USING STOCK PENS



The text describes how to use stock pens in Windows programming. Stock pens are predefined pens that Windows provides.

The three stock pens are BLACK\_PEN, WHITE\_PEN, and NULL\_PEN.

* BLACK\_PEN draws a solid black line with a width of one pixel.
* WHITE\_PEN draws a solid white line with a width of one pixel.
* NULL\_PEN is a pen that doesn't draw.

To use a stock pen, you first need to obtain a handle to it using the GetStockObject function. The GetStockObject function takes the name of the stock pen as an argument and returns a handle to the pen. For example, the following code obtains a handle to the WHITE\_PEN:

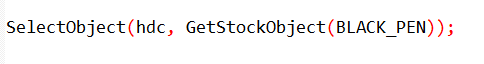


Once you have a handle to a pen, you need to select it into the device context using the SelectObject function. The SelectObject function takes two arguments: the device context and the pen handle. The following code selects the WHITE\_PEN into the device context:

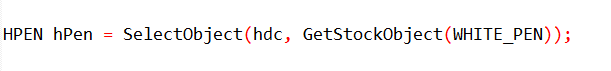


Now any lines that you draw will use the WHITE\_PEN until you select another pen into the device context or release the device context handle.

To return to using the BLACK\_PEN, you can get the handle to that stock object and select it into the device context in one statement:



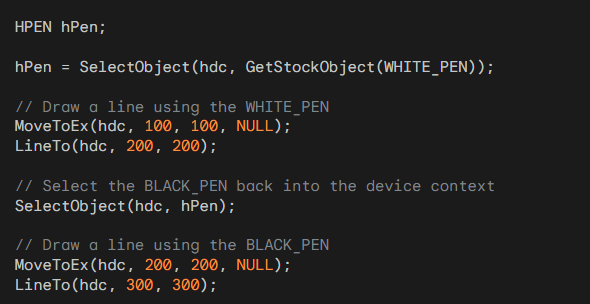
The SelectObject function returns the handle to the pen that had been previously selected into the device context. If you start off with a fresh device context and call:



The current pen in the device context will be WHITE\_PEN and the variable hPen will be the handle to BLACK\_PEN. You can then select BLACK\_PEN into the device context by calling:



Example Code:



This code will draw a white line from (100, 100) to (200, 200) and a black line from (200, 200) to (300, 300).

CREATING, SELECTING AND DELETING PENS

Creating Pens

To create a custom pen, you use the CreatePen or CreatePenIndirect functions. These functions take several parameters that define the appearance of the pen, such as the line style, line width, and color. The functions return a handle to the pen, which you can then select into the device context using the SelectObject function.

Selecting Pens

Only one pen can be selected into the device context at a time. To select a pen, you call the SelectObject function with the device context handle and the pen handle as arguments. Once a pen is selected, all lines that you draw will use that pen until you select another pen or release the device context.

Deleting Pens

When you are finished with a pen, you should delete it using the DeleteObject function. This will free up the resources that the pen was using. However, you should not delete a pen while it is still selected into a device context.

GDI Objects

A logical pen is a type of GDI object. GDI objects are resources that are managed by the Graphics Device Interface (GDI). There are six types of GDI objects: brushes, bitmaps, regions, fonts, palettes, and pens. GDI objects are selected into the device context using the SelectObject function.

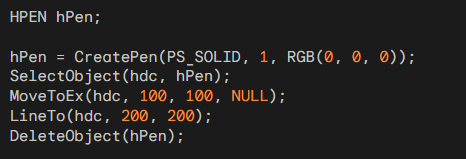
*Rules for Using GDI Objects.*

There are three rules for using GDI objects:

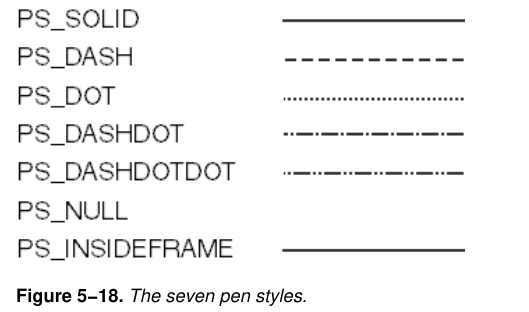
* You should eventually delete all GDI objects that you create.
* Don't delete GDI objects while they are selected in a valid device context.
* Don't delete stock objects.

