi.h)

Article 11/19/2022

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

C++

```
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.

 Expand table

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.

PT_BEZIERTO 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 PT_BEZIERTO points are the control points, and the third PT_BEZIERTO 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.

 Expand table

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PathToRegion function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

SetMiterLimit function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

StrokeAndFillPath function (wingdi.h)

Article10/13/2021

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

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

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

Syntax

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

WidenPath function (wingdi.h)

Article02/22/2024

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

C++

```
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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Pens

Article • 01/07/2021

A pen is a graphics tool that an application can use to draw lines and curves. Drawing applications use pens to draw freehand lines, straight lines, and curves. Computer-aided design (CAD) applications use pens to draw visible lines, hidden lines, section lines, center lines, and so on. Word processing and desktop publishing applications use pens to draw borders and rules. Spreadsheet applications use pens to designate trends in graphs and to outline bar graphs and pie charts.

- [About Pens](#)
- [Using Pens](#)
- [Pen Reference](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Pens

Article • 01/07/2021

There are two types of pen: cosmetic and geometric. A *cosmetic pen* is used with applications requiring lines of fixed width and lines that are quickly drawn. A CAD application, for example, uses a cosmetic pen to generate hidden, section, center, and dimension lines that are between .015 and .022 inches wide regardless of the scale factor. A *geometric pen* is used with applications requiring scalable lines, lines with unique end or join styles, and lines that are wider than a single pixel. A spreadsheet application, for example, uses a geometric pen to define each of the bars in a bar graph as a wide line.

- [Cosmetic Pens](#)
- [Geometric Pens](#)
- [Pen Attributes](#)
- [ICM-Enabled Pen Functions](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Cosmetic Pens

Article • 01/07/2021

The dimensions of a cosmetic pen are specified in device units. Therefore, lines drawn with a cosmetic pen always have a fixed width. Lines drawn with a cosmetic pen are generally drawn 3 to 10 times faster than lines drawn with a geometric pen. Cosmetic pens have three attributes: width, style, and color. For more information about these attributes, see [Pen Attributes](#).

To create a cosmetic pen, use the [CreatePen](#), [CreatePenIndirect](#), or [ExtCreatePen](#) function. To retrieve one of the three stock cosmetic pens managed by the system, use the [GetStockObject](#) function.

After you create a pen (or obtain a handle to one of the stock pens), select the pen into the application's device context (DC) using the [SelectObject](#) function. From this point on, the application uses this pen for any line-drawing operations in its client area.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Geometric Pens

Article • 01/07/2021

The dimensions of a geometric pen are specified in logical units. Therefore, lines drawn with a geometric pen can be scaled that is, they may appear wider or narrower, depending on the current world transformation. For more information about world transformation, see [Coordinate Spaces and Transformations](#).

In addition to the three attributes shared with cosmetic pens (width, style, and color), geometric pens possess the following four attributes: pattern, optional hatch, end style, and join style. For more information about these attributes, see [Pen Attributes](#).

To create a geometric pen, an application uses the [ExtCreatePen](#) function. As with cosmetic pens, the [SelectObject](#) function selects a geometric pen into the application's DC.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Attributes

Article • 01/07/2021

There are seven pen attributes that define the type of pen and its characteristics: width, style, color, pattern, hatch, end style, and join style. Both cosmetic and geometric pens have the width, style, and color attributes. Only geometric pens have the pattern, hatch, end style, and join style attributes. The pattern and optional hatch attribute are usually associated with a brush but can also be used with geometric pens.

For more information, see the following topics:

- [Pen Width](#)
- [Pen Style](#)
- [Pen Color](#)
- [Pen Pattern](#)
- [Pen Hatch](#)
- [Pen End Cap](#)
- [Pen Join](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Width

Article • 01/07/2021

The width attribute specifies a cosmetic pen width in device units. When used with a geometric pen, however, it specifies the pen's width in logical units. For more information about device units, see [Coordinate Spaces and Transformations](#).

Currently, the system limits the width of cosmetic pens to a single pixel; however, future versions may remove this limitation.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Pen Style

Article • 01/07/2021

The style attribute specifies the line pattern that appears when a particular cosmetic or geometric pen is used. There are eight predefined pen styles. The following illustration shows the seven of these styles that are defined by the system.

Solid	_____
Dash	-----
Dot
Dash-dot	-.-
Dash-dot-dot	-...-
Null	
Inside-frame	_____

The inside-frame style is identical to the solid style for cosmetic pens. However, it operates differently when used with a geometric pen. If the geometric pen is wider than a single pixel and a drawing function uses the pen to draw a border around a filled object, the system draws the border inside the object's frame. By using the inside-frame style, an application can ensure that an object appears entirely within the specified dimensions, regardless of the geometric pen width.

In addition to the seven styles defined by the system, there is an eighth style that is user (or application) defined. A user-defined style generates lines with a customized series of dashes and dots.

Use the [CreatePen](#), [CreatePenIndirect](#), or [ExtCreatePen](#) function to create a pen that has the system-defined styles. Use the [ExtCreatePen](#) function to create a pen that has a user-defined style.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Color

Article • 01/07/2021

The color attribute specifies the pen's color. An application can create a cosmetic pen with a unique color by using the [RGB](#) macro to store the red, green, blue triplet that specifies the desired color in a [COLORREF](#) structure and passing this structure's address to the [CreatePen](#), [CreatePenIndirect](#), or [ExtCreatePen](#) function. (The stock pens are limited to black, white, and invisible.) For more information about RGB triplets and color, see [Colors](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Pattern

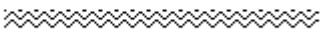
Article • 01/07/2021

The pattern attribute specifies the pattern of a geometric pen.

The following illustration shows lines drawn with different geometric pens. Each pen was created using a different pattern attribute.

Hatch 

Hollow

Custom 

Solid 

The first line in the previous illustration is drawn using one of the six available hatch patterns; for more information about hatch patterns, see [Pen Hatch](#). The next line is drawn using the hollow pattern, identical to the null pattern. The third line is drawn using a custom pattern created from an 8-by-8-pixel bitmap. (For more information about bitmaps and their creation, see [Bitmaps](#).) The last line is drawn using a solid pattern. Creating a brush and passing its handle to the [ExtCreatePen](#) function creates a pattern.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Hatch

Article • 01/07/2021

The hatch attribute specifies the hatch type of a geometric pen with the hatch pattern attribute. There are six patterns available. The following illustration shows lines drawn using different hatch patterns.

Forward diagonal	
Cross	
Diagonal cross	
Backward diagonal	
Horizontal	
Vertical	

Feedback

Was this page helpful?

