2 GDI32.lib

2.1 AbortDoc

The **AbortDoc** function stops the current print job and erases everything drawn since the last call to the **StartDoc** function.

```
AbortDoc: procedure
(
     hdc:dword
);
     stdcall;
     returns( "eax" );
     external( "__imp__AbortDoc@4" );
```

Parameters

hdc

[in] Handle to the device context for the print job.

Return Values

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is SP_ERROR.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Remarks

Applications should call the **AbortDoc** function to stop a print job if an error occurs, or to stop a print job after the user cancels that job. To end a successful print job, an application should call the **EndDoc** function.

If Print Manager was used to start the print job, calling **AbortDoc** erases the entire spool job, so that the printer receives nothing. If Print Manager was not used to start the print job, the data may already have been sent to the printer. In this case, the printer driver resets the printer (when possible) and ends the print job.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.
Library: Use Gdi32.lib.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, EndDoc, SetAbort-

2.2 AbortPath

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

```
AbortPath: procedure
(
     hdc:dword
);
     stdcall;
     returns( "eax" );
     external( "__imp__AbortPath@4" );
```

Parameters

hdc

[in] Handle to the device context from which a path will be discarded.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Paths Overview, Path Functions, BeginPath, EndPath

2.3 AddFontMemResourceEx

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

```
AddFontMemResourceEx: procedure
(
    var pbFont: var;
```

```
        cbFont: dword;
    var pdv:    var;
    var pcFonts:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__AddFontMemResourceEx@16" );
```

Parameters

pbFont

[in] Pointer to a font resource.

cbFont

[in] Number of bytes in the font resource that is pointed to by *pbFont*.

pdv

[in] Reserved. Must be 0.

pcFonts

[in] Pointer to a variable that specifies the number of fonts installed.

Return Values

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.

Remarks

This function allows an application to get a font that is embedded in a document or a Web page. 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

Windows NT/2000: Requires Windows 2000 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf.
Library: Use Gdi32.lib.

See Also

Fonts and Text Overview, Font and Text Functions, RemoveFontMemResourceEx, SendMessage, DESIGNVECTOR

2.4 AddFontResource

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 Win32-based application.

To mark a font as private or no enumerable, use the **AddFontResourceEx** function.

```
AddFontResource: procedure
(
    lpszFilename: string
);
    stdcall;
    returns( "eax" );
    external( "__imp__AddFontResourceA@4" );
```

Parameters

lpszFilename

[in] Pointer to a null-terminated character string that contains a valid font file name. This parameter can specify any of the following files.

File extension	Description
.fon	Font resource file.
.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	**Windows 95/98 East Asian and Windows NT:** 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.

Windows 2000: 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 Values

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Fonts and Text Overview, Font and Text Functions, AddFontResourceEx, RemoveFontResource, SendMessage

---

2.5 AddFontResourceEx

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.

```
AddFontResourceEx: procedure
(
        lpszFilename:   string;
        fl:             dword;
    var pdv:            var
);
    stdcall;
    returns( "eax" );
    external( "__imp__AddFontResourceExA@12" );
```

Parameters

lpszFilename

[in] Pointer to a null-terminated character string that contains a valid font file file name. This parameter can specify any of the following files.

File extension	Description
.fon	Font resource file.

.fnt	Raw bitmap font file.
.ttf	Raw TrueType file.
.ttc	**Windows 95/98 East Asian and Windows NT:** True Type 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, abcxxxxx.pfm | abcxxxxx.pfb.

fl

[in] Specifies characteristics of the font to be added to the system. This parameter can be one of the following values.

Value	Meaning
FR_PRIVATE	Specifies that only the process that called the **AddFontResourceEx** function can use this font. When the font name matches a public font, the private font will be chosen. When the process terminates, the system will remove all fonts installed by the process with the **AddFontResourceEx** function.
FR_NOT_ENUM	Specifies that no process, including the process that called the **AddFontResourceEx** function, can enumerate this font.

pdv

[in] Reserved. It must be 0.

Return Values

If the function succeeds, the return value specifies the number of fonts added.

If the function fails, the return value is zero.

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.

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

Fonts and Text Overview, Font and Text Functions, RemoveFontResourceEx, SendMessage

---

2.6 AngleArc

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.

```
AngleArc: procedure
(
    hdc:        dword;
    x:          dword;
    y:          dword;
    dwRadius:   dword;
    eStartAngle:dword;
    eSweepAngle:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__AngleArc@24" );
```

Parameters

hdc

[in] Handle to a device context.

X

[in] Specifies the logical x-coordinate of the center of the circle.

Y

[in] Specifies the logical y-coordinate of the center of the circle.

dwRadius

[in] Specifies the radius, in logical units, of the circle. This value must be positive.

eStartAngle

[in] Specifies the start angle, in degrees, relative to the x-axis.

eSweepAngle

[in] Specifies the sweep angle, in degrees, relative to the starting angle.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Lines and Curves Overview, Line and Curve Functions, Arc, ArcTo, MoveToEx

---

2.7 AnimatePalette

The **AnimatePalette** function replaces entries in the specified logical palette.

```
AnimatePalette: procedure
(
        hpal:       dword;
        iStartIndex:dword;
        cEntries:   dword;
    var ppe:        PALETTEENTRY
);
    stdcall;
    returns( "eax" );
    external( "__imp__AnimatePalette@16" );
```

Parameters

hpal

[in] Handle to the logical palette.

iStartIndex

[in] Specifies the first logical palette entry to be replaced.

cEntries

[in] Specifies the number of entries to be replaced.

ppe

[in] Pointer to the first member in an array of **PALETTEENTRY** structures used to replace the current entries.

Return Values

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

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Colors Overview, Color Functions, CreatePalette, GetDeviceCaps, LOGPALETTE, PALETTEENTRY

2.8 Arc

The **Arc** function draws an elliptical arc.

```
Arc: procedure
(
    hdc:            dword;
    nLeftRect:      dword;
    nTopRect:       dword;
    nRightRect:     dword;
    nBottomRect:    dword;
    nXStartArc:     dword;
    nYStartArc:     dword;
    nXEndArc:       dword;
```

```
        nYEndArc:       dword
    );
        stdcall;
        returns( "eax" );
        external( "__imp__Arc@36" );
```

Parameters

hdc

[in] Handle to the device context where drawing takes place.

nLeftRect

[in] Specifies the logical x-coordinate of the upper-left corner of the bounding rectangle.

Windows 95/98: The sum of *nLeftRect* plus *nRightRect* must be less than 32768.

nTopRect

[in] Specifies the logical y-coordinate of the upper-left corner of the bounding rectangle.

Windows 95/98: The sum of *nTopRect* plus *nBottomRect* must be less than 32768.

nRightRect

[in] Specifies the logical x-coordinate of the lower-right corner of the bounding rectangle.

Windows 95/98: The sum of *nLeftRect* plus *nRightRect* must be less than 32768.

nBottomRect

[in] Specifies the logical y-coordinate of the lower-right corner of the bounding rectangle.

Windows 95/98: The sum of *nTopRect* plus *nBottomRect* must be less than 32768.

nXStartArc

[in] Specifies the logical x-coordinate of the ending point of the radial line defining the starting point of the arc.

nYStartArc

[in] Specifies the logical y-coordinate of the ending point of the radial line defining the starting point of the arc.

nXEndArc

[in] Specifies the logical x-coordinate of the ending point of the radial line defining the ending point of the arc.

nYEndArc

[in] Specifies the logical y-coordinate of the ending point of the radial line defining the ending point of the arc.

Return Values

If the arc is drawn, the return value is nonzero.

If the arc is not drawn, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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**.

Windows 95/98: The drawing direction is always counterclockwise.

Windows NT/2000: Use the **GetArcDirection** and **SetArcDirection** functions to get and set the current drawing direction for a device context. The default drawing direction is counterclockwise.

Windows 95/98: The sum of the coordinates of the bounding rectangle cannot exceed 32,767. The sum of *nLeftRect* and *nRightRect* or *nTopRect* and *nBottomRect* parameters cannot exceed 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Lines and Curves Overview, Line and Curve Functions, AngleArc, ArcTo, Chord, Ellipse, GetArcDirection, Pie, SetArcDirection

---

2.9 ArcTo

The **ArcTo** function draws an elliptical arc.

```
ArcTo: procedure
(
    hdc:            dword;
    nLeftRect:      dword;
    nTopRect:       dword;
    nRightRect:     dword;
    nBottomRect:    dword;
    nXRadial1:      dword;
    nYRadial1:      dword;
    nXRadial2:      dword;
    nYRadial2:      dword
);
    stdcall;
    returns( "eax" );
```

```
external( "__imp__ArcTo@36" );
```

Parameters

hdc

[in] Handle to the device context where drawing takes place.

nLeftRect

[in] Specifies the logical x-coordinate of the upper-left corner of the bounding rectangle.

nTopRect

[in] Specifies the logical y-coordinate of the upper-left corner of the bounding rectangle.

nRightRect

[in] Specifies the logical x-coordinate of the lower-right corner of the bounding rectangle.

nBottomRect

[in] Specifies the logical y-coordinate of the lower-right corner of the bounding rectangle.

nXRadial1

[in] Specifies the logical x-coordinate of the endpoint of the radial defining the starting point of the arc.

nYRadial1

[in] Specifies the logical y-coordinate of the endpoint of the radial defining the starting point of the arc.

nXRadial2

[in] Specifies the logical x-coordinate of the endpoint of the radial defining the ending point of the arc.

nYRadial2

[in] Specifies the logical y-coordinate of the endpoint of the radial defining the ending point of the arc.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **`GetLastError`**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Lines and Curves Overview, Line and Curve Functions, AngleArc, Arc, SetArcDirection

---

2.10 BeginPath

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

```
BeginPath: procedure
(
    hdc:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__BeginPath@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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 table shows which drawing functions can be used on the different Windows operating systems.

Drawing function	Operating system

AngleArc	Windows NT/2000
Arc	Windows NT/2000
ArcTo	Windows NT/2000
Chord	Windows NT/2000
CloseFigure	Windows 95/98 and Windows NT/2000
Ellipse	Windows NT/2000
ExtTextOut	Windows 95/98 and Windows NT/2000
LineTo	Windows 95/98 and Windows NT/2000
MoveToEx	Windows 95/98 and Windows NT/2000
Pie	Windows NT/2000
PolyBezier	Windows 95/98 and Windows NT/2000
PolyBezierTo	Windows 95/98 and Windows NT/2000
PolyDraw	Windows NT/2000
Polygon	Windows 95/98 and Windows NT/2000
Polyline	Windows 95/98 and Windows NT/2000
PolylineTo	Windows 95/98 and Windows NT/2000
PolyPolygon	Windows 95/98 and Windows NT/2000
PolyPolyline	Windows 95/98 and Windows NT/2000
Rectangle	Windows NT/2000
RoundRect	Windows NT/2000
TextOut	Windows 95/98 and Windows NT/2000

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Paths Overview, Path Functions, EndPath, FillPath, PathToRegion, SelectClipPath, StrokeAndFillPath, StrokePath, WidenPath

2.11 BitBlt

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

```
BitBlt: procedure
(
    hdcDest     :dword;
    nXDest      :dword;
    nYDest      :dword;
    nWidth      :dword;
    nHeight     :dword;
    hdcSrc      :dword;
    nXSrc       :dword;
    nYSrc       :dword;
    dwRop       :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__BitBlt@36" );
```

Parameters

hdcDest

[in] Handle to the destination device context.

nXDest

[in] Specifies the logical x-coordinate of the upper-left corner of the destination rectangle.

nYDest

[in] Specifies the logical y-coordinate of the upper-left corner of the destination rectangle.

nWidth

[in] Specifies the logical width of the source and destination rectangles.

nHeight

[in] Specifies the logical height of the source and the destination rectangles.

hdcSrc

[in] Handle to the source device context.

nXSrc

[in] Specifies the logical x-coordinate of the upper-left corner of the source rectangle.

nYSrc

[in] Specifies the logical y-coordinate of the upper-left corner of the source rectangle.

dwRop

[in] Specifies a raster-operation code. These codes define how the color data for the source rectangle is to be combined with the color data for the destination rectangle to achieve the final color.

The following list shows some common raster operation codes.

Value	Description
BLACKNESS	Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.)
CAPTUREBLT	**Windows 98, Windows 2000:** Includes any windows that are layered on top of your window in the resulting image. By default, the image only contains your window.
DSTINVERT	Inverts the destination rectangle.
MERGECOPY	Merges the colors of the source rectangle with the brush currently selected in *hdcDest*, by using the Boolean AND operator.
MERGEPAINT	Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator.
NOMIRRORBITMAP	**Windows 98, Windows 2000:** Prevents the bitmap from being mirrored.
NOTSRCCOPY	Copies the inverted source rectangle to the destination.
NOTSRCERASE	Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color.
PATCOPY	Copies the brush currently selected in *hdcDest*, into the destination bitmap.
PATINVERT	Combines the colors of the brush currently selected in *hdcDest*, with the colors of the destination rectangle by using the Boolean XOR operator.
PATPAINT	Combines the colors of the brush currently selected in *hdcDest*, with the colors of the inverted source rectangle by using the Boolean OR operator. The result of this operation is combined with the colors of the destination rectangle by using the Boolean OR operator.
SRCAND	Combines the colors of the source and destination rectangles by using the Boolean AND operator.
SRCCOPY	Copies the source rectangle directly to the destination rectangle.

SRCERASE	Combines the inverted colors of the destination rectangle with the colors of the source rectangle by using the Boolean AND operator.
SRCINVERT	Combines the colors of the source and destination rectangles by using the Boolean XOR operator.
SRCPAINT	Combines the colors of the source and destination rectangles by using the Boolean OR operator.
WHITENESS	Fills the destination rectangle using the color associated with index 1 in the physical palette. (This color is white for the default physical palette.)

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call `GetLastError`.

Remarks

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.

ICM: No color management is performed when blits occur.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Bitmaps Overview, Bitmap Functions

2.12 CancelDC

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

```
CancelDC: procedure
(
    hdc:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CancelDC@4" );
```

Parameters

hdc

[in] Handle to the DC.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call `GetLastError`.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Device Contexts Overview, Device Context Functions, CreateThread, GetCurrentThread

2.13 CheckColorsInGamut

The **CheckColorsInGamut** function determines whether a specified set of RGB triples lies in the output gamut of a specified device. The RGB triples are interpreted in the input logical color space.

```
CheckColorsInGamut: procedure
(
```

```
        hdc             :dword;
    var lpRGBTriples    :var;
    var lpBuffer        :var;
        nCount          :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CheckColorsInGamut@16" );
```

hDC

Handle to the device context whose output gamut to be checked.

lpRGBTriples

Pointer to an array of RGB triples to check.

lpBuffer

Pointer to the buffer in which the results are to be placed. This buffer must be at least as large as *nCount* bytes.

nCount

The number of elements in the array of triples.

Return Values

If this function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero.

Remarks

The function places the test results in the buffer pointed to by *lpBuffer*. Each byte in the buffer corresponds to an *RGB triple*, and has an unsigned value between CM_IN_GAMUT (= 0) and CM_OUT_OF_GAMUT (= 255). The value 0 denotes that the color is in gamut, while a nonzero value denotes that it is out of gamut. For any integer *n* such that $0 < n < 255$, a result value of $n + 1$ indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of *n*, as specified by the ICC Profile Format Specification. For more information on the ICC Profile Format Specification, see the sources listed in Further Information.

Note that for this function to succeed, ICM must be enabled for the device context handle that is passed in through the *hDC* parameter. ICM can be enabled for a device context handle by calling the **SetICMMode** function.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Import Library: Use gdi32.lib.

See Also

Basic Color Management Concepts, Functions, SetICMMode

2.14 ChoosePixelFormat

The **ChoosePixelFormat** function attempts to match an appropriate pixel format supported by a device context to a given pixel format specification.

```
ChoosePixelFormat: procedure
(
        hdc     :dword;
    var ppfd    :PIXELFORMATDESCRIPTOR
);
    stdcall;
    returns( "eax" );
    external( "__imp__ChoosePixelFormat@8" );
```

Parameters

hdc

Specifies the device context that the function examines to determine the best match for the pixel format descriptor pointed to by *ppfd*.

ppfd

Pointer to a **PIXELFORMATDESCRIPTOR** structure that specifies the requested pixel format. In this context, the members of the **PIXELFORMATDESCRIPTOR** structure that *ppfd* points to are used as follows:

nSize

Specifies the size of the **PIXELFORMATDESCRIPTOR** data structure. Set this member to **sizeof(PIXELFORMATDESCRIPTOR)**.

nVersion

Specifies the version number of the **PIXELFORMATDESCRIPTOR** data structure. Set this member to 1.

dwFlags

A set of bit flags that specify properties of the pixel buffer. You can combine the following bit flag constants by using bitwise-OR.

If any of the following flags are set, the **ChoosePixelFormat** function attempts to match pixel formats that also have that flag or flags set. Otherwise, **ChoosePixelFormat** ignores that flag in the pixel formats:

PFD_DRAW_TO_WINDOW
PFD_DRAW_TO_BITMAP
PFD_SUPPORT_GDI
PFD_SUPPORT_OPENGL

If any of the following flags are set, **ChoosePixelFormat** attempts to match pixel formats that also have that flag or flags set. Otherwise, it attempts to match pixel formats without that flag set:

PFD_DOUBLEBUFFER

PFD_STEREO

If the following flag is set, the function ignores the PFD_DOUBLEBUFFER flag in the pixel formats:

PFD_DOUBLEBUFFER_DONTCARE

If the following flag is set, the function ignores the PFD_STEREO flag in the pixel formats:

PFD_STEREO_DONTCARE

iPixelType

Specifies the type of pixel format for the function to consider:

PFD_TYPE_RGBA
PFD_TYPE_COLORINDEX

cColorBits

Zero or greater.

cRedBits

Not used.

cRedShift

Not used.

cGreenBits

Not used.

cGreenShift

Not used.

cBlueBits

Not used.

cBlueShift

Not used.

cAlphaBits

Zero or greater.

cAlphaShift

Not used.

cAccumBits

Zero or greater.

cAccumRedBits

Not used.

cAccumGreenBits

Not used.

cAccumBlueBits

Not used.

cAccumAlphaBits

Not used.

cDepthBits

Zero or greater.

cStencilBits

Zero or greater.

cAuxBuffers

Zero or greater.

iLayerType

Specifies one of the following layer type values:

PFD_MAIN_PLANE
PFD_OVERLAY_PLANE
PFD_UNDERLAY_PLANE

bReserved

Not used.

dwLayerMask

Not used.

dwVisibleMask

Not used.

dwDamageMask

Not used.

Return Values

If the function succeeds, the return value is a pixel format index (one-based) that is the closest match to the given pixel format descriptor.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

You must ensure that the pixel format matched by the **ChoosePixelFormat** function satisfies your requirements. For example, if you request a pixel format with a 24-bit RGB color buffer but the device context offers only 8-bit RGB color buffers, the function returns a pixel format with an 8-bit RGB color buffer.

The following code sample shows how to use **ChoosePixelFormat** to match a specified pixel format:

```
PIXELFORMATDESCRIPTOR pfd = {
    sizeof(PIXELFORMATDESCRIPTOR),  //  size of this pfd
```

```
    1,                      // version number
    PFD_DRAW_TO_WINDOW |    // support window
    PFD_SUPPORT_OPENGL |    // support OpenGL
    PFD_DOUBLEBUFFER,       // double buffered
    PFD_TYPE_RGBA,          // RGBA type
    24,                     // 24-bit color depth
    0, 0, 0, 0, 0, 0,       // color bits ignored
    0,                      // no alpha buffer
    0,                      // shift bit ignored
    0,                      // no accumulation buffer
    0, 0, 0, 0,             // accum bits ignored
    32,                     // 32-bit z-buffer
    0,                      // no stencil buffer
    0,                      // no auxiliary buffer
    PFD_MAIN_PLANE,         // main layer
    0,                      // reserved
    0, 0, 0                 // layer masks ignored
    };
    HDC  hdc;
    int  iPixelFormat;

iPixelFormat = ChoosePixelFormat(hdc, &pfd);
```

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in gdi32.hhf
Import Library: Use gdi32.lib.

See Also

OpenGL on Windows NT, Windows 2000, and Windows 95/98, Win32 Functions, DescribePixelFormat, GetPixelFormat, SetPixelFormat

---

2.15 Chord

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

```
    Chord: procedure
    (
        hdc         :dword;
        nLeftRect   :dword;
        nTopRect    :dword;
        nRightRect  :dword;
        nBottomRect :dword;
        nXRadial1   :dword;
        nYRadial1   :dword;
        nXRadial2   :dword;
        nYRadial2   :dword
    );
        stdcall;
        returns( "eax" );
        external( "__imp__Chord@36" );
```

Parameters

hdc

[in] Handle to the device context in which the chord appears.

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the bounding rectangle.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the bounding rectangle.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the bounding rectangle.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the bounding rectangle.

nXRadial1

[in] Specifies the x-coordinate of the endpoint of the radial defining the beginning of the chord.

nYRadial1

[in] Specifies the y-coordinate of the endpoint of the radial defining the beginning of the chord.

nXRadial2

[in] Specifies the x-coordinate of the endpoint of the radial defining the end of the chord.

nYRadial2

[in] Specifies the y-coordinate of the endpoint of the radial defining the end of the chord.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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**.

Windows 95/98: The sum of the coordinates of the bounding rectangle cannot exceed 32,767. The sum of *nLeftRect* and *nRightRect* or *nTopRect* and *nBottomRect* parameters cannot exceed

32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Filled Shapes Overview, Filled Shape Functions, AngleArc, Arc, ArcTo, Pie

---

2.16 CloseEnhMetaFile

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

```
CloseEnhMetaFile: procedure
(
    hdc:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CloseEnhMetaFile@4" );
```

Parameters

hdc

[in] Handle to an enhanced-metafile device context.

Return Values

If the function succeeds, the return value is a handle to an enhanced metafile.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Metafiles Overview, Metafile Functions, CopyEnhMetaFile, CreateEnhMetaFile, DeleteEnhMetaFile, EnumEnhMetaFile, GetEnhMetaFileBits, GetWinMetaFileBits, PlayEnhMetaFile

---

2.17 CloseFigure

The **CloseFigure** function closes an open figure in a path.

```
CloseFigure: procedure
(
    hdc:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CloseFigure@4" );
```

Parameters

hdc

[in] Handle to the device context in which the figure will be closed.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Paths Overview, Path Functions, BeginPath, EndPath, ExtCreatePen, LineTo, MoveToEx

---

2.18 CloseMetaFile

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 16-bit versions of Windows. Win32-based applications should use the **CloseEnhMetaFile** function.

```
CloseMetaFile: procedure
(
    hdc:dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CloseMetaFile@4" );
```

Parameters

hdc

[in] Handle to a metafile device context used to create a Windows-format metafile.

Return Values

If the function succeeds, the return value is a handle to a Windows-format metafile.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

A Windows-format metafile does not support the new curve, path, and transformation functions, such as **PolyBezier**, **BeginPath**, and **SetWorldTransform**. Applications that use these new functions *and* use metafiles to store pictures created by these functions should call the enhanced-format metafile functions.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.
Library: Use Gdi32.lib.

See Also

Metafiles Overview, Metafile Functions, BeginPath, CloseEnhMetaFile, CopyMetaFile, CreateMetaFile, DeleteMetaFile, EnumMetaFile, GetMetaFileBitsEx, PlayMetaFile, PolyBezier, SetWinMetaFileBits, SetWorldTransform

---

2.19 ColorCorrectPalette

The **ColorCorrectPalette** function corrects the entries of a palette using the ICM 2.0 parameters in the specified device context.

```
ColorCorrectPalette: procedure
(
    hdc             :dword;
    hPalette        :dword;
    dwFirstEntry    :dword;
    dwNumOfEntries  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ColorCorrectPalette@16" );
```

hDC

Specifies a device context whose ICM parameters to use.

hPalette

Specifies the handle to the palette to be color corrected.

dwFirstEntry

Specifies the first entry in the palette to be color corrected.

dwNumOfEntries

Specifies the number of entries to color correct.

Return Values

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Header: Declared in gdi32.hhf

Import Library: Use gdi32.lib.

See Also

Basic Color Management Concepts, Functions

2.20 ColorMatchToTarget

The **ColorMatchToTarget** function enables you to preview colors as they would appear on the target device.

```
ColorMatchToTarget: procedure
(
    hdc         :dword;
    hdcTarget   :dword;
    uiAction    :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ColorMatchToTarget@12" );
```

hDC

Specifies the device context for previewing, generally the screen.

hdcTarget

Specifies the target device context, generally a printer.

uiAction

A constant that can have one of the following values.

Constant	Meaning
CS_ENABLE	Map the 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	Disable color proofing.
CS_DELETE_TRANSFORM	If color management is enabled for the target profile, disable it and delete the concatenated transform.

Return Values

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

Remarks

ColorMatchToTarget can be used to proof the colors of a color output device on another color output device. Setting the *uiAction* parameter to CS_ENABLE causes all subsequent drawing

commands to the DC to render colors as they would appear on the target device. If *uiAction* is set to CS_DISABLE, proofing is turned off. However, the current color transform is not deleted from the DC. It is just inactive.

When **ColorMatchToTarget** is called, the color transform for the target device is performed first, and then the transform to the preview device is applied to the results of the first transform. This is used primarily for checking gamut mapping conditions. Before using this function, you must enable ICM for both device contexts.

This function cannot be cascaded. While color mapping to the target is enabled by setting *uiAction* to CS_ENABLE, application changes to the color space or gamut mapping method are ignored. Those changes then take effect when color mapping to the target is disabled.

Note A memory leak will not occur if an application does not delete a transform using CS_DELETE_TRANSFORM. The transform will be deleted when either the device context (DC) is closed, or when the application color space is deleted. However if the transform is not going to be used again, or if the application will not be performing any more color matching on the DC, it should explicitly delete the transform to free the memory it occupies.

The *uiAction* parameter should only be set to CS_DELETE_TRANSFORM if color management is enabled before the **ColorMatchToTarget** function is called.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Header: Declared in gdi32.hhf
Import Library: Use gdi32.lib.

See Also

Basic Color Management Concepts, Functions

2.21 CombineRgn

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

```
CombineRgn: procedure
(
    hrgnDest        :dword;
    hrgnSrc1        :dword;
    hrgnSrc2        :dword;
    fnCombineMode   :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CombineRgn@16" );
```

Parameters

hrgnDest

[in] Handle to a new region with dimensions defined by combining two other regions. (This region must exist before **CombineRgn** is called.)

hrgnSrc1

[in] Handle to the first of two regions to be combined.

hrgnSrc2

[in] Handle to the second of two regions to be combined.

fnCombineMode

[in] Specifies a mode indicating how the two regions will be combined. This parameter can be one of the following values.

Value	Description
RGN_AND	Creates the intersection of the two combined regions.
RGN_COPY	Creates a copy of the region identified by *hrgnSrc1*.
RGN_DIFF	Combines the parts of *hrgnSrc1* that are not part of *hrgnSrc2*.
RGN_OR	Creates the union of two combined regions.
RGN_XOR	Creates the union of two combined regions *except* for any overlapping areas.

Return Values

The return value specifies the type of the resulting region. It can be one of the following values.

Value	Meaning
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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf
Library: Use Gdi32.lib.

See Also

Regions Overview, Region Functions, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn

---

2.22 CombineTransform

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

```
CombineTransform: procedure
(
        lpxformResult   :dword;
    var lpxform1        :XFORM;
    var lpxform2        :XFORM
);
    stdcall;
    returns( "eax" );
    external( "__imp__CombineTransform@12" );
```

Parameters

lpxformResult

[out] Pointer to an **XFORM** structure that receives the combined transformation.

lpxform1

[in] Pointer to an **XFORM** structure that specifies the first transformation.

lpxform2

[in] Pointer to an **XFORM** structure that specifies the second transformation.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf.
Library: Use Gdi32.lib.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWorldTransform, ModifyWorldTransform, SetWorldTransform, XFORM

2.23 CopyEnhMetaFile

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

```
CopyEnhMetaFile: procedure
(
    hemfSrc     :dword;
    lpszFile    :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__CopyEnhMetaFileA@8" );
```

Parameters

hemfSrc

[in] Handle to the enhanced metafile to be copied.

lpszFile

[in] Pointer to the name of the destination file. If this parameter is NULL, the source metafile is copied to memory.

Return Values

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.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile

2.24 CopyMetaFile

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

Note This function is provided only for compatibility with 16-bit versions of Windows. Win32-based applications should use the **CopyEnhMetaFile** function.

```
CopyMetaFile: procedure
(
    hmfSrc      :dword;
    lpszFile    :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__CopyMetaFileA@8" );
```

Parameters

hmfSrc

[in] Handle to the source Windows-format metafile.

lpszFile

[in] Pointer to the name of the destination file. If this parameter is NULL, the source metafile is copied to memory.

Return Values

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.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

The **CopyMetaFile** function supports only 16-bit Windows-based applications. It does not record or play back the new graphics device interface functions, such as **PolyBezier**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, DeleteMetaFile

2.25 CreateBitmap

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

```
CreateBitmap: procedure
(
        nWidth      :dword;
        nHeight     :dword;
        cPlanes     :dword;
        cBitsPerPel :dword;
    var lpvBits     :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateBitmap@20" );
```

Parameters

nWidth

[in] Specifies the bitmap width, in pixels.

nHeight

[in] Specifies the bitmap height, in pixels.

cPlanes

[in] Specifies the number of color planes used by the device.

cBitsPerPel

[in] Specifies the number of bits required to identify the color of a single pixel.

lpvBits

[in] Pointer to an array of color data used to set the colors in a rectangle of pixels. Each scan line in the rectangle must be word aligned (scan lines that are not word aligned must be padded with zeros). If this parameter is NULL, the contents of the new bitmap is undefined.

Return Values

If the function succeeds, the return value is a handle to a bitmap.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**. This can have the following value.

Value	Meaning

ERROR_INVALID_BITMAP	The calculated size of the bitmap is less than zero.

Remarks

After a bitmap is created, it can be selected into a device context by calling the **SelectObject** function.

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. When a color bitmap returned from **CreateBitmap** 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. Since **CreateCompatibleBitmap** takes a device context, it returns a bitmap that has the same format as the specified device context. Because of this, subsequent calls to **SelectObject** are faster 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.

Windows 95/98: The created bitmap cannot exceed 16MB in size.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, CreateBitmapIndirect, CreateCompatibleBitmap, CreateDIBitmap, DeleteObject, GetBitmapBits, SelectObject, SetBitmapBits

---

2.26 CreateBitmapIndirect

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

```
CreateBitmapIndirect: procedure
(
    var lpbm    :BITMAP
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateBitmapIndirect@4" );
```

Parameters

lpbm

[in] 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 Values

If the function succeeds, the return value is a handle to the bitmap.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**. This can have the following values.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more of the input parameters was invalid.
ERROR_NOT_ENOUGH_MEMORY	The bitmap is too big for memory to be allocated.

Remarks

After a bitmap is created, it can be selected into a device context by calling the **SelectObject** function.

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. When a color bitmap returned 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. Since **CreateCompatibleBitmap** takes a device context, it returns a bitmap that has the same format as the specified device context. Because of this, subsequent calls to **SelectObject** are faster 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.

Windows 95/98: The created bitmap cannot exceed 16MB in size.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BitBlt, BITMAP, CreateBitmap, CreateCompatibleBitmap, CreateDIBitmap, DeleteObject, SelectObject

2.27 CreateBrushIndirect

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

```
CreateBrushIndirect: procedure
(
    var lplb    :LOGBRUSH
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateBrushIndirect@4" );
```

Parameters

lplb

[in] Pointer to a **LOGBRUSH** structure that contains information about the brush.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Windows 95/98: Creating brushes from bitmaps or DIBs larger than 8 by 8 pixels is not supported. If a larger bitmap is specified, only a portion of the bitmap is used.

Windows NT/ 2000: Brushes can be created from bitmaps or DIBs larger than 8 by 8 pixels.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, DeleteObject, GetBrushOrgEx, LOGBRUSH, SelectObject, SetBrushOrgEx

2.28 CreateColorSpace

The **CreateColorSpace** function creates a logical color space.