*Example Code*

The following code shows how to create a pen, select it into the device context, draw a line with the pen, and then delete the pen:

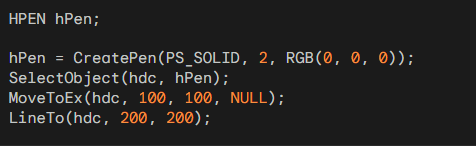


This code will draw a black line from (100, 100) to (200, 200).



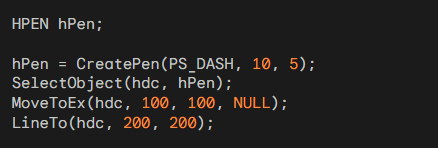
PS\_SOLID

A solid pen draws a solid line with a constant width. The width of the line is specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a solid black line with a width of 2 pixels:



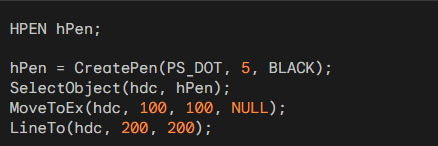
PS\_DASH

A dashed pen draws a line that is made up of a series of dashes. The length of the dashes and the spacing between them is specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a dashed black line with a dash length of 10 pixels and a spacing of 5 pixels:



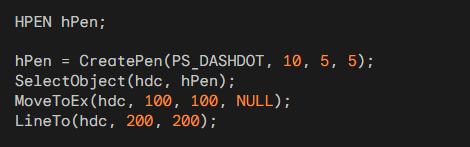
PS\_DOT

A dotted pen draws a line that is made up of a series of dots. The size of the dots is specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a dotted black line with a dot size of 5 pixels:



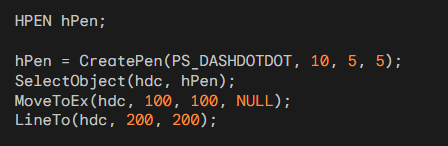
PS\_DASHDOT

A dash-dot pen draws a line that is made up of a series of alternating dashes and dots. The length of the dashes and the size of the dots are specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a dash-dot black line with a dash length of 10 pixels, a dot size of 5 pixels, and a spacing of 5 pixels between the dashes and dots:



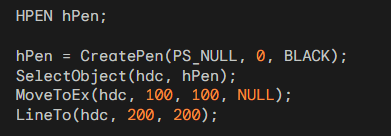
PS\_DASHDOTDOT

A dash-dot-dot pen draws a line that is made up of a series of alternating dashes and double dots. The length of the dashes and the size of the dots are specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a dash-dot-dot black line with a dash length of 10 pixels, a dot size of 5 pixels, and a spacing of 5 pixels between the dashes and dots:



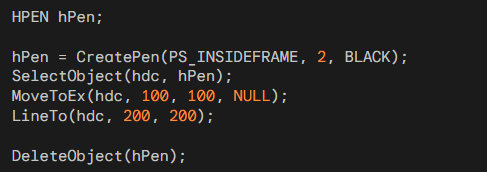
PS\_NULL

A null pen does not draw anything. The following code shows how to draw a null line:



PS\_INSIDEFRAME

The PS\_INSIDEFRAME pen style is a special pen style that is used to draw lines that are clipped to the inside of a frame. The frame is specified by the iWidth parameter to the CreatePen function. The following code shows how to draw a solid black line with a width of 2 pixels that is clipped to the inside of a frame with a width of 10 pixels:



The provided code snippet is a complete and functional code segment that draws a line using the PS\_INSIDEFRAME pen style, a width of 2 pixels, and a black color. It also deletes the pen when it is no longer needed.

Creating and Managing Pens

Pens are used to draw lines and shapes in Windows programming. There are three types of pens: solid pens, dotted pens, and dashed pens.

* *Solid pens draw solid lines, dotted pens draw lines that consist of a series of dots, and dashed pens draw lines that consist of a series of dashes.*

The CreatePen and CreatePenIndirect functions are used to create pens. The CreatePen function takes three arguments: the pen style, the line width, and the color of the pen. The CreatePenIndirect function takes a pointer to a structure of type LOGPEN, which contains the pen style, line width, and color.

Once a pen has been created, it must be selected into the device context before it can be used. The SelectObject function is used to select a pen into the device context. The SelectObject function takes two arguments: the device context and the pen handle.

When a pen is no longer needed, it should be deleted using the DeleteObject function. The DeleteObject function takes a pen handle as an argument.