 Yes

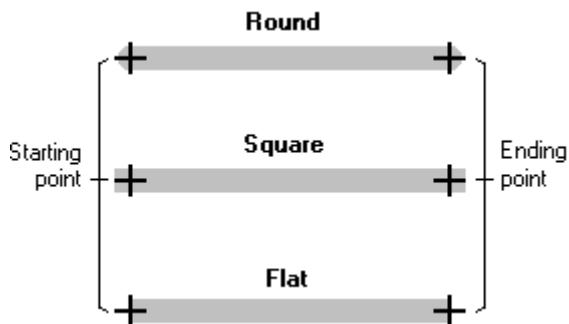
 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Pen End Cap

Article • 01/07/2021

The end cap attribute specifies the shape of a geometric pen: round, square, or flat. The following illustration shows parallel lines drawn using each type of end cap.



The round and square end caps extend past the starting and ending points of a line drawn with a geometric pen; the flat end cap does not.

Feedback

Was this page helpful?

Yes

No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Pen Join

Article • 01/07/2021

The join attribute specifies how the ends of two geometric lines are joined: beveled, mitered, or round. The following illustration shows pairs of connected lines drawn using each type of join.

Bevel join 

Round join 

Miter join 

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ICM-Enabled Pen Functions

Article • 01/07/2021

Microsoft Image Color Management (ICM) ensures that a color image, graphic, or text object is rendered as close as possible to its original intent on any device, despite differences in imaging technologies and color capabilities among devices. Whether you are scanning an image or other graphic on a color scanner, downloading it over the Internet, viewing or editing it on the screen, or outputting it to paper, film, or other media, ICM version 2.0 helps you keep its colors consistent and accurate. For more information about ICM, see [Windows Color System](#)

There are various functions in the graphics device interface (GDI) that use or operate on color data. The following pen functions are enabled for use with ICM:

- [CreatePen](#)
- [ExtCreatePen](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Using Pens

Article • 01/07/2021

This section contains sample code that demonstrates the appearance of lines drawn using various pen styles and attributes.

C++

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam,
                         LPARAM lParam)
{
    PAINTSTRUCT ps;
    LOGBRUSH lb;
    RECT rc;
    HDC hdc;
    int i;
    HGDIOBJ hPen = NULL;
    HGDIOBJ hPenOld;
    DWORD dwPenStyle[] = {
        PS_DASH,
        PS_DASHDOT,
        PS_DOT,
        PS_INSIDEFRAME,
        PS_NULL,
        PS_SOLID
    };
    UINT uHatch[] = {
        HS_BDIAGONAL,
        HS_CROSS,
        HS_DIAGCROSS,
        HS_FDIAGONAL,
        HS_HORIZONTAL,
        HS_VERTICAL
    };

    switch (uMsg)
    {
        case WM_PAINT:
        {
            GetClientRect(hWnd, &rc);
            rc.left += 10;
            rc.top += 10;
            rc.bottom -= 10;

            // Initialize the pen's brush.
            lb.lbStyle = BS_SOLID;
            lb.lbColor = RGB(255,0,0);
            lb.lbHatch = 0;

            hdc = BeginPaint(hWnd, &ps);
            for (i = 0; i < 6; i++)
                DrawLine(hdc, rc.left + 10 * i, rc.top + 10 * i, rc.left + 10 * i, rc.bottom - 10 * i);
        }
    }
}
```

```

    {
        hPen = ExtCreatePen(PS_COSMETIC | dwPenStyle[i],
                            1, &lb, 0, NULL);
        hPenOld = SelectObject(hdc, hPen);
        MoveToEx(hdc, rc.left + (i * 20), rc.top, NULL);
        LineTo(hdc, rc.left + (i * 20), rc.bottom);
        SelectObject(hdc, hPenOld);
        DeleteObject(hPen);
    }
    rc.left += 150;
    for (i = 0; i < 6; i++)
    {
        lb.lbStyle = BS_HATCHED;
        lb.lbColor = RGB(0,0,255);
        lb.lbHatch = uHatch[i];
        hPen = ExtCreatePen(PS_GEOMETRIC,
                            5, &lb, 0, NULL);
        hPenOld = SelectObject(hdc, hPen);
        MoveToEx(hdc, rc.left + (i * 20), rc.top, NULL);
        LineTo(hdc, rc.left + (i * 20), rc.bottom);
        SelectObject(hdc, hPenOld);
        DeleteObject(hPen);
    }
    EndPaint(hWnd, &ps);
}

break;

case WM_DESTROY:
    DeleteObject(hPen);
    PostQuitMessage(0);
    break;

default:
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

return FALSE;
}

```

Feedback

Was this page helpful?

 Yes

 No

Pen Reference

Article • 01/07/2021

The following elements are used with pens:

- [Pen Functions](#)
- [Pen Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Pen Functions (Windows GDI)

Article • 01/07/2021

The following functions are used with pens.

 Expand table

Function	Description
CreatePen	Creates a logical pen that has the specified style, width, and color.
CreatePenIndirect	Creates a logical cosmetic pen that has the style, width, and color specified in a structure.
ExtCreatePen	Creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.
SetDCPenColor	Sets the current device context pen color.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

CreatePen function (wingdi.h)

Article 08/23/2022

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

C++

```
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.

[Expand table](#)

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

ExtCreatePen function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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.

 Expand table

Value	Meaning
PS_GEOMETRIC	The pen is geometric.
PS_COSMETIC	The pen is cosmetic.

The pen style can be one of the following values.

 Expand table

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.

 [Expand table](#)

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.

 [Expand table](#)

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] plbrush

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 *lplb* 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

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib

Requirement	Value
DLL	Gdi32.dll

See also

[CreateDIBSection](#)

[CreatePen](#)

[CreatePenIndirect](#)

[DeleteObject](#)

[GetObject](#)

[LOGBRUSH](#)

[Pen Functions](#)

[Pens Overview](#)

[SelectObject](#)

[SetMiterLimit](#)

Pen Structures

Article • 01/07/2021

The following structures are used with pens:

- [EXTLOGPEN](#)
- [LOGPEN](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

EXTLOGOPEN structure (wingdi.h)

Article09/01/2022

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

C++

```
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.

[] [Expand table](#)

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:

[Expand table](#)

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.

[Expand table](#)

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.

[Expand table](#)

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.

 Expand table

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.

[+] Expand table

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.

[+] Expand table

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[1]`

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

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

LOGPEN structure (wingdi.h)

Article 02/22/2024

The **LOGPEN** structure defines the style, width, and color of a pen. The [CreatePenIndirect](#) function uses the **LOGPEN** structure.

Syntax

C++

```
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.

[+] [Expand table](#)

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 `x` member is `NULL`, then the pen is one pixel wide on raster devices. The `y` member in the [POINT](#) structure for `lopnWidth` isn't 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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	<code>wingdi.h</code> (include <code>Windows.h</code>)

See also

[COLORREF](#)

[CreatePenIndirect](#)

[POINT](#)

[Pen Structures](#)

[Pens Overview](#)

[RGB](#)

Feedback

Was this page helpful?

 Yes

 No

Rectangles

Article • 01/07/2021

Applications use *rectangles* to specify rectangular areas on the screen or in a window.