```
CreateColorSpace: procedure
(
    var lpLogColorSpace :LOGCOLORSPACE
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateColorSpaceA@4" );
```

lpLogColorSpace

Pointer to the **LOGCOLORSPACE** data structure.

Return Values

If this function succeeds, the return value is a handle that identifies a color space.

If this function fails, the return value is NULL.

Remarks

When the color space is no longer needed, use **DeleteColorSpace** to delete it.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions, DeleteColorSpace

2.29 CreateCompatibleBitmap

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

```
CreateCompatibleBitmap: procedure
(
    hdc     :dword;
    nWidth  :dword;
    nHeight :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateCompatibleBitmap@12" );
```

Parameters

hdc

[in] Handle to a device context.

nWidth

[in] Specifies the bitmap width, in pixels.

nHeight

[in] Specifies the bitmap height, in pixels.

Return Values

If the function succeeds, the return value is a handle to the bitmap.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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 non-memory device context always possesses the same color format and uses the same color palette as the specified device context.

Note: When a memory device context is created, it initially has a 1-by-1 monochrome bitmap selected into it. If this memory device context is used in **CreateCompatibleBitmap**, the bitmap that is created is a *monochrome* bitmap. To create a color bitmap, use the *hDC* that was used to create the memory device context, as shown in the following code:

```
HDC memDC = CreateCompatibleDC ( hDC );
HBITMAP memBM = CreateCompatibleBitmap ( hDC );
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.

Windows 95/98: The created bitmap cannot exceed 16MB in size.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, CreateDIBSection, DeleteObject, SelectObject

2.30 CreateCompatibleDC

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

```
CreateCompatibleDC: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateCompatibleDC@4" );
```

Parameters

hdc

[in] Handle to an existing DC. If this handle is NULL, the function creates a memory DC compatible with the application's current screen.

Return Values

If the function succeeds, the return value is the handle to a memory DC.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

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.

ICM: If the DC that is passed to this function is enabled for Independent Color Management (ICM), the DC created by the function is ICM-enabled. The source and destination color spaces are specified in the DC.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CreateCompatibleBitmap, DeleteDC, GetDeviceCaps

2.31 CreateDC

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

```
CreateDC: procedure
(
        lpszDriver  :string;
        lpszDevice  :string;
        lpszOutput  :string;
    var lpInitData  :DEVMODE
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDCA@16" );
```

Parameters

lpszDriver

Windows 95/98: In Win32-based applications, *lpszDriver* can be NULL, WINSPL16 (a print provider), or (to obtain a display DC) it can be either the null-terminated string DISPLAY or the device name of a specific display device. If *lpszDevice* specifies a particular device, you must use NULL for *lpszDriver*.

Windows NT 4.0: Pointer to a null-terminated character string that specifies either DISPLAY or the name of a print provider, which is usually WINSPOOL.

Windows NT/2000: Pointer to a null-terminated character string that specifies either DISPLAY or the name of a specific display device or the name of a print provider, which is usually WINSPOOL.

lpszDevice

[in] 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.

Windows NT 3.51 and 4.0: There is only one (thus the primary) display device. Set *lpszDevice* to NULL.

lpszOutput

This parameter is ignored for Win32-based applications, and should be set to NULL. It is provided only for compatibility with 16-bit Windows. For more information, see the Remarks

section.

lpInitData

[in] 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 *lpInitData* parameter must be NULL if the device driver is to use the default initialization (if any) specified by the user.

If *lpszDriver* is DISPLAY, then *lpInitData* must be NULL. The display device's current **DEVMODE** is used.

Return Values

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. The function will return NULL for a **DEVMODE** structure other than the current **DEVMODE**.

Windows NT/2000: To get extended error information, call **GetLastError**.

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.

Applications written for 16-bit versions of Windows used the *lpszOutput* parameter to specify a port name or to print to a file. Win32-based applications do not need to specify a port name. Win32-based applications can print to a file by calling the **StartDoc** function with a DOCINFO structure whose **lpszOutput** member specifies the path of the output file name.

When you no longer need the DC, call the **DeleteDC** function.

ICM: To enable ICM, set the **dmICMMethod** member of the **DEVMODE** structure (pointed to by the *pInitData* parameter) to the appropriate value.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, Multiple Display Monitors, DeleteDC, DEVMODE, EnumDisplayDevices, DOCINFO, DocumentProperties, StartDoc

2.32 CreateDIBPatternBrush

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. Win32-based applications should use the **CreateDIBPatternBrushPt** function.

```
CreateDIBPatternBrush: procedure
(
    hglbDIBPacked   :dword;
    fuColorSpec     :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDIBPatternBrush@8" );
```

Parameters

hglbDIBPacked

[in] 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.

Windows 95/98: Creating brushes from bitmaps or DIBs larger than 8 by 8 pixels is not supported. If a larger bitmap is specified, only a portion of the bitmap is used.

Windows NT/ 2000: Brushes can be created from bitmaps or DIBs larger than 8 by 8 pixels.

fuColorSpec

[in] Specifies whether the **bmiColors** member of the **BITMAPINFO** structure is initialized and, if so, whether this member contains explicit red, green, blue (RGB) values or indexes into a logical palette. The *fuColorSpec* parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the brush is to be selected.
DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, BITMAPINFO, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, DeleteObject, SetBkColor, SetTextColor

---

2.33 CreateDIBPatternBrushPt

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

```
CreateDIBPatternBrushPt: procedure
(
    var lpPackedDIB :var;
        iUsage      :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDIBPatternBrushPt@8" );
```

Parameters

lpPackedDIB

[in] Pointer to a packed DIB consisting of a **BITMAPINFO** structure immediately followed by an array of bytes defining the pixels of the bitmap.

Windows 95/98: Creating brushes from bitmaps or DIBs larger than 8 by 8 pixels is not supported. If a larger bitmap is specified, only a portion of the bitmap is used.

Windows NT/ 2000: Brushes can be created from bitmaps or DIBs larger than 8 by 8 pixels.

iUsage

[in] Specifies whether the **bmiColors** member of the **BITMAPINFO** structure contains a valid color table and, if so, whether the entries in this color table contain explicit red, green, blue (RGB) values or palette indexes. The *iUsage* parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the brush is to be selected.

DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, BITMAPINFO, CreateDIBPatternBrush, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, DeleteObject, GetBrushOrgEx, SelectObject, SetBrushOrgEx

2.34 CreateDIBSection

The **CreateDIBSection** function creates a DIB that applications can write to directly. The function gives you a pointer to the location of the bitmap's 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.

```
CreateDIBSection: procedure
(
        hdc         :dword;
    var pbmi        :BITMAPINFO;
        iUsage      :dword;
    var ppvBits     :var;
        hSection    :dword;
        dwOffset    :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDIBSection@24" );
```

Parameters

hdc

[in] 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's colors.

pbmi

[in] Pointer to a **BITMAPINFO** structure that specifies various attributes of the DIB, including the bitmap's dimensions and colors.

iUsage

[in] Specifies the type of data contained in the **bmiColors** array member of the **BITMAPINFO** structure pointed to by *pbmi* (either logical palette indexes or literal RGB values). The following values are defined.

Value	Meaning
DIB_PAL_COLORS	The **bmiColors** member is an array of 16-bit indexes into the logical palette of the device context specified by *hdc*.
DIB_RGB_COLORS	The **BITMAPINFO** structure contains an array of literal RGB values.

ppvBits

[out] Pointer to a variable that receives a pointer to the location of the DIB's bit values.

hSection

[in] 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's 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.

dwOffset

[in] Specifies the offset from the beginning of the file-mapping object referenced by *hSection* where storage for the bitmap's bit values is to begin. This value is ignored if *hSection* is NULL. The bitmap's bit values are aligned on doubleword boundaries, so *dwOffset* must be a multiple of the size of a **DWORD**.

Return Values

If the function succeeds, the return value is a handle to the newly created DIB, and **ppvBits* points to the bitmap's bit values.

If the function fails, the return value is NULL, and **ppvBits* is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**. This can be the following value.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more 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 dibsection from one application into another application.

Windows NT/ 2000: 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's bit values, including passing the pointer in calls to functions such as **SetDIBits**.

ICM: No color management is done.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFO, CreateFileMapping, DeleteObject, DIBSECTION, GetDIBColorTable, GetObject, GdiFlush, SetDIBits, SetDIBColorTable

---

2.35 CreateDIBitmap

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

```
CreateDIBitmap: procedure
(
        hdc         :dword;
    var lpbmih      :BITMAPINFOHEADER;
        fdwInit     :dword;
    var lpbInit     :var;
    var lpbmi       :BITMAPINFO;
```

```
        fuUsage     :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDIBitmap@24" );
```

Parameters

hdc

[in] Handle to a device context.

lpbmih

[in] Pointer to a bitmap information header structure, which may be one of those shown in the following table.

Operating system	Bitmap information header
Windows NT 3.51 and earlier	**BITMAPINFOHEADER**
Windows NT 4.0 and Windows 95	**BITMAPV4HEADER**
Windows 2000 and Windows 98	**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.

fdwInit

[in] Specifies how the system initializes the bitmap bits. The following values is defined.

Value	Meaning
CBM_INIT	If this flag is set, the system uses the data pointed to by the *lpbInit* and *lpbmi* parameters to initialize the bitmap's 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's bits.

lpbInit

[in] 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.

lpbmi

[in] Pointer to a **BITMAPINFO** structure that describes the dimensions and color format of the array pointed to by the *lpbInit* parameter.

fuUsage

[in] Specifies whether the **bmiColors** member of the **BITMAPINFO** structure was initialized and, if so, whether **bmiColors** contains explicit red, green, blue (RGB) values or palette indexes. The *fuUsage* parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	A color table is provided and consists of an array of 16-bit indexes into the logical palette of the device context into which the bitmap is to be selected.
DIB_RGB_COLORS	A color table is provided and contains literal RGB values.

Return Values

If the function succeeds, the return value is a handle to the bitmap.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Windows 95/98: The created bitmap cannot exceed 16MB in size.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFOHEADER, BITMAPINFO, CreateCompatibleBitmap, CreateDIBSection, DeleteObject, GetDeviceCaps, GetSystemPaletteEntries, SelectObject, SetDIBits

2.36 CreateDiscardableBitmap

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. Win32-based applications should use the **CreateCompatibleBitmap** function.

```
CreateDiscardableBitmap: procedure
(
    hdc     :dword;
    nWidth  :dword;
    nHeight :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateDiscardableBitmap@12" );
```

Parameters

hdc

[in] Handle to a device context.

nWidth

[in] Specifies the width, in pixels, of the bitmap.

nHeight

[in] Specifies the height, in pixels, of the bitmap.

Return Values

If the function succeeds, the return value is a handle to the bitmap.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

When you no longer need the bitmap, call the **DeleteObject** function to delete it.

Windows 95/98: The created bitmap cannot exceed 16MB in size.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, CreateCompatibleBitmap, DeleteObject

2.37 CreateEllipticRgn

The **CreateEllipticRgn** function creates an elliptical region.

```
CreateEllipticRgn: procedure
(
```

```
        nLeftRect   :dword;
        nTopRect    :dword;
        nRightRect  :dword;
        nBottomRect :dword
    );
        stdcall;
        returns( "eax" );
        external( "__imp__CreateEllipticRgn@16" );
```

Parameters

nLeftRect

[in] Specifies the x-coordinate in logical units, of the upper-left corner of the bounding rectangle of the ellipse.

nTopRect

[in] Specifies the y-coordinate in logical units, of the upper-left corner of the bounding rectangle of the ellipse.

nRightRect

[in] Specifies the x-coordinate in logical units, of the lower-right corner of the bounding rectangle of the ellipse.

nBottomRect

[in] Specifies the y-coordinate in logical units, of the lower-right corner of the bounding rectangle of the ellipse.

Return Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **`GetLastError`**.

Remarks

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateEllipticRgnIndirect, DeleteObject, SelectObject

2.38 CreateEllipticRgnIndirect

The **CreateEllipticRgnIndirect** function creates an elliptical region.

```
CreateEllipticRgnIndirect: procedure
(
    var lprc    :RECT
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateEllipticRgnIndirect@4" );
```

Parameters

lprc

[in] 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 Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateEllipticRgn, DeleteObject, RECT, SelectObject

2.39 CreateEnhMetaFile

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

```
CreateEnhMetaFile: procedure
(
        hdcRef          :dword;
        lpFilename      :string;
    var lpRect          :RECT;
        lpDescription   :string
);
```

```
stdcall;
returns( "eax" );
external( "__imp__CreateEnhMetaFileA@16" );
```

Parameters

hdcRef

[in] Handle to a reference device for the enhanced metafile.

lpFilename

[in] 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.

lpRect

[in] Pointer to a **RECT** structure that specifies the dimensions (in .01-millimeter units) of the picture to be stored in the enhanced metafile.

lpDescription

[in] Pointer to a string that specifies the name of the application that created the picture, as well as the picture's title.

Return Values

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.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CloseEnhMetaFile, DeleteEnhMetaFile, GetEnhMetaFileDescription, GetEnhMetaFileHeader, GetWinMetaFileBits, PlayEnhMetaFile, RECT

2.40 CreateFont

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

```
CreateFont: procedure
(
    nHeight             :dword;
    nWidth              :dword;
    nEscapement         :dword;
    nOrientation        :dword;
    fnWeight            :dword;
    fdwItalic           :dword;
    fdwUnderline        :dword;
    fdwStrikeOut        :dword;
    fdwCharSet          :dword;
    fdwOutputPrecision  :dword;
    fdwClipPrecision    :dword;
    fdwQuality          :dword;
    fdwPitchAndFamily   :dword;
    lpszFace            :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateFontA@56" );
```

Parameters

nHeight

[in] Specifies the height, in logical units, of the font's character cell or character. The character height value (also known as the em height) is the character cell height value minus the internal-leading value. The font mapper interprets the value specified in *nHeight* in the following manner.

Value	Meaning
> 0	The font mapper transforms this value into device units and matches it against the cell height of the available fonts.
0	The font mapper uses a default height value when it searches for a match.
< 0	The font mapper transforms this value into device units and matches its absolute value against the character height of the available fonts.

For all height comparisons, the font mapper looks for the largest font that does not exceed the requested size.

This mapping occurs when the font is used for the first time.

For the MM_TEXT mapping mode, you can use the following formula to specify a height for a font with a specified point size:

```
nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
```

nWidth

[in] Specifies 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.

nEscapement

[in] Specifies 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.

Windows NT/ 2000: 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.

Windows 95: The *nEscapement* parameter specifies both the escapement and orientation. You should set *nEscapement* and *nOrientation* to the same value.

nOrientation

[in] Specifies the angle, in tenths of degrees, between each character's base line and the x-axis of the device.

fnWeight

[in] Specifies the weight of the font in the range 0 through 1000. For example, 400 is normal

and 700 is bold. If this value is zero, a default weight is used.

The following values are defined for convenience.

Value	Weight
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

fdwItalic

[in] Specifies an italic font if set to TRUE.

fdwUnderline

[in] Specifies an underlined font if set to TRUE.

fdwStrikeOut

[in] Specifies a strikeout font if set to TRUE.

fdwCharSet

[in] Specifies the character set. The following values are predefined:

ANSI_CHARSET
BALTIC_CHARSET
CHINESEBIG5_CHARSET
DEFAULT_CHARSET
EASTEUROPE_CHARSET

GB2312_CHARSET
GREEK_CHARSET
HANGUL_CHARSET
MAC_CHARSET
OEM_CHARSET
RUSSIAN_CHARSET
SHIFTJIS_CHARSET
SYMBOL_CHARSET
TURKISH_CHARSET

Windows NT/ 2000 or Middle-Eastern Windows 3.1 or later:

HEBREW_CHARSET
ARABIC_CHARSET

Windows NT/ 2000 or Thai Windows 3.1 or later:

THAI_CHARSET

The OEM_CHARSET value specifies a character set that is operating-system dependent.

Windows 95/98: You can use the DEFAULT_CHARSET value to allow the name and size of a font to fully describe the logical font. If the specified font name does not exist, a font from any character set can be substituted for the specified font, so you should use DEFAULT_CHARSET sparingly to avoid unexpected results.

Windows NT/ 2000: DEFAULT_CHARSET is set to a value based on the current system locale. For example, when the system locale is English (United States), it is set as ANSI_CHARSET.

Fonts with other character sets may exist in the operating system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are rendered with that font.

To ensure consistent results when creating a font, do not specify OEM_CHARSET or DEFAULT_CHARSET. If you specify a typeface name in the *lpszFace* parameter, make sure that the *fdwCharSet* value matches the character set of the typeface specified in *lpszFace.*

fdwOutputPrecision

[in] Specifies the output precision. The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type. It can be one of the following values.

Value	Meaning
OUT_CHARACTER_PRECIS	Not used.
OUT_DEFAULT_PRECIS	Specifies the default font mapper behavior.
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.

OUT_OUTLINE_PRECIS	**Windows NT/ 2000:** This value instructs the font mapper to choose from TrueType and other outline-based fonts.
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	**Windows NT/ 2000:** This value is not used by the font mapper, but it is returned when TrueType, other outline-based fonts, and vector fonts are enumerated. **Windows 95/98:** This value is used to map vector fonts, and is returned when TrueType or 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, and OUT_TT_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.

fdwClipPrecision

[in] Specifies the clipping precision. The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values.

Value	Meaning
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.
CLIP_CHARACTER_PRECIS	Not used.
CLIP_STROKE_PRECIS	Not used by the font mapper, but is returned when raster, vector, or TrueType fonts are enumerated. **Windows NT/ 2000:** For compatibility, this value is always returned when enumerating fonts.
CLIP_MASK	Not used.

CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system. For more information about the orientation of coordinate systems, see the description of the *nOrientation* parameter
CLIP_TT_ALWAYS	Not used.

fdwQuality

[in] Specifies the output quality. The output quality defines how carefully GDI must attempt to match the logical-font attributes to those of an actual physical font. It can be one of the following values.

Value	Meaning
ANTIALIASED_QUALITY	**Windows NT 4.0 and later:** Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large. **Windows 95 with Plus! and later:** In addition to the comments for Windows NT, the display must greater than 8-bit color, it must be a single plane device, it cannot be a palette display, and it cannot be in a multiple display monitor setup. In addition, you must select a TrueType font into a screen DC prior to using it in a DIBSection, otherwise antialiasing does not happen.
DEFAULT_QUALITY	Appearance of the font does not matter.
DRAFT_QUALITY	Appearance of the font is less important than when the PROOF_QUALITY value is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.
NONANTIALIASED_QUALITY	**Windows 95 with Plus!, Windows 98, Windows NT 4.0, and Windows 2000:** Font is never antialiased, that is, font smoothing is not done.

PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.

If neither ANTIALIASED_QUALITY nor NONANTIALIASED_QUALITY is selected, the font is antialiased only if the user chooses "smooth screen fonts" in Control Panel.

fdwPitchAndFamily

[in] Specifies the pitch and family of the font. The two low-order bits specify the pitch of the font and can be one of the following values:

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

The four high-order bits specify the font family and can be one of the following values.

Value	Description
FF_DECORATIVE	Novelty fonts. Old English is an example.
FF_DONTCARE	Don't care or don't know.
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.

lpszFace

[in] 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 null terminator. The **EnumFontFamilies** function can be used to enumerate the typeface names of all currently available fonts. For

more information, see the Remarks.

If *lpszFace* is NULL or empty string, GDI uses the first font that matches the other specified attributes.

Return Values

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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 95/98 and Windows NT/Windows 2000, Win32-based 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.

Windows 95/98 and Windows NT 4.0: 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 on a system locale that matches the language, but 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.

Windows 2000: The font mapper for **CreateFont, CreateFontIndirect**, and **CreateFontIndirectEx** recognizes both the English and the localized typeface name, regardless of locale.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, CreateFontIndirect, CreateFontIndirectEx, DeleteObject, EnumFonts, EnumFontFamilies, EnumFontFamiliesEx, SelectObject, EnumFontFamilies

2.41 CreateFontIndirect

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.

```
CreateFontIndirect: procedure
(
    var lplf:LOGFONT
);
    stdcall;
```

```
returns( "eax" );
external( "__imp__CreateFontIndirectA@4" );
```

Parameters

lplf

[in] Pointer to a **LOGFONT** structure that defines the characteristics of the logical font.

Return Values

If the function succeeds, the return value is a handle to a logical font.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

When you no longer need the font, call the **DeleteObject** function to delete it.

Windows 95/98 and Windows NT 4.0: 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 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.

Windows 2000: The font mapper for **CreateFont**, **CreateFontIndirect**, and **CreateFontIndirectEx** recognizes both the English and the localized typeface name, regardless of locale.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, CreateFont, CreateFontIndirectEx, DeleteObject, EnumFonts, EnumFontFamilies, EnumFontFamiliesEx, LOGFONT, SelectObject

---

2.42 CreateFontIndirectEx

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 con-

text.

```
CreateFontIndirectEx: procedure
(
    var penumlfex   :ENUMLOGFONTEXDV
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateFontIndirectExA@4" );
```

Parameters

penumlfex

[in] Pointer to an **ENUMLOGFONTEXDV** structure that defines the characteristics of a multiple master font.

Note, this function ignores the **elfDesignVector** member in **ENUMLOGFONTEXDV**.

Return Values

If the function succeeds, the return value is the handle to the new **ENUMLOGFONTEXDV** structure.

If the function fails, the return value is zero.

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.

Windows 95/98 and Windows NT 4.0: 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 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.

Windows 2000: The font mapper for **CreateFont**, **CreateFontIndirect**, and **CreateFontIndirectEx** recognizes both the English and the localized typeface name, regardless of locale.

Requirements

Windows NT/2000: Requires Windows 2000 or later.
Windows 95/98: Unsupported.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, CreateFont, CreateFontIndirect, EnumFonts, EnumFontFamilies, EnumFontFamiliesEx, ENUMLOGFONTEXDV

2.43 CreateHalftonePalette

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

```
CreateHalftonePalette: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateHalftonePalette@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is a handle to a logical halftone palette.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call `GetLastError`.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, DeleteObject, RealizePalette, SelectPalette, SetStretchBltMode, StretchDIBits, StretchBlt

2.44 CreateHatchBrush

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

```
CreateHatchBrush: procedure
(
    fnStyle     :dword;
    clrref      :COLORREF
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateHatchBrush@8" );
```

Parameters

fnStyle

[in] Specifies the hatch style of the brush. This parameter can be one of the following values.

Value	Meaning
HS_BDIAGONAL	45-degree upward left-to-right hatch
HS_CROSS	Horizontal and vertical crosshatch
HS_DIAGCROSS	45-degree crosshatch
HS_FDIAGONAL	45-degree downward left-to-right hatch
HS_HORIZONTAL	Horizontal hatch
HS_VERTICAL	Vertical hatch

clrref

[in] Specifies the foreground color of the brush that is used for the hatches. To create a **COLORREF** color value, use the **RGB** macro.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

If an application uses a hatch brush to fill the backgrounds of both a parent and a child window with matching color, it may be necessary to set the brush origin before painting the background of the child window. You can do this by having your application call 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 done at brush creation. However, color management is performed when the brush is selected into an ICM-enabled device context.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreatePatternBrush, CreateSolidBrush, DeleteObject, GetBrushOrgEx, SelectObject, SetBrushOrgEx, COLORREF, RGB

2.45 CreateIC

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.

```
CreateIC: procedure
(
        lpszDriver  :string;
        lpszDevice  :string;
        lpszOutput  :string;
    var lpdvmInit   :DEVMODE
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateICA@16" );
```

Parameters

lpszDriver

[in] Pointer to a null-terminated character string that specifies the name of the device driver (for example, Epson).

lpszDevice

[in] 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.

lpszOutput

This parameter is ignored for Win32-based applications, and should be set to NULL. It is provided only for compatibility with 16-bit Windows. For more information, see the Remarks section.

lpdvmInit

[in] 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 Values

If the function succeeds, the return value is the handle to an information context.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

Applications written for 16-bit versions of Windows used the *lpszOutput* parameter to specify a port name or to print to a file. Win32-based applications do not need to specify a port name.

When you no longer need the information DC, call the **DeleteDC** function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, DeleteDC, DocumentProperties, DEVMODE, GetDeviceCaps

---

2.46 CreateMetaFile

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

Note This function is provided only for compatibility with earlier 16-bit versions of Windows. Win32-based applications should use the **CreateEnhMetaFile** function.

```
CreateMetaFile: procedure
(
    lpszFile    :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateMetaFileA@4" );
```

Parameters

lpszFile

[in] 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 Values

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 the new Win32 graphics device interface (GDI) functions such as **PolyBezier**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CloseMetaFile, CreateEnhMetaFile, DeleteMetaFile, GetTextColor, PolyBezier, SelectClipRgn, SelectObject

---

2.47 CreatePalette

The **CreatePalette** function creates a logical palette.

```
CreatePalette: procedure
(
    var lplgpl  :LOGPALETTE
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePalette@4" );
```

Parameters

lplgpl

[in] Pointer to a **LOGPALETTE** structure that contains information about the colors in the logical palette.

Return Values

If the function succeeds, the return value is a handle to a logical palette.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, DeleteObject, GetDeviceCaps, LOGPALETTE, RealizePalette, SelectPalette

---

2.48 CreatePatternBrush

The **CreatePatternBrush** function creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the **CreateDIBSection** function.

```
CreatePatternBrush: procedure
(
    hbmp    :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePatternBrush@4" );
```

Parameters

hbmp

[in] Handle to the bitmap to be used to create the logical brush.

Windows 95/98: Creating brushes from bitmaps or DIBs larger than 8 by 8 pixels is not supported. If a larger bitmap is specified, only a portion of the bitmap is used.

Windows NT/ 2000: Brushes can be created from bitmaps or DIBs larger than 8 by 8 pixels.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, CreateBitmap, CreateBitmapIndirect, CreateCompatibleBitmap, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreateDIBSection, CreateHatchBrush, DeleteObject, GetBrushOrgEx, LoadBitmap, SelectObject, SetBrushOrgEx

---

2.49 CreatePen

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

```
CreatePen: procedure
(
    fnPenStyle  :dword;
    nWidth      :dword;
    crColor     :COLORREF
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePen@12" );
```

Parameters

fnPenStyle

[in] Specifies the pen style. It can be any one of the following values.

Value	Meaning

PS_SOLID	The pen is solid.
PS_DASH	The pen is dashed. This style is valid only when the pen width is one or less in device units.
PS_DOT	The pen is dotted. This style is valid only when the pen width is one or less in device units.
PS_DASHDOT	The pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.
PS_DASHDOTDOT	The pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.
PS_NULL	The pen is invisible.
PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

nWidth

[in] Specifies 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 bit 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.

crColor

[in] Specifies a color reference for the pen color. To generate a **COLORREF** structure, use the **RGB** macro.

Return Values

If the function succeeds, the return value is a handle that identifies a logical pen.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

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 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Pens Overview, Pen Functions, CreatePenIndirect, COLORREF, DeleteObject, ExtCreatePen, GetObject, RGB, SelectObject

---

2.50 CreatePenIndirect

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

```
CreatePenIndirect: procedure
(
    var lplgpn  :LOGPEN
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePenIndirect@4" );
```

Parameters

lplgpn

[in] Pointer to a **LOGPEN** structure that specifies the pen's style, width, and color.

Return Values

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.

Windows NT/2000: To get extended error information, call **GetLastError**.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Pens Overview, Pen Functions, CreatePen, DeleteObject, ExtCreatePen, GetObject, LOGPEN, RGB, SelectObject

---

2.51 CreatePolyPolygonRgn

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

```
CreatePolyPolygonRgn: procedure
(
    var lppt            :POINT;
    var lpPolyCounts    :dword;
        nCount          :dword;
        fnPolyFillMode  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePolyPolygonRgn@16" );
```

Parameters

lppt

[in] 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.

lpPolyCounts

[in] 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*.

nCount

[in] Specifies the total number of integers in the array pointed to by *lpPolyCounts*.

fnPolyFillMode

[in] Specifies the fill mode used to determine which pixels are in the region. This parameter can be one of the following values.

Value	Meaning
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).

WINDING	Selects winding mode (fills any region with a nonzero winding value).

For more information about these modes, see the **SetPolyFillMode** function.

Return Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreatePolygonRgn, DeleteObject, POINT, SelectObject, SetPolyFillMode

---

2.52 CreatePolygonRgn

The **CreatePolygonRgn** function creates a polygonal region.

```
CreatePolygonRgn: procedure
(
    var lppt            :POINT;
        cPoints         :dword;
        fnPolyFillMode  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreatePolygonRgn@12" );
```

Parameters

lppt

[in] 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.

cPoints

[in] Specifies the number of points in the array.

fnPolyFillMode

[in] Specifies the fill mode used to determine which pixels are in the region. This parameter can be one of the following values.

Value	Meaning
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).

WINDING Selects winding mode (fills any region with a nonzero winding value).

For more information about these modes, see the **SetPolyFillMode** function.

Return Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreatePolyPolygonRgn, DeleteObject, POINT, SelectObject, SetPolyFillMode

---

2.53 CreateRectRgn

The **CreateRectRgn** function creates a rectangular region.

```
CreateRectRgn: procedure
(
    nLeftRect   :dword;
    nTopRect    :dword;
    nRightRect  :dword;
    nBottomRect :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateRectRgn@16" );
```

Parameters

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the region in logical units.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the region in logical units.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the region in logical units.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the region in logical units.

Return Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

The region will be exclusive of the bottom and right edges.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateRectRgnIndirect, CreateRoundRectRgn, DeleteObject, SelectObject

---

2.54 CreateRectRgnIndirect

The **CreateRectRgnIndirect** function creates a rectangular region.

```
CreateRectRgnIndirect: procedure
(
    VAR lprc    :rect
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateRectRgnIndirect@4" );
```

Parameters

lprc

[in] 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 Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

The region will be exclusive of the bottom and right edges.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateRectRgn, CreateRoundRectRgn, DeleteObject, RECT, SelectObject

2.55 CreateRoundRectRgn

The **CreateRoundRectRgn** function creates a rectangular region with rounded corners.

