Win32 API Reference for HLA

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-

Proc, StartDoc

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 wingdi.h.
Import Library: Use gdi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 `GetDe-viceCaps` 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Device Contexts Overview, Device Context Functions, Multiple Display Monitors, DeleteDC, DEVMODE, Enum-DisplayDevices, 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Brushes Overview, Brush Functions, BITMAPINFO, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePattern-Brush, 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Brushes Overview, Brush Functions, BITMAPINFO, CreateDIBPatternBrush, CreateHatchBrush, CreatePattern-Brush, 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Bitmaps Overview, Bitmap Functions, BITMAPINFO, CreateFileMapping, DeleteObject, DIBSECTION, GetDIB-ColorTable, 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

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_QUALI TY	**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

² FIXED_PITCH

² 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 Wingdi.h; include Windows.h.
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, 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 Wingdi.h; include Windows.h.
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, 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 context.

```
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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

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

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 (typically with a .fot file name extension) and then uses the `AddFontResource` function to install the font. The TrueType font file (typically with a .ttf file name extension) must be in the System subdirectory of the Windows directory to be used by the **AddFontResource** function.

The **CreateScalableFontResource** function currently supports only TrueType-technology scalable fonts.

When the *lpszFontFile* parameter specifies only a file name and extension, the *lpszCurrentPath* parameter must specify a path. When the *lpszFontFile* parameter specifies a full path, the *lpszCurrentPath* parameter must be NULL or a pointer to NULL.

When only a file name and extension are specified in the *lpszFontFile* parameter and a path is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file as the .ttf file that belongs to this resource. When the `AddFontResource` function is called, the operating system assumes that the .ttf file has been copied into the System directory (or into the main Windows directory in the case of a network installation). The .ttf file need not be in this directory when the **CreateScalableFontResource** function is called, because the *lpszCurrentPath* parameter contains the directory information. A resource created in this manner does not contain absolute path information and can be used in any installation.

When a path is specified in the *lpszFontFile* parameter and NULL is specified in the *lpszCurrentPath* parameter, the string in *lpszFontFile* is copied into the .fot file. In this case, when the

`AddFontResource` function is called, the .ttf file must be at the location specified in the *lpszFont-File* 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 Wingdi.h; include Windows.h.
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, AddFontResource, EnumFonts, EnumFont-Families

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 wingdi.h.
Import Library: Use gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 wingdi.h.
Import Library: Use gdi32.lib.

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 *pOut-put* 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 Wingdi.h; include Windows.h.
Library: Use Winspool.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 `EnhMeta-FileProc` 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 transform 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 **EnumFont-Families** 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
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, 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 Wingdi.h; include Windows.h.
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, 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 wingdi.h.
Import Library: Use gdi32.lib.
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 `SetWinMetaFileBits` 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Pens Overview, Pen Functions, CreateDIBSection, CreatePen, CreatePenIndirect, DeleteObject, GetObject, LOG-BRUSH, 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Post-Script 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 indicates 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.

Library: Use Gdi32.lib.

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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

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 `GetTextAlign` 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 Wingdi.h; include Windows.h.
Library: Use Gdi32.lib.

See Also

Fonts and Text Overview, Font and Text Functions, GetTextAlign, RECT, SetBkColor, SelectObject, SetTextAlign, SetTextColor