- [About Rectangles](#)
 - [Using Rectangles](#)
 - [Rectangle Reference](#)
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

About Rectangles

Article • 01/07/2021

Rectangles are used for the cursor clipping region, the invalid portion of the client area, an area for displaying formatted text, or the scroll area. Your applications can also use rectangles to fill, frame, or invert a portion of the client area with a given brush, and to retrieve the coordinates of a window or a window's client area.

- [Rectangle Coordinates](#)
- [Rectangle Operations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Rectangle Coordinates

Article • 11/19/2022

An application must use a **RECT** structure to define a rectangle. The structure specifies the coordinates of two points: the upper left and lower right corners of the rectangle. The sides of the rectangle extend from these two points and are parallel to the x-axis and y-axis.

The coordinate values for a rectangle are expressed as signed integers. The coordinate value of a rectangle's right side must be greater than that of its left side. Likewise, the coordinate value of the bottom must be greater than that of the top.

Because applications can use rectangles for many 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. For more information about coordinates and mapping modes, see [Coordinate Spaces and Transformations](#).

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Rectangle Operations

Article • 01/07/2021

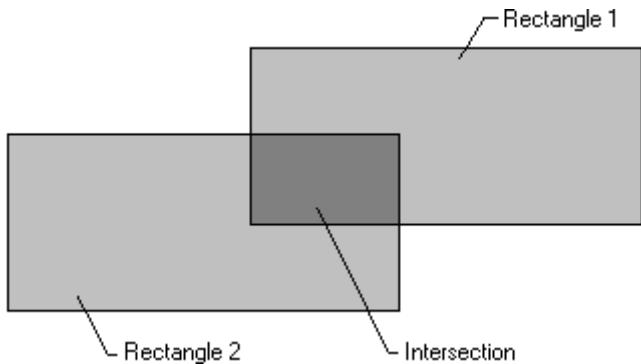
The [SetRect](#) function creates a rectangle, the [CopyRect](#) function makes a copy of a given rectangle, and the [SetRectEmpty](#) function creates an empty rectangle. An empty rectangle is any rectangle that has zero width, zero height, or both. The [IsRectEmpty](#) function determines whether a given rectangle is empty. The [EqualRect](#) function determines whether two rectangles are identical that is, whether they have the same coordinates.

The [InflateRect](#) function increases or decreases the width or height of a rectangle, or both. It can add or remove width from both ends of the rectangle; it can add or remove height from both the top and bottom of the rectangle.

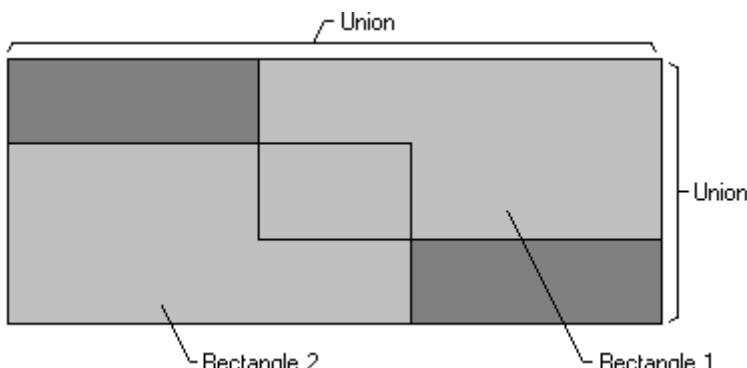
The [OffsetRect](#) function moves a rectangle by a given amount. It moves the rectangle by adding the given x-amount, y-amount, or x- and y-amounts to the corner coordinates.

The [PtInRect](#) function determines whether a given point lies within a given rectangle. The point is in the rectangle if it lies on the left or top side or is completely within the rectangle. The point is not in the rectangle if it lies on the right or bottom side.

The [IntersectRect](#) function creates a new rectangle that is the intersection of two existing rectangles, as shown in the following figure.



The [UnionRect](#) function creates a new rectangle that is the union of two existing rectangles, as shown in the following figure.



For information about functions that draw ellipses and polygons, see [Filled Shapes](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Rectangles (Windows GDI)

Article • 01/07/2021

The example in this section illustrates how to use the rectangle functions. It consists of the main window procedure from an application that enables the user to move and size a bitmap.

When the application starts, it draws a 32-pixel by 32-pixel bitmap in the upper left corner of the screen. The user can move the bitmap by dragging it. To size the bitmap, the user creates a target rectangle by dragging the mouse, then drags the bitmap and "drops" it on the target rectangle. The application responds by copying the bitmap into the target rectangle.

The window procedure that allows the user to move and size the bitmap is given in the following example.

C++

```
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;                      // device context (DC) for window
    RECT rcTmp;                   // temporary rectangle
    PAINTSTRUCT ps;               // paint data for BeginPaint and EndPaint
    POINT ptClientUL;             // client area upper left corner
    POINT ptClientLR;             // client area lower right corner
    static HDC hdcCompat;          // DC for copying bitmap
    static POINT pt;                // x and y coordinates of cursor
    static RECT rcBmp;              // rectangle that encloses bitmap
    static RECT rcTarget;            // rectangle to receive bitmap
    static RECT rcClient;           // client-area rectangle
    static BOOL fDragRect;           // TRUE if bitmap rect. is dragged
    static HBITMAP hbmp;             // handle of bitmap to display
    static HBRUSH hbrBkgnd;          // handle of background-color brush
    static COLORREF crBkgnd;         // color of client-area background
    static HPEN hpenDot;              // handle of dotted pen

    switch (uMsg)
    {
        case WM_CREATE:

            // Load the bitmap resource.

            hbmp = LoadBitmap(hinst, MAKEINTRESOURCE(1));

            // Create a device context (DC) to hold the bitmap.
            // The bitmap is copied from this DC to the window's DC
            // whenever it must be drawn.
```

```
hdc = GetDC(hwnd);
hdcCompat = CreateCompatibleDC(hdc);
SelectObject(hdcCompat, hbmp);

// Create a brush of the same color as the background
// of the client area. The brush is used later to erase
// the old bitmap before copying the bitmap into the
// target rectangle.

crBkgnd = GetBkColor(hdc);
hbrBkgnd = CreateSolidBrush(crBkgnd);
ReleaseDC(hwnd, hdc);

// Create a dotted pen. The pen is used to draw the
// bitmap rectangle as the user drags it.

hpenDot = CreatePen(PS_DOT, 1, RGB(0, 0, 0));

// Set the initial rectangle for the bitmap. Note that
// this application supports only a 32- by 32-pixel
// bitmap. The rectangle is slightly larger than the
// bitmap.

SetRect(&rcBmp, 1, 1, 34, 34);
return 0;

case WM_PAINT:

    // Draw the bitmap rectangle and copy the bitmap into
    // it. The 32-pixel by 32-pixel bitmap is centered
    // in the rectangle by adding 1 to the left and top
    // coordinates of the bitmap rectangle, and subtracting 2
    // from the right and bottom coordinates.

    BeginPaint(hwnd, &ps);
    Rectangle(ps.hdc, rcBmp.left, rcBmp.top,
              rcBmp.right, rcBmp.bottom);
    StretchBlt(ps.hdc, rcBmp.left + 1, rcBmp.top + 1,
               (rcBmp.right - rcBmp.left) - 2,
               (rcBmp.bottom - rcBmp.top) - 2, hdcCompat,
               0, 0, 32, 32, SRCCOPY);
    EndPaint(hwnd, &ps);
    break;

case WM_MOVE:
case WM_SIZE:

    // Convert the client coordinates of the client-area
    // rectangle to screen coordinates and save them in a
    // rectangle. The rectangle is passed to the ClipCursor
    // function during WM_LBUTTONDOWN processing.

    GetClientRect(hwnd, &rcClient);
    ptClientUL.x = rcClient.left;
    ptClientUL.y = rcClient.top;
```

```

ptClientLR.x = rcClient.right;
ptClientLR.y = rcClient.bottom;
ClientToScreen(hwnd, &ptClientUL);
ClientToScreen(hwnd, &ptClientLR);
SetRect(&rcClient, ptClientUL.x, ptClientUL.y,
        ptClientLR.x, ptClientLR.y);
return 0;

case WM_LBUTTONDOWN:

    // Restrict the mouse cursor to the client area. This
    // ensures that the window receives a matching
    // WM_LBUTTONUP message.

    ClipCursor(&rcClient);

    // Save the coordinates of the mouse cursor.

    pt.x = (LONG) LOWORD(lParam);
    pt.y = (LONG) HIWORD(lParam);

    // If the user has clicked the bitmap rectangle, redraw
    // it using the dotted pen. Set the fDragRect flag to
    // indicate that the user is about to drag the rectangle.

    if (PtInRect(&rcBmp, pt))
    {
        hdc = GetDC(hwnd);
        SelectObject(hdc, hpenDot);
        Rectangle(hdc, rcBmp.left, rcBmp.top, rcBmp.right,
                  rcBmp.bottom);
        fDragRect = TRUE;
        ReleaseDC(hwnd, hdc);
    }
    return 0;

case WM_MOUSEMOVE:

    // Draw a target rectangle or drag the bitmap rectangle,
    // depending on the status of the fDragRect flag.

    if ((wParam && MK_LBUTTON)
        && !fDragRect)
    {
        // Set the mix mode so that the pen color is the
        // inverse of the background color. The previous
        // rectangle can then be erased by drawing
        // another rectangle on top of it.

        hdc = GetDC(hwnd);
        SetROP2(hdc, R2_NOTXORPEN);

        // If a previous target rectangle exists, erase
        // it by drawing another rectangle on top of it.