```
CreateRoundRectRgn: procedure
(
    nLeftRect       :dword;
    nTopRect        :dword;
    nRightRect      :dword;
    nBottomRect     :dword;
    nWidthEllipse   :dword;
    nHeightEllipse  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateRoundRectRgn@24" );
```

Parameters

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the region in logical units.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the region in logical units.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the region in logical units.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the region in logical units.

nWidthEllipse

[in] Specifies the width of the ellipse used to create the rounded corners in logical units.

nHeightEllipse

[in] Specifies the height of the ellipse used to create the rounded corners in logical units.

Return Values

If the function succeeds, the return value is the handle to the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateRectRgn, CreateRectRgnIndirect, DeleteObject, SelectObject

2.56 CreateScalableFontResource

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

```
CreateScalableFontResource: procedure
(
    fdwHidden       :dword;
    lpszFontRes     :string;
    lpszFontFile    :string;
    lpszCurrentPath :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateScalableFontResourceA@16" );
```

Parameters

fdwHidden

[in] Specifies whether the font is a read-only font. This parameter can be one of the following values.

Value	Meaning
0	The font has read-write permission.
1	The font has read-only permission and should be hidden from other applications in the system. When this flag is set, the font is not enumerated by the **EnumFonts** or **EnumFontFamilies** function.

lpszFontRes

[in] 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.

lpszFontFile

[in] Pointer to a null-terminated string specifying the name of the scalable font file that this function uses to create the font resource file.

lpszCurrentPath

[in] Pointer to a null-terminated string specifying the path to the scalable font file.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**. 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 (typ-

ically 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, AddFontResource, EnumFonts, EnumFontFamilies

---

2.57 CreateSolidBrush

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

```
CreateSolidBrush: procedure
(
    crColor     :COLORREF
);
    stdcall;
    returns( "eax" );
    external( "__imp__CreateSolidBrush@4" );
```

Parameters

crColor

[in] Specifies the color of the brush. To create a **COLORREF** color value, use the **RGB** macro.

Return Values

If the function succeeds, the return value identifies a logical brush.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, CreateDIBPatternBrush, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, DeleteObject, GetSysColorBrush, SelectObject, COLORREF, RGB

2.58 DPtoLP

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

```
DPtoLP: procedure
(
        hdc         :dword;
    var lpPoints    :POINT;
        nCount      :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__DPtoLP@12" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoints

[in/out] Pointer to an array of **POINT** structures. The x- and y-coordinates contained in each **POINT** structure will be transformed.

nCount

[in] Specifies the number of points in the array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, LPtoDP, POINT

2.59 DeleteColorSpace

The **DeleteColorSpace** function removes and destroys a specified color space.

```
DeleteColorSpace: procedure
(
    hColorSpace :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__DeleteColorSpace@4" );
```

hColorSpace

Specifies the handle to a color space to delete.

Return Values

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions

2.60 DeleteDC

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

```
DeleteDC: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__DeleteDC@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CreateDC, GetDC, ReleaseDC

2.61 DeleteEnhMetaFile

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

```
DeleteMetaFile: procedure
(
    hemf    :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__DeleteMetaFile@4" );
```

Parameters

hemf

[in] Handle to an enhanced metafile.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

If the *hemf* parameter identifies an enhanced metafile stored in memory, the **DeleteEnhMetaFile** function deletes the metafile. If *hemf* 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CopyEnhMetaFile, CreateEnhMetaFile, GetEnhMetaFile

2.62 DeleteObject

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

```
DeleteObject: procedure
(
    hObject :dword
);
    stdcall;
```

```
    returns( "eax" );
    external( "__imp__DeleteObject@4" );
```

Parameters

hObject

[in] Handle to a logical pen, brush, font, bitmap, region, or palette.

Return Values

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.

Windows NT/2000: To get extended error information, call **GetLastError**.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, SelectObject

---

2.63 DescribePixelFormat

The **DescribePixelFormat** function obtains information about the pixel format identified by *iPixelFormat* of the device associated with *hdc*. The function sets the members of the **PIXELFORMATDESCRIPTOR** structure pointed to by *ppfd* with that pixel format data.

```
DescribePixelFormat: procedure
(
        hdc             :dword;
        iPixelFormat    :dword;
        nBytes          :dword;
    var ppfd            :PIXELFORMATDESCRIPTOR
);
    stdcall;
    returns( "eax" );
    external( "__imp__DescribePixelFormat@16" );
```

Parameters

hdc

Specifies the device context.

iPixelFormat

Index that specifies the pixel format. The pixel formats that a device context supports are identified by positive one-based integer indexes.

nBytes

The size, in bytes, of the structure pointed to by *ppfd*. The **DescribePixelFormat** function stores no more than *nBytes* bytes of data to that structure. Set this value to **sizeof(PIXELFORMATDESCRIPTOR)**.

ppfd

Pointer to a **PIXELFORMATDESCRIPTOR** structure whose members the function sets with pixel format data. The function stores the number of bytes copied to the structure in the structure's **nSize** member. If, upon entry, *ppfd* is NULL, the function writes no data to the structure. This is useful when you only want to obtain the maximum pixel format index of a device context.

Return Values

If the function succeeds, the return value is the maximum pixel format index of the device context. In addition, the function sets the members of the **PIXELFORMATDESCRIPTOR** structure pointed to by *ppfd* according to the specified pixel format.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The following code sample shows **DescribePixelFormat** usage:

```
PIXELFORMATDESCRIPTOR  pfd;
HDC  hdc;
int  iPixelFormat;

iPixelFormat = 1;

// obtain detailed information about
// the device context's first pixel format
DescribePixelFormat(hdc, iPixelFormat,
        sizeof(PIXELFORMATDESCRIPTOR), &pfd);
```

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in gdi32.hhf.

See Also

OpenGL on Windows NT, Windows 2000, and Windows 95/98, Win32 Functions, ChoosePixelFormat, GetPixelFormat, SetPixelFormat

2.64 DeviceCapabilities

The **DeviceCapabilities** function retrieves the capabilities of a printer device driver.

```
DWORD DeviceCapabilities(
```

```
  LPCTSTR pDevice,          // printer name
  LPCTSTR pPort,            // port name
  WORD fwCapability,        // device capability
  LPTSTR pOutput,           // output buffer
  CONST DEVMODE *pDevMode   // device data buffer
);
```

Parameters

pDevice

[in] Pointer to a null-terminated string that contains the name of the printer. Note that this is the name of the printer, not of the printer driver.

pPort

[in] Pointer to a null-terminated string that contains the name of the port to which the device is connected, such as LPT1.

fwCapability

[in] Specifies the capabilities to query. This parameter can be one of the following values.

Value	Meaning
DC_BINADJUST	**Windows 95/98:** Retrieves the page positioning for the paper source specified in the **DEVMODE** structure pointed to by *pdevMode*. The return value can be one of the following: DCBA_FACEUPNONE DCBA_FACEUPCENTER DCBA_FACEUPLEFT DCBA_FACEUPRIGHT DCBA_FACEDOWNNONE DCBA_FACEDOWNCENTER DCBA_FACEDOWNLEFT DCBA_FACEDOWNRIGHT **Windows NT/2000:** Not supported.
DC_BINNAMES	Retrieves the names of the printer's paper bins. The *pOutput* buffer receives an array of string buffers. Each string buffer is 24 characters long and contains the name of a paper bin. The return value indicates the number of entries in the array. The name strings are null-terminated unless the name is 24 characters long. If *pOutput* is NULL, the return value is the number of bin entries required.

DC_BINS	Retrieves a list of available paper bins. The *pOutput* buffer receives an array of **WORD** values that indicate the available paper sources for the printer. The return value indicates the number of entries in the array. For a list of the possible array values, see the description of the **dmDefaultSource** member of the **DEVMODE** structure. If *pOutput* is NULL, the return value indicates the required number of entries in the array.
DC_COLLATE	If the printer supports collating, the return value is 1; otherwise, the return value is zero. The *pOutput* parameter is not used.
DC_COLORDEVICE	**Windows 2000:** If the printer supports color printing, the return value is 1; otherwise, the return value is zero. The *pOutput* parameter is not used.
DC_COPIES	Returns the number of copies the device can print.
DC_DRIVER	Returns the version number of the printer driver.
DC_DATATYPE_PRODUCED	**Windows 95/98:** The return value is the number of datatypes supported by the printer driver. If the function returns -1, the driver recognizes only the"RAW" datatype. The names of the supported datatypes are copied to an array. Use the names in the **DOCINFO** structure when calling the **StartDoc** function to specify the datatype. **Windows NT/2000:** Not supported.
DC_DUPLEX	If the printer supports duplex printing, the return value is 1; otherwise, the return value is zero. The *pOutput* parameter is not used.
DC_EMF_COMPLIANT	**Windows 95/98:** Determines if a printer driver supports enhanced metafiles (EMF). A return value of 1 means the driver supports EMF. A return value of -1 means the driver does not support EMF. **Windows NT/2000:** Not supported.
DC_ENUMRESOLUTIONS	Retrieves a list of the resolutions supported by the printer. The *pOutput* buffer receives an array of **LONG** values. For each supported resolution, the array contains a pair of **LONG** values that specify the x and y dimensions of the resolution, in dots per inch. The return value indicates the number of supported resolutions. If *pOutput* is NULL, the return value indicates the number of supported resolutions.
DC_EXTRA	Returns the number of bytes required for the device-specific portion of the **DEVMODE** structure for the printer driver.

DC_FIELDS	Returns the **dmFields** member of the printer driver's **DEVMODE** structure. The **dmFields** member indicates which members in the device-independent portion of the structure are supported by the printer driver.
DC_FILEDEPENDENCIES	Retrieves the names of any additional files that need to be loaded when a driver is installed. The *pOutput* buffer receives an array of string buffers. Each string buffer is 64 characters long and contains the name of a file. The return value indicates the number of entries in the array. The name strings are null-terminated unless the name is 64 characters long. If *pOutput* is NULL, the return value is the number of files.
DC_MANUFACTURER	**Windows 95/98:** The return value is the identification number of the printer manufacturer. This value is used with Image Color Management (ICM). **Windows NT/2000:** Not supported.
DC_MAXEXTENT	Returns the maximum paper size that the **dmPaperLength** and **dmPaperWidth** members of the printer driver's **DEVMODE** structure can specify. The **LOWORD** of the return value contains the maximum **dmPaperWidth** value, and the **HIWORD** contains the maximum **dmPaperLength** value.
DC_MEDIAREADY	**Windows 2000:** Retrieves the names of the paper forms that are currently available for use. The *pOutput* buffer receives an array of string buffers. Each string buffer is 64 characters long and contains the name of a paper form. The return value indicates the number of entries in the array. The name strings are null-terminated unless the name is 64 characters long. If *pOutput* is NULL, the return value is the number of paper forms.
DC_MINEXTENT	Returns the minimum paper size that the **dmPaperLength** and **dmPaperWidth** members of the printer driver's **DEVMODE** structure can specify. The **LOWORD** of the return value contains the minimum **dmPaperWidth** value, and the **HIWORD** contains the minimum **dmPaperLength** value.
DC_MODEL	**Windows 95/98:** The return value is the identification of the printer model. This value is used with Image Color Management (ICM). **Windows NT/2000:** Not supported.

DC_ORIENTATION

Returns the relationship between portrait and landscape orientations for a device, in terms of the number of degrees that portrait orientation is rotated counterclockwise to produce landscape orientation. The return value can be one of the following:

0

No landscape orientation.

90

Portrait is rotated 90 degresss to produce landscape.

270

Portrait is rotated 270 degrees to produce landscape.

DC_NUP

Windows 2000: Retrieves an array of integers that indicate that printer's ability to print multiple document pages per printed page. The *pOutput* buffer receives an array of **DWORD** values. Each value represents a supported number of document pages per printed page. The return value indicates the number of entries in the array. If *pOutput* is NULL, the return value indicates the required number of entries in the array.

DC_PAPERNAMES

Retrieves a list of supported paper names (for example, Letter or Legal). The *pOutput* buffer receives an array of string buffers. Each string buffer is 64 characters long and contains the name of a paper form. The return value indicates the number of entries in the array. The name strings are null-terminated unless the name is 64 characters long. If *pOutput* is NULL, the return value is the number of paper forms.

DC_PAPERS

Retrieves a list of supported paper sizes. The *pOutput* buffer receives an array of **WORD** values that indicate the available paper sizes for the printer. The return value indicates the number of entries in the array. For a list of the possible array values, see the description of the **dmPaperSize** member of the **DEVMODE** structure. If *pOutput* is NULL, the return value indicates the required number of entries in the array.

DC_PAPERSIZE

Retrieves the dimensions, in tenths of a millimeter, of each supported paper size. The *pOutput* buffer receives an array of **POINT** structures. Each structure contains the width (x-dimension) and length (y-dimension) of a paper size as if the paper were in the DMORIENT_PORTRAIT orientation. The return value indicates the number of entries in the array.

DC_PERSONALITY

Windows 2000: Retrieves a list of printer description languages supported by the printer. The *pOutput* buffer receives an array of string buffers. Each buffer is 32 characters long and contains the name of a printer description language. The return value indicates the number of entries in the array. The name strings are null-terminated unless the name is 32 characters long. If *pOutput* is NULL, the return value indicates the required number of array entries.

DC_PRINTERMEM

Windows 2000: The return value is the amount of available printer memory, in kilobytes. The *pOutput* parameter is not used.

DC_PRINTRATE

Windows 2000: The return value indicates the printer's print rate. The value returned for DC_PRINTRATEUNIT indicates the units of the DC_PRINTRATE value. The *pOutput* parameter is not used.

DC_PRINTRATEPPM

Windows 2000: The return value indicates the printer's print rate, in pages per minute. The *pOutput* parameter is not used.

DC_PRINTRATEUNIT

Windows 2000: The return value is one of the following values that indicate the print rate units for the value returned for the DC_PRINTRATE flag. The *pOutput* parameter is not used.

PRINTRATEUNIT_CPS

Characters per second.

PRINTRATEUNIT_IPM

Inches per minute.

PRINTRATEUNIT_LPM

Lines per minute.

PRINTRATEUNIT_PPM

Pages per minute.

DC_SIZE

Returns the **dmSize** member of the printer driver's **DEVMODE** structure.

DC_STAPLE

Windows 2000: If the printer supports stapling, the return value is a nonzero value; otherwise, the return value is zero. The *pOutput* parameter is not used.

DC_TRUETYPE	Retrieves the abilities of the driver to use TrueType fonts. For DC_TRUETYPE, the *pOutput* parameter should be NULL. The return value can be one or more of the following: DCTT_BITMAP Device can print TrueType fonts as graphics. DCTT_DOWNLOAD Device can down-load TrueType fonts. DCTT_DOWNLOAD_OUTLINE **Windows 95/98:** Device can download outline TrueType fonts. DCTT_SUBDEV Device can substitute device fonts for TrueType fonts.
DC_VERSION	Returns the specification version to which the printer driver conforms.

pOutput

[out] Pointer to an array. The format of the array depends on the setting of the *fwCapability* parameter. If *pOutput* is NULL, **DeviceCapabilities** returns the number of bytes required for the output data.

pDevMode

[in] Pointer to a **DEVMODE** structure. If this parameter is NULL, **DeviceCapabilities** retrieves the current default initialization values for the specified printer driver. Otherwise, the function retrieves the values contained in the structure to which *pDevMode* points.

Return Values

If the function succeeds, the return value depends on the setting of the *fwCapability* parameter. A return value of zero generally indicates that, while the function completed successfully, there was some type of failure, such as a capability that is not supported. For more details, see the descriptions for the *fwCapability* values.

If the function fails, the return value is -1.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Remarks

For 16-bit programs, **DeviceCapabilities** was implemented in the printer driver. To get a pointer to the function, call **LoadLibrary** and **GetProcAddress**. For 32-bit applications on both Windows 95/98 and Windows NT, **DeviceCapabilities** is part of the Win32 API, so you should not call **LoadLibrary** on the printer driver.

The **DEVMODE** structure pointed to by the *pDevMode* parameter may be obtained by calling the **DocumentProperties** function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, DEVMODE, DOCINFO, DocumentProperties, GetDeviceCaps, GetProcAddress, LoadLibrary, POINT, StartDoc

---

2.65 DrawEscape

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

```
DrawEscape: procedure
(
    hdc         :dword;
    nEscape     :dword;
    cbInput     :dword;
    lpszInData  :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__DrawEscape@16" );
```

Parameters

hdc

[in] Handle to the DC for the specified video display.

nEscape

[in] Specifies the escape function to be performed.

cbInput

[in] Specifies the number of bytes of data pointed to by the *lpszInData* parameter.

lpszInData

[in] Pointer to the input structure required for the specified escape.

Return Values

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 .

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

When an application calls the **DrawEscape** function, the data identified by *cbInput* and *lpszInData* is passed directly to the specified display driver.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions

2.66 Ellipse

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

```
Ellipse: procedure
(
    hdc         :dword;
    nLeftRect   :dword;
    nTopRect    :dword;
    nRightRect  :dword;
    nBottomRect :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__Ellipse@20" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the bounding rectangle.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the bounding rectangle.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the bounding rectangle.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the bounding rectangle.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The current position is neither used nor updated by **Ellipse**.

Windows 95/98: The sum of the coordinates of the bounding rectangle cannot exceed 32,767. The sum of *nLeftRect* and *nRightRect* or *nTopRect* and *nBottomRect* parameters cannot exceed 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, Arc, ArcTo

2.67 EndDoc

The **EndDoc** function ends a print job.

```
EndDoc: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__EndDoc@4" );
```

Parameters

hdc

[in] Handle to the device context for the print job.

Return Values

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is less than or equal to zero.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Remarks

Applications should call **EndDoc** immediately after finishing a print job.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, StartDoc

2.68 EndPage

The **EndPage** function notifies the device that the application has finished writing to a page. This function is typically used to direct the device driver to advance to a new page.

```
EndPage: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__EndPage@4" );
```

Parameters

hdc

[in] Handle to the device context for the print job.

Return Values

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is less than or equal to zero.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Remarks

Use the **ResetDC** function to change the device mode, if necessary, after calling the **EndPage** function. Note that a call to **ResetDC** resets all device context attributes back to default values:

- **Windows 3.x: EndPage** resets the device context attributes back to default values. You must re-select objects and set up the mapping mode again before printing the next page.
- ∑ **Windows 95: EndPage** does not reset the device context attributes. However, the next **StartPage** call does reset the device context attributes to default values. At that time, you must re-select objects and set up the mapping mode again before printing the next page.
- ∑ **Windows NT/Windows 2000:** Beginning with Windows NT Version 3.5, neither **EndPage** or **StartPage** resets the device context attributes. Device context attributes remain constant across subsequent pages. You do not need to re-select objects and set up the mapping mode again before printing the next page; however, doing so will produce the same results and reduce code differences between Windows 95 and Windows NT.

Windows 2000: When a page in a spooled file exceeds approximately 350 MB, it may fail to print and not send an error message. For example, this can occur when printing large EMF files. The page size limit depends on many factors including the amount of virtual memory available, the amount of memory allocated by calling processes, and the amount of fragmentation in the process heap.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, ResetDC, StartPage

---

2.69 EndPath

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

```
EndPath: procedure
(
    hdc :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__EndPath@4" );
```

Parameters

hdc

[in] Handle to the device context into which the new path is selected.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call `GetLastError`. **GetLastError** may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, BeginPath

2.70 EnumEnhMetaFile

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.

```
EnumEnhMetaFile: procedure
(
        hdc             :dword;
        hemf            :dword;
        lpEnhMetaFunc   :ENHMFENUMPROC;
    var lpData          :dword;
    var lpRect          :RECT
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumEnhMetaFile@20" );
```

Parameters

hdc

[in] Handle to a device context. This handle is passed to the callback function.

hemf

[in] Handle to an enhanced metafile.

lpEnhMetaFunc

[in] Pointer to the application-supplied callback function. For more information, see the **EnhMetaFileProc** function.

lpData

[in] Pointer to optional callback-function data.

lpRect

[in] Pointer to a **RECT** structure that specifies the coordinates of the picture's upper-left and lower-right corners. The dimensions of this rectangle are specified in logical units.

Return Values

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 trans-

form the picture displayed by the **PlayEnhMetaFileRecord** function.

You can use the **EnumEnhMetaFile** function to embed one enhanced-metafile within another.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, EnhMetaFileProc, PlayEnhMetaFile, PlayEnhMetaFileRecord, RECT

2.71 EnumFontFamilies

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. Win32-based applications should use the **EnumFontFamiliesEx** function.

```
EnumFontFamilies: procedure
(
        hdc                 :dword;
        lpszFamily          :string;
        lpEnumFontFamProc   :FONTENUMPROC;
    var lParam              :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumFontFamiliesA@16" );
```

Parameters

hdc

[in] Handle to the device context.

lpszFamily

[in] 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.

lpEnumFontFamProc

[in] Point to the application defined—callback function. For information, see **EnumFontFamProc**.

lParam

[in] Pointer to application-supplied data. The data is passed to the callback function along with the font information.

Return Values

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.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, EnumFonts, EnumFontFamiliesEx, EnumFontFamProc

---

2.72 EnumFontFamiliesEx

The **EnumFontFamiliesEx** function enumerates all 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.

```
EnumFontFamiliesEx: procedure
(
        hdc                 :dword;
    var lpLogfont           :LOGFONT;
        lpEnumFontFamExProc :FONTENUMPROC;
    var lParam              :var;
        dwFlags             :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumFontFamiliesExA@20" );
```

Parameters

hdc

[in] Handle to the device context.

lpLogfont

[in] Pointer to a **LOGFONT** structure that contains information about the fonts to enumerate. The function examines the following members.

Member	Description
lfCharset	If set to DEFAULT_CHARSET, the function enumerates all fonts in all character sets. If set to a valid character set value, the function enumerates only fonts in the specified character set.
lfFaceName	If set to an empty string, the function enumerates one font in each available typeface name. If set to a valid typeface name, the function enumerates all fonts with the specified name.
lfPitchAndFamily	Must be set to zero for all language versions of the operating system.

lpEnumFontFamExProc

[in] Pointer to the application defined—callback function. For more information, see the **EnumFontFamExProc** function.

lParam

[in] Specifies 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 Values

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 the values of the **lfCharset** and **lfFacename** 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 **lfCharSet** and **lfFaceName**.

Values	Meaning
lfCharSet = DEFAULT_CHARSET **lfFaceName** = '\0'	Enumerates all fonts in all character sets.

lfCharSet = DEFAULT_CHARSET **lfFaceName** = a specific font	Enumerates all character sets and styles in a specific font.
lfCharSet =a specific character set **lfFaceName** = '\0'	Enumerates all styles of all fonts in the specific character set.
lfCharSet =a specific character set **lfFaceName** = a specific font	Enumerates all styles of a font in a specific character set.

The following code sample shows how these values are used.

```
//to enumerate all styles and charsets of all fonts:
lf.lfFaceName[0] = '\0';
lf.lfCharSet = DEFAULT_CHARSET;

//to enumerate all styles and character sets of the Arial font:
lstrcpy( (LPSTR)&lf.lfFaceName, "Arial" );
lf.lfCharSet = DEFAULT_CHARSET;

//to enumerate all styles of all fonts for the ANSI character set
lf.lfFaceName[0] = '\0';
lf.lfCharSet = ANSI_CHARSET;

//to enumerate all styles of Arial font that cover the ANSI charset
lstrcpy( (LPSTR)&lf.lfFaceName, "Arial" );
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.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, EnumFontFamExProc, EnumFonts, EnumFontFamilies, LOGFONT

2.73 EnumFonts

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. Win32-based applications should use the **EnumFontFamiliesEx** function.

```
EnumFonts: procedure
(
        hdc         :dword;
        lpFaceName  :string;
        lpFontFunc  :FONTENUMPROC;
    var lParam      :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumFontsA@16" );
```

Parameters

hdc

[in] Handle to the device context.

lpFaceName

[in] 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.

lpFontFunc

[in] Pointer to the application defined—callback function. For more information, see **EnumFontsProc**.

lParam

[in] Pointer to any application-defined data. The data is passed to the callback function along with the font information.

Return Values

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, EnumFontFamilies, EnumFontFamiliesEx EnumFontsProc, GetDeviceCaps

---

2.74 EnumICMProfiles

The **EnumICMProfiles** function enumerates the different output color profiles that the system supports for a given device context.

```
EnumICMProfiles: procedure
(
        hdc                     :dword;
        lpEnumICMProfilesFunc   :ICMENUMPROC;
    var lParam                  :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumICMProfilesA@12" );
```

hDC

Specifies the device context.

lpEnumICMProfilesFunc

Specifies the procedure instance address of a callback function defined by the application. (See **EnumICMProfilesProcCallback**.)

lParam

Data supplied by the application that is passed to the callback function along with the color profile information.

Return Values

This function returns zero if the application interrupted the enumeration. The return value is -1 if there are no color profiles to enumerate. Otherwise, the return value is the last value returned by the callback function.

Remarks

The **EnumICMProfiles** function returns a list of profiles that are associated with a device context (DC), and whose settings match those of the DC. It is possible for a device context to contain device profiles that are not associated with particular hardware devices, or device profiles that do not match the settings of the DC. The sRGB profile is an example. The **SetICMProfile** function is used to associate these types of profiles with a DC. The **GetICMProfile** function can be used to retrieve a profile that is not enumerated by the **EnumICMProfiles** function.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.
See Also

Basic Color Management Concepts, Functions, EnumICMProfilesProcCallback, SetICMProfile, GetICMProfile

---

2.75 EnumMetaFile

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 16-bit versions of Windows. Win32-based applications should use the **EnumEnhMetaFile** function.

```
EnumMetaFile: procedure
(
        hdc         :dword;
        hmf         :dword;
        lpMetaFunc  :MFENUMPROC;
    var lParam      :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumMetaFile@16" );
```

Parameters

hdc

[in] Handle to a device context. This handle is passed to the callback function.

hmf

[in] Handle to a Windows-format metafile.

lpMetaFunc

[in] Pointer to an application-supplied callback function. For more information, see **EnumMetaFileProc**.

lParam

[in] Pointer to optional data.

Return Values

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

A Windows-format metafile does not support the new curve, path, and transformation functions, such as **PolyBezier, BeginPath**, and **SetWorldTransform**. Applications that use these new functions *and* use metafiles to store pictures created by these functions should use the enhanced-format metafile functions.

To convert a Windows-format metafile into an enhanced-format metafile, use the **SetWinMeta-FileBits** function.

You can use the **EnumMetaFile** function to embed one Windows-format metafile within another.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, BeginPath, EnumEnhMetaFile, EnumMetaFileProc, PlayMetaFile, PlayMetaFileRecord, PolyBezier, SetWinMetaFileBits, SetWorldTransform

---

2.76 EnumObjects

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.

```
EnumObjects: procedure
(
        hdc             :dword;
        nObjectType     :dword;
        lpObjectFunc    :GOBJENUMPROC;
    var lParam          :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__EnumObjects@16" );
```

Parameters

hdc

[in] Handle to the DC.

nObjectType

[in] Specifies the object type. This parameter can be OBJ_BRUSH or OBJ_PEN.

lpObjectFunc

[in] Pointer to the application-defined callback function. For more information about the callback function, see the **EnumObjectsProc** function.

lParam

[in] Pointer to the application-defined data. The data is passed to the callback function along with the object information.

Return Values

If the function succeeds, the return value is the last value returned by the callback function. Its meaning is user-defined.

If there are too many objects to enumerate, the function returns —1. In this case, the callback function is not called.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, EnumObjectsProc, GetObject

---

2.77 EqualRgn

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

```
EqualRgn: procedure
(
    hSrcRgn1    :dword;
    hSrcRgn2    :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__EqualRgn@8" );
```

Parameters

hSrcRgn1

[in] Handle to a region.

hSrcRgn2

[in] Handle to a region.

Return Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateRectRgn, CreateRectRgnIndirect

2.78 Escape

The **Escape** function enables applications to access capabilities of a particular device not directly available through GDI. Escape calls made by an application are translated and sent to the driver.

```
Escape: procedure
(
        hdc             :dword;
        nEscape         :dword;
        cbInput         :dword;
        lpvInData       :string;
    var lpvOutData      :var
);
    stdcall;
    returns( "eax" );
    external( "__imp__Escape@20" );
```

Parameters

hdc

[in] Handle to the device context.

nEscape

[in] Specifies the escape function to be performed. This parameter must be one of the predefined escape values listed in the Remarks section. Use the **ExtEscape** function if your application defines a private escape value.

cbInput

[in] Specifies the number of bytes of data pointed to by the *lpvInData* parameter.

lpvInData

[in] Pointer to the input structure required for the specified escape.

lpvOutData

[out] Pointer to the structure that receives output from this escape. This parameter should be NULL if no data is returned.

Return Values

If the function succeeds, the return value is greater than zero, except with the QUERYESCSUPPORT printer escape, which checks for implementation only. If the escape is not implemented, the return value is zero.

If the function fails, the return value is an error.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Errors

If the function fails, the return value is one of the following values.

Value	Meaning
SP_ERROR	General error. If SP_ERROR is returned, **Escape** may set the last error code to: ERROR_INVALID_PARAMETER ERROR_DISK_FULL ERROR_NOT_ENOUGH_MEMORY ERROR_PRINT_CANCELLED
SP_OUTOFDISK	Not enough disk space is currently available for spooling, and no more space will become available.
SP_OUTOFMEMORY	Not enough memory is available for spooling.
SP_USERABORT	The user terminated the job through Print Manager.

Remarks

Of the original printer escapes, only the following can be used by Win32-based applications.

Escape	Description
QUERYESCSUPPORT	Determines whether a particular escape is implemented by the device driver.
PASSTHROUGH	Allows the application to send data directly to a printer.

The following printer escapes are obsolete. They are provided only for compatibility with 16-bit versions of Windows.

Escape	Description
ABORTDOC	Stops the current print job and erases everything the application has written to the device since the last ENDDOC escape. In the Win32 API, this is superseded by **AbortDoc**.
ENDDOC	Ends a print job started by the STARTDOC escape. In the Win32 API, this is superseded by **EndDoc**.
GETPHYSPAGESIZE	Retrieves the physical page size and copies it to the specified location. In the Win32 API, this is superseded by PHYSICALWIDTH and PHYSICALHEIGHT in **GetDeviceCaps**.

GETPRINTINGOFFSET	Retrieves the offset from the upper-left corner of the physical page where the actual printing or drawing begins. In the Win32 API, this is superseded by PHYSICALOFFSETX and PHYSICALOFFSETY in **GetDeviceCaps**.
GETSCALINGFACTOR	Retrieves the scaling factors for the x-axis and the y-axis of a printer. In the Win32 API, this is superseded by SCALINGFACTORX and SCALINGFACTORY in **GetDeviceCaps**.
NEWFRAME	Informs the printer that the application has finished writing to a page. In the Win32 API, this is superseded by **EndPage** which ends a page. Unlike NEWFRAME, **EndPage** is always called after printing a page.
NEXTBAND	Informs the printer that the application has finished writing to a band. Band information is not used in Win32 applications.
SETABORTPROC	Sets the Abort function for a print job. In the Win32 API, this is superseded by **SetAbortProc**.
SETCOPYCOUNT	Sets the number of copies. In the Win32 API, this is superseded by **DocumentProperties** or **PrinterProperties**.
STARTDOC	Informs a printer driver that a new print job is starting. In the Win32 API, this is superseded by **StartDoc**.

In addition, the Win32 API has **StartPage** which is used to prepare the printer driver to receive data.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, AbortDoc,DocumentProperties, EndDoc, EndPage, ExtEscape, GetDeviceCaps, PrinterProperties SetAbortProc, StartDoc, StartPage, ResetDC

2.79 ExcludeClipRect

The **ExcludeClipRect** function creates a new clipping region that consists of the existing clipping

region minus the specified rectangle.