```

```

    if (!IsRectEmpty(&rcTarget))
    {
        Rectangle(hdc, rcTarget.left, rcTarget.top,
                  rcTarget.right, rcTarget.bottom);
    }

    // Save the coordinates of the target rectangle. Avoid
    // invalid rectangles by ensuring that the value of
    // the left coordinate is lesser than the
    // right coordinate, and that the value of the top
    // coordinate is lesser than the bottom coordinate.

    if ((pt.x < (LONG) LOWORD(lParam)) &&
        (pt.y > (LONG) HIWORD(lParam)))
    {
        SetRect(&rcTarget, pt.x, HIWORD(lParam),
                LOWORD(lParam), pt.y);
    }
    else if ((pt.x > (LONG) LOWORD(lParam)) &&
              (pt.y > (LONG) HIWORD(lParam)))
    {
        SetRect(&rcTarget, LOWORD(lParam),
                HIWORD(lParam), pt.x, pt.y);
    }
    else if ((pt.x > (LONG) LOWORD(lParam)) &&
              (pt.y < (LONG) HIWORD(lParam)))
    {
        SetRect(&rcTarget, LOWORD(lParam), pt.y,
                pt.x, HIWORD(lParam));
    }
    else
    {
        SetRect(&rcTarget, pt.x, pt.y, LOWORD(lParam),
                HIWORD(lParam));
    }

    // Draw the new target rectangle.

    Rectangle(hdc, rcTarget.left, rcTarget.top,
              rcTarget.right, rcTarget.bottom);
    ReleaseDC(hwnd, hdc);
}

else if ((wParam && MK_LBUTTON)
          && fDragRect)
{
    // Set the mix mode so that the pen color is the
    // inverse of the background color.

    hdc = GetDC(hwnd);
    SetROP2(hdc, R2_NOTXOROPEN);

    // Select the dotted pen into the DC and erase
    // the previous bitmap rectangle by drawing
    // another rectangle on top of it.
}

```

```

        SelectObject(hdc, hpenDot);
        Rectangle(hdc, rcBmp.left, rcBmp.top,
                   rcBmp.right, rcBmp.bottom);

        // Set the new coordinates of the bitmap rectangle,
        // then redraw it.

        OffsetRect(&rcBmp, LOWORD(lParam) - pt.x,
                   HIWORD(lParam) - pt.y);
        Rectangle(hdc, rcBmp.left, rcBmp.top,
                   rcBmp.right, rcBmp.bottom);
        ReleaseDC(hwnd, hdc);

        // Save the coordinates of the mouse cursor.

        pt.x = (LONG) LOWORD(lParam);
        pt.y = (LONG) HIWORD(lParam);
    }

    return 0;
}

case WM_LBUTTONDOWN:
    // If the bitmap rectangle and target rectangle
    // intersect, copy the bitmap into the target
    // rectangle. Otherwise, copy the bitmap into the
    // rectangle bitmap at its new location.

    if (IntersectRect(&rcTmp, &rcBmp, &rcTarget))
    {

        // Erase the bitmap rectangle by filling it with
        // the background color.

        hdc = GetDC(hwnd);
        FillRect(hdc, &rcBmp, hbrBkgnd);

        // Redraw the target rectangle because the part
        // that intersected with the bitmap rectangle was
        // erased by the call to FillRect.

        Rectangle(hdc, rcTarget.left, rcTarget.top,
                   rcTarget.right, rcTarget.bottom);

        // Copy the bitmap into the target rectangle.

        StretchBlt(hdc, rcTarget.left + 1, rcTarget.top + 1,
                   (rcTarget.right - rcTarget.left) - 2,
                   (rcTarget.bottom - rcTarget.top) - 2, hdcCompat,
                   0, 0, 32, 32, SRCCOPY);

        // Copy the target rectangle to the bitmap
        // rectangle, set the coordinates of the target
        // rectangle to 0, then reset the fDragRect flag.

```

```

        CopyRect(&rcBmp, &rcTarget);
        SetRectEmpty(&rcTarget);
        ReleaseDC(hwnd, hdc);
        fDragRect = FALSE;
    }

    else if (fDragRect)
    {

        // Draw the bitmap rectangle, copy the bitmap into
        // it, and reset the fDragRect flag.

        hdc = GetDC(hwnd);
        Rectangle(hdc, rcBmp.left, rcBmp.top,
                  rcBmp.right, rcBmp.bottom);
        StretchBlt(hdc, rcBmp.left + 1, rcBmp.top + 1,
                    (rcBmp.right - rcBmp.left) - 2,
                    (rcBmp.bottom - rcBmp.top) - 2, hdcCompat,
                    0, 0, 32, 32, SRCCOPY);
        ReleaseDC(hwnd, hdc);
        fDragRect = FALSE;
    }

    // Release the mouse cursor.

    ClipCursor((LPRECT) NULL);
    return 0;

case WM_DESTROY:

    // Destroy the background brush, compatible bitmap,
    // and the bitmap.

    DeleteObject(hbrBkgnd);
    DeleteDC(hdcCompat);
    DeleteObject(hbmp);
    PostQuitMessage(0);
    break;

default:
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

return (LRESULT) NULL;
}

```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Rectangle Reference

Article • 01/07/2021

The following elements are used with rectangles:

- Rectangle Functions
- Rectangle Structures
- Rectangle Macros

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Rectangle Functions

Article • 01/07/2021

The following functions are used with rectangles.

 Expand table

Function	Description
CopyRect	Copies the coordinates of one rectangle to another.
EqualRect	Determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.
InflateRect	Increases or decreases the width and height of the specified rectangle.
IntersectRect	Calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle.
IsRectEmpty	Determines whether the specified rectangle is empty.
OffsetRect	Moves the specified rectangle by the specified offsets.
PtInRect	Determines whether the specified point lies within the specified rectangle.
SetRect	Sets the coordinates of the specified rectangle.
SetRectEmpty	Creates an empty rectangle in which all coordinates are set to zero.
SubtractRect	Determines the coordinates of a rectangle formed by subtracting one rectangle from another.
UnionRect	Creates the union of two rectangles.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

CopyRect function (winuser.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

EqualRect function (winuser.h)

Article10/13/2021

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

InflateRect function (winuser.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

IsRectEmpty function (winuser.h)

Article02/22/2024

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

OffsetRect function (winuser.h)

Article02/22/2024

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

PtInRect function (winuser.h)

Article11/19/2022

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

C++

```
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

 [Expand table](#)

Requirement	Value
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](#)

SetRect function (winuser.h)

Article02/22/2024

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

C++

```
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

 Expand table

Requirement	Value
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)

Article10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

SubtractRect function (winuser.h)

Article 10/13/2021

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

UnionRect function (winuser.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
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](#)

Rectangle Structures

Article • 11/19/2022

The following structures are used with rectangles.

- [POINT](#)
 - [POINTS](#)
 - [RECT](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

POINT structure (windef.h)

Article 02/22/2024

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

Syntax

C++

```
typedef struct tagPOINT {  
    LONG x;  
    LONG y;  
} POINT, *PPOINT, *NPPOINT, *LPPOINT;
```

Members

x

Specifies the x-coordinate of the point.

y

Specifies the y-coordinate of the point.

Remarks

The POINT structure is identical to the [POINTL](#) structure.

Requirements

 Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

[POINTL](#)

POINTS

Feedback

Was this page helpful?

 Yes

 No

POINTS structure (windef.h)

Article 02/22/2024

The POINTS structure defines the x- and y-coordinates of a point.

Syntax

C++

```
typedef struct tagPOINTS {
    #if ...
        SHORT x;
    #if ...
        SHORT y;
    #else
        SHORT y;
    #endif
    #else
        SHORT x;
    #endif
} POINTS, *PPOINTS, *LPPPOINTS;
```

Members

x

Specifies the x-coordinate of the point.

y

Specifies the y-coordinate of the point.

Remarks

The POINTS structure is similar to the [POINT](#) and [POINTL](#) structures. The difference is that the members of the POINTS structure are of type `SHORT`, while those of the other two structures are of type `LONG`.

Requirements

 Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

[POINT](#)

[POINTL](#)

Feedback

Was this page helpful?

 Yes

 No

RECT structure (windef.h)

Article 02/22/2024

The RECT structure defines a rectangle by the coordinates of its upper-left and lower-right corners.

Syntax

C++

```
typedef struct tagRECT {  
    LONG left;  
    LONG top;  
    LONG right;  
    LONG bottom;  
} RECT, *PRECT, *NPRECT, *LPRECT;
```

Members

`left`

Specifies the x-coordinate of the upper-left corner of the rectangle.

`top`

Specifies the y-coordinate of the upper-left corner of the rectangle.

`right`

Specifies the x-coordinate of the lower-right corner of the rectangle.

`bottom`

Specifies the y-coordinate of the lower-right corner of the rectangle.

Remarks

The RECT structure is identical to the [RECTL](#) structure.

Requirements

Expand table

Requirement	Value
Header	windef.h (include Windows.h)

See also

[RECTL](#)

Feedback

Was this page helpful?

 Yes

 No

Rectangle Macros

Article • 01/07/2021

The following macros are used with rectangles.

- [MAKEPOINTS](#)
- [POINTSTOPOINT](#)
- [POINTTOPOINTS](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

MAKEPOINTS macro (wingdi.h)

07/09/2025

The **MAKEPOINTS** macro converts a value that contains the x- and y-coordinates of a point into a [POINTS](#) structure.

Syntax

C++

```
POINTS MAKEPOINTS(
    DWORD l
);
```

Parameters

l

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

Type: [POINTS](#)

The return value is a pointer to a [POINTS](#) structure.

Requirements

 Expand table

Requirement	Value
Minimum supported client	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](#)

POINTSTOPOINT macro (winuser.h)

07/01/2025

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

Syntax

C++

```
void POINTSTOPOINT(
    POINT pt,
    POINTS pts
);
```

Parameters

pt

The **POINT** structure to receive the contents of the **POINTS** structure.

pts

The **POINTS** structure to copy.

Return value

None

Requirements

 Expand table

Requirement	Value
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](#)

[POINTTOPOLY](#)

[Rectangle Macros](#)

[Rectangles Overview](#)

POINTTOPPOINTS macro (winuser.h)

07/09/2025

The **POINTTOPPOINTS** macro converts a [POINT](#) structure to a [POINTS](#) structure.

Syntax

C++

```
POINTS POINTTOPPOINTS(  
    POINT pt  
)
```

Parameters

pt

The [POINT](#) structure to convert.

Return value

Type: [POINTS](#)

The return value is a [POINTS](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
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

Regions (Windows GDI)

Article • 01/07/2021

A *region* is a rectangle, polygon, or ellipse (or a combination of two or more of these shapes) that can be filled, painted, inverted, framed, and used to perform hit testing (testing for the cursor location).

- [About Regions](#)
 - [Using Regions](#)
 - [Region Reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

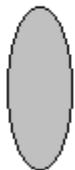
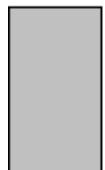
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

About Regions

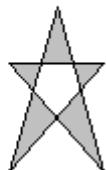
Article • 01/07/2021

Following are three types of regions that have been filled and framed.

Rectangular region Elliptical region Polygonal region



Polygonal region



- Region Creation and Selection
- Region Operations

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Region Creation and Selection

Article • 01/07/2021

An application creates a region by calling a function associated with a specific shape. The following table shows the function(s) associated with each of the standard shapes.