```
ExcludeClipRect: procedure
(
    hdc         :dword;
    nLeftRect   :dword;
    nTopRect    :dword;
    nRightRect  :dword;
    nBottomRect :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExcludeClipRect@20" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the logical x-coordinate of the upper-left corner of the rectangle.

nTopRect

[in] Specifies the logical y-coordinate of the upper-left corner of the rectangle.

nRightRect

[in] Specifies the logical x-coordinate of the lower-right corner of the rectangle.

nBottomRect

[in] Specifies the logical y-coordinate of the lower-right corner of the rectangle.

Return Values

The return value specifies the new clipping region's complexity; it can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	No region was created.

Remarks

The lower and right edges of the specified rectangle are not excluded from the clipping region.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, IntersectClipRect

2.80 ExtCreatePen

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

```
ExtCreatePen: procedure
(
        dwPenStyle      :dword;
        dwWidth         :dword;
    var lplb            :LOGBRUSH;
        dwStyleCount    :dword;
    var lpStyle         :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExtCreatePen@20" );
```

Parameters

dwPenStyle

[in] Specifies a combination of type, style, end cap, and join attributes. The values from each category are combined by using the bitwise OR operator (|).

The pen type can be one of the following values.

Value	Meaning
PS_GEOMETRIC	The pen is geometric.
PS_COSMETIC	The pen is cosmetic.

The pen style can be one of the following values.

Value	Meaning
PS_ALTERNATE	**Windows NT/2000:** 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. **Windows 95:** This style is not supported for geometric lines. **Windows 98:** Not supported.

PS_DOT	The pen is dotted. **Windows 95/98:** This style is not supported for geometric lines.
PS_DASHDOT	The pen has alternating dashes and dots. **Windows 95:** This style is not supported for geometric lines. **Windows 98:** Not supported.
PS_DASHDOTDOT	The pen has alternating dashes and double dots. **Windows 95:** This style is not supported for geometric lines. **Windows 98:** Not supported.
PS_NULL	The pen is invisible.
PS_USERSTYLE	**Windows NT/2000:** The pen uses a styling array supplied by the user.
PS_INSIDEFRAME	The pen is solid. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

The end cap is only specified for geometric pens. The end cap can be one of the following values.

Value	Meaning
PS_ENDCAP_ROUND	End caps are round.
PS_ENDCAP_SQUARE	End caps are square.
PS_ENDCAP_FLAT	End caps are flat.

The join is only specified for geometric pens. The join can be one of the following values.

Value	Meaning
PS_JOIN_BEVEL	Joins are beveled.
PS_JOIN_MITER	Joins are mitered when they are within the current limit set by the `SetMiterLimit` function. If it exceeds this limit, the join is beveled.
PS_JOIN_ROUND	Joins are round.

Windows 95/98: The PS_ENDCAP_ROUND, PS_ENDCAP_SQUARE, PS_ENDCAP_FLAT, PS_JOIN_BEVEL, PS_JOIN_MITER, and PS_JOIN_ROUND styles are supported only for

geometric pens when used to draw paths.

dwWidth

[in] Specifies 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.

lplb

[in] Pointer to a **LOGBRUSH** structure. If *dwPenStyle* is PS_COSMETIC, the **lbColor** member specifies the color of the pen and the **lbStyle** 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.

dwStyleCount

[in] Specifies the length, in **DWORD** units, of the *lpStyle* array. This value must be zero if *dwPenStyle* is not PS_USERSTYLE.

lpStyle

[in] 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.

Return Values

If the function succeeds, the return value is a handle that identifies a logical pen.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call **GetLastError**.

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 **lbStyle** member of the **LOGBRUSH** structure pointed to by *lplb* is BS_PATTERN, the bitmap pointed to by the **lbHatch** 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Pens Overview, Pen Functions, CreateDIBSection, CreatePen, CreatePenIndirect, DeleteObject, GetObject, LOGBRUSH, SelectObject, SetMiterLimit

---

2.81 ExtCreateRegion

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

```
ExtCreateRegion: procedure
(
    var lpXform     :XFORM;
        nCount      :dword;
    var lpRgnData   :RGNDATA
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExtCreateRegion@12" );
```

Parameters

lpXform

[in] 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.

nCount

[in] Specifies the number of bytes pointed to by *lpRgnData*.

lpRgnData

[in] Pointer to a **RGNDATA** structure that contains the region data in logical units.

Return Values

If the function succeeds, the return value is the value of the region.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call **GetLastError**.

Remarks

An application can retrieve data for a region by calling the **GetRegionData** function.

Windows 95/98: Regions are no longer limited to the 64K heap.

Windows 95/98: World transforms that involve either shearing or rotations are not supported. **ExtCreateRegion** fails if the transformation matrix is anything other than a scaling or translation of the region.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, GetRegionData, RGNDATA, XFORM

2.82 ExtEscape

The **ExtEscape** function enables applications to access capabilities of a particular device that are not available through GDI.

```
ExtEscape: procedure
(
    hdc         :dword;
    nEscape     :dword;
    cbInput     :dword;
    lpszInData  :string;
    cbOutput    :dword;
    lpszOutData :string
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExtEscape@24" );
```

Parameters

hdc

[in] Handle to the device context.

nEscape

[in] Specifies the escape function to be performed. It can be one of the following or it can be an application-defined escape function.

Value	Meaning
CHECKJPEGFORMAT	**Windows 2000:** Checks whether the printer supports a JPEG image.

CHECKPNGFORMAT	**Windows 2000:** Checks whether the printer supports a PNG image.
DRAWPATTERNRECT	Draws a white, gray-scale, or black rectangle.
GET_PS_FEATURESETTING	**Windows 2000:** Gets information on a specified feature setting for a PostScript driver.
PASSTHROUGH	Allows the application to send data directly to a printer. Supported in compatibility mode and GDI-centric mode.
POSTSCRIPT_DATA	Allows the application to send data directly to a printer. Supported only in compatibility mode.
POSTSCRIPT_IDENTIFY	**Windows 2000:** Sets a PostScript driver to GDI-centric or PostScript-centric mode.
POSTSCRIPT_INJECTION	**Windows 2000:** Inserts a block of raw data in a PostScript job stream.
POSTSCRIPT_PASSTHROUGH	**Windows 2000:** Sends data directly to a PostScript printer driver. Supported in compatibility mode and PS-centric mode.
QUERYESCSUPPORT	Determines whether a particular escape is implemented by the device driver.
SPCLPASSTHROUGH2	**Windows 2000:** Allows applications to include private procedures and other resources at the document level-save context.

cbInput

[in] Specifies the number of bytes of data pointed to by the *lpszInData* parameter.

lpszInData

[in] Pointer to the input structure required for the specified escape.

cbOutput

[in] Specifies the number of bytes of data pointed to by the *lpszOutData* parameter.

lpszOutData

[out] Pointer to the structure that receives output from this escape. This parameter must not be NULL if **ExtEscape** is called as a query function. If no data is to be returned in this structure, set *cbOutput* to 0.

Return Values

The return value specifies the outcome of the function. It is greater than zero if the function is successful, except for the QUERYESCSUPPORT printer escape, which checks for implementation only. The return value is zero if the escape is not implemented. A return value less than zero indi-

cates an error.

Windows NT/Windows 2000: To get extended error information, call **GetLastError**.

Remarks

Use this function to pass a driver-defined escape value to a device.

Use the **Escape** function to pass one of the system-defined escape values to a device, unless the escape is one of the defined escapes in *nEscape*. **ExtEscape** might not work properly with the system-defined escapes. In particular, escapes in which *lpszInData* is a pointer to a structure that contains a member that is a pointer will fail.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, Escape, GetDeviceCaps

---

2.83 ExtFloodFill

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

```
ExtFloodFill: procedure
(
    hdc         :dword;
    nXStart     :dword;
    nYStart     :dword;
    crColor     :dword;
    fuFillType  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExtFloodFill@20" );
```

Parameters

hdc

[in] Handle to a device context.

nXStart

[in] Specifies the logical x-coordinate of the point where filling is to start.

nYStart

[in] Specifies the logical y-coordinate of the point where filling is to start.

crColor

[in] Specifies the color of the boundary or of the area to be filled. The interpretation of *crColor* depends on the value of the *fuFillType* parameter. To create a **COLORREF** color value, use the **RGB**

macro.

fuFillType

[in] Specifies the type of fill operation to be performed. This parameter must be one of the following values.

Value	Meaning
FLOODFILLBORDER	The fill area is bounded by the color specified by the *crColor* 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 *crColor*. Filling continues outward in all directions as long as the color is encountered. This style is useful for filling areas with multicolored boundaries.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

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 *crColor* parameter (if FLOODFILLBORDER was requested).
- ∑ The specified point does not have the color specified by *crColor* (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 *crColor* 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 *crColor*.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, FloodFill, GetDeviceCaps, COLORREF, RGB

2.84 ExtSelectClipRgn

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

```
ExtSelectClipRgn: procedure
(
    hdc     :dword;
    hrgn    :dword;
    fnMode  :dword
);
    stdcall;
    returns( "eax" );
    external( "__imp__ExtSelectClipRgn@12" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region to be selected. This handle can only be NULL when the RGN_COPY mode is specified.

fnMode

[in] Specifies the operation to be performed. It must be one of the following values.

Value	Meaning
RGN_AND	The new clipping region combines the overlapping areas of the current clipping region and the region identified by *hrgn*.
RGN_COPY	The new clipping region is a copy of the region identified by *hrgn*. This is identical to **SelectClipRgn**. If the region identified by *hrgn* 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 *hrgn*.
RGN_OR	The new clipping region combines the current clipping region and the region identified by *hrgn*.

RGN_XOR	The new clipping region combines the current clipping region and the region identified by *hrgn* but excludes any overlapping areas.

Return Values

The return value specifies the new clipping region's complexity; it can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred.

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

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, SelectClipRgn

2.85 ExtTextOut

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

```
ExtTextOut: procedure
(
        hdc         :dword;
        x           :dword;
        y           :dword;
        fuOptions   :dword;
    var lprc        :RECT;
        lpString    :string;
        cbCount     :dword;
    var lpDx        :var
);
```

```
stdcall;
returns( "eax" );
external( "__imp__ExtTextOutA@32" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the logical x-coordinate of the reference point used to position the string.

Y

[in] Specifies the logical y-coordinate of the reference point used to position the string.

fuOptions

[in] Specifies how to use the application-defined rectangle. This parameter can be one or more of the following values.

Value	Meaning
ETO_CLIPPED	The text will be clipped to the rectangle.
ETO_GLYPH_INDEX	The *lpString* 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_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 *lpDx* 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-Eastern 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 compatability.

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.

lprc

[in] Pointer to an optional **RECT** structure that specifies the dimensions of a rectangle that is used for clipping, opaquing, or both.

lpString

[in] 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.

cbCount

[in] Specifies the length of the 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 **WORD**s.

Windows 95/98: This value may not exceed 8192.

lpDx

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call **GetLastError**.

Remarks

Although not true in general, Windows 95/98 supports the Unicode version of this function as well as the ANSI version.

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 **GetTex-**

`tAlign` function. The text-alignment settings are altered by calling the `SetTextAlign` function.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextAlign, RECT, SetBkColor, SelectObject, SetTextAlign, SetTextColor

---

2.86 FillPath

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

```
FillPath: procedure
(
     hdc                :dword
);
     @stdcall;
     returns( "eax" );
     external( "__imp__FillPath@4" );
```

Parameters

hdc

[in] Handle to a device context that contains a valid path.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError**. GetLastError may return one of the following error codes:**

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY

Remarks

After its interior is filled, the path is discarded from the DC identified by the hdc parameter.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, BeginPath, SetPolyFillMode, StrokeAndFillPath, StrokePath

---

2.87 FillRgn

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

```
FillRgn: procedure
(
     hdc                 :dword;
     hrgn           :dword;
     hbr                 :dword
);
     @stdcall;
     returns( "eax" );
     external( "__imp__FillRgn@12" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region to be filled. The region's coordinates are presumed to be in logical units.

hbr

[in] Handle to the brush to be used to fill the region.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError**.**

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf

See Also

Regions Overview, Region Functions, CreateBrushIndirect, CreateDIBPatternBrush, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, PaintRgn

---

2.88 FlattenPath

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

```
FlattenPath: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__FlattenPath@4" );
```

Parameters

hdc

[in] Handle to a DC that contains a valid path.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, WidenPath

2.89 FloodFill

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

Note The FloodFill function is included only for compatibility with 16-bit versions of Windows. For Win32-based applications, use the ExtFloodFill function with FLOODFILLBORDER specified.

```
FloodFill: procedure
(
     hdc                 :dword;
     nXStart        :dword;
     nYStart        :dword;
     crFill         :dword
);
     @stdcall;
     returns( "eax" );
     external( "__imp__FloodFill@16" );
```

Parameters

hdc

[in] Handle to a device context.

nXStart

[in] Specifies the logical x-coordinate of the point where filling is to start.

nYStart

[in] Specifies the logical y-coordinate of the point where filling is to start.

crFill

[in] Specifies the color of the boundary or the area to be filled. To create a COLORREF color value, use the RGB macro.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 *crFill parameter.*

The given point lies outside the current clipping region that is, it is not visible on the device.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, ExtFloodFill, COLORREF, RGB

2.90 FrameRgn

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

```
FrameRgn: procedure
(
    hdc             :dword;
    hrgn            :dword;
    hbr             :dword;
    nWidth          :dword;
    nHeight         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__FrameRgn@20" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region to be enclosed in a border. The region's coordinates are presumed to be in logical units.

hbr

[in] Handle to the brush to be used to draw the border.

nWidth

[in] Specifies the width, in logical units, of vertical brush strokes.

nHeight

[in] Specifies the height, in logical units, of horizontal brush strokes.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf

See Also

Regions Overview, Region Functions, FillRgn, PaintRgn

2.91 GdiComment

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

```
GdiComment: procedure
(
            hdc     :dword;
            cbSize  :dword;
    var     lpData  :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GdiComment@12" );
```

Parameters

hdc

[in] Handle to an enhanced-metafile device context.

cbSize

[in] Specifies the length of the comment buffer, in bytes.

lpData

[in] Pointer to the buffer that contains the comment.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 _IDENTIFIER. The following public comments are defined.

_WINDOWS_METAFILE

The GDICOMMENT_WINDOWS_METAFILE public comment contains a Windows-format metafile that is equivalent to an enhanced-format metafile. This comment is written only by the SetWinMetaFileBits function. The comment record, if given, follows the ENHMETAHEADER metafile record. The comment has the following form:

```
DWORD ident;          // This contains GDICOMMENT_IDENTIFIER.
DWORD iComment;       // This contains GDICOMMENT_WINDOWS_METAFILE.
DWORD nVersion;       // This contains the version number of the
                      // Windows-format metafile.
DWORD nChecksum;      // This is the additive DWORD checksum for
                      // the enhanced metafile.  The checksum
                      // for the enhanced metafile data including
                      // this comment record must be zero.
                        // Otherwise, the enhanced metafile has been
                      //  modified and the Windows-format
                      // metafile is no longer valid.
DWORD fFlags;         // This must be zero.
DWORD cbWinMetaFile; // This is the size, in bytes. of the
                      // Windows-format metafile data that follows.
```

_BEGINGROUP

The GDICOMMENT_BEGINGROUP public comment identifies the beginning of a group of drawing records. It identifies an object within an enhanced metafile. The comment has the following form:

```
DWORD   ident;          // This contains GDICOMMENT_IDENTIFIER.
DWORD   iComment;       // This contains GDICOMMENT_BEGINGROUP.
RECTL   rclOutput;      // This is the bounding rectangle for the
                        // object in logical coordinates.
DWORD   nDescription;  // This is the number of characters in the
                        // optional Unicode description string that
                        // follows. This is zero if there is no
                        // description string.
```

_ENDGROUP

The GDICOMMENT_ENDGROUP public comment identifies the end of a group of drawing records. The GDICOMMENT_BEGINGROUP comment and the GDICOMMENT_ENDGROUP comment must be included in a pair and may be nested. The comment has the following form:

```
DWORD   ident;        // This contains GDICOMMENT_IDENTIFIER.
DWORD   iComment;     // This contains GDICOMMENT_ENDGROUP.
```

_MULTIFORMATS

Windows NT 4.0 SP4 and earlier, Windows 95/98: The GDICOMMENT_MULTIFORMATS public comment allows multiple definitions of a picture to be included in an enhanced metafile. Using this comment, for example, an application can include an encapsulated PostScript definition as well as an enhanced metafile definition of a picture. When the record is played back, GDI selects and renders the first format recognized by the device. The comment has the following form:

```
DWORD   ident;          // This contains GDICOMMENT_IDENTIFIER.
DWORD   iComment;       // This contains GDICOMMENT_MULTIFORMATS.
RECTL   rclOutput;      // This is the bounding rectangle for the
                        // picture in logical coordinates.
DWORD   nFormats;       // This contains the number of formats in
                        // the comment.
EMRFORMAT aemrformat[ 1];// This is an array of EMRFORMAT structures
                        // in the order of preference.  The data
                        // for each format follows the last
                        // EMRFORMAT structure.
```

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CreateEnhMetaFile, SetWinMetaFileBits

---

2.92 GdiFlush

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

```
GdiFlush: procedure;
    @stdcall;
    returns( "eax" );
    external( "__imp__GdiFlush@0" );
```

Parameters

This function has no parameters.

Return Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GdiGetBatchLimit, GdiSetBatchLimit

---

2.93 GdiGetBatchLimit

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

```
GdiGetBatchLimit: procedure;
    @stdcall;
    returns( "eax" );
    external( "__imp__GdiGetBatchLimit@0" );
```

Parameters

This function has no parameters.

Return Values

If the function succeeds, the return value is the batch limit.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GdiFlush, GdiSetBatchLimit

---

2.94 GdiSetBatchLimit

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

```
GdiSetBatchLimit: procedure
(
    dwLimit     :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GdiSetBatchLimit@4" );
```

Parameters

dwLimit

[in] Specifies the batch limit to be set. A value of 0 sets the default limit. A value of 1 disables batching.

Return Values

If the function succeeds, the return value is the previous batch limit.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GdiFlush, GdiGetB

2.95 GetArcDirection

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

```
GetArcDirection: procedure
(
    hdc         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetArcDirection@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

The return value specifies the current arc direction; it can be any one of the following values:

Value	Meaning
AD_COUNTERCLOCKWISE	Arcs and rectangles are drawn counterclockwise.
AD_CLOCKWISE	Arcs and rectangles are drawn clockwise.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, SetArcDirection

---

2.96 GetAspectRatioFilterEx

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

```
GetAspectRatioFilterEx: procedure
(
        hdc             :dword;
    var lpAspectRatio   :SIZE;
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetAspectRatioFilterEx@8" );
```

Parameters

hdc

[in] Handle to a device context.

lpAspectRatio

[out] Pointer to a SIZE structure that receives the current aspect-ratio filter.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, SetMapperFlags, SIZE

2.97 GetBitmapBits

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

Note This function is provided only for compatibility with 16-bit versions of Windows. Win32-based applications should use the GetDIBits function.

```
GetBitmapBits: procedure
(
        hbmp            :dword;
        cbBuffer        :dword;
    var lpvBits         :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBitmapBits@12" );
```

Parameters

hbmp

[in] Handle to the bitmap of interest.

cbBuffer

[in] Specifies the number of bytes to copy from the bitmap into the buffer.

lpvBits

[out] Pointer to a buffer to receive the bitmap bits. The bits are stored as an array of byte values.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions

2.98 GetBitmapDimensionEx

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

```
GetBitmapDimensionEx: procedure
(
        hBitmap         :dword;
    var lpDimension     :SIZE
```

```
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBitmapDimensionEx@8" );
```

Parameters

hBitmap

[in] Handle to the bitmap.

lpDimension

[out] Pointer to a SIZE structure to receive the bitmap dimensions.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 zeroes in those fields.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, SetBitmapDimensionEx, SIZE

2.99 GetBkColor

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

```
GetBkColor: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBkColor@4" );
```

Parameters

hdc

[in] Handle to the device context whose background color is to be returned.

Return Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GetBkMode, SetBkColor, COLORREF

2.100 GetBkMode

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

```
GetBkMode: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBkMode@4" );
```

Parameters

hdc

[in] Handle to the device context whose background mode is to be returned.

Return Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

Library: Use Gdi32.lib.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GetBkColor, SetBkMode

2.101 GetBoundsRect

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.

```
GetBoundsRect: procedure
(
        hdc         :dword;
    var lprcBounds  :RECT;
        flags       :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBoundsRect@12" );
```

Parameters

hdc

[in] Handle to the device context whose bounding rectangle the function will return.

lprcBounds

[out] 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.

flags

[in] Specifies how the GetBoundsRect function will behave. This parameter can be the following value.

Value	Meaning
DCB_RESET	Clears the bounding rectangle after returning it. If this flag is not set, the bounding rectangle will not be cleared.

Return Values

The return value specifies the state of the accumulated bounding rectangle; it can be one of the following values.

Value	Meaning
0	An error occurred. The specified device context handle is invalid.
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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, SetBoundsRect

---

2.102 GetBrushOrgEx

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

```
GetBrushOrgEx: procedure
(
        hdc         :dword;
    var lppt        :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetBrushOrgEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lppt

[out] Pointer to a POINT structure that receives the brush origin, in device coordinates.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows NT/ 2000: 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.

Windows 95/98: Automatic tracking of the brush origin is not supported. Applications must use the UnrealizeObject, SetBrushOrgEx, and SelectObject functions to align the brush before using it.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, POINT, SelectObject, SetBrushOrgEx, UnrealizeObject

---

2.103 GetCharABCWidths

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.

```
GetCharABCWidths: procedure
(
        hdc         :dword;
        uFirstChar  :dword;
        uLastChar   :dword;
    var lpabc       :ABC
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCharABCWidthsA@16" );
```

Parameters

hdc

[in] Handle to the device context.

uFirstChar

[in] Specifies the first character in the group of consecutive characters from the current font.

uLastChar

[in] Specifies the last character in the group of consecutive characters from the current font.

lpabc

[out] Pointer to an array of ABC structures that receives the character widths. 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 Values

If the function succeeds, the return value is nonzero

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 otmEMSquare 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetCharWidth, GetOutlineTextMetrics, OUTLINETEXTMETRIC, ABC

2.104 GetCharABCWidthsFloat

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

```
GetCharABCWidthsFloat: procedure
(
        hdc         :dword;
        iFirstChar  :dword;
        iLastChar   :dword;
    var lpABCF      :ABCFLOAT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCharABCWidthsFloatA@16" );
```

Parameters

hdc

[in] Handle to the device context.

iFirstChar

[in] Specifies the code point of the first character in the group of consecutive characters where

the ABC widths are sought.

iLastChar

[in] Specifies the code point of the last character in the group of consecutive characters where the ABC widths are sought. This range is inclusive. An error is returned if the specified last character precedes the specified first character.

lpABCF

[out] Pointer to an array of ABCFLOAT structures that receives the character widths.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ABCFLOAT, GetCharABCWidths, GetCharWidth, GetCharWidthFloat

---

2.105 GetCharABCWidthsI

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

```
GetCharABCWidthsI: procedure
(
        hdc         :dword;
```

```
        giFirst     :dword;
        cgi         :dword;
    var pgi         :word;
    var lpabc       :ABC
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCharABCWidthsI@20" );
```

Parameters

hdc

[in] Handle to the device context.

giFirst

[in] Specifies 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.

cgi

[in] Specifies the number of glyph indices.

pgi

[in] 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.

lpabc

[out] Pointer to an array of ABC structures that receives the character widths. This array must contain at least as many ABC structures as there are glyph indices specified by the cgi parameter.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 otmEMSquare 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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetCharWidth, GetOutlineTextMetrics, OUTLINETEXTMETRIC, ABC

2.106 GetCharWidth32

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

```
GetCharWidth: procedure
(
        hdc         :dword;
        iFirstChar  :dword;
        iLastChar   :dword;
    var lpBuffer    :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCharWidthA@16" );
```

Parameters

hdc

[in] Handle to the device context.

iFirstChar

[in] Specifies the first character in the group of consecutive characters.

iLastChar

[in] Specifies the last character in the group of consecutive characters, which must not precede the specified first character.

lpBuffer

[out] Pointer to a buffer that receives the character widths.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

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.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetCharABCWidths, GetCharABCWidths-Float, GetCharWidthFloat

---

2.107 GetCharWidthFloat

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

```
GetCharWidthFloat: procedure
(
        hdc         :dword;
        iFirstChar  :dword;
        iLastChar   :dword;
    var pxBuffer    :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCharWidthFloatA@16" );
```

Parameters

hdc

[in] Handle to the device context.

iFirstChar

[in] Specifies the code point of the first character in the group of consecutive characters.

iLastChar

[in] Specifies the code point of the last character in the group of consecutive characters.

pxBuffer

[out] Pointer to a buffer that receives the character widths.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetCharABCWidths, GetCharABCWidths-Float, GetCharWidth32

2.108 GetCharWidthI

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

```
BOOL GetCharWidthI(
  HDC hdc,          // handle to DC
  UINT giFirst,     // first glyph index in range
  UINT cgi,         // number of glyph indicies in range
  LPWORD pgi,       // array of glyph indices
  LPINT lpBuffer    // buffer for widths
);
```

Parameters

hdc

[in] Handle to the device context.

giFirst

[in] Specifies the first glyph index in the group of consecutive glyph indices.

cgi

[in] Specifies the number of glyph indices.

pgi

[in] Pointer to an array of glyph indices. If this parameter is not NULL, it is used instead of the giFirst parameter.

lpBuffer

[out] Pointer to a buffer that receives the widths.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetCharABCWidths, GetCharABCWidthsFloat, GetCharWidth32, GetCharWidthFloat

---

2.109 GetCharacterPlacement

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.

This function has been superseded by the functionality of the Uniscribe module. For more information, see Uniscribe.

```
GetCharacterPlacement: procedure
(
         hdc           :dword;
         lpString:string;
         nCount    :dword;
         nMaxExtent:dword;
     var  lpResults:var;
         flags     :dword
);
     @stdcall;
     returns( "eax" );
     external( "__imp__GetCharacterPlacementA@24" );
```

Parameters

hdc

[in] Handle to the device context.

lpString

[in] Pointer to the character string to process.

nCount

[in] Specifies the length of the 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.
Windows 95/98: This value may not exceed 8192.

nMaxExtent
[in] Specifies 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.
lpResults
[in/out] Pointer to a GCP_RESULTS structure that receives the results of the function.
dwFlags
[in] Specifies how to process the string into the required arrays. This parameter can be one or more of the following values. Value Meaning
GCP_CLASSIN Specifies that the lpClass 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 lpGlyphs and lpDx 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 lpDx array so that the string length is the same as nMaxExtent. GCP_JUSTIFY may only be used in conjunction with GCP_MAXEXTENT.
GCP_KASHIDA 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 nMaxExtent. In the lpDx 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.
Using Kashidas to justifiy 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 dxPage-Width/dxAveCharWidth, where dxPageWidth is the width of the document and dxAveCharWidth is the average character width as returned from a GetTextMetrics call).

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.

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.

GCP_MAXEXTENT Compute extents of the string only as long as the resulting extent, in logical units, does not exceed the values specified by the nMaxExtent parameter.
GCP_NEUTRALOVERRIDE Certain languages only. Override the normal handling of neutrals and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.
GCP_NUMERICOVERRIDE Certain languages only. Override the normal handling of numerics and treat them as strong characters that match the strings reading order. Useful only with the GCP_REORDER flag.
GCP_NUMERICSLATIN Arabic/Thai only. Use standard Latin glyphs for numbers and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_NUMERICSLOCAL Arabic/Thai only. Use local glyphs for numeric characters and override the system default. To determine if this option is available in the language of the font, use GetStringTypeEx to see if the language supports more than one number format.
GCP_REORDER 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 lpClass array is used, the first two elements of the array are used to specify the reading order beyond the bounds of the string. GCP_CLASS_PREBOUNDRTL and GCP_CLASS_PREBOUNDLTR can be used to set the order. If no preset order is required, set the values to zero. These values can be combined with other values if theGCPCLASSIN 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 Use kerning pairs in the font (if any) when creating the widths arrays. Use GetFontLanguage-Info to determine whether the current font supports kerning pairs.
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.

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 Values
If the function succeeds, the return value is the same as the return value from GetTextExtentPoint32, the width and height of the string in logical units.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 lpGlyphs output array is used in the call to ExtTextOut, the ETO_GLYPH_INDEX flag must be set.

GetCharacterPlacement checks the lpOrder, lpDx, lpCaretPos, lpOutString, and lpGlyphs 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 lpDx and/or lpCaretPos members must have valid addresses.

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.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in gid32.hhf Library: Use Gdi32.lib.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GCP_RESULTS, GetCharABCWidths, GetCharWidth32, GetFontLanguageInfo, GetStringTypeEx, GetTextExtentPoint32, GetTextMetrics

---

2.110 GetClipBox

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.

```
GetClipBox: procedure
(
        hdc         :dword;
    var lprc        :RECT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetClipBox@8" );
```

Parameters

hdc

[in] Handle to the device context.

lprc

[out] Pointer to a RECT structure that is to receive the rectangle dimensions.

Return Values

If the function succeeds, the return value specifies the clipping box's complexity and can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred.

GetClipBox returns logical coordinates based on the given device context.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, RECT

2.111 GetClipRgn

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

```
GetClipRgn: procedure
(
    hdc             :dword;
    hrgn            :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetClipRgn@8" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, BeginPaint, SelectClipRgn

---

2.112 GetColorAdjustment

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

```
GetColorAdjustment: procedure
(
        hdc         :dword;
    var lpca        :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetColorAdjustment@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpca

[out] Pointer to a COLORADJUSTMENT structure that receives the color adjustment values.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, SetColorAdjustment, COLORADJUSTMENT

2.113 GetColorSpace

The GetColorSpace function retrieves the handle to the input color space from a specified device context.

```
GetColorSpace: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetColorSpace@4" );
```

hDC

Specifies a device context that is to have its input color space handle retrieved.

Return Values

If the function succeeds, the return value is the current input color space handle.

If this function fails, the return value is NULL.

Remarks

GetColorSpace obtains the handle to the input color space regardless of whether color management is enabled for the device context.

Requirements

Windows NT/2000: Requires Windows 2000.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions

2.114 GetCurrentObject

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

```
GetCurrentObject: procedure
(
        hdc         :dword;
        uObjectType :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCurrentObject@8" );
```

Parameters

hdc

[in] Handle to the DC.

uObjectType

[in] Specifies the object type to be queried. This parameter can be one of the following values.

Value	Meaning
OBJ_BITMAP	Returns the current selected bitmap.
OBJ_BRUSH	Returns the current selected brush.
OBJ_COLORSPACE	Returns the current color space.
OBJ_FONT	Returns the current selected font.
OBJ_PAL	Returns the current selected palette.
OBJ_PEN	Returns the current selected pen.

Return Values

If the function succeeds, the return value is a handle to the specified object.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

An application can use the GetCurrentObject and GetObject functions to retrieve descriptions of the graphic objects currently selected into the specified DC.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, DeleteObject, GetObject, SelectObject, CreateColorSpace

---

2.115 GetCurrentPositionEx

The GetCurrentPositionEx function retrieves the current position in logical coordinates.

```
GetCurrentPositionEx: procedure
(
        hdc         :dword;
    var lpPoint     :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetCurrentPositionEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoint

[out] Pointer to a POINT structure that receives the coordinates of the current position.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, MoveToEx, POINT

2.116 GetDCBrushColor

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

```
GetDCBrushColor: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDCBrushColor@4" );
```

Parameters

hdc

[in] Handle to the DC whose brush color is to be returned.

Return Values

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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Requires Windows 98 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, SetDCBrushColor, COLORREF

2.117 GetDCOrgEx

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).

```
GetDCOrgEx: procedure
(
        hdc         :dword;
    var lpPoint     :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDCOrgEx@8" );
```

Parameters

hdc

[in] Handle to the DC whose final translation origin is to be retrieved.

lpPoint

[out] Pointer to a POINT structure that receives the final translation origin, in device coordinates.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The final translation origin is relative to the physical origin of the screen.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CreateIC, POINT

2.118 GetDCPenColor

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

```
GetDCPenColor: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDCPenColor@4" );
```

Parameters

hdc

[in] Handle to the DC whose brush color is to be returned.

Return Values

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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Requires Windows 98 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, COLORREF, SetDCPenColor

2.119 GetDIBColorTable

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

```
GetDIBColorTable: procedure
(
        hdc         :dword;
        uStartIndex :dword;
        cEntries    :dword;
```

```
    var pColors     :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDIBColorTable@16" );
```

Parameters

hdc

[in] Handle to a device context. A DIB section bitmap must be selected into this device context.

uStartIndex

[in] A zero-based color table index that specifies the first color table entry to retrieve.

cEntries

[in] Specifies the number of color table entries to retrieve.

pColors

[out] Pointer to a buffer that receives an array of RGBQUAD data structures containing color information from the DIB's color table. The buffer must be large enough to contain as many RGBQUAD data structures as the value of cEntries.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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's 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

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFOHEADER, CreateDIBSection, DIBSECTION, GetObject, RGBQUAD, SetDIBColorTable

---

2.120 GetDIBits

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

```
GetDIBits: procedure
```

```
(
        hdc         :dword;
        hbmp        :dword;
        uStartScan  :dword;
        cScanLines  :dword;
    var lpvBits     :var;
    var lpbi        :BITMAPINFO;
        uUsage      :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDIBits@28" );
```

Parameters

hdc

[in] Handle to the device context.

hbmp

[in] Handle to the bitmap.

uStartScan

[in] Specifies the first scan line to retrieve.

cScanLines

[in] Specifies the number of scan lines to retrieve.

lpvBits

[out] 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 lpbi parameter.

lpbi

[in/out] Pointer to a BITMAPINFO structure that specifies the desired format for the DIB data.

uUsage

[in] Specifies the format of the bmiColors member of the BITMAPINFO structure. It must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	The color table should consist of an array of 16-bit indexes into the current logical palette.
DIB_RGB_COLORS	The color table should consist of literal red, green, blue (RGB) values.

Return Values

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

Windows 95/98: If the lpvBits parameter is NULL and GetDIBits successfully fills the BITMAP-INFO structure, the return value is the total number of scan lines in the bitmap.

Windows NT/ 2000: If the lpvBits parameter is NULL and GetDIBits successfully fills the BIT-MAPINFO structure, the return value is non-zero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError. This can be the following value.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more input parameters is invalid.

Remarks

If the requested format for the DIB matches its internal format, the RGB values for the bitmap are copied. If the requested format doesn't match the internal format, a color table is synthesized. The following table describes the color table synthesized for each format.

Value	Meaning
1_BPP	The color table consists of a black and a white entry.
4_BPP	The color table consists of a mix of colors identical to the standard VGA palette.
8_BPP	The color table consists of a general mix of 256 colors defined by GDI. (Included in these 256 colors are the 20 colors found in the default logical palette.)
24_BPP	No color table is returned.

If the 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's 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, SetDIBits

2.121 GetDeviceCaps

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

```
GetDeviceCaps: procedure
```

```
(
    hdc             :dword;
    nIndex          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDeviceCaps@8" );
```

Parameters

hdc

[in] Handle to the DC.

nIndex

[in] Specifies the item to return. This parameter can be one of the following values.

Index	Meaning
DRIVERVERSION	The device driver version.
TECHNOLOGY	Device technology. It can be any one of the following values.
	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 hdc 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.
VERTRES	Height, in raster lines, of the screen.
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	Windows NT/2000: 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 Win32 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	Windows NT/2000: 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	Windows 98, Windows 2000: Value that indicates the shading and blending capabilities of the device.		

	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.	
	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_GDI20_OUTPUT	Capable of supporting features of 16-bit Windows 2.0.
	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:	
	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:		
	LC_NONE	Device does not support lines.	
	LC_INTERIORS	Device can draw interiors.	
	LC_MARKER	Device can draw a marker.	
	LC_POLYLINE	Device can draw a polyline.	
	LC_POLYMARKER	Device can draw multiple markers.	
	LC_STYLED	Device can draw styled lines.	
	LC_WIDE	Device can draw wide lines.	
	LC_WIDESTYLED	Device can draw lines that are wide and styled.	
POLYGONALCAPS	Value that indicates the polygon capabilities of the device, as shown in the following table.		
	PC_NONE	Device does not support polygons.	
	PC_INTERIORS	Device can draw interiors.	
	PC_POLYGON	Device can draw alternate-fill polygons.	
	PC_RECTANGLE	Device can draw rectangles.	
	PC_SCANLINE	Device can draw a single scanline.	
	PC_STYLED	Device can draw styled borders.	
	PC_WIDE	Device can draw wide borders.	
	PC_WIDESTYLED	Device can draw borders that are wide and styled.	
	PC_WINDPOLYGON	Device can draw winding-fill polygons.	
TEXTCAPS	Value that indicates the text capabilities of the device, as shown in the following table.		
	TC_OP_CHARACTER	Device is capable of character output precision.	
	TC_OP_STROKE	Device is capable of stroke output precision.	

	TC_CP_STROKE	Device is capable of stroke clip precision.
	TC_CR_90	Device is capable of 90-degree character rotation.
	TC_CR_ANY	Device is capable of any character rotation.
	TC_SF_X_YINDEP	Device can scale independently in the x- and y-directions.
	TC_SA_DOUBLE	Device is capable of doubled character for scaling.
	TC_SA_INTEGER	Device uses integer multiples only for character scaling.
	TC_SA_CONTIN	Device uses any multiples for exact character scaling.
	TC_EA_DOUBLE	Device can draw double-weight characters.
	TC_IA_ABLE	Device can italicize.
	TC_UA_ABLE	Device can underline.
	TC_SO_ABLE	Device can draw strikeouts.
	TC_RA_ABLE	Device can draw raster fonts.
	TC_VA_ABLE	Device can draw vector fonts.
	TC_RESERVED	Reserved; must be zero.
	TC_SCROLLBLT	Device cannot scroll using a bit-block transfer. Note that this meaning may be the opposite of what you expect.
COLORMGMTCAPS	Windows 2000: Value that indicates the color management capabilities of the device.	
	CM_CMYK_COLOR	Device can accept CMYK color space ICC color profile.
	CM_DEVICE_ICM	Device can perform ICM on either the device driver or the device itself.
	CM_GAMMA_RAMP	Device supports GetDeviceGammaRamp and SetDeviceGammaRamp
	CM_NONE	Device does not support ICM.

Return Values

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

GetDeviceCaps provides the following six indexes in place of printer escapes.

Index	Printer escape replaced
PHYSICALWIDTH	GETPHYSPAGESIZE
PHYSICALHEIGHT	GETPHYSPAGESIZE
PHYSICALOFFSETX	GETPRINTINGOFFSET
PHYSICALOFFSETY	GETPHYSICALOFFSET
SCALINGFACTORX	GETSCALINGFACTOR
SCALINGFACTORY	GETSCALINGFACTOR

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CreateEnhMetaFile, CreateIC, DeviceCapabilities, GetDIBits, GetObjectType, SetDIBits, SetDIBitsToDevice, StretchBlt, StretchDIBits

---

2.122 GetDeviceGammaRamp

The GetDeviceGammaRamp function gets the gamma ramp on direct color display boards having drivers that support downloadable gamma ramps in hardware.

```
GetDeviceGammaRamp: procedure
(
        hdc         :dword;
    var lpRamp      :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetDeviceGammaRamp@8" );
```

hDC

Specifies the device context of the direct color display board in question.

lpRamp

Points to a buffer where the function can place the current gamma ramp of the color display board. The gamma ramp is specified in three arrays of 256 WORD elements each, which contain the mapping between RGB values in the frame buffer and digital-analog-converter (DAC) values. The sequence of the arrays is red, green, blue.

Return Values

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

Remarks

Direct color display modes do not use color lookup tables and are usually 16, 24, or 32 bit. Not all direct color video boards support loadable gamma ramps. GetDeviceGammaRamp succeeds only for devices with drivers that support downloadable gamma ramps in hardware.

Requirements

Windows NT/2000: Requires Windows 2000.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions

2.123 GetEnhMetaFile

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

```
GetEnhMetaFile: procedure
(
    lpszMetaFile    :string
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFileA@4" );
```

Parameters

lpszMetaFile

[in] Pointer to a null-terminated string that specifies the name of an enhanced metafile.

Return Values

If the function succeeds, the return value is a handle to the enhanced metafile.

If the function fails, the return value is NULL.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: The maximum length of the description string for an enhanced metafile is 16,384 bytes.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile, GetEnhMetaFile, SetWinMetaFileBits

---

2.124 GetEnhMetaFileBits

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

```
GetEnhMetaFileBits: procedure
(
        hemf        :dword;
        cbBuffer    :dword;
    var lpbBuffer   :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFileBits@12" );
```

Parameters

hemf

[in] Handle to the enhanced metafile.

cbBuffer

[in] Specifies the size, in bytes, of the buffer to receive the data.

lpbBuffer

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile, GetWinMetaFileBits, SetEnhMetaFileBits

---

2.125 GetEnhMetaFileDescription

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

```
GetEnhMetaFileDescription: procedure
(
    hemf            :dword;
    cchBuffer       :dword;
    lpszDescription :string
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFileDescriptionA@12" );
```

Parameters

hemf

[in] Handle to the enhanced metafile.

cchBuffer

[in] Specifies the size, in characters, of the buffer to receive the data. Only this many characters will be copied.

lpszDescription

[out] Pointer to a buffer that receives the optional text description.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: The maximum length of the description string for an enhanced metafile is 16,384 bytes.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CreateEnhMetaFile

2.126 GetEnhMetaFileHeader

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

```
GetEnhMetaFileHeader: procedure
(
        hemf        :dword;
        cbBuffer    :dword;
    var lpemh       :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFileHeader@12" );
```

Parameters

hemf

[in] Handle to the enhanced metafile for which the header is to be retrieved.

cbBuffer

[in] Specifies the size, in bytes, of the buffer to receive the data. Only this many bytes will be copied.

lpemh

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: The maximum length of the description string for an enhanced metafile is 16,384 bytes.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, ENHMETAHEADER, PlayEnhMetaFile

---

2.127 GetEnhMetaFilePaletteEntries

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

```
GetEnhMetaFilePaletteEntries: procedure
(
        hemf        :dword;
        cEntries    :dword;
    var lppe        :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFilePaletteEntries@12" );
```

Parameters

hemf

[in] Handle to the enhanced metafile.

cEntries

[in] Specifies the number of entries to be retrieved from the optional palette.

lppe

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CreatePalette, PALETTEENTRY, PlayEnhMetaFile, RealizePalette, SelectPalette

---

2.128 GetEnhMetaFilePixelFormat

The GetEnhMetaFilePixelFormat function retrieves pixel format information for an enhanced metafile.

```
GetEnhMetaFilePixelFormat: procedure
(
        hemf        :dword;
        cbBuffer    :dword;
    var ppfd        :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetEnhMetaFilePixelFormat@12" );
```

Parameters

hemf

Identifies the enhanced metafile.

cbBuffer

Specifies the size, in bytes, of the buffer into which the pixel format information is copied.

ppfd

Pointer to a PIXELFORMATDESCRIPTOR structure that contains the logical pixel format specification. The metafile uses this structure to record the logical pixel format specification.

Return Values

If the function succeeds and finds a pixel format, the return value is the size of the metafile's pixel format.

If no pixel format is present, the return value is zero.

If an error occurs and the function fails, the return value is GDI_ERROR. To get extended error information, call GetLastError.

Remarks

When an enhanced metafile specifies a pixel format in its ENHMETAHEADER structure and the pixel format fits in the buffer, the pixel format information is copied into ppfd. When cbBuffer is too small to contain the pixel format of the metafile, the pixel format is not copied to the buffer. In either case, the function returns the size of the metafile's pixel format.

For information on metafile recording and other operations, see Enhanced Metafile Operations.

See Also

OpenGL on Windows NT, Windows 2000, and Windows 95/98, Win32 Functions, ENHMETAHEADER, PIXELFORMATDESCRIPTOR

2.129 GetFontData

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

```
GetFontData: procedure
(
        hdc         :dword;
        dwTable     :dword;
        dwOffset    :dword;
    var lpvBuffer   :var;
        cbData      :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetFontData@20" );
```

Parameters

hdc

[in] Handle to the device context.

dwTable

[in] Specifies 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 font file.

dwOffset

[in] Specifies 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.

lpvBuffer

[out] 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.

cbData

[in] Specifies 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 Values

If the function succeeds, the return value is the number of bytes returned.

If the function fails, the return value is GDI_ERROR.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextMetrics, OUTLINETEXTMETRIC

2.130 GetFontLanguageInfo

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

```
GetFontLanguageInfo: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetFontLanguageInfo@4" );
```

Parameters

hdc

[in] Handle to a display device context.

Return Values

The return value identifies characteristics of the currently selected font. The function returns 0 if the font is "normalized" and can be treated as a simple Latin font; it returns GCP_ERROR if an error occurs. Otherwise, the function returns a combination of the following values.

Value	Meaning
GCP_DBCS	The character set is DBCS.
GCP_DIACRITIC	The font/language contains diacritic glyphs.
FLI_GLYPHS	The font contains extra glyphs not normally accessible using the code page. Use GetCharacterPlacement to access the glyphs. This value is for information only and is not intended to be passed to GetCharacterPlacement.
GCP_GLYPHSHAPE	The font/language contains multiple glyphs per code point or per code point combination (supports shaping and/or ligation), and the font contains advanced glyph tables to provide extra glyphs for the extra shapes. If this value is specified, the lpGlyphs array must be used with the GetCharacterPlacement function and the ETO_GLYPHINDEX value must be passed to the ExtTextOut function when the string is drawn.
GCP_KASHIDA	The font/ language permits Kashidas.
GCP_LIGATE	The font/language contains ligation glyphs which can be substituted for specific character combinations.
GCP_USEKERNING	The font contains a kerning table which can be used to provide better spacing between the characters and glyphs.
GCP_REORDER	The language requires reordering for display—for example, Hebrew or Arabic.

The return value, when masked with FLI_MASK, can be passed directly to the GetCharacter-Placement function.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GetCharacterPlacement

2.131 GetFontUnicodeRanges

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

```
GetFontUnicodeRanges: procedure
(
        hdc         :dword;
    var lpgs        :GLYPHSET
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetFontUnicodeRanges@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpgs

[out] 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 Values

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.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GLYPHSET

2.132 GetGlyphIndices

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.

```
GetGlyphIndices: procedure
(
        hdc         :dword;
        lpstr       :string;
        c           :dword;
    var pgi         :var;
```

```
        fl          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetGlyphIndicesA@20" );
```

Parameters

hdc

[in] Handle to the device context.

lpstr

[in] Pointer to the string to be converted.

c

[in] Length of the string in pgi. 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.

pgi

[out] Array of glyph indices corresponding to the characters in the string.

fl

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

Fonts and Text Overview, Font and Text Functions, GetFontUnicodeRanges

2.133 GetGlyphOutline

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

```
GetGlyphOutline: procedure
(
        hdc         :dword;
        uChar       :char;
        uFormat     :dword;
    var lpgm        :var;
        dbBuffer    :dword;
    var lpvBuffer   :var;
    var lpmat2      :MAT2
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetGlyphOutline@28" );
```

Parameters

hdc

[in] Handle to the device context.

uChar

[in] Specifies the character for which data is to be returned.

uFormat

[in] Specifies the format of the data that the function retrieves. This parameter can be one of the following values.

Value	Meaning
GGO_BEZIER	Windows 2000: The function retrieves the curve data as a cubic Bézier spline (not in quadratic spline format).
GGO_BITMAP	The function retrieves the glyph bitmap. For information about memory allocation, see the following Remarks section.
GGO_GLYPH_INDEX	Windows 95, Windows NT 4.0, and Windows 2000: Indicates that the uChar parameter is a TrueType Glyph Index rather than a character code. See the ExtTextOut function for additional remarks on Glyph Indexing.
GGO_GRAY2_BITMAP	Windows 95, Windows NT 4.0, and Windows 2000: The function retrieves a glyph bitmap that contains five levels of gray.
GGO_GRAY4_BITMAP	Windows 95, Windows NT 4.0, and Windows 2000: The function retrieves a glyph bitmap that contains 17 levels of gray.
GGO_GRAY8_BITMAP	Windows 95, Windows NT 4.0, and Windows 2000: The function retrieves a glyph bitmap that contains 65 levels of gray.
GGO_METRICS	The function only retrieves the GLYPHMETRICS structure specified by lpgm. The other buffers are 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	Windows 2000: 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 con-

tains n^2+1 (n squared plus one) levels of gray.

lpgm

[out] Pointer to the GLYPHMETRICS structure describing the placement of the glyph in the character cell.

cbBuffer

[in] Specifies 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.

lpvBuffer

[out] 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.

lpmat2

[in] Pointer to a MAT2 structure specifying a transformation matrix for the character.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 otmEMSquare 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 preceeding curves and additional contours are packed into the buffer following preceeding 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, FORM_INFO_1, GetOutline-TextMetrics, GLYPHMETRICS, MAT2, OUTLINETEXTMETRIC, POINT , POINTFX, TTPOLYCURVE, TTPOLYGONHEADER

2.134 GetGraphicsMode

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

```
GetGraphicsMode: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetGraphicsMode@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is the current graphics mode. It can be one of the following values.

Value	Meaning

GM_COMPATIBLE — The current graphics mode is the compatible graphics mode, a mode that is compatible with 16-bit Windows. In this graphics mode, an application cannot set or modify the world transformation for the specified device context. The compatible graphics mode is the default graphics mode.

GM_ADVANCED — Windows NT/ 2000: 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.

Windows 95/98: The GM_ADVANCED value is not supported.

Otherwise, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

An application can set the graphics mode for a device context by calling the SetGraphicsMode function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, SetGraphicsMode

---

2.135 GetICMProfile

The GetICMProfile function retrieves the file name of the current output color profile for a specified device context.

```
GetICMProfile: procedure
(
        hdc         :dword;
    var lpcbName    :dword;
        lpszFilename:string
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetICMProfileA@12" );
```

hDC

Specifies a device context from which to retrieve the color profile.

lpcbName

Points to a DWORD that contains the size of the buffer pointed at by lpszFilename. On return this parameter contains the size of the buffer actually used if the function is successful. If the buffer is not large enough, this function will return value FALSE. The GetLastError() function will return ERROR_INSUFFICIENT_BUFFER. The DWORD pointed to by this parameter

will contain the size needed for the lpscFilename buffer.

lpszFilename

Points to the buffer that receives the path name of the profile.

Return Values

If this function succeeds, the return value is TRUE. It also returns TRUE if the lpszFilename parameter is NULL and the size required for the buffer is copied into lpcbName.

If this function fails, the return value is FALSE.

Remarks

GetICMProfile obtains the file name of the current output profile regardless of whether or not color management is enabled for the device context.

Given a device context, GetICMProfile will output, through the parameter lpszFilename, the path name of the file containing the color profile currently being used by the device context. It will also output, through the parameter lpcbName, the length of the string containing the path name.

It is possible that the profile name returned by GetICMProfile will not be in the list of profiles returned by EnumICMProfiles. The EnumICMProfiles function returns all color space profiles that are associated with a device context (DC) whose settings match that of the DC. If the SetICMProfile function is used to set the current profile, a profile may be associated with the DC that does not match its settings. For instance, the SetICMProfile function can be used to associate the device-independent sRGB profile with a DC. This profile will be used as the current ICM profile for that DC, and calls to GetICMProfile will return its file name. However, the profile will not appear in the list of profiles that is returned from EnumICMProfiles.

If this function is called before any calls to the SetICMProfile function, it can be used to get the default profile for a device context.

Requirements

Windows NT/2000: Requires Windows 2000.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions, EnumICMProfiles, SetICMProfile

---

2.136 GetKerningPairs

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

```
GetKerningPairs: procedure
(
        hdc         :dword;
        nNumPairs   :dword;
    var lpkrnpair   :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetKerningPairs@12" );
```

Parameters

hdc

[in] Handle to the device context.

nNumPairs

[in] Specifies the number of pairs in the lpkrnpair array. If the font has more than nNumPairs kerning pairs, the function returns an error.

lpkrnpair

[out] 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 Values

If the function succeeds, the return value is the number of kerning pairs returned.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, KERNINGPAIR

2.137 GetLogColorSpace

The GetLogColorSpace function retrieves the color space definition identified by a specified handle.

```
GetLogColorSpace: procedure
(
        hColorSpace :dword;
    var lpBuffer    :var;
        nSize:dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetLogColorSpaceA@12" );
```

hColorSpace

Specifies the handle to a color space.

lpBuffer

Points to a buffer to receive the LOGCOLORSPACE structure.

nSize

Specifies the maximum size of the buffer.

Return Values

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

Requirements

Windows NT/2000: Requires Windows 2000.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Basic Color Management Concepts, Functions

2.138 GetMapMode

The GetMapMode function retrieves the current mapping mode.

```
GetMapMode: procedure
(
    hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetMapMode@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value specifies the mapping mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The following are the various mapping modes.

Mode	Description
MM_ANISOTROPIC	Logical units are mapped to arbitrary units with arbitrarily scaled axes. Use the SetWindowExtEx and SetViewportExtEx functions to specify the units, orientation, and scaling requiredt.
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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, SetMapMode, SetWindowExtEx, SetViewportExtEx

---

2.139 GetMetaFileBitsEx

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 16-bit versions of Windows. Win32-based applications should use the GetEnhMetaFileBits function.

```
GetMetaFileBitsEx: procedure
(
        hmf         :dword;
        nSize       :dword;
    var lpvData     :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetMetaFileBitsEx@12" );
```

Parameters

hmf

[in] Handle to a Windows-format metafile.

nSize

[in] Specifies the size, in bytes, of the buffer to receive the data.

lpvData

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

A Windows-format metafile does not support the new curve, path, and transformation functions, such as PolyBezier, BeginPath, and SetWorldTransform. Applications that use these new functions and use metafiles to store pictures created by these functions should use the enhanced format metafile functions.

To convert a Windows-format metafile into an enhanced-format metafile, use the SetWinMetaFileBits function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, BeginPath, DeleteMetaFile, GetEnhMetaFileBits, PolyBezier, SetMetaFileBitsEx, SetWinMetaFileBits, SetWorldTransform

---

2.140 GetMetaRgn

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

```
GetMetaRgn: procedure
(
    hdc             :dword;
    hrgn            :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetMetaRgn@8" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, SetMetaRgn

2.141 GetMiterLimit

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

```
GetMiterLimit: procedure
(
        hdc         :dword;
    var peLimit     :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetMiterLimit@8" );
```

Parameters

hdc

[in] Handle to the device context.

peLimit

[out] Pointer to a floating-point value that receives the current miter limit.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The miter limit is used when drawing geometric lines that have miter joins.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

: Use Gdi32.lib.

See Also

Paths Overview, Path Functions, ExtCreatePen, SetMiterLimit

2.142 GetNearestColor

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

```
GetNearestColor: procedure
(
    hdc             :dword;
    crColor         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetNearestColor@8" );
```

Parameters

hdc

[in] Handle to the device context.

crColor

[in] Specifies a color value that identifies a requested color. To create a COLORREF color value, use the RGB macro.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetNearestPaletteIndex, COLORREF, RGB

2.143 GetNearestPaletteIndex

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

```
GetNearestPaletteIndex: procedure
(
    hpal            :dword;
    crColor         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetNearestPaletteIndex@8" );
```

Parameters

hpal

[in] Handle to a logical palette.

crColor

[in] Specifies a color to be matched. To create a COLORREF color value, use the RGB macro.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetNearestColor, GetPaletteEntries, GetSystemPaletteEntries, COLORREF, RGB

2.144 GetObject

The GetObject function retrieves information for the specified graphics object.

```
GetObject: procedure
(
        hgdiobj         :dword;
        cbBuffer        :dword;
    var lpvObject       :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetObjectA@12" );
```

Parameters

hgdiobj

[in] 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.

cbBuffer

[in] Specifies the number of bytes of information to be written to the buffer.

lpvObject

[out] 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.

Object type	Data written to buffer
HBITMAP	BITMAP
HBITMAP returned from a call to CreateDIBSection	DIBSECTION, if *cbBuffer* is set to **sizeof**(**DIBSECTION**), or **BITMAP**, if *cbBuffer* is set to **sizeof**(**BITMAP**)
HPALETTE	A WORD count of the number of entries in the logical palette
HPEN returned from a call to ExtCreatePen	EXTLOGPEN
HPEN	LOGPEN
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.

Return Values

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.

Windows NT/2000: To get extended error information, call GetLastError.

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, EXTLOGPEN, LOGBRUSH, LOGFONT, or LOGPEN 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gid32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CreateDIBSection, GetBitmapBits, GetDIBits, GetPaletteEntries, GetRegionData, BITMAP, DIBSECTION, EXTLOGPEN, LOGBRUSH, LOGFONT, LOGPALETTE, LOGPEN

2.145 GetObjectType

The GetObjectType retrieves the type of the specified object.

```
GetObjectType: procedure
(
    h                   :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetObjectType@4" );
```

Parameters

h

[in] Handle to the graphics object.

Return Values

If the function succeeds, the return value identifies the object. This value can be one of the following.

Value	Meaning
OBJ_BITMAP	Bitmap
OBJ_BRUSH	Brush
OBJ_COLORSPACE	Color space
OBJ_DC	Device context
OBJ_ENHMETADC	Enhanced metafile DC
OBJ_ENHMETAFILE	Enhanced metafile
OBJ_EXTPEN	Extended pen
OBJ_FONT	Font
OBJ_MEMDC	Memory DC
OBJ_METAFILE	Metafile
OBJ_METADC	Metafile DC
OBJ_PAL	Palette
OBJ_PEN	Pen
OBJ_REGION	Region

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

: See Also

Device Contexts Overview, Device Context Functions, GetObject, SelectObject

---

2.146 GetOutlineTextMetrics

The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.

```
GetOutlineTextMetrics: procedure
(
        hdc             :dword;
        cbData          :dword;
    var lpOTM           :OUTLINETEXTMETRIC
```

```
    );
        @stdcall;
        returns( "eax" );
        external( "__imp__GetOutlineTextMetricsA@12" );
```

Parameters

hdc

[in] Handle to the device context.

cbData

[in] Specifies the size, in bytes, of the array that receives the text metrics.

lpOTM

[out] Pointer to an array of OUTLINETEXTMETRIC structures. If this parameter is NULL, the function returns the size of the buffer required for the retrieved metric data.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The OUTLINETEXTMETRIC structure contains most of the text metric information provided for TrueType fonts (including a TEXTMETRIC structure). The sizes returned in the OUTLINE-TEXTMETRIC structures are in logical units; they depend on the current mapping mode.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextMetrics, OUTLINETEXTMETRIC, TEXTMETRIC

2.147 GetPaletteEntries

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

```
GetPaletteEntries: procedure
(
        hpal            :dword;
        iStartIndex     :dword;
        nEntries        :dword;
    var lppe            :PALETTEENTRY
);
    @stdcall;
```

```
    returns( "eax" );
    external( "__imp__GetPaletteEntries@16" );
```

Parameters

hpal

[in] Handle to the logical palette.

iStartIndex

[in] Specifies the first entry in the logical palette to be retrieved.

nEntries

[in] Specifies the number of entries in the logical palette to be retrieved.

lppe

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetSystemPaletteEntries, SetPaletteEntries, PALETTEENTRY

2.148 GetPath

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

```
GetPath: procedure
(
```

```
        hdc             :dword;
    var lpPoints        :POINT;
    var lpTypes         :var;
        nSize           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetPath@16" );
```

Parameters

hdc

[in] Handle to a device context that contains a closed path.

lpPoints

[out] Pointer to an array of POINT structures that receives the line endpoints and curve control points.

lpTypes

[out] Pointer to an array of bytes that receives the vertex types. This parameter can be one of the following values.

Type	Description
PT_MOVETO	Specifies that the corresponding point in the lpPoints parameter starts a disjoint figure.
PT_LINETO	Specifies that the previous point and the corresponding point in lpPoints are the endpoints of a line.
PT_BEZIERTO	Specifies that the corresponding point in lpPoints 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.

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.

nSize

[in] Specifies 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 Values

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.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_BUFFER_OVERFLOW

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, FlattenPath, POINT, PolyDraw, WidenPath

---

2.149 GetPixel

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

```
GetPixel: procedure
(
    hdc                 :dword;
    nXPos               :dword;
    nYPos               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetPixel@12" );
```

Parameters

hdc

[in] Handle to the device context.

nXPos

[in] Specifies the logical x-coordinate of the pixel to be examined.

nYPos

[in] Specifies the logical y-coordinate of the pixel to be examined.

Return Values

The return value is the RGB value of the pixel. If the pixel is outside of the current clipping region, the return value is CLR_INVALID.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, COLORREF, GetDeviceCaps, SetPixel

---

2.150 GetPixelFormat

Returns a globally unique identifier (GUID) that specifies image pixel format.

Syntax

```
GetPixelFormat: procedure
(
    var pFormat         :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetPixelFormat@4" );
```

Parameters

pFormat

[out] Pointer to a DWORD that returns the pixel format identifier (ID). Possible values are defined in GDIPlusPixelFormats.h.

Return Value

Returns S_OK if successful, or an error value otherwise.

See Also

IShellImageData::GetProperties, IShellImageData::GetSize

2.151 GetPolyFillMode

The GetPolyFillMode function retrieves the current polygon fill mode.

```
GetPolyFillMode: procedure
(
```

```
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetPolyFillMode@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value specifies the polygon fill mode, which can be one of the following values.