[+] Expand table

Shape	Function
Rectangular region	CreateRectRgn , CreateRectRgnIndirect , SetRectRgn
Rectangular region with rounded corners	CreateRoundRectRgn
Elliptical region	CreateEllipticRgn , CreateEllipticRgnIndirect
Polygonal region	CreatePolygonRgn , CreatePolyPolygonRgn

Each region creation function returns a handle that identifies the new region. An application can use this handle to select the region into a device context by calling the [SelectObject](#) function and supplying this handle as the second argument. After a region is selected into a device context, the application can perform various operations on it.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Region Operations

Article • 01/07/2021

Applications can combine regions, compare them, paint or invert their interiors, draw a frame around them, retrieve their dimensions, and test whether the cursor lies within their boundaries.

- [Combining Regions](#)
- [Comparing Regions](#)
- [Filling Regions](#)
- [Painting Regions](#)
- [Inverting Regions](#)
- [Framing Regions](#)
- [Retrieving a Bounding Rectangle](#)
- [Moving Regions](#)
- [Hit Testing Regions](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Combining Regions

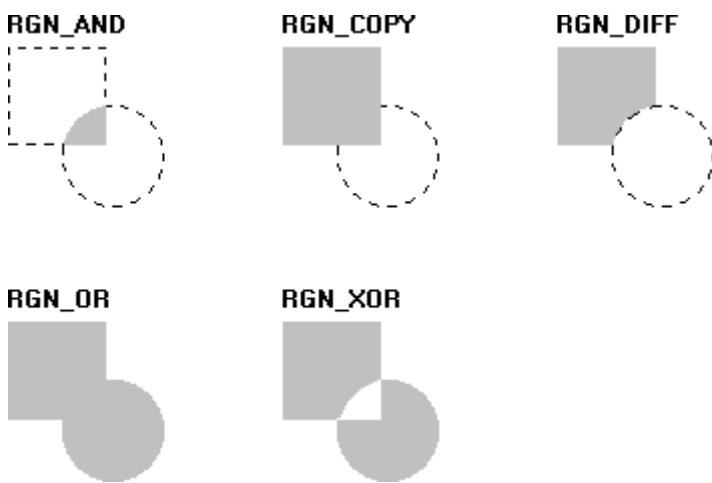
Article • 01/07/2021

An application combines two regions by calling the [CombineRgn](#) function. Using this function, an application can combine the intersecting parts of two regions, all but the intersecting parts of two regions, the two original regions in their entirety, and so on. Following are five values that define the region combinations.

[Expand table](#)

Value	Meaning
RGN_AND	The intersecting parts of two original regions define a new region.
RGN_COPY	A copy of the first (of the two original regions) defines a new region.
RGN_DIFF	The part of the first region that does not intersect the second defines a new region.
RGN_OR	The two original regions define a new region.
RGN_XOR	Those parts of the two original regions that do not overlap define a new region.

The following illustration shows the five possible combinations of a square and a circular region resulting from a call to [CombineRgn](#).



Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Comparing Regions

Article • 01/07/2021

An application compares two regions to determine whether they are identical by calling the [EqualRgn](#) function. EqualRgn considers two regions identical if they are equal in size and shape.

Feedback

Was this page helpful?

 Yes

 No

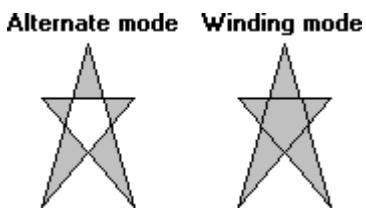
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Filling Regions

Article • 01/07/2021

An application fills the interior of a region by calling the [FillRgn](#) function and supplying a handle that identifies a specific brush. When an application calls `FillRgn`, the system fills the region with the brush by using the current fill mode for the specified device context. There are two fill modes: alternate and winding. The application can set the fill mode for a device context by calling the [SetPolyFillMode](#) function. The application can retrieve the current fill mode for a device context by calling the [GetPolyFillMode](#) function.

The following illustration shows two identical regions: one filled using alternate mode and the other filled using winding mode.



Alternate Mode

To determine which pixels the system highlights when alternate mode is specified, perform the following test:

1. Select a pixel within the region's interior.
2. Draw an imaginary ray, in the positive x-direction, from that pixel toward infinity.
3. Each time the ray intersects a boundary line, increment a count value.

The system highlights the pixel if the count value is an odd number.

Winding Mode

To determine which pixels the system highlights when winding mode is specified, perform the following test:

1. Determine the direction in which each boundary line is drawn.
2. Select a pixel within the region's interior.
3. Draw an imaginary ray, in the positive x-direction, from the pixel toward infinity.
4. Each time the ray intersects a boundary line with a positive y-component, increment a count value. Each time the ray intersects a boundary line with a negative y-component, decrement the count value.

The system highlights the pixel if the count value is nonzero.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Painting Regions

Article • 01/07/2021

An application fills the interior of a region by using the brush currently selected into a device context by the [PaintRgn](#) function. This function uses the current polygon fill modes (alternate and winding).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Inverting Regions

Article • 01/07/2021

An application inverts the colors that appear within a region by calling the [InvertRgn](#) function. On monochrome displays, **InvertRgn** 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.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Framing Regions

Article • 01/07/2021

An application draws a border around a region by calling the [FrameRgn](#) function and specifying the border width and brush pattern that the system uses when drawing the frame.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Retrieving a Bounding Rectangle

Article • 01/07/2021

An application retrieves the dimensions of a region's bounding rectangle by calling the **GetRgnBox** function. If the region is rectangular, **GetRgnBox** returns the dimensions of the region. If the region is elliptical, the function returns the dimensions of the smallest rectangle that can be drawn around the ellipse. The long sides of the rectangle are the same length as the ellipse's major axis, and the short sides of the rectangle are the same length as the ellipse's minor axis. If the region is polygonal, **GetRgnBox** returns the dimensions of the smallest rectangle that can be drawn around the entire polygon.

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Moving Regions

Article • 01/07/2021

An application moves a region by calling the [OffsetRgn](#) function. The given offsets along the x-axis and y-axis determine the number of logical units to move left or right and up or down.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Hit Testing Regions

Article • 01/07/2021

An application performs hit testing on regions to determine the coordinates of the current cursor position. Then it passes these coordinates as well as a handle identifying the region to the [PtInRegion](#) function. The cursor coordinates can be retrieved by processing the various mouse messages, such as [WM_LBUTTONDOWN](#), [WM_LBUTTONUP](#), [WM_RBUTTONDOWN](#), and [WM_RBUTTONUP](#). The return value for PtInRegion indicates whether the cursor position is within the given region.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Regions (Windows GDI)

Article • 01/07/2021

This topic has code examples for the following tasks.

- [Using regions to clip output](#)
- [Using regions to perform hit testing](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Regions to Clip Output

Article • 01/07/2021

This section contains a single example that demonstrates how you can use regions to enable the user to define how a part of client area output can appear. Regions used for this purpose are called clipping regions.