Value	Meaning
ALTERNATE	Selects alternate mode (fills area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

If an error occurs, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, SetPolyFillMode

2.152 GetROP2

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

```
GetROP2: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetROP2@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value specifies the foreground mix mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

Following are the foreground mix modes.

Mix mode	Description
R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
R2_MERGEPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEPENNOT	Pixel is a combination of the pen color and the inverse of the screen color.
R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the screen color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, SetROP2

2.153 GetRandomRgn

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

```
GetRandomRgn: procedure
(
    hdc                 :dword;
    hrgn                :dword;
    iNum                :dword
);
```

```
    @stdcall;
    returns( "eax" );
    external( "__imp__GetRandomRgn@12" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] 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.

iNum

[in] This parameter must be SYSRGN.

Return Values

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.

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 bracket 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.

In Windows NT/ 2000, the region returned is in screen coordinates. In Windows 95/98, the region returned is in window coordinates.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, BeginPaint, EndPaint, ExtSelectClipRgn, GetClipRgn, GetClipBox, GetRegionData, OffsetRgn

---

2.154 GetRasterizerCaps

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

```
GetRasterizerCaps: procedure
(
    var lprs            :RASTERIZER_STATUS;
```

```
        cb              :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetRasterizerCaps@8" );
```

Parameters

lprs

[out] Pointer to a RASTERIZER_STATUS structure that receives information about the rasterizer.

cb

[in] Specifies the number of bytes to be copied into the structure pointed to by the lprs parameter.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetOutlineTextMetrics, RASTERIZER_STATUS

2.155 GetRegionData

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

```
GetRegionData: procedure
(
        hRgn            :dword;
        dwCount         :dword;
    var lpRgnData       :RGNDATA
```

```
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetRegionData@12" );
```

Parameters

hRgn

[in] Handle to the region.

dwCount

[in] Specifies the size, in bytes, of the lpRgnData buffer.

lpRgnData

[out] Pointer to a RGNDATA structure that receives the information. The dimensions of the region are in logical units. If this parameter is NULL, the return value contains the number of bytes needed for the region data.

Return Values

If the function succeeds and dwCount specifies an adequate number of bytes, the return value is always dwCount. If dwCount is too small or the function fails, the return value is 0. If lpRgnData is NULL, the return value is the required number of bytes.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The GetRegionData function is used in conjunction with the ExtCreateRegion function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, ExtCreateRegion, RGNDATA

2.156 GetRgnBox

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

```
GetRgnBox: procedure
(
        hrgn            :dword;
    var lprc            :RECT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetRgnBox@8" );
```

Parameters

hrgn

[in] Handle to the region.

lprc

[out] Pointer to a RECT structure that receives the bounding rectangle in logical units.

Return Values

The return value specifies the region's complexity. It can be one of the following values:

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than a single rectangle.

If the hrgn parameter does not identify a valid region, the return value is zero.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Regions Overview, Region Functions, RECT

2.157 GetStockObject

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

```
GetStockObject: procedure
(
    fnObject            :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetStockObject@4" );
```

Parameters

fnObject

[in] Specifies the type of stock object. This parameter can be one of the following values.

Value	Meaning
BLACK_BRUSH	Black brush.
DKGRAY_BRUSH	Dark gray brush.
DC_BRUSH	Windows 98, Windows 2000: 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	Windows 98, Windows 2000: Solid pen color. The default color is white. The color can be changed by using the SetDCPenColor function. For more information, see the Remarks section.
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	Windows NT/2000: Device-dependent font.
DEFAULT_GUI_FONT	Default font for user interface objects such as menus and dialog boxes. This is MS Sans Serif. Compare this with SYSTEM_FONT.
OEM_FIXED_FONT	Original equipment manufacturer (OEM) dependent fixed-pitch (monospace) font.
SYSTEM_FONT	System font. By default, the system uses the system font to draw menus, dialog box controls, and text. Windows 95/98 and NT: The system font is MS Sans Serif. Windows 2000: The system font is Tahoma
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 Values

If the function succeeds, the return value is a handle to the requested logical object.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

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.

The font used by the DEFAULT_GUI_FONT stock object could change. Use this stock object when you want to use the font that menus, dialog boxes, and other user interface objects use.

It is not necessary (but it is not harmful) to delete stock objects by calling DeleteObject.

Windows 98, Windows 2000: 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Device Contexts Overview, Device Context Functions, DeleteObject, SelectObject

2.158 GetStretchBltMode

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

```
GetStretchBltMode: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetStretchBltMode@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is the current stretching mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gid32.hhf.

See Also

Bitmaps Overview, Bitmap Functions

2.159 GetSystemPaletteEntries

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

```
GetSystemPaletteEntries: procedure
(
```

```
        hdc                 :dword;
        iStartIndex:dword;
        nEntries  :dword;
    var  lppe           :PALETTEENTRY
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetSystemPaletteEntries@16" );
```

Parameters

hdc

[in] Handle to the device context.

iStartIndex

[in] Specifies the first entry to be retrieved from the system palette.

nEntries

[in] Specifies the number of entries to be retrieved from the system palette.

lppe

[out] 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. If this parameter is NULL, the function returns the total number of entries in the palette.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

An application can determine whether a device supports palette operations by calling the GetDeviceCaps function and specifying the RASTERCAPS constant.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetPaletteEntries, PALETTEENTRY

2.160 GetSystemPaletteUse

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

```
GetSystemPaletteUse: procedure
(
    hdc                 :dword
```

```
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetSystemPaletteUse@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is the current state of the system palette. This parameter can be one of the following values.

Value	Meaning
SYSPAL_NOSTATIC	The system palette contains no static colors except black and white.
SYSPAL_STATIC	The system palette contains static colors that will not change when an application realizes its logical palette.
SYSPAL_ERROR	The given device context is invalid or does not support a color palette.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 or not a device supports color palettes by calling the GetDeviceCaps function and specifying the RASTERCAPS constant.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, SetSystemPaletteUse

---

2.161 GetTextAlign

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

```
GetTextAlign: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
```

```
external( "__imp__GetTextAlign@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is the status of the text-alignment flags. For more information about the return value, see the Remarks section. The return value is a combination of the following values.

Value	Meaning
TA_BASELINE	The reference point is on the base line of the text.
TA_BOTTOM	The reference point is on the bottom edge of the bounding rectangle.
TA_TOP	The reference point is on the top edge of the bounding rectangle.
TA_CENTER	The reference point is aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point is on the left edge of the bounding rectangle.
TA_RIGHT	The reference point is on the right edge of the bounding rectangle.
TA_RTLREADING	Middle-Eastern Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This only applies when the font selected into the device context is either Hebrew or Arabic.
TA_NOUPDATECP	The current position is not updated after each text output call.
TA_UPDATECP	The current position is updated after each text output call.

When the current font has a vertical default base line (as with Kanji), the following values are used instead of TA_BASELINE and TA_CENTER.

Value	Meaning
VTA_BASELINE	The reference point is on the base line of the text.
VTA_CENTER	The reference point is aligned vertically with the center of the bounding rectangle.

If the function fails, the return value is GDI_ERROR.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Apply the bitwise OR operator to the flag and its related flags.

Apply the bitwise AND operator to the result and the return value.

Test for the equality of this result and the flag.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GetTextExtentPoint32, SetTextAlign, TextOut

---

2.162 GetTextCharacterExtra

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

```
GetTextCharacterExtra: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextCharacterExtra@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is the current intercharacter spacing.

If the function fails, the return value is 0x8000000.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, SetTextCharacterExtra, TextOut

---

2.163 GetTextCharset

The GetTextCharset function retrieves a character-set identifier for the font that is currently selected into a specified device context.

The function call GetTextCharset(hdc) is equivalent to the function call GetTextCharsetInfo(hdc, NULL, 0).

```
GetTextCharset: procedure
(
    hdc                     :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextCharset@4" );
```

Parameters

hdc

[in] Handle to a device context. The function obtains a character-set identifier for the font that is selected into this device context.

Return Values

If the function succeeds, the return value identifies the character set of the font that is currently selected into the specified device context. The following character-set identifiers are defined:

ANSI_CHARSET

BALTIC_CHARSET

CHINESEBIG5_CHARSET

DEFAULT_CHARSET

EASTEUROPE_CHARSET

GB2312_CHARSET

GREEK_CHARSET

HANGUL_CHARSET

MAC_CHARSET

OEM_CHARSET

RUSSIAN_CHARSET

SHIFTJIS_CHARSET

SYMBOL_CHARSET

TURKISH_CHARSET

Korean Windows:

JOHAB_CHARSET

Middle-Eastern Windows:

HEBREW_CHARSET

ARABIC_CHARSET

Thai Windows:

THAI_CHARSET

If the function fails, the return value is DEFAULT_CHARSET.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Unicode and Character Sets Overview, Unicode and Character Set Functions, GetTextCharsetInfo

---

2.164 GetTextCharsetInfo

The GetTextCharsetInfo function retrieves information about the character set of the font that is currently selected into a specified device context.

```
GetTextCharsetInfo: procedure
(
        hdc             :dword;
    var lpSig           :FONTSIGNATURE;
        dwFlags         :dword
);
    @stdcall;
```

```
returns( "eax" );
external( "__imp__GetTextCharsetInfo@12" );
```

Parameters

hdc

[in] Handle to a device context. The function obtains information about the font that is selected into this device context.

lpSig

[out] Pointer to a FONTSIGNATURE data structure that receives font-signature information.

If a TrueType font is currently selected into the device context, the FONTSIGNATURE structure receives information that identifies the code page and Unicode subranges for which the font provides glyphs.

If a font other than TrueType is currently selected into the device context, the FONTSIGNATURE structure receives zeroes. In this case, use the TranslateCharsetInfo function to obtain generic font-signature information for the character set.

The lpSig parameter can be NULL if you do not need the FONTSIGNATURE information. In this case, you can also call the GetTextCharset function, which is equivalent to calling GetTextCharsetInfo with lpSig set to NULL.

dwFlags

This parameter is reserved for future use. It must be set to zero.

Return Values

If the function succeeds, the return value identifies the character set of the font currently selected into the specified device context. The following character-set identifiers are defined:

ANSI_CHARSET

BALTIC_CHARSET

CHINESEBIG5_CHARSET

DEFAULT_CHARSET

EASTEUROPE_CHARSET

GB2312_CHARSET

GREEK_CHARSET

HANGUL_CHARSET

MAC_CHARSET

OEM_CHARSET

RUSSIAN_CHARSET

SHIFTJIS_CHARSET

SYMBOL_CHARSET

TURKISH_CHARSET

Korean Windows:

JOHAB_CHARSET

Middle-Eastern Windows:

HEBREW_CHARSET

ARABIC_CHARSET

Thai Windows:

THAI_CHARSET

If the function fails, the return value is DEFAULT_CHARSET.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Unicode and Character Sets Overview, Unicode and Character Set Functions, FONTSIGNATURE, GetTextCharset, TranslateCharsetInfo

---

2.165 GetTextColor

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

```
GetTextColor: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextColor@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

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.

Remarks

The text color defines the foreground color of characters drawn by using the TextOut or ExtTextOut function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, SetTextColor, TextOut, COLOR-

2.166 GetTextExtentExPoint

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.

```
GetTextExtentExPoint: procedure
(
        hdc             :dword;
        lpszStr         :string;
        cchString       :dword;
        nMaxExtent      :dword;
    var lpnFit          :dword;
    var alpDx           :var;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextExtentExPointA@28" );
```

Parameters

hdc

[in] Handle to the device context.

lpszStr

[in] Pointer to the null-terminated string for which extents are to be retrieved.

cchString

[in] Specifies 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.

Windows 95/98: This value may not exceed 8192.

nMaxExtent

[in] Specifies the maximum allowable width, in logical units, of the formatted string.

lpnFit

[out] 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.

alpDx

[out] 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.

lpSize

[out] Pointer to a SIZE structure that receives the dimensions of the string, in logical units. This parameter cannot be NULL.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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 returning the text extent, 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextExtentPoint, SIZE

2.167 GetTextExtentExPointI

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

```
GetTextExtentExPointI: procedure
```

```
(
        hdc             :dword;
    var pgiIn           :var;
        cgi             :dword;
        nMaxExtent      :dword;
    var lpnFit          :dword;
    var alpDx           :var;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextExtentExPointI@28" );
```

Parameters

hdc

[in] Handle to the device context.

pgiIn

[in] Pointer to an array of glyph indices for which extents are to be retrieved.

cgi

[in] Specifies the number of glyphs in the array pointed to by the pgiIn parameter.

nMaxExtent

[in] Specifies the maximum allowable width, in logical units, of the formatted string.

lpnFit

[out] 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.

alpDx

[out] 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 lpnDx is NULL, the function does not compute partial string widths.

lpSize

[out] Pointer to a SIZE structure that receives the dimensions of the glyph indices array, in logical units. This value cannot be NULL.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError.

Remarks

If both the lpnFit and lpnDx parameters are NULL, calling the GetTextExtentExPointI function is equivalent to calling the GetTextExtentPointI function.

When returning the text extent, 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.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextExtentPoint, SIZE

2.168 GetTextExtentPoint32

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

```
GetTextExtentPoint32: procedure
(
        hdc             :dword;
        lpString        :string;
        cbString        :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextExtentPoint32A@16" );
```

Parameters

hdc

[in] Handle to the device context.

lpString

[in] Pointer to a buffer that specifies the text string. The string does not need to be zero-terminated, because the cbString parameter specifies the length of the string.

cbString

[in] Specifies the length of the lpString buffer. 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.

Windows 95/98: This value may not exceed 8192.

lpSize

[out] Pointer to a SIZE structure that receives the dimensions of the string.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

When returning the text extent, 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.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Fonts and Text Overview, Font and Text Functions, SetTextCharacterExtra, SIZE

2.169 GetTextExtentPoint

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. Win32-based applications should call the GetTextExtentPoint32 function, which provides more accurate results.

```
GetTextExtentPoint: procedure
(
        hdc             :dword;
        lpString        :string;
        cpString        :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextExtentPointA@16" );
```

Parameters

hdc

[in] Handle to the device context.

lpString

[in] 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.

cbString

[in] Specifies the length of the string pointed to by lpString. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count. Note, 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)) take one WORD while Unicode surrogates take two WORDs.

Windows 95/98: This value may not exceed 8192.

lpSize

[out] Pointer to a SIZE structure that receives the dimensions of the string.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextExtentPoint32, SetTextCharacterExtra, SIZE

2.170 GetTextExtentPointI

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

indices.

```
GetTextExtentPointI: procedure
(
        hdc             :dword;
    var pgiIn           :word;
        cgi             :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextExtentPointI@16" );
```

Parameters

hdc

[in] Handle to the device context.

pgiIn

[in] Pointer to array of glyph indices.

cgi

[in] Specifies the number of glyph indices.

lpSize

[out] Pointer to a SIZE structure that receives the dimensions of the string.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError.

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 returning the text extent, 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 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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextExtentPoint, GetTextExtentPoint32, SetTextCharacterExtra, SIZE

2.171 GetTextFace

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

```
GetTextFace: procedure
(
    hdc                 :dword;
    nCount              :dword;
    lpFaceName          :string
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextFaceA@12" );
```

Parameters

hdc

[in] Handle to the device context.

nCount

[in] Specifies 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.

lpFaceName

[out] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextAlign, GetTextColor, GetTextExtentPoint32, GetTextMetrics

---

2.172 GetTextMetrics

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

```
GetTextMetrics: procedure
(
        hdc             :dword;
    var lptm            :TEXTMETRIC
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetTextMetricsA@8" );
```

Parameters

hdc

[in] Handle to the device context.

lptm

[out] Pointer to the TEXTMETRIC structure that receives the text metrics.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextAlign, GetTextExtentPoint32, GetTextFace, SetTextJustification, TEXTMETRIC

2.173 GetViewportExtEx

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

```
GetViewportExtEx: procedure
(
        hdc             :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetViewportExtEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpSize

[out] Pointer to a SIZE structure that receives the x- and y-extents, in device units.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWindowExtEx, SetViewportExtEx, SetWindowExtEx

2.174 GetViewportOrgEx

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

```
GetViewportOrgEx: procedure
(
        hdc             :dword;
    var lpPoint         :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetViewportOrgEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoint

[out] Pointer to a POINT structure that receives the coordinates of the origin, in device units.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWindowOrgEx, POINT, SetViewportOrgEx, SetWindowOrgEx

---

2.175 GetWinMetaFileBits

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

```
GetWinMetaFileBits: procedure
(
        hemf            :dword;
        cbBuffer        :dword;
    var lpbBuffer       :var;
        fnMapMode       :dword;
        hdc             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetWinMetaFileBits@20" );
```

Parameters

hemf

[in] Handle to the enhanced metafile.

cbBuffer

[in] Specifies the size, in bytes, of the buffer into which the converted records are to be copied.

lpbBuffer

[out] 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.

fnMapMode

[in] Specifies the mapping mode to use in the converted metafile.

hdcRef

[in] Handle to the reference device context.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Due to the limitations of the Windows-format metafile, some information can be lost in the retrieved metafile contents. For example, an original call to the PolyBezier function in the enhanced metafile may be converted into a call to the Polyline function in the Windows-format metafile, because there is no equivalent PolyBezier function in the Windows format.

16-bit Windows-based applications define the viewport origin and extents of a picture stored in a Windows-format metafile. As a result, the Windows-format records created by GetWinMetaFileBits do not contain the SetViewportOrgEx and SetViewportExtEx functions. However, GetWinMetaFileBits does create Windows-format records for the SetWindowExtEx and SetMapMode functions.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile, PolyBezier, Polyline, SetMapMode, SetViewportOrgEx, SetViewportExtEx, SetWindowExtEx, SetWinMetaFileBits

2.176 GetWindowExtEx

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

```
GetWindowExtEx: procedure
(
        hdc             :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetWindowExtEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpSize

[out] Pointer to a SIZE structure that receives the x- and y-extents in page-space units, that is, logical units.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportExtEx, SetViewportExtEx, SetWindowExtEx

2.177 GetWindowOrgEx

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

```
GetWindowOrgEx: procedure
(
        hdc             :dword;
    var lpPoint         :POINT
);
```

```
    @stdcall;
    returns( "eax" );
    external( "__imp__GetWindowOrgEx@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoint

[out] Pointer to a POINT structure that receives the coordinates, in logical units, of the window origin.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportOrgEx, SetViewportOrgEx, SetWindowOrgEx

---

2.178 GetWorldTransform

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

```
GetWorldTransform: procedure
(
        hdc             :dword;
    var lpXform         :XFORM
);
    @stdcall;
    returns( "eax" );
    external( "__imp__GetWorldTransform@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpXform

[out] Pointer to an XFORM structure that receives the current world-space to page-space transformation.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, ModifyWorldTransform, SetWorldTransform

---

2.179 IntersectClipRect

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

```
IntersectClipRect: procedure
(
    hdc                 :dword;
    nLeftRect           :dword;
    nTopRect            :dword;
    nRightRect          :dword;
    nBottomRect         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__IntersectClipRect@20" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the logical x-coordinate of the upper-left corner of the rectangle.

nTopRect

[in] Specifies the logical y-coordinate of the upper-left corner of the rectangle.

nRightRect

[in] Specifies the logical x-coordinate of the lower-right corner of the rectangle.

nBottomRect

[in] Specifies the logical y-coordinate of the lower-right corner of the rectangle.

Return Values

The return value specifies the new clipping region's type and can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The current clipping region is unaffected.)

Remarks

The lower and right-most edges of the given rectangle are excluded from the clipping region.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Clipping Overview, Clipping Functions, ExcludeClipRect

2.180 InvertRgn

The InvertRgn function inverts the colors in the specified region.

```
InvertRgn: procedure
(
    hdc                 :dword;
    hrgn                :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__InvertRgn@8" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region for which colors are inverted. The region's coordinates are presumed to be logical coordinates.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gid32.hff.

See Also

Regions Overview, Region Functions, FillRgn, PaintRgn

2.181 LPtoDP

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

```
LPtoDP: procedure
(
        hdc             :dword;
    var lpPoints        :POINT;
        nCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__LPtoDP@12" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoints

[in/out] Pointer to an array of POINT structures. The x-coordinates and y-coordinates contained in each of the POINT structures will be transformed.

nCount

[in] Specifies the number of points in the array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

This 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, DPtoLP, POINT

2.182 LineTo

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

```
LineTo: procedure
(
    hdc             :dword;
    nXEnd           :dword;
    nYEnd           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__LineTo@12" );
```

Parameters

hdc

[in] Handle to a device context.

nXEnd

[in] Specifies the x-coordinate of the line's ending point.

nYEnd

[in] Specifies the y-coordinate of the line's ending point.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The coordinates of the line's ending point are specified in logical units.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, MoveToEx, Polyline, PolylineTo

---

2.183 MaskBlt

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

```
MaskBlt: procedure
(
    hdcDest         :dword;
    nXDest          :dword;
    nYDest          :dword;
    nWidth          :dword;
    hHeight         :dword;
    hdcSrc          :dword;
    nXSrc           :dword;
    nYSrc           :dword;
    hbmMask         :dword;
    xMask           :dword;
    yMask           :dword;
    dwRop           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__MaskBlt@48" );
```

Parameters

hdcDest

[in] Handle to the destination device context.

nXDest

[in] Specifies the logical x-coordinate of the upper-left corner of the destination rectangle.

nYDest

[in] Specifies the logical y-coordinate of the upper-left corner of the destination rectangle.

nWidth

[in] Specifies the width, in logical units, of the destination rectangle and source bitmap.

nHeight

[in] Specifies the height, in logical units, of the destination rectangle and source bitmap.

hdcSrc

[in] 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.

nXSrc

[in] Specifies the logical x-coordinate of the upper-left corner of the source bitmap.

nYSrc

[in] Specifies the logical y-coordinate of the upper-left corner of the source bitmap.

hbmMask

[in] Handle to the monochrome mask bitmap combined with the color bitmap in the source device context.

xMask

[in] Specifies the horizontal pixel offset for the mask bitmap specified by the hbmMask parameter.

yMask

[in] Specifies the vertical pixel offset for the mask bitmap specified by the hbmMask parameter.

dwRop

[in] Specifies both foreground and background ternary raster operation codes 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, see the BitBlt function.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

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 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.

Windows 98, Windows 2000: When used in a multimonitor system, both hdcSrc and hdcDest must refer to the same device or the function will fail.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BitBlt, GetDeviceCaps, PlgBlt, StretchBlt

---

2.184 ModifyWorldTransform

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

```
ModifyWorldTransform: procedure
(
        hdc         :dword;
    var lpXform     :XFORM;
        iMode       :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__ModifyWorldTransform@12" );
```

Parameters

hdc

[in] Handle to the device context.

lpXform

[in] Pointer to an XFORM structure used to modify the world transformation for the given device context.

iMode

[in] Specifies how the transformation data modifies the current world transformation. This parameter must be one of the following values.

Value	Description
MWT_IDENTITY	Resets the current world transformation by using the identity matrix. If this mode is specified, the XFORM structure pointed to by lpXform 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWorldTransform, SetWorldTransform, SetGraphicsMode, XFORM

2.185 MoveToEx

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

```
MoveToEx: procedure
(
        hdc         :dword;
        X           :dword;
        Y           :dword;
    var lpPoint     :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__MoveToEx@16" );
```

Parameters

hdc

[in] Handle to a device context.

X

[in] Specifies the x-coordinate of the new position, in logical units.

Y

[in] Specifies the y-coordinate of the new position, in logical units.

lpPoint

[out] 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The MoveToEx function affects all drawing functions.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, AngleArc, LineTo, POINT, PolyBezierTo, PolylineTo

---

2.186 OffsetClipRgn

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

```
OffsetClipRgn: procedure
(
    hdc             :dword;
    nXOffset        :dword;
    nYOffset        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__OffsetClipRgn@12" );
```

Parameters

hdc

[in] Handle to the device context.

nXOffset

[in] Specifies the number of logical units to move left or right.

nYOffset

[in] Specifies the number of logical units to move up or down.

Return Values

The return value specifies the new region's complexity and can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The current clipping region is unaffected.)

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, SelectClipRgn

2.187 OffsetRgn

The OffsetRgn function moves a region by the specified offsets.

```
OffsetRgn: procedure
(
    hrgn            :dword;
    nXOffset        :dword;
    nYOffset        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__OffsetRgn@12" );
```

Parameters

hrgn

[in] Handle to the region to be moved.

nXOffset

[in] Specifies the number of logical units to move left or right.

nYOffset

[in] Specifies the number of logical units to move up or down.

Return Values

The return value specifies the new region's complexity. It can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred; region is unaffected.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions

2.188 OffsetViewportOrgEx

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

```
OffsetWindowOrgEx: procedure
(
        hdc             :dword;
        nXOffset        :dword;
        nYOffset        :dword;
    var lpPoint         :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__OffsetWindowOrgEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

nXOffset

[in] Specifies the horizontal offset, in device units.

nYOffset

[in] Specifies the vertical offset, in device units.

lpPoint

[out] 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 Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportOrgEx, OffsetWindowOrgEx, SetViewportOrgEx

---

2.189 PaintRgn

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

```
PaintRgn: procedure
(
    hdc                 :dword;
    hrgn                :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PaintRgn@8" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region to be filled. The region's coordinates are presumed to be logical coordinates.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, FillRgn

2.190 PatBlt

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

```
PatBlt: procedure
(
    hdc                 :dword;
    nXLeft              :dword;
    nYLeft              :dword;
    nWidth              :dword;
    nHeight             :dword;
    dwRop               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PatBlt@24" );
```

Parameters

hdc

[in] Handle to the device context.

nXLeft

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

nYLeft

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the rectangle to be filled.

nWidth

[in] Specifies the width, in logical units, of the rectangle.

nHeight

[in] Specifies the height, in logical units, of the rectangle.

dwRop

[in] Specifies 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, GetDeviceCaps

2.191 PathToRegion

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

```
PathToRegion: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PathToRegion@4" );
```

Parameters

hdc

[in] Handle to a device context that contains a closed path.

Return Values

If the function succeeds, the return value identifies a valid region.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

Remarks

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Paths Overview, Path Functions, BeginPath, EndPath, SetPolyFillMode

---

2.192 Pie

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

```
Pie: procedure
(
    hdc                 :dword;
    nLeftRect           :dword;
    nTopRect            :dword;
    nRightRect          :dword;
    nBottomRect         :dword;
    nXRadial1           :dword;
    nYRadial1           :dword;
    nXRadial2           :dword;
    nYRadial2           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__Pie@36" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the bounding rectangle.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the bounding rectangle.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the bounding rectangle.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the bounding rectangle.

nXRadial1

[in] Specifies the x-coordinate of the endpoint of the first radial.

nYRadial1

[in] Specifies the y-coordinate of the endpoint of the first radial.

nXRadial2

[in] Specifies the x-coordinate of the endpoint of the second radial.

nYRadial2

[in] Specifies the y-coordinate of the endpoint of the second radial.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: The sum of the coordinates of the bounding rectangle cannot exceed 32,767. The sum of nLeftRect and nRightRect or nTopRect and nBottomRect parameters cannot exceed 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, AngleArc, Arc, ArcTo, Chord

---

2.193 PlayEnhMetaFile

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

```
PlayEnhMetaFile: procedure
(
        hdc             :dword;
        hemf            :dword;
    var lpRect          :RECT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PlayEnhMetaFile@12" );
```

Parameters

hdc

[in] Handle to the device context for the output device on which the picture will appear.

hemf

[in] Handle to the enhanced metafile.

lpRect

[in] 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: PlayEnhMetaFile is subject to the limitations of the GDI. For example, Windows 95/98 supports only 16-bit signed coordinates. For records that contain 32-bit values, Windows 95/98 fails to play the record if the values are not in the range –32,768 to 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, CancelDC, GetEnhMetaFileHeader, GetEnhMetaFile-

2.194 PlayEnhMetaFileRecord

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

```
PlayEnhMetaFileRecord: procedure
(
        hdc             :dword;
    var lpHandletable   :HANDLETABLE;
    var lpEnhMetaRecord :ENHMETARECORD;
        nHandles        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PlayEnhMetaFileRecord@16" );
```

Parameters

hdc

[in] Handle to the device context passed to the EnumEnhMetaFile function.

lpHandletable

[in] 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.

lpEnhMetaRecord

[in] Pointer to the enhanced-metafile record to be played.

nHandles

[in] Specifies the number of handles in the handle table.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: PlayEnhMetaFileRecord is subject to the limitations of GDI. For example, Windows 95/98 supports only 16-bit signed coordinates. For records that contain 32-bit values, Windows 95/98 fails to play the record if the values are not in the range –32,768 to 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, EnumEnhMetaFile, PlayEnhMetaFile

---

2.195 PlayMetaFile

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 16-bit versions of Windows. Win32-based applications should use the PlayEnhMetaFile function.

```
PlayMetaFile: procedure
(
    hdc                 :dword;
    hmf                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PlayMetaFile@8" );
```

Parameters

hdc

[in] Handle to a device context.

hmf

[in] Handle to a Windows-format metafile.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

A Windows-format metafile does not support the new curve, path, and transformation functions such as PolyBezier, BeginPath, and SetWorldTransform. Applications that use these new functions and use metafiles to store pictures created by these functions should use the enhanced-format metafile functions.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf

See Also

Metafiles Overview, Metafile Functions, BeginPath, CancelDC, PolyBezier, SetWinMetaFileBits, SetWorldTransform

---

2.196 PlayMetaFileRecord

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 16-bit versions of Windows. Win32-based applications should use the PlayEnhMetaFileRecord function.

```
PlayMetaFileRecord: procedure
(
        hdc             :dword;
    var lpHandletable   :HANDLETABLE;
    var lpMetaRecord    :METARECORD;
        nHandles        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PlayMetaFileRecord@16" );
```

Parameters

hdc

[in] Handle to a device context.

lpHandletable

[in] Pointer to a HANDLETABLE structure representing the table of handles to GDI objects used when playing the metafile.

lpMetaRecord

[in] Pointer to the Windows-format metafile record.

nHandles

[in] Specifies the number of handles in the handle table.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

A Windows-format metafile does not support the new curve, path, and transformation functions, such as PolyBezier, BeginPath, and SetWorldTransform. Applications that use these new functions and use metafiles to store pictures created by these functions, should use the enhanced format metafile functions.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, BeginPath, EnumMetaFile, HANDLETABLE, METARECORD, PlayMetaFile, PolyBezier, SetWinMetaFileBits, SetWorldTransform

---

2.197 PlgBlt

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

```
PlgBlt: procedure
(
        hdcDest         :dword;
    var lpPoint         :POINT;
        hdcSrc          :dword;
        nXSrc           :dword;
        nYSrc           :dword;
        nWidth          :dword;
        nHeight         :dword;
        hbmMask         :dword;
        xMask           :dword;
        yMask           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PlgBlt@40" );
```

Parameters

hdcDest

[in] Handle to the destination device context.

lpPoint

[in] 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.

hdcSrc

[in] Handle to the source device context.

nXSrc

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the source rectangle.

nYSrc

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the source rectangle.

nWidth

[in] Specifies the width, in logical units, of the source rectangle.

nHeight

[in] Specifies the height, in logical units, of the source rectangle.

hbmMask

[in] Handle to an optional monochrome bitmap that is used to mask the colors of the source rectangle.

xMask

[in] Specifies the x-coordinate of the upper-left corner of the the monochrome bitmap.

yMask

[in] Specifies the y-coordinate of the upper-left corner of the the monochrome bitmap.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

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 + C – A.