The example for this section is taken from an application that enables a user to capture the entire desktop as a bitmap and then isolate and save a part of this image as a .BMP file.

By clicking **Define Clip Region** from the application's menu, the user is able to select a clipping region by clicking the left mouse button and dragging the mouse. As the user drags the mouse, the application draws a rectangle that corresponds to the new clipping region.

By clicking **Clip** from the application's menu, the user is able to redraw the isolated part of the image within the boundaries of the specified rectangle.

This section provides information on the following topics.

- [Defining the Clipping Region](#)
- [Clipping Output](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Defining the Clipping Region

Article • 01/07/2021

When the user clicks Define Clip Region , the system issues a [WM_COMMAND](#) message. The *wParam* parameter of this message contains an application-defined constant, IDM_DEFINE, that indicates that the user selected this option from the menu. The application processes this input by setting a Boolean flag, fDefineRegion, as shown in the following code sample.

C++

```
case WM_COMMAND:  
    switch (wParam)  
    {  
  
        case IDM_DEFINE:  
            fDefineRegion = TRUE;  
            break;  
    }
```

After clicking **Define Clipping Region** , the user can begin drawing the rectangle by clicking and dragging the mouse while the cursor is in the application's client area.

When the user presses the left button, the system issues a [WM_LBUTTONDOWN](#) message. The *lParam* parameter of this message contains the cursor coordinates, which correspond to the upper left corner of a rectangle used to define the clipping region. The application processes the **WM_LBUTTONDOWN** message, as follows.

C++

```
// These variables are required for clipping.  
  
static POINT ptUpperLeft;  
static POINT ptLowerRight;  
static POINT aptRect[5];  
static POINT ptTmp;  
static POINTS ptsTmp;  
static BOOL fDefineRegion;  
static BOOL fRegionExists;  
static HRGN hrgn;  
static RECT rctTmp;  
int i;  
  
switch (message)  
{  
  
    case WM_LBUTTONDOWN:  
        if (fDefineRegion)
```

```
{  
  
    // Retrieve the new upper left corner.  
  
    ptsTmp = MAKEPOINTS(lParam);  
    ptUpperLeft.x = (LONG) ptsTmp.x;  
    ptUpperLeft.y = (LONG) ptsTmp.y;  
}  
  
if (fRegionExists)  
{  
  
    // Erase the previous rectangle.  
  
    hdc = GetDC(hwnd);  
    SetROP2(hdc, R2_NOTXORPEN);  
  
    if (!Polyline(hdc, (CONST POINT *) aptRect, 5))  
        errhandler("Polyline Failed", hwnd);  
    ReleaseDC(hwnd, hdc);  
  
    // Clear the rectangle coordinates.  
  
    for (i = 0; i < 4; i++)  
    {  
        aptRect[i].x = 0;  
        aptRect[i].y = 0;  
    }  
  
    // Clear the temporary point structure.  
  
    ptTmp.x = 0;  
    ptTmp.y = 0;  
  
    // Clear the lower right coordinates.  
  
    ptLowerRight.x = 0;  
    ptLowerRight.y = 0;  
  
    // Reset the flag.  
  
    fRegionExists = FALSE;  
    fDefineRegion = TRUE;  
  
    // Retrieve the new upper left corner.  
  
    ptsTmp = MAKEPOINTS(lParam);  
    ptUpperLeft.x = (LONG) ptsTmp.x;  
    ptUpperLeft.y = (LONG) ptsTmp.y;  
}  
break;  
}
```

As the user drags the mouse, the system issues **WM_MOUSEMOVE** messages and stores the new cursor coordinates in the *lParam* parameter. Each time the application receives a new **WM_MOUSEMOVE** message, it erases the previous rectangle (if one exists) and draws the new rectangle by calling the **Polyline** function, passing it the coordinates of the four corners of the rectangle. The application performs the following tasks.

C++

```
// These variables are required for clipping.

static POINT ptUpperLeft;
static POINT ptLowerRight;
static POINT aptRect[5];
static POINT ptTmp;
static POINTS ptsTmp;
static BOOL fDefineRegion;
static BOOL fRegionExists;
static HRGN hrgn;
static RECT rctTmp;
int i;

switch (message)
{
    case WM_MOUSEMOVE:

        if (wParam & MK_LBUTTON && fDefineRegion)
        {

            // Get a window DC.

            hdc = GetDC(hwnd);

            if (!SetROP2(hdc, R2_NOTXORPEN))
                errhandler("SetROP2 Failed", hwnd);

            // If previous mouse movement occurred, store the original
            // lower right corner coordinates in a temporary structure.

            if (ptLowerRight.x)
            {
                ptTmp.x = ptLowerRight.x;
                ptTmp.y = ptLowerRight.y;
            }

            // Get the new coordinates of the clipping region's lower
            // right corner.

            ptsTmp = MAKEPOINTS(lParam);
            ptLowerRight.x = (LONG) ptsTmp.x;
            ptLowerRight.y = (LONG) ptsTmp.y;

            // If previous mouse movement occurred, erase the original
```

```
// rectangle.

if (ptTmp.x)
{
    aptRect[0].x = ptUpperLeft.x;
    aptRect[0].y = ptUpperLeft.y;
    aptRect[1].x = ptTmp.x;
    aptRect[1].y = ptUpperLeft.y;
    aptRect[2].x = ptTmp.x;
    aptRect[2].y = ptTmp.y;
    aptRect[3].x = ptUpperLeft.x;
    aptRect[3].y = ptTmp.y;
    aptRect[4].x = aptRect[0].x;
    aptRect[4].y = aptRect[0].y;

    if (!Polyline(hdc, (CONST POINT *) aptRect, 5))
        errhandler("Polyline Failed", hwnd);
}

aptRect[0].x = ptUpperLeft.x;
aptRect[0].y = ptUpperLeft.y;
aptRect[1].x = ptLowerRight.x;
aptRect[1].y = ptUpperLeft.y;
aptRect[2].x = ptLowerRight.x;
aptRect[2].y = ptLowerRight.y;
aptRect[3].x = ptUpperLeft.x;
aptRect[3].y = ptLowerRight.y;
aptRect[4].x = aptRect[0].x;
aptRect[4].y = aptRect[0].y;

if (!Polyline(hdc, (CONST POINT *) aptRect, 5))
    errhandler("Polyline Failed", hwnd);

ReleaseDC(hwnd, hdc);
}
break;
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Clipping Output

Article • 01/07/2021

After the user clicks Clip from the menu, the application uses the coordinates of the rectangle the user created to define a clipping region. After defining the clipping region and selecting it into the application's device context, the application redraws the bitmapped image. The application performs these tasks, as follows.

C++

```
// These variables are required for clipping.

static POINT ptUpperLeft;
static POINT ptLowerRight;
static POINT aptRect[5];
static POINT ptTmp;
static POINTS ptsTmp;
static BOOL fDefineRegion;
static BOOL fRegionExists;
static HRGN hrgn;
static RECT rctTmp;
int i;

case WM_COMMAND:
    switch (wParam)
    {

        case IDM_CLIP:

            hdc = GetDC(hwnd);

            // Retrieve the application's client rectangle and paint
            // with the default (white) brush.

            GetClientRect(hwnd, &rctTmp);
            FillRect(hdc, &rctTmp, GetStockObject(WHITE_BRUSH));

            // Use the rect coordinates to define a clipping region.

            hrgn = CreateRectRgn(aptRect[0].x, aptRect[0].y,
                aptRect[2].x, aptRect[2].y);
            SelectClipRgn(hdc, hrgn);

            // Transfer (draw) the bitmap into the clipped rectangle.

            BitBlt(hdc,
                0, 0,
                bmp.bmWidth, bmp.bmHeight,
                hdcCompatible,
                0, 0,
                SRCCOPY);
```

```
ReleaseDC(hwnd, hdc);  
break;  
}
```

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Using Regions to Perform Hit Testing

Article • 01/07/2021

The example in [Brushes](#) uses regions to simulate a "zoomed" view of an 8- by 8-pixel monochrome bitmap. By clicking on the pixels in this bitmap, the user creates a custom brush suitable for drawing operations. The example shows how to use the [PtInRegion](#) function to perform hit testing and the [InvertRgn](#) function to invert the colors in a region.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Region Reference

Article • 01/07/2021

The following elements are used with regions.

- [Region Functions](#)
- [Region Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Region Functions (Windows GDI)

Article • 01/07/2021

The following functions are used with regions.

[+] Expand table

Function	Description
CombineRgn	Combines two regions and stores the result in a third region.
CreateEllipticRgn	Creates an elliptical region.
CreateEllipticRgnIndirect	Creates an elliptical region.
CreatePolygonRgn	Creates a polygonal region.
CreatePolyPolygonRgn	Creates a region consisting of a series of polygons.
CreateRectRgn	Creates a rectangular region.
CreateRectRgnIndirect	Creates a rectangular region.
CreateRoundRectRgn	Creates a rectangular region with rounded corners.
EqualRgn	Checks the two specified regions to determine whether they are identical.
ExtCreateRegion	Creates a region from the specified region and transformation data.
FillRgn	Fills a region by using the specified brush.
FrameRgn	Draws a border around the specified region by using the specified brush.
GetPolyFillMode	Retrieves the current polygon fill mode.
GetRegionData	Fills the specified buffer with data describing a region.
GetRgnBox	Retrieves the bounding rectangle of the specified region.
InvertRgn	Inverts the colors in the specified region.
OffsetRgn	Moves a region by the specified offsets.
PaintRgn	Paints the specified region by using the brush currently selected into the device context.
PtInRegion	Determines whether the specified point is inside the specified region.

Function	Description
RectInRegion	Determines whether any part of the specified rectangle is within the boundaries of a region.
SetPolyFillMode	Sets the polygon fill mode for functions that fill polygons.
SetRectRgn	Converts a region into a rectangular region with the specified coordinates.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

CombineRgn function (wingdi.h)

Article 02/22/2024

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

C++

```
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.

 Expand table

Value	Meaning
RGN_AND	Creates the intersection of the two combined regions.
RGN_COPY	Creates a copy of the region identified by <code>hrgnSrc1</code> .

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.

 Expand table

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

CreateEllipticRgn function (wingdi.h)

Article 02/22/2024

The **CreateEllipticRgn** function creates an elliptical region.

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

The `CreateEllipticRgnIndirect` function creates an elliptical region.

Syntax

C++

```
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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateEllipticRegion](#)

[DeleteObject](#)

[RECT](#)

[Region Functions](#)

[Regions Overview](#)

[SelectObject](#)

CreatePolygonRgn function (wingdi.h)

Article11/19/2022

The `CreatePolygonRgn` function creates a polygonal region.

Syntax

C++

```
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.

 Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 11/19/2022

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

Syntax

C++

```
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.

[] Expand table

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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article02/22/2024

The **CreateRectRgn** function creates a rectangular region.

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

The `CreateRectRgnIndirect` function creates a rectangular region.

Syntax

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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)

Article 02/22/2024

The **CreateRoundRectRgn** function creates a rectangular region with rounded corners.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

EqualRgn function (wingdi.h)

Article02/22/2024

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

C++

```
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

[] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[CreateRectRgn](#)

[CreateRectRgnIndirect](#)

[Region Functions](#)

[Regions Overview](#)

ExtCreateRegion function (wingdi.h)

Article10/13/2021

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

Syntax

C++

```
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 *lpRgnData*.

[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

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

FillRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

FrameRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetPolyFillMode function (wingdi.h)

Article02/22/2024

The **GetPolyFillMode** function retrieves the current polygon fill mode.

Syntax

C++

```
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.

 Expand table

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

GetRegionData function (wingdi.h)

Article01/29/2024

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

C++

```
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**, then the return value contains the number of bytes needed for the region data.

Return value

If the function succeeds and *nCount* specifies an adequate number of bytes, then the return value equals the actual number of bytes used (less than or equal to *nCount*).

If *lpRgnData* is **NULL**, then the return value is the required number of bytes.

If the function fails, then the return value is zero.

If the failure is due to *hrgn* being invalid, then **GetLastError** returns **ERROR_INVALID_HANDLE**. Otherwise, **GetLastError** returns **ERROR_INVALID_PARAMETER**.

If *nCount* is too small, then the function fails.

Remarks

The `GetRegionData` function is used in conjunction with the [ExtCreateRegion](#) function.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article 02/22/2024

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

Syntax

C++

```
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:

 Expand table

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

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

InvertRgn function (wingdi.h)

Article02/22/2024

The **InvertRgn** function inverts the colors in the specified region.

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

OffsetRgn function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

PaintRgn function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wingdi.h (include Windows.h)
Library	Gdi32.lib
DLL	Gdi32.dll

See also

[FillRgn](#)

[Region Functions](#)

[Regions Overview](#)

PtInRegion function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

RectInRegion function (wingdi.h)

Article02/22/2024

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

Syntax

C++

```
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

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported 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](#)

SetPolyFillMode function (wingdi.h)

Article 02/22/2024

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

Syntax

C++

```
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.

 Expand table

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

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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)

Article10/13/2021

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

Syntax

C++

```
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

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported 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](#)

Region Structures

Article • 01/07/2021

The following structures are used with regions:

- [RGNDATA](#)
- [RGNDATAHEADER](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

RGNDATA structure (wingdi.h)

Article02/22/2024

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

C++

```
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[1]

Specifies an arbitrary-size buffer that contains the [RECT](#) structures that make up the region.

Requirements

 Expand table

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No

RGNDATAHEADER structure (wingdi.h)

Article 02/22/2024

The **RGNDATAHEADER** structure describes the data returned by the [GetRegionData](#) function.

Syntax

C++

```
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

Requirement	Value
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](#)

Feedback

Was this page helpful?

 Yes

 No