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; how-

ever, 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, PlgBlt converts the source rectangle to match the destination rectangle.

Not all devices support the PlgBlt 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, PlgBlt returns an error.

Windows 98, Windows 2000: When used in a multimonitor system, both hdcSrc and hdcDest must refer to the same device or the function will fail.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BitBlt, GetDeviceCaps, MaskBlt, SetStretchBltMode, StretchBlt

---

2.198 PolyBezier

The PolyBezier function draws one or more Bézier curves.

```
PolyBezier: procedure
(
        hdc             :dword;
    var lppt            :POINT;
        cPoints         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyBezier@12" );
```

Parameters

hdc

[in] Handle to a device context.

lppt

[in] Pointer to an array of POINT structures that contain the endpoints and control points of the curve(s).

cPoints

[in] Specifies 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

Th 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.

Windows 95/98: PolyBezier cannot draw more than a certain number of points. The limit depends on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum points allowed
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but device does not support wideline	approximately 1360 (that is, approximately 16K / 12)

Any extra points are ignored. To draw a line with more points, divide the data into groups that have less than the maximum number of points and call the function for each group of points. Remember to connect the line segments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, MoveToEx, POINT, PolyBezierTo

2.199 PolyBezierTo

The PolyBezierTo function draws one or more Bézier curves.

```
PolyBezierTo: procedure
(
        hdc             :dword;
    var lppt            :POINT;
        cCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyBezierTo@12" );
```

Parameters

hdc

[in] Handle to a device context.

lppt

[in] Pointer to an array of POINT structures that contains the endpoints and control points.

cCount

[in] Specifies 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Windows 95/98: PolyBezierTo cannot draw more than a certain number of points. The limit depends on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum points allowed
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but device does not support wideline	approximately 1360 (that is, approximately 16K / 12)

Any extra points are ignored. To draw a line with more points, divide the data into groups that have less than the maximum number of points and call the function for each group of points. Remember to connect the line segments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Lines and Curves Overview, Line and Curve Functions, MoveToEx, POINT, PolyBezier

---

2.200 PolyDraw

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

```
PolyDraw: procedure
(
        hdc             :dword;
    var lppt            :POINT;
    var lpbTypes        :var;
        cCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyDraw@16" );
```

Parameters

hdc

[in] Handle to a device context.

lppt

[in] 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.

lpbTypes

[in] Pointer to an array that specifies how each point in the lppt array is used. This parameter can be one of the following values.

Type	Meaning
PT_MOVETO	Specifies that this point starts a disjoint figure. This point becomes the new current position.
PT_LINETO	Specifies that a line is to be drawn from the current position to this point, which then becomes the new current position.
PT_BEZIERTO	Specifies that this point is a control point or ending point for a Bézier curve. PT_BEZIERTO types always occur in sets of three. The current position defines the starting point for the Bézier curve. The first two PT_BEZIERTO points are the control points, and the third PT_BEZIERTO point is the ending point. The ending point becomes the new current position. If there are not three consecutive PT_BEZIERTO points, an error results.

A PT_LINETO or PT_BEZIERTO type can be combined with the following value by using the bitwise operator OR to indicate that the corresponding point is the last point in a figure and the figure is closed.

Value	Meaning
PT_CLOSEFIGURE	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. 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. The current position is set to the ending point of the closing line.

cCount

[in] Specifies the total number of points in the lppt array, the same as the number of bytes in the lpbTypes array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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 lppt 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, BeginPath, EndPath, LineTo, MoveToEx, POINT, PolyBezierTo, PolyLine

---

2.201 PolyPolygon

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

```
PolyPolygon: procedure
```

```
(
        hdc             :dword;
    var lpPoints        :POINT;
    var lpPolyCounts    :var;
        nCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyPolygon@16" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoints

[in] Pointer to an array of POINT structures that define the vertices of the polygons. 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.

lpPolyCounts

[in] 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.

nCount

[in] Specifies the total number of polygons.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The current position is neither used nor updated by this function.

Windows 95/98: Polypolygon is limited in the number of points it can draw, depending on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum number of points
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but the device does not support wideline	approximately 1360 (that is, a few less than 16K / 12)

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, GetPolyFillMode, POINT, Polygon, Polyline, PolylineTo, SetPolyFillMode

---

2.202 PolyPolyline

The PolyPolyline function draws multiple series of connected line segments.

```
PolyPolyline: procedure
(
        hdc             :dword;
    var lppt            :POINT;
    var lpdwPolyPoints  :var;
        cCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyPolyline@16" );
```

Parameters

hdc

[in] Handle to the device context.

lppt

[in] Pointer to an array of POINT structures that contains the vertices of the polylines. The polylines are specified consecutively.

lpdwPolyPoints

[in] 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.

cCount

[in] Specifies the total number of entries in the lpdwPolyPoints array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Windows 95/98: PolyPolyline cannot draw more than a certain number of points. The limit depends on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum points allowed
line width is 1	16K

line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but device does not support wideline	approximately 1360 (that is, approximately 16K / 12)

Any extra points are ignored. To draw a line with more points, divide the data into groups that have less than the maximum number of points and call the function for each group of points. Remember to connect the line segments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, POINT, Polyline, PolylineTo

---

2.203 PolyTextOut

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

```
PolyTextOut: procedure
(
        hdc             :dword;
    var pptxt           :POLYTEXT;
        cStrings        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolyTextOutA@12" );
```

Parameters

hdc

[in] Handle to the device context.

pptxt

[in] Pointer to an array of POLYTEXT structures describing the strings to be drawn. The array contains one structure for each string to be drawn.

cStrings

[in] Specifies the number of POLYTEXT structures in the pptxt array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GetTextAlign, POLYTEXT, SetTextAlign

---

2.204 Polygon

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

```
Polygon: procedure
(
        hdc             :dword;
    var lpPoints        :POINT;
        nCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__Polygon@12" );
```

Parameters

hdc

[in] Handle to the device context.

lpPoints

[in] Pointer to an array of POINT structures that specify the vertices of the polygon.

nCount

[in] Specifies the number of vertices in the array. This value must be greater than or equal to 2.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: Polygon is limited in the number of points it can draw, depending on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum number of points
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but the device does not support wideline	approximately 1360 (that is, a few less than 16K / 12)

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, GetPolyFillMode, POINT, Polyline, PolylineTo, PolyPolygon, SetPolyFillMode

---

2.205 Polyline

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

```
Polyline: procedure
(
        hdc             :dword;
    var lppt            :POINT;
        cPoints         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__Polyline@12" );
```

Parameters

hdc

[in] Handle to a device context.

lppt

[in] Pointer to an array of POINT structures. Each structure in the array identifies a point in logical space.

cPoints

[in] Specifies the number of points in the array. This number must be greater than or equal to two.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Windows 95/98: Polyline cannot draw more than a certain number of points. The limit depends on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum points allowed
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but device does not support wideline	approximately 1360 (that is, approximately 16K / 12)

Any extra points are ignored. To draw a line with more points, divide the data into groups that have less than the maximum number of points and call the function for each group of points. Remember to connect the line segments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, LineTo, MoveToEx, POINT, PolylineTo, PolyPolyline

---

2.206 PolylineTo

The PolylineTo function draws one or more straight lines.

```
PolylineTo: procedure
(
        hdc             :dword;
    var lppt            :POINT;
        cCount          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PolylineTo@12" );
```

Parameters

hdc

[in] Handle to the device context.

lppt

[in] Pointer to an array of POINT structures that contains the vertices of the line.

cCount

[in] Specifies the number of points in the array.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Windows 95/98: PolylineTo cannot draw more than a certain number of points. The limit depends on the line width (that is, the width of the pen selected into the DC), as shown in the following table.

Line width	Maximum points allowed
line width is 1	16K
line width > 1 (that is, wideline) and device supports wideline	16K
line width > 1 but device does not support wideline	approximately 1360 (that is, approximately 16K / 12)

Any extra points are ignored. To draw a line with more points, divide the data into groups that have less than the maximum number of points and call the function for each group of points. Remember to connect the line segments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions, LineTo, MoveToEx, POINT, Polyline, PolyPolyline

2.207 PtInRegion

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

```
PtInRegion: procedure
(
    hrgn                :dword;
    X                   :dword;
    Y                   :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__PtInRegion@12" );
```

Parameters

hrgn

[in] Handle to the region to be examined.

X

[in] Specifies the x-coordinate of the point in logical units.

Y

[in] Specifies the y-coordinate of the point in logical units.

Return Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, RectInRegion

2.208 PtVisible

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

```
PtVisible: procedure
(
    hdc                 :dword;
    X                   :dword;
    Y                   :dword
);
    @stdcall;
    returns( "eax" );
```

```
    external( "__imp__PtVisible@12" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the logical x-coordinate of the point.

Y

[in] Specifies the logical y-coordinate of the point.

Return Values

If the specified point is within the clipping region of the device context, the return value is non-zero.

If the specified point is not within the clipping region of the device context, the return value is zero.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, RectVisible

2.209 RealizePalette

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

```
RealizePalette: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RealizePalette@4" );
```

Parameters

hdc

[in] Handle to the device context into which a logical palette has been selected.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, CreatePalette, GetDeviceCaps, SelectPalette

---

2.210 RectInRegion

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

```
RectInRegion: procedure
(
        hrgn            :dword;
    var lprc            :RECT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RectInRegion@8" );
```

Parameters

hrgn

[in] Handle to the region.

lprc

[in] 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 Values

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, PtInRegion, RECT

2.211 RectVisible

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

```
RectVisible: procedure
(
        hdc             :dword;
    var lprc            :RECT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RectVisible@8" );
```

Parameters

hdc

[in] Handle to the device context.

lprc

[in] Pointer to a RECT structure that contains the logical coordinates of the specified rectangle.

Return Values

If some portion of the specified rectangle lies within the clipping region, the return value is nonzero.

If no portion of the specified rectangle lies within the clipping region, the return value is zero.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, CreateRectRgn, PtVisible, RECT, SelectClipRgn

---

2.212 Rectangle

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

```
Rectangle: procedure
(
        hdc             :dword;
        nLeftRect       :dword;
        nTopRect        :dword;
        nRightRect      :dword;
        nBottomRect     :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__Rectangle@20" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the logical x-coordinate of the upper-left corner of the rectangle.

nTopRect

[in] Specifies the logical y-coordinate of the upper-left corner of the rectangle.

nRightRect

[in] Specifies the logical x-coordinate of the lower-right corner of the rectangle.

nBottomRect

[in] Specifies the logical y-coordinate of the lower-right corner of the rectangle.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, RoundRect

2.213 RemoveFontMemResourceEx

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

```
RemoveFontMemResourceEx: procedure
(
    fh                  :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RemoveFontMemResourceEx@4" );
```

Parameters

fh

[in] Handle to the font-resource. This handle is returned by the AddFontMemResourceEx function.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

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

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, AddFontMemResourceEx, SendMessage

2.214 RemoveFontResource

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.

```
RemoveFontResource: procedure
(
    lpFileName          :string
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RemoveFontResourceA@4" );
```

Parameters

lpFileName

[in] Pointer to a null-terminated string that names a font resource file.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

An 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 system. The application 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Fonts and Text Overview, Font and Text Functions, AddFontResource, RemoveFontResourceEx, SendMessage

2.215 RemoveFontResourceEx

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

```
RemoveFontResourceEx: procedure
(
        lpFileName      :string;
        fl              :dword;
    var pdf             :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RemoveFontResourceExA@12" );
```

Parameters

lpFileName

[in] Pointer to a null-terminated string that names a font resource file.

fl

[in] Specifies 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.

pdv

[in] Reserved. It must be 0.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This function will only remove the font if the flags specified are the same as when then font was added with the AddFontResourceEx function.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, AddFontResourceEx, SendMessage

---

2.216 ResetDC

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

```
ResetDC: procedure
(
        hdc             :dword;
    var lpInitData      :DEVMODE
);
    @stdcall;
```

```
    returns( "eax" );
    external( "__imp__ResetDCA@8" );
```

Parameters

hdc

[in] Handle to the DC to update.

lpInitData

[in] Pointer to a DEVMODE structure containing information about the new DC.

Return Values

If the function succeeds, the return value is a handle to the original DC.

If the function fails, the return value is NULL.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

An application will typically use the ResetDC function when a window receives a WM_DEVMODECHANGE message. ResetDC can also be used to change the paper orientation or paper bins while printing a document.

The ResetDC function cannot be used to change the driver name, device name, or the output port. When the user changes the port connection or device name, the application must delete the original DC and create a new DC with the new information.

An application can pass an information DC to the ResetDC function. In that situation, ResetDC will always return a printer DC.

ICM: The color profile of the DC specified by the hdc parameter will be reset based on the information contained in the lpInitData member of the DEVMODE structure.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, DeviceCapabilities, DEVMODE, Escape

---

2.217 ResizePalette

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

```
ResizePalette: procedure
(
    hpal                :dword;
    nEntries            :dword
);
    @stdcall;
```

```
    returns( "eax" );
    external( "__imp__ResizePalette@8" );
```

Parameters

hpal

[in] Handle to the palette to be changed.

nEntries

[in] Specifies the number of entries in the palette after it has been resized. Windows NT/ 2000: the number of entries is limited to 1024.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps

2.218 RestoreDC

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

```
RestoreDC: procedure
(
    hdc                 :dword;
    nSavedDC            :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RestoreDC@8" );
```

Parameters

hdc

[in] Handle to the DC.

nSavedDC

[in] Specifies 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The stack can contain the state information for several instances of the DC. If the state specified by the specified parameter is not at the top of the stack, RestoreDC deletes all state information between the top of the stack and the specified instance.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, SaveDC

2.219 RoundRect

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

```
RoundRect: procedure
(
    hdc                 :dword;
    nLeftRect           :dword;
    nTopRect            :dword;
    nRightRect          :dword;
    nBottomRect         :dword;
    nWidth              :dword;
    nHeight             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__RoundRect@28" );
```

Parameters

hdc

[in] Handle to the device context.

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the rectangle.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the rectangle.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the rectangle.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the rectangle.

nWidth

[in] Specifies the width of the ellipse used to draw the rounded corners.

nHeight

[in] Specifies the height of the ellipse used to draw the rounded corners.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The current position is neither used nor updated by this function.

Windows 95/98: The sum of the coordinates of the bounding rectangle cannot exceed 32,767. The sum of nLeftRect and nRightRect or nTopRect and nBottomRect parameters cannot exceed 32,767.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Filled Shapes Overview, Filled Shape Functions, Rectangle

2.220 SaveDC

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

```
SaveDC: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SaveDC@4" );
```

Parameters

hdc

[in] Handle to the DC whose state is to be saved.

Return Values

If the function succeeds, the return value identifies the saved state.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, RestoreDC

2.221 ScaleViewportExtEx

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

```
ScaleViewportExtEx: procedure
(
        hdc             :dword;
        Xnum            :dword;
        Xdenom          :dword;
        Ynum            :dword;
        Ydenom          :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__ScaleViewportExtEx@24" );
```

Parameters

hdc

[in] Handle to the device context.

Xnum

[in] Specifies the amount by which to multiply the current horizontal extent.

Xdenom

[in] Specifies the amount by which to divide the current horizontal extent.

Ynum

[in] Specifies the amount by which to multiply the current vertical extent.

Ydenom

[in] Specifies the amount by which to divide the current vertical extent.

lpSize

[out] Pointer to a SIZE structure that receives the previous viewport extents, in device units. If lpSize is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The viewport extents are modified as follows:

xNewVE = (xOldVE * Xnum) / Xdenom

yNewVE = (yOldVE * Ynum) / Ydenom

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportExtEx, SIZE

2.222 ScaleWindowExtEx

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

```
ScaleWindowExtEx: procedure
(
        hdc             :dword;
```

```
        Xnum            :dword;
        Xdenom          :dword;
        Ynum            :dword;
        Ydenom          :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__ScaleWindowExtEx@24" );
```

Parameters

hdc

[in] Handle to the device context.

Xnum

[in] Specifies the amount by which to multiply the current horizontal extent.

Xdenom

[in] Specifies the amount by which to divide the current horizontal extent.

Ynum

[in] Specifies the amount by which to multiply the current vertical extent.

Ydenom

[in] Specifies the amount by which to divide the current vertical extent.

lpSize

[out] Pointer to a SIZE structure that receives the previous window extents, in logical units. If lpSize is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The window extents are modified as follows:

xNewWE = (xOldWE * Xnum) / Xdenom

yNewWE = (yOldWE * Ynum) / Ydenom

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWindowExtEx, SIZE

2.223 SelectClipPath

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

```
SelectClipPath: procedure
(
    hdc                 :dword;
    iMode               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SelectClipPath@8" );
```

Parameters

hdc

[in] Handle to the device context of the path.

iMode

[in] Specifies the way to use the path. This parameter can be one of the following values.

Value	Meaning
RGN_AND	The new clipping region includes the intersection (overlapping areas) of the current clipping region and the current path.
RGN_COPY	The new clipping region is the current path.
RGN_DIFF	The new clipping region includes the areas of the current clipping region with those of the current path excluded.
RGN_OR	The new clipping region includes the union (combined areas) of the current clipping region and the current path.
RGN_XOR	The new clipping region includes the union of the current clipping region and the current path but without the overlapping areas.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

Remarks

The device context identified by the hdc parameter must contain a closed path.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, BeginPath, EndPath

2.224 SelectClipRgn

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

```
SelectClipRgn: procedure
(
    hdc                 :dword;
    hrgn                :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SelectClipRgn@8" );
```

Parameters

hdc

[in] Handle to the device context.

hrgn

[in] Handle to the region to be selected.

Return Values

The return value specifies the region's complexity and can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.
COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (The previous clipping region is unaffected.)

Windows NT/Windows 2000: To get extended error Information, call GetLastError.

Remarks

Only a copy of the selected region is used. The region itself can be selected for any number of other device contexts or it can be deleted.

The SelectClipRgn function assumes that the coordinates for a region are specified in device units.

To remove a device-context's clipping region, specify a NULL region handle.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, ExtSelectClipRgn

2.225 SelectObject

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

```
SelectObject: procedure
(
    hdc                 :dword;
    hgdiobj             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SelectObject@8" );
```

Parameters

hdc

[in] Handle to the DC.

hgdiobj

[in] Handle to the object to be selected. The specified object must have been created by using one of the following functions.

Object	Functions
Bitmap	CreateBitmap, **CreateBitmapIndirect**, **CreateCompatibleBitmap**, **CreateDIBitmap**, **CreateDIBSection** (Bitmaps can be selected for memory DCs only, and for only one DC at a time.)
Brush	**CreateBrushIndirect**, **CreateDIBPatternBrush**, **CreateDIBPatternBrushPt**, **CreateHatchBrush**, **CreatePatternBrush**, **CreateSolidBrush**
Font	**CreateFont**, **CreateFontIndirect**
Pen	**CreatePen**, **CreatePenIndirect**
Region	**CombineRgn**, **CreateEllipticRgn**, **CreateEllipticRgnIndirect**, **CreatePolygonRgn**, **CreateRectRgn**, **CreateRectRgnIndirect**

Return Values

If the selected object is not a region and the function succeeds, the return value is a handle to the object being replaced. If the selected object is a region and the function succeeds, the return value is one of the following values.

Value	Meaning
SIMPLEREGION	Region consists of a single rectangle.
COMPLEXREGION	Region consists of more than one rectangle.
NULLREGION	Region is empty.

If an error occurs and the selected object is not a region, the return value is NULL. Otherwise, it is GDI_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 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, CombineRgn, CreateBitmap, CreateBitmapIndirect, CreateBrushIndirect, CreateCompatibleBitmap, CreateDIBitmap, CreateDIBPatternBrush, CreateEllipticRgn, CreateEllipticRgnIndirect, CreateFont, CreateFontIndirect, CreateHatchBrush, CreatePatternBrush, CreatePen, CreatePenIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateSolidBrush, SelectClipRgn, SelectPalette

---

2.226

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

```
SelectPalette: procedure
(
    hdc                 :dword;
    hpal                :dword;
    bForceBackground    :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SelectPalette@12" );
```

Parameters

hdc

[in] Handle to the device context.

hpal

[in] Handle to the logical palette to be selected.

bForceBackground

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, CreateCompatibleDC, CreatePalette, GetDeviceCaps, RealizePalette

---

2.227 SetArcDirection

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

```
SetArcDirection: procedure
(
    hdc                 :dword;
    ArcDirection        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetArcDirection@8" );
```

Parameters

hdc

[in] Handle to the device context.

ArcDirection

[in] Specifies the new arc direction. This parameter can be one of the following values.

Value	Meaning
AD_COUNTERCLOCKWISE	Figures drawn counterclockwise.
AD_CLOCKWISE	Figures drawn clockwise.

Return Values

If the function succeeds, the return value specifies the old arc direction.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 98 or later.

Header: Declared in gdi32.hhf.

See Also

Lines and Curves Overview, Line and Curve Functions

2.228 SetBitmapBits

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. Win32-based applications should use the SetDIBits function.

```
SetBitmapBits: procedure
(
        hbmp            :dword;
        cBytes          :dword;
    var lpBits          :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetBitmapBits@12" );
```

Parameters

hbmp

[in] Handle to the bitmap to be set.

cBytes

[in] Specifies the number of bytes pointed to by the lpBits parameter.

lpBits

[in] Pointer to an array of bytes that contain color data for the specified bitmap.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The array identified by lpBits must be WORD aligned.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, GetBitmapBits, SetDIBits

2.229 SetBitmapDimensionEx

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

```
SetBitmapDimensionEx: procedure
(
        hBitmap         :dword;
        nWidth          :dword;
        nHeight         :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetBitmapDimensionEx@16" );
```

Parameters

hBitmap

[in] Handle to the bitmap. The bitmap cannot be a DIB-section bitmap.

nWidth

[in] Specifies the width, in 0.1-millimeter units, of the bitmap.

nHeight

[in] Specifies the height, in 0.1-millimeter units, of the bitmap.

lpSize

[out] Pointer to a SIZE structure to receive the previous dimensions of the bitmap. This pointer can be NULL.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, CreateDIBSection, GetBitmapDimensionEx, SIZE

2.230 SetBkColor

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

```
SetBkColor: procedure
(
    hdc                 :dword;
    crColor             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetBkColor@8" );
```

Parameters

hdc

[in] Handle to the device context.

crColor

[in] Specifies the new background color. To make a COLORREF value, use the RGB macro.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, COLORREF, CreatePen, ExtCreatePen, GetBKColor, GetBkMode, SetBkMode

---

2.231 SetBkMode

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

```
SetBkMode: procedure
(
    hdc                 :dword;
    iBkMode             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetBkMode@8" );
```

Parameters

hdc

[in] Handle to the device context.

iBkMode

[in] Specifies the background mode. This parameter can be one of the following values.

Value	Description
OPAQUE	Background is filled with the current background color before the text, hatched brush, or pen is drawn.
TRANSPARENT	Background remains untouched.

Return Values

If the function succeeds, the return value specifies the previous background mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

The iBkMode parameter can also be set to driver-specific values. GDI passes such values to the device driver and otherwise ignores them.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

Library: Use Gdi32.lib.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, CreatePen, ExtCreatePen, GetBkMode

---

2.232 SetBoundsRect

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.

```
SetBoundsRect: procedure
(
        hdc             :dword;
    var lprcBounds      :RECT;
        flags           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetBoundsRect@12" );
```

Parameters

hdc

[in] Handle to the device context for which to accumulate bounding rectangles.

lprcBounds

[in] Pointer to a RECT structure used to set the bounding rectangle. Rectangle dimensions are in logical coordinates. This parameter can be NULL.

flags

[in] Specifies how the new rectangle will be combined with the accumulated rectangle. This parameter can be one of more of the following values.

Value	Description

DCB_ACCUMULATE	Adds the rectangle specified by the lprcBounds parameter to the bounding rectangle (using a rectangle union operation). Using both DCB_RESET and DCB_ACCUMULATE sets the bounding rectangle to the rectangle specified by the lprcBounds 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 Values

If the function succeeds, the return value specifies the previous state of the bounding rectangle. This state can be a combination of the following values.

Value	Meaning
DCB_DISABLE	Boundary accumulation is off.
DCB_ENABLE	Boundary accumulation is on. DCB_ENABLE and DCB_DISABLE are mutually exclusive.
DCB_RESET	Bounding rectangle is empty.
DCB_SET	Bounding rectangle is not empty. DCB_SET and DCB_RESET are mutually exclusive.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GetBoundsRect, RECT

2.233 SetBrushOrgEx

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

```
SetBrushOrgEx: procedure
(
        hdc             :dword;
        nXOrg           :dword;
        nYOrg           :dword;
    var lppt            :POINT
);
    @stdcall;
    returns( "eax" );
```

```
external( "__imp__SetBrushOrgEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

nXOrg

[in] Specifies 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 mod brush width).

nYOrg

[in] Specifies 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 mod brush height).

lppt

[out] 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows NT/ 2000: 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.

Windows 95/98: Automatic tracking of the brush origin is not supported. Applications must use the UnrealizeObject, SetBrushOrgEx, and SelectObject functions to align the brush before using it.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Brushes Overview, Brush Functions, GetBrushOrgEx, POINT, SelectObject, SetStretchBltMode, UnrealizeObject

---

2.234 SetColorAdjustment

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

```
SetColorAdjustment: procedure
(
        hdc             :dword;
    var lpca            :COLORADJUSTMENT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetColorAdjustment@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpca

[in] Pointer to a COLORADJUSTMENT structure containing the color adjustment values.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetColorAdjustment, StretchBlt, StretchDIBits, SetStretch-BltMode, COLORADJUSTMENT

2.235 SetDCBrushColor

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

```
SetDCBrushColor: procedure
(
    hdc                 :dword;
    crColor             :COLORREF
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetDCBrushColor@8" );
```

Parameters

hdc

[in] Handle to the DC.

crColor

[in] Specifies the new brush color.

Return Values

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.

Example

For an example of setting colors, see Setting the Pen or Brush Color.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, GetDCBrushColor, COLORREF

2.236 SetDCPenColor

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.

```
SetDCPenColor: procedure
(
        hdc             :dword;
        crColor         :COLORREF
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetDCPenColor@8" );
```

Parameters

hdc

[in] Handle to the DC.

crColor

[in] Specifies the new pen color.

Return Values

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.

Example

For an example of setting colors, see Setting the Pen or Brush Color.

Requirements

Windows NT/2000: Requires Windows 2000 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Device Contexts Overview, Device Context Functions, GetDCPenColor, COLORREF

2.237 SetDIBColorTable

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

```
SetDIBColorTable: procedure
(
        hdc             :dword;
    uStartIndex         :dword;
    cEntries            :dword;
    pColors             :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetDIBColorTable@16" );
```

Parameters

hdc

[in] Specifies a device context. A DIB must be selected into this device context.

uStartIndex

[in] A zero-based color table index that specifies the first color table entry to set.

cEntries

[in] Specifies the number of color table entries to set.

pColors

[in] Pointer to an array of RGBQUAD structures containing new color information for the DIB's color table.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Note A bitmap information header structure may be one of the following.

Operating system	Bitmap information header
Windows NT 3.51 and earlier	BITMAPINFOHEADER
Windows NT 4.0 and Windows 95	**BITMAPV4HEADER**
Windows 2000 and Windows 98	**BITMAPV5HEADER**

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.

Windows NT 4.0 and Windows 95:The bV4BitCount member of a bitmap's associated BITMAPV4HEADER structure specifies the number of bits-per-pixel. Device-independent bitmaps with a bV4BitCount value greater than 8 do not have a color table.

Windows 2000 and Windows 98: 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

Windows NT/2000: Requires Windows NT 3.5 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFOHEADER, CreateDIBSection, GetDIBColorTable, GetObject, DIBSECTION, RGBQUAD

2.238 SetDIBits

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

```
SetDIBits: procedure
(
        hdc             :dword;
        hbmp            :dword;
        uStartScan      :dword;
        cScanLines      :dword;
    var lpvBits         :var;
    var lpbmi           :BITMAPINFO;
        fuColorUse      :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetDIBits@28" );
```

Parameters

hdc

[in] Handle to a device context.

hbmp

[in] Handle to the bitmap that is to be altered using the color data from the specified DIB.

uStartScan

[in] Specifies the starting scan line for the device-independent color data in the array pointed to by the lpvBits parameter.

cScanLines

[in] Specifies the number of scan lines found in the array containing device-independent color data.

lpvBits

[in] 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.

lpbmi

[in] Pointer to a BITMAPINFO structure that contains information about the DIB.

fuColorUse

[in] Specifies whether the bmiColors member of the BITMAPINFO structure was provided and, if so, whether bmiColors contains explicit red, green, blue (RGB) values or palette indexes. The fuColorUse parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the logical palette of the device context identified by the hdc parameter.
DIB_RGB_COLORS	The color table is provided and contains literal RGB values.

Return Values

If the function succeeds, the return value is the number of scan lines copied.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError. This can be the following value.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more input parameters are 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. The color profile of the current device context is used as the source color space profile and the sRGB color space is used.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, GetDIBits, GetSystemPaletteEntries, BITMAPINFO

2.239 SetDIBitsToDevice

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

Windows 98 and Windows 2000: SetDIBitsToDevice has been extended to allow a JPEG or PNG image to be passed as the source image.

```
SetDIBitsToDevice: procedure
(
        hdc         :dword;
        XDest       :dword;
        YDest       :dword;
        dwWidth     :dword;
        dwHeight    :dword;
        XSrc        :dword;
        YSrc        :dword;
        uStartScan  :dword;
        cScanLines  :dword;
    var lpvBits     :var;
    var lpbmi       :BITMAPINFO;
        fuColorUse  :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetDIBitsToDevice@48" );
```

Parameters

hdc

[in] Handle to the device context.

XDest

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

YDest

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

dwWidth

[in] Specifies the width, in logical units, of the DIB.

dwHeight

[in] Specifies the height, in logical units, of the DIB.

XSrc

[in] Specifies the x-coordinate, in logical units, of the lower-left corner of the DIB.

YSrc

[in] Specifies the y-coordinate, in logical units, of the lower-left corner of the DIB.

uStartScan

[in] Specifies the starting scan line in the DIB.

cScanLines

[in] Specifies the number of DIB scan lines contained in the array pointed to by the lpvBits

parameter.

lpvBits

[in] Pointer to DIB color data stored as an array of bytes. For more information, see the following Remarks section.

lpbmi

[in] Pointer to a BITMAPINFO structure that contains information about the DIB.

fuColorUse

[in] Specifies whether the bmiColors member of the BITMAPINFO structure contains explicit red, green, blue (RGB) values or indexes into a palette. For more information, see the following Remarks section.

The fuColorUse parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	The color table consists of an array of 16-bit indexes into the currently selected logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.

Return Values

If the function succeeds, the return value is the number of scan lines set.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Windows 98 and Windows 2000: 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 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.

Windows 98, Windows 2000:

- 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. The color profile of the current device context is used as the source color space profile. The sRGB color space is used.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFO, GetSystemPaletteEntries, SetDIBits, StretchDIBits

---

2.240 SetEnhMetaFileBits

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

```
SetEnhMetaFileBits: procedure
(
        cbBuffer        :dword;
    var lpData          :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetEnhMetaFileBits@8" );
```

Parameters

cbBuffer

[in] Specifies the size, in bytes, of the data provided.

lpData

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in Wingdi.h; include Windows.h.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile, GetEnhMetaFileBits, SetWinMetaFileBits

---

2.241 SetGraphicsMode

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

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

```
SetGraphicsMode: procedure
(
    hdc                 :dword;
    iMode               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetGraphicsMode@8" );
```

Parameters

hdc

[in] Handle to the device context.

iMode

[in] Specifies the graphics mode. This parameter can be one of the following values.

Value	Meaning
GM_COMPATIBLE	Sets the graphics mode that is compatible with 16-bit Windows. This is the default mode. If this value is specified, the application can only modify the world-to-device transform by calling functions that set window and viewport extents and origins, but not by using SetWorldTransform or ModifyWorldTransform; calls to those functions will fail. Examples of functions that set window and viewport extents and origins are SetViewportExtEx and SetWindowExtEx.

GM_ADVANCED — Windows NT/ 2000: 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.

Windows 95/98 :The GM_ADVANCED value is not supported. When playing enhanced metafiles, Windows 95/98 attempts to make enhanced metafiles on Windows 95/98 look the same as they do on Windows NT/Windows 2000. To accomplish this, Windows 95/98 may simulate GM_ADVANCED mode when playing specific enhanced metafile records.

Return Values

If the function succeeds, the return value is the old graphics mode.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

There are three areas in which graphics output differs according to the graphics mode:

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.

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 bottom-right inclusive.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, CreateDC, GetArcDirection, GetDC, GetGraphicsMode, ModifyWorldTransform, SetArcDirection, SetViewportExtent, SetViewportExtEx, SetWindowExtent, SetWindowExtEx, SetWorldTransform

---

2.242 SetMapMode

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

```
SetMapMode: procedure
(
    hdc             :dword;
    fnMapMode       :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetMapMode@8" );
```

Parameters

hdc

[in] Handle to the device context.

fnMapMode

[in] Specifies the new mapping mode. This parameter can be one of the following values.

Value	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.
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 (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 Values

If the function succeeds, the return value identifies the previous mapping mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetMapMode, SetViewportExtEx, SetViewportOrgEx, SetWindowExtEx, SetWindowOrgEx

---

2.243 SetMapperFlags

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

```
SetMapperFlags: procedure
(
    hdc                 :dword;
    dwFlag              :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetMapperFlags@8" );
```

Parameters

hdc

[in] Handle to the device context that contains the font-mapper flag.

dwFlag

[in] Specifies whether the font mapper should attempt to match a font's aspect ratio to the cur-

rent device's aspect ratio. If bit zero is set, the mapper selects only matching fonts.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetAspectRatioFilterEx

---

2.244 SetMetaFileBitsEx

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

Note This function is provided only for compatibility with 16-bit versions of Windows. Win32-based applications should use the SetEnhMetaFileBits function.

```
SetMetaFileBitsEx: procedure
(
        nSize               :dword;
    var lpData              :var
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetMetaFileBitsEx@8" );
```

Parameters

nSize

[in] Specifies the size, in bytes, of the Windows-format metafile.

lpData

[in] Pointer to a buffer that contains the Windows-format metafile. (It is assumed that the data was obtained by using the GetMetaFileBitsEx function.)

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

A Windows-format metafile does not support the new curve, path, and transformation functions such as PolyBezier, BeginPath, and SetWorldTransform. Applications that use these new functions and use metafiles to store pictures created by these functions should use the enhanced-format metafile functions.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Metafiles Overview, Metafile Functions, BeginPath, DeleteMetaFile, GetMetaFileBitsEx, PolyBezier, SetEnhMetaFileBits, SetWinMetaFileBits, SetWorldTransform

---

2.245 SetMetaRgn

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

```
SetMetaRgn: procedure
(
    hdc                     :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetMetaRgn@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

The return value specifies the new clipping region's complexity and can be one of the following values.

Value	Meaning
NULLREGION	Region is empty.
SIMPLEREGION	Region is a single rectangle.

COMPLEXREGION	Region is more than one rectangle.
ERROR	An error occurred. (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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Clipping Overview, Clipping Functions, GetMetaRgn, SaveDC

2.246 SetMiterLimit

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

```
SetMiterLimit: procedure
(
        hdc             :dword;
        eNewLimit       :real32;
    var peLoldLimit     :real32
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetMiterLimit@12" );
```

Parameters

hdc

[in] Handle to the device context.

eNewLimit

[in] Specifies the new miter limit for the device context.

peOldLimit

[out] 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 Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, ExtCreatePen, GetMiterLimit

2.247 SetPaletteEntries

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

```
UINT SetPaletteEntries(
  HPALETTE hpal,             // handle to logical palette
  UINT iStart,               // index of first entry to set
  UINT cEntries,             // number of entries to set
  CONST PALETTEENTRY *lppe   // array of palette entries
);
```

Parameters

hpal

[in] Handle to the logical palette.

iStart

[in] Specifies the first logical-palette entry to be set.

cEntries

[in] Specifies the number of logical-palette entries to be set.

lppe

[in] Pointer to the first member of an array of PALETTEENTRY structures containing the RGB values and flags.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

An application can determine whether or not 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetPaletteEntries, RealizePalette, PALETTEENTRY

2.248 SetPixel

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

```
SetPixel: procedure
(
    hdc                 :dword;
    X                   :dword;
    Y                   :dword;
    crColor             :COLORREF
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetPixel@16" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the x-coordinate, in logical units, of the point to be set.

Y

[in] Specifies the y-coordinate, in logical units, of the point to be set.

crColor

[in] Specifies the color to be used to paint the point. To create a COLORREF color value, use the RGB macro.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError. This can be the following value.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more input parameters are 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, GetDeviceCaps, GetPixel, SetPixelV, COLORREF, RGB

2.249 SetPixelV

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

```
SetPixelV: procedure
(
    hdc                 :dword;
    X                   :dword;
    Y                   :dword;
    crColor             :COLORREF
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetPixelV@16" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the x-coordinate, in logical units, of the point to be set.

Y

[in] Specifies the y-coordinate, in logical units, of the point to be set.

crColor

[in] Specifies the color to be used to paint the point. To create a COLORREF color value, use the RGB macro.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, GetDeviceCaps, SetPixel, COLORREF, RGB

2.250 SetPolyFillMode

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

```
SetPolyFillMode: procedure
(
    hdc                 :dword;
    iPolyFillMode       :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetPolyFillMode@8" );
```

Parameters

hdc

[in] Handle to the device context.

iPolyFillMode

[in] Specifies the new fill mode. This parameter can be one of the following values.

Value	Meaning
ALTERNATE	Selects alternate mode (fills the area between odd-numbered and even-numbered polygon sides on each scan line).
WINDING	Selects winding mode (fills any region with a nonzero winding value).

Return Values

The return value specifies the previous filling mode. If an error occurs, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, GetPolyFillMode

2.251 SetROP2

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.

```
SetROP2: procedure
(
    hdc                 :dword;
    fnDrawMode          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetROP2@8" );
```

Parameters

hdc

[in] Handle to the device context.

fnDrawMode

[in] Specifies the mix mode. This parameter can be one of the following values.

Mix mode	Description

R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
R2_MERGEPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEPENNOT	Pixel is a combination of the pen color and the inverse of the screen color.
R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the screen color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.

Return Values

If the function succeeds, the return value specifies the previous mix mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Painting and Drawing Overview, Painting and Drawing Functions, GetROP2

2.252 SetRectRgn

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

```
SetRectRgn: procedure
(
```

```
    hrgn                :dword;
    nLeftRect           :dword;
    nTopRect            :dword;
    nRightRect          :dword;
    nBottomRect         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetRectRgn@20" );
```

Parameters

hrgn

[in] Handle to the region.

nLeftRect

[in] Specifies the x-coordinate of the upper-left corner of the rectangular region in logical units.

nTopRect

[in] Specifies the y-coordinate of the upper-left corner of the rectangular region in logical units.

nRightRect

[in] Specifies the x-coordinate of the lower-right corner of the rectangular region in logical units.

nBottomRect

[in] Specifies the y-coordinate of the lower-right corner of the rectangular region in logical units.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError.

Remarks

The region does not include the lower and right boundaries of the rectangle.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Regions Overview, Region Functions, CreateRectRgn

2.253 SetStretchBltMode

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

```
SetStretchBltMode: procedure
(
    hdc                 :dword;
    iStretchMode        :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetStretchBltMode@8" );
```

Parameters

hdc

[in] Handle to the device context.

iStretchMode

[in] Specifies the stretching mode. This parameter can be one of the following values.

Value	Description
BLACKONWHITE	Performs a Boolean AND operation using the color values for the eliminated and existing pixels. If the bitmap is a monochrome bitmap, this mode preserves black pixels at the expense of white pixels.
COLORONCOLOR	Deletes the pixels. This mode deletes all eliminated lines of pixels without trying to preserve their information.
HALFTONE	Maps pixels from the source rectangle into blocks of pixels in the destination rectangle. The average color over the destination block of pixels approximates the color of the source pixels. After setting the HALFTONE stretching mode, an application must call the SetBrushOrgEx function to set the brush origin. If it fails to do so, brush misalignment occurs. This is not supported on Windows 95/98.
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 Values

If the function succeeds, the return value is the previous stretching mode.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError. This can be the following value.

Value	Meaning
ERROR_INVALID_PARAMETER	One or more input parameters are 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, GetStretchBltMode, SetBrushOrgEx, StretchBlt

---

2.254 SetSystemPaletteUse

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.)

```
SetSystemPaletteUse: procedure
(
    hdc                 :dword;
    uUsage              :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetSystemPaletteUse@8" );
```

Parameters

hdc

[in] Handle to the device context. This device context must refer to a device that supports color palettes.

uUsage

[in] Specifies the new use of the system palette. This parameter can be one of the following values.

Value	Meaning

SYSPAL_NOSTATIC	The system palette contains two static colors (black and white).
SYSPAL_NOSTATIC256	Windows 2000: The system palette contains no static colors.
SYSPAL_STATIC	The system palette contains static colors that will not change when an application realizes its logical palette.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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:

Realize the 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. For example, adjacent or overlapping items (such as window frames and borders) should be set to black and white, respectively.

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:

Call SetSystemPaletteUse with the uUsage parameter set to SYSPAL_STATIC.

Realize the logical palette.

Restore the system colors to their previous values.

Send the WM_SYSCOLORCHANGE message.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, GetSysColor, SetSysColors, GetSystemPaletteUse

---

2.255 SetTextAlign

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

```
SetTextAlign: procedure
(
    hdc                 :dword;
    fMode               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetTextAlign@8" );
```

Parameters

hdc

[in] Handle to the device context.

fMode

[in] Specifies the text alignment by using a mask of the values in the following list. Only one flag can be chosen from those that affect horizontal and vertical alignment. In addition, only one of the two flags that alter the current position can be chosen.

Value	Meaning
TA_BASELINE	The reference point will be on the base line of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call. The reference point is passed to the text output function.
TA_RTLREADING	Middle-Eastern Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic.
TA_UPDATECP	The current position is updated after each text output call. The current position is used as the reference point.

When the current font has a vertical default base line, as with Kanji, the following values must be used instead of TA_BASELINE and TA_CENTER.

Value	Meaning
VTA_BASELINE	The reference point will be on the base line of the text.
VTA_CENTER	The reference point will be aligned vertically with the center of the bounding rectangle.

The default values are TA_LEFT, TA_TOP, and TA_NOUPDATECP.

Return Values

If the function succeeds, the return value is the previous text-alignment setting.

If the function fails, the return value is GDI_ERROR.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The TextOut and ExtTextOut functions use the text-alignment flags to position a string of text on a display or other device. The flags specify the relationship between a reference point and a rectangle that bounds the text. The reference point is either the current position or a point passed to a text output function.

The rectangle that bounds the text is formed by the character cells in the text string.

The best way to get left-aligned text is to use either

SetTextAlign (hdc, GetTextAlign(hdc) & (~TA_CENTER))

– or –

SetTextAlign (hdc,TA_LEFT | <other flags>)

You can also use SetTextAlign (hdc, TA_LEFT) for this purpose, but this loses any vertical or right-to-left settings.

Note that 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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GetTextAlign, ScriptStringOut, TextOut

---

2.256 SetTextCharacterExtra

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

```
SetTextCharacterExtra: procedure
(
    hdc                 :dword;
    nCharExtra          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetTextCharacterExtra@8" );
```

Parameters

hdc

[in] Handle to the device context.

nCharExtra

[in] Specifies 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 Values

If the function succeeds, the return value is the previous intercharacter spacing.

If the function fails, the return value is 0x80000000.

Windows NT/ 2000: To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, DrawText, GetTextCharacterExtra, TextOut

---

2.257 SetTextColor

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

```
SetTextColor: procedure
(
    hdc                 :dword;
    crColor             :COLORREF
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetTextColor@8" );
```

Parameters

hdc

[in] Handle to the device context.

crColor

[in] Specifies the color of the text.

Return Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, BitBlt, ExtTextOut, GetTextColor, RGB, SetBkColor, StretchBlt, TextOut, COLORREF

2.258 SetTextJustification

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

```
SetTextJustification: procedure
(
    hdc                 :dword;
    nBreakExtra         :dword;
    nBreakCount         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetTextJustification@12" );
```

Parameters

hdc

[in] Handle to the device context.

nBreakExtra

[in] Specifies 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.

nBreakCount

[in] Specifies the number of break characters in the line.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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. The GetTextExtentPoint32 function computes the width of a specified line before justification. 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, GetTextExtent-Point 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, ExtTextOut, GetTextExtentPoint32, GetText-Metrics, TextOut

---

2.259 SetViewportExtEx

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

```
SetViewportExtEx: procedure
(
        hdc             :dword;
        nXExtent        :dword;
        nYExtent        :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetViewportExtEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

nXExtent

[in] Specifies the horizontal extent, in device units, of the viewport.

nYExtent

[in] Specifies the vertical extent, in device units, of the viewport.

lpSize

[out] Pointer to a SIZE structure that receives the previous viewport extents, in device units. If lpSize is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportExtEx, SetWindowExtEx, SIZE

2.260 SetViewportOrgEx

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

```
SetViewportOrgEx: procedure
(
        hdc             :dword;
        X               :dword;
        Y               :dword;
    var lpPoint         :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetViewportOrgEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the x-coordinate, in device units, of the new viewport origin.

Y

[in] Specifies the y-coordinate, in device units, of the new viewport origin.

lpPoint

[out] Pointer to a POINT structure that receives the previous viewport origin, in device coordinates. If lpPoint is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

This function (along with SetViewportExtEx and SetWindowExtEx) helps define the mapping from the logical coordinate space (also known as a window) to the device coordinate space (the viewport). SetViewportOrgEx specifies which device point maps to the logical point (0,0). It has the effect of shifting the axes so that the logical point (0,0) no longer refers to the upper-left corner.

//map the logical point (0,0) to the device point (xViewOrg, yViewOrg)

SetViewportOrgEx (hdc, xViewOrg, yViewOrg, NULL)

This is related to the 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportOrgEx, POINT, SetWindowOrgEx

---

2.261 SetWinMetaFileBits

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

```
SetWinMetaFileBits: procedure
(
        cbBuffer        :dword;
    var lpbBuffer       :var;
        hdcRef          :dword;
    var lpmfp           :METAFILEPICT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetWinMetaFileBits@16" );
```

Parameters

cbBuffer

[in] Specifies the size, in bytes, of the buffer that contains the Windows-format metafile.

lpbBuffer

[in] 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.)

hdcRef

[in] Handle to a reference device context.

lpmfp

[in] 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 Values

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.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

The Win32 API 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 field in 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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

: Use Gdi32.lib.

See Also

Metafiles Overview, Metafile Functions, DeleteEnhMetaFile, GetWinMetaFileBits, GetMetaFileBitsEx, METAFILEPICT, PlayEnhMetaFile

---

2.262 SetWindowExtEx

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

```
SetWindowExtEx: procedure
(
        hdc             :dword;
        nXExtent        :dword;
        nYExtent        :dword;
    var lpSize          :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetWindowExtEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

nXExtent

[in] Specifies the window's horizontal extent in logical units.

nYExtent

[in] Specifies the window's vertical extent in logical units.

lpSize

[out] Pointer to a SIZE structure that receives the previous window extents, in logical units. If lpSize is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWindowExtEx, SetViewportExtEx, SIZE

2.263 SetWindowOrgEx

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

```
SetWindowOrgEx: procedure
(
        hdc             :dword;
        X               :dword;
        Y               :dword;
    var lpPoint         :POINT
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetWindowOrgEx@16" );
```

Parameters

hdc

[in] Handle to the device context.

X

[in] Specifies the logical x-coordinate of the new window origin.

Y

[in] Specifies the logical y-coordinate of the new window origin.

lpPoint

[out] Pointer to a POINT structure that receives the previous origin of the window. If lpPoint is NULL, this parameter is not used.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

This helps define the mapping from the logical coordinate space (also known as a window) to the device coordinate space (the viewport). SetWindowOrgEx specifies which logical point maps to the device point (0,0). It has the effect of shifting the axes so that the logical point (0,0) no longer refers to the upper-left corner.

//map the logical point (xWinOrg, yWinOrg) to the device point (0,0)

SetWindowOrgEx (hdc, xWinOrg, yWinOrg, NULL)

This is related to the SetViewportOrgEx function. Generally, you will use one function or the other, but not both. Regardless of your use of SetWindowOrgEx and SetViewportOrgEx, the device point (0,0) is always the upper-left corner.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetViewportOrgEx, GetWindowOrgEx, POINT, SetViewportOrgEx

---

2.264 SetWorldTransform

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

```
SetWorldTransform: procedure
```

```
(
        hdc             :dword;
    var lpXform         :XFORM
);
    @stdcall;
    returns( "eax" );
    external( "__imp__SetWorldTransform@8" );
```

Parameters

hdc

[in] Handle to the device context.

lpXform

[in] Pointer to an XFORM structure that contains the transformation data.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

For any coordinates (x, y) in world space, the transformed coordinates in page space (x', y') can be determined by the following algorithm:

x' = x * eM11 + y * eM21 + eDx,

y' = x * eM12 + y * eM22 + eDy,

where the transformation matrix is represented by the following:

| eM11 eM12 0 |

| eM21 eM22 0 |

| eDx eDy 1 |

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Unsupported.

Header: Declared in gdi32.hhf.

See Also

Coordinate Spaces and Transformations Overview, Coordinate Space and Transformation Functions, GetWorldTransform, ModifyWorldTransform, SetGraphicsMode, SetMapMode, SetViewportExtEx, SetViewportOrgEx, SetWindowExtEx, SetWindowOrgEx, XFORM

2.265 StartDoc

The StartDoc function starts a print job.

```
StartDoc: procedure
(
        hdc             :dword;
    var lpdi            :DOCINFO
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StartDocA@8" );
```

Parameters

hdc

[in] Handle to the device context for the print job.

lpdi

[in] Pointer to a DOCINFO structure containing the name of the document file and the name of the output file.

Return Values

If the function succeeds, the return value is greater than zero. This value is the print job identifier for the document.

Windows 95/98: If the function succeeds, the return value is always 1.

If the function fails, the return value is less than or equal to zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError.

Remarks

Applications should call the StartDoc function immediately before beginning a print job. Using this function ensures that multipage documents are not interspersed with other print jobs.

Applications can use the value returned by StartDoc to retrieve or set the priority of a print job. Call the GetJob or SetJob function and supply this value as one of the required arguments.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, DOCINFO, EndDoc,

2.266 StartPage

The StartPage function prepares the printer driver to accept data.

```
StartPage: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StartPage@4" );
```

Parameters

hDC

[in] Handle to the device context for the print job.

Return Values

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is less than or equal to zero.

Windows NT/Windows 2000: To get extended error information, call GetLastError.

Remarks

The system disables the ResetDC function between calls to the StartPage and EndPage functions. This means that you cannot change the device mode except at page boundaries. After calling EndPage, you can call ResetDC to change the device mode, if necessary. Note that a call to ResetDC resets all device context attributes back to default values.

Windows 3.x: EndPage resets the device context attributes back to default values. You must re-select objects and set up the mapping mode again before printing the next page.

Windows 95: EndPage does not reset the device context attributes. However, the next StartPage call does reset the device context attributes to default values. At that time, you must re-select objects and set up the mapping mode again before printing the next page. Note that StartPage also resets the device context state stack used by the SaveDC and RestoreDC functions to default values.

Windows NT/Windows 2000: Beginning with Windows NT Version 3.5, neither EndPage or StartPage resets the device context attributes. Device context attributes remain constant across subsequent pages. You do not need to re-select objects and set up the mapping mode again before printing the next page; however, doing so will produce the same results and reduce code differences between Windows and Windows NT.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Printing and Print Spooler Overview, Printing and Print Spooler Functions, EndPage, ResetDC

2.267 StretchBlt

The StretchBlt function copies a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary. The system stretches or compresses the bitmap according to the stretching mode currently set in the destination device context.

```
StretchBlt: procedure
(
    hdcDest             :dword;
    nXOriginDest        :dword;
    nYOriginDest        :dword;
    nWidthDest          :dword;
    nHeightDest         :dword;
    hdcSrc              :dword;
    nXOriginSrc         :dword;
    nYOriginSrc         :dword;
    nWidthSrc           :dword;
    nHeightSrc          :dword;
    dwRop               :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StretchBlt@44" );
```

Parameters

hdcDest

[in] Handle to the destination device context.

nXOriginDest

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

nYOriginDest

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

nWidthDest

[in] Specifies the width, in logical units, of the destination rectangle.

nHeightDest

[in] Specifies the height, in logical units, of the destination rectangle.

hdcSrc

[in] Handle to the source device context.

nXOriginSrc

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the source rectangle.

nYOriginSrc

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the source rectangle.

nWidthSrc

[in] Specifies the width, in logical units, of the source rectangle.

nHeightSrc

[in] Specifies the height, in logical units, of the source rectangle.

dwRop

[in] Specifies 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.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

StretchBlt stretches or compresses the source bitmap in memory and then copies the result to the destination rectangle. 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 of 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.

Windows 98, Windows 2000: When used in a multimonitor system, both hdcSrc and hdcDest must refer to the same device or the function will fail.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, BitBlt, GetDeviceCaps, MaskBlt, PlgBlt, SetStretchBltMode

---

2.268 StretchDIBits

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, this function compresses the rows and columns by using the specified raster operation.

Windows 98 and Windows 2000: StretchDIBits has been extended to allow a JPEG or PNG image to be passed as the source image.

```
StretchDIBits: procedure
(
        hdc             :dword;
        XDest           :dword;
        YDest           :dword;
        nDestWidth      :dword;
        nDestHeight     :dword;
        XSrc            :dword;
        YSrc            :dword;
        nSrcWidth       :dword;
        nSrcHeight      :dword;
    var lpBits          :var;
    var lpBitsInfo      :BITMAPINFO;
        iUsage          :dword;
        dwRop           :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StretchDIBits@52" );
```

Parameters

hdc

[in] Handle to the destination device context.

XDest

[in] Specifies the x-coordinate, in logical units, of the upper-left corner of the destination rectangle.

YDest

[in] Specifies the y-coordinate, in logical units, of the upper-left corner of the destination rectangle.

nDestWidth

[in] Specifies the width, in logical units, of the destination rectangle.

nDestHeight

[in] Specifies the height, in logical units, of the destination rectangle.

XSrc

[in] Specifies the x-coordinate, in pixels, of the source rectangle in the DIB.

YSrc

[in] Specifies the y-coordinate, in pixels, of the source rectangle in the DIB.

nSrcWidth

[in] Specifies the width, in pixels, of the source rectangle in the DIB.

nSrcHeight

[in] Specifies the height, in pixels, of the source rectangle in the DIB.

lpBits

[in] Pointer to the DIB bits, which are stored as an array of bytes. For more information, see the Remarks section.

lpBitsInfo

[in] Pointer to a BITMAPINFO structure that contains information about the DIB.

iUsage

[in] Specifies whether the bmiColors member of the BITMAPINFO structure was provided and, if so, whether bmiColors contains explicit red, green, blue (RGB) values or indexes. The iUsage parameter must be one of the following values.

Value	Meaning
DIB_PAL_COLORS	The array contains 16-bit indexes into the logical palette of the source device context.
DIB_RGB_COLORS	The color table contains literal RGB values.

For more information, see the Remarks section.

dwRop

[in] 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 more information, see the following Remarks section.

Return Values

If the function succeeds, the return value is the number of scan lines copied.

If the function fails, the return value is GDI_ERROR.

Windows NT/ 2000: To get extended error information, call GetLastError.

Windows 98/Windows 2000: 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 bottom-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 nDest-Width 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.

Windows 98/Windows 2000: 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 BITMAPIN-FOHEADER's biSizeImage member 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 CHECK-JPEGFORMAT or CHECKPNGFORMAT escape to verify that the printer recognizes the JPEG or PNG image, respectively, before calling StretchDIBits.

ICM: Color management is performed. The color profile of the current device context is used as the source color space profile and the sRGB space is used.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Bitmaps Overview, Bitmap Functions, SetMapMode, SetStretchBltMode, BITMAPINFO

2.269 StrokeAndFillPath

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

```
StrokeAndFillPath: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StrokeAndFillPath@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

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.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, BeginPath, FillPath, SetPolyFillMode, StrokePath

---

2.270 StrokePath

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

```
StrokePath: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__StrokePath@4" );
```

Parameters

hdc

[in] Handle to a device context that contains a closed path.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

Remarks

The device context identified by the hdc parameter must contain a closed path.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, BeginPath, EndPath, ExtCreatePen

---

2.271 TextOut

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

```
StrokePath: procedure
(
hdc                 :dword
);
@stdcall;
returns( "eax" );
external( "__imp__StrokePath@4" );
```

Parameters

hdc

[in] Handle to the device context.

nXStart

[in] Specifies the logical x-coordinate of the reference point that the system uses to align the string.

nYStart

[in] Specifies the logical y-coordinate of the reference point that the system uses to align the string.

lpString

[in] Pointer to the string to be drawn. The string does not need to be zero-terminated, since

cbString specifies the length of the string.

cbString

[in] Specifies the length of the 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.

Windows 95/98: This value may not exceed 8192.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

Although not true in general, Windows 95/98 supports the Unicode version of this function as well as the ANSI version.

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.

By default, the current position is not used or updated by this function. However, an application can call the SetTextAlign function with the fMode parameter set to TA_UPDATECP to permit the system to use and update the current position each time the application calls TextOut for a specified device context. When this flag is set, the system ignores the nXStart and nYStart parameters on subsequent TextOut calls.

When the TextOut function is placed inside a path bracket, the system generates a path for the TrueType text that includes each character plus its character box. The region generated is the character box minus the text, rather than the text itself. You can obtain the region enclosed by the outline of the TrueType text by setting the background mode to transparent before placing the TextOut function in the path bracket. Following is sample code that demonstrates this procedure.

```
// Obtain the window's client rectangle
GetClientRect(hwnd, &r);

// THE FIX: by setting the background mode
// to transparent, the region is the text itself
// SetBkMode(hdc, TRANSPARENT);

// Bracket begin a path
BeginPath(hdc);

// Send some text out into the world
TextOut(hdc, r.left, r.top, "Defenestration can be hazardous", 4);
```

```
// 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));
```

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextAlign, SelectObject, SetBkColor, SetTextAlign, SetTextColor, TabbedTextOut

---

2.272 TranslateCharsetInfo

The TranslateCharsetInfo function translates based on the specified character set, code page, or font signature value, setting all members of the destination structure to appropriate values.

```
TranslateCharsetInfo: procedure
(
    var lpSrc           :var;
        lpCs            :CHARSETINFO;
        dwFlags         :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__TranslateCharsetInfo@12" );
```

Parameters

lpSrc

[in/out] If dwFlags is TCI_SRCFONTSIG, this parameter is the address of the fsCsb member of a FONTSIGNATURE structure. Otherwise, this parameter is a DWORD value.

lpCs

[out] Pointer to a CHARSETINFO structure that receives the translated character set information.

dwFlags

[in] Specifies how to perform the translation. This parameter can be one of the following values.

Value	Meaning
TCI_SRCCHARSET	Source contains the character set value in the low word, and zero in the high word.
TCI_SRCCODEPAGE	Source is a code-page identifier in the low word and zero in the high word.
TCI_SRCFONTSIG	Source is the code-page bitfield portion of a FONTSIGNATURE structure. On input this should have only one Windows code-page bit set, either for an ANSI code-page value or for a common ANSI and OEM value (for OEM values, bits 32-63 must be clear.). On output this will have only one bit set. If the TCI_SRCFONTSIG value is given, the lpSrc parameter must be the address of the code-page bitfield. If any other TCI_ value is given, the lpSrc parameter must be a value not an address.

Return Values

If the function succeeds, it returns a nonzero value.

If the function fails, it returns zero. To get extended error information, call GetLastError.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Unicode and Character Sets Overview, Unicode and Character Set Functions, CHARSETINFO, FONTSIGNATURE

2.273 UnrealizeObject

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.

```
UnrealizeObject: procedure
(
    hgdiobj             :dword
);
    @stdcall;
```

```
    returns( "eax" );
    external( "__imp__UnrealizeObject@4" );
```

Parameters

hgdiobj

[in] Handle to the logical palette to be reset.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

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.

Windows 95/98: Automatic tracking of the brush origin is not supported. Applications must use the UnrealizeObject, SetBrushOrgEx, and SelectObject functions to align the brush before using it.

Windows 2000: If hgdiobj is a brush, UnrealizeObject does nothing, and the function returns TRUE. Use SetBrushOrgEx to set the origin of a brush.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetStockObject, RealizePalette, SetBrushOrgEx

---

2.274 UpdateColors

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

```
UpdateColors: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__UpdateColors@4" );
```

Parameters

hdc

[in] Handle to the device context.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/ 2000: To get extended error information, call GetLastError.

Remarks

An application can determine whether a device supports palette operations by calling the GetDeviceCaps function and specifying the RASTERCAPS constant.

An inactive window with a realized logical palette may call UpdateColors as an alternative to redrawing its client area when the system palette changes.

The UpdateColors function typically updates a client area faster than redrawing the area. However, because UpdateColors performs the color translation based on the color of each pixel before the system palette changed, each call to this function results in the loss of some color accuracy.

This function must be called soon after a WM_PALETTECHANGED message is received.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Colors Overview, Color Functions, GetDeviceCaps, RealizePalette

---

2.275 WidenPath

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

```
WidenPath: procedure
(
    hdc                 :dword
);
    @stdcall;
    returns( "eax" );
    external( "__imp__WidenPath@4" );
```

Parameters

hdc

[in] Handle to a device context that contains a closed path.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT/2000: To get extended error information, call GetLastError. GetLastError may return one of the following error codes:

ERROR_CAN_NOT_COMPLETE

ERROR_INVALID_PARAMETER

ERROR_NOT_ENOUGH_MEMORY

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

Windows NT/2000: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in gdi32.hhf.

See Also

Paths Overview, Path Functions, BeginPath, CreatePen, EndPath, ExtCreatePen, SetMiterLimit