Line Width

The line width is the width of the line that the pen draws. The line width is specified by the iWidth parameter to the CreatePen function. The iWidth parameter can be a positive integer or zero. A positive integer specifies the width of the line in pixels. Zero specifies a line width of one pixel.

Color

The color of the pen is the color of the line that the pen draws. The color is specified by the crColor parameter to the CreatePen function. The crColor parameter is a COLORREF value, which is a 32-bit value that contains the red, green, and blue components of the color.

Dithered Colors

The PS\_INSIDEFRAME pen style is the only pen style that can use a dithered color. A dithered color is a color that is made up of a pattern of other colors. Dithered colors are used to simulate colors that cannot be displayed by the device.

Creating Pens at Initialization

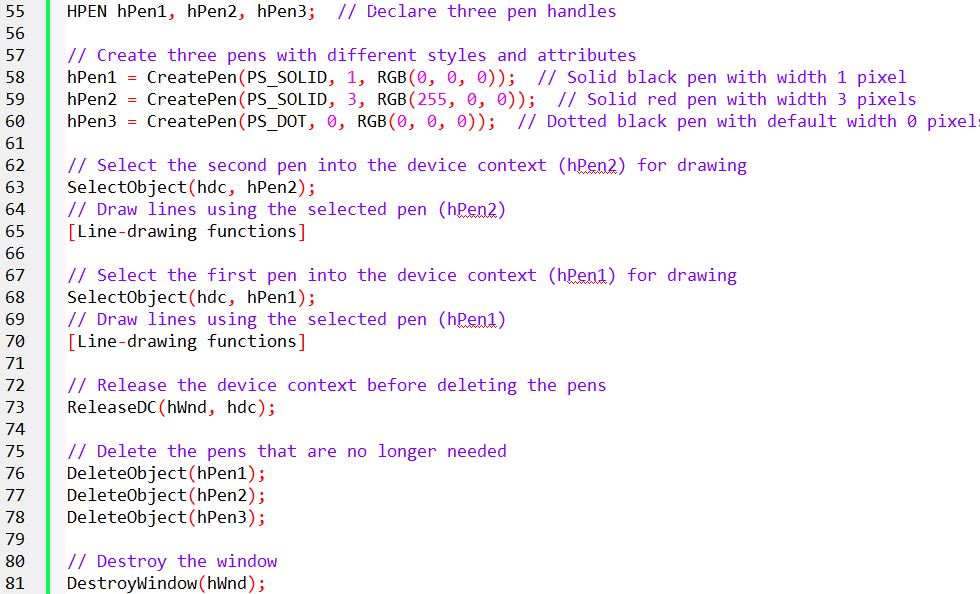
If your program uses a lot of different pens that you initialize in your source code, you can create the pens using the CreatePenIndirect function and store the pen handles in static variables. This can be more efficient than creating the pens each time you need to use them.

Selecting and Deleting Pens

To select a pen into the device context, you use the SelectObject function. The SelectObject function takes two arguments: the device context and the pen handle. To delete a pen, you use the DeleteObject function. The DeleteObject function takes a pen handle as an argument.

Example Code

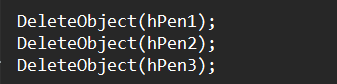
The following code shows how to create, select, and delete pens:



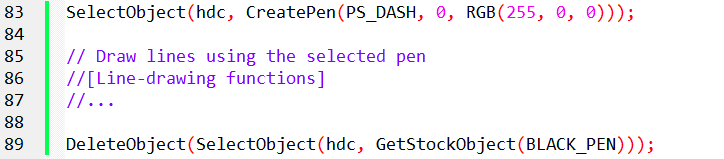
Deleting Pens

There are two main methods for deleting pens:

Delete the pens during WM\_DESTROY processing: This method involves deleting the pens when the window is destroyed. This is the most straightforward approach, but it requires that your program knows which pens will be needed beforehand.

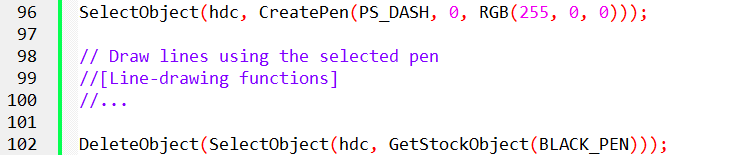


Create and delete pens during WM\_PAINT processing: This method involves creating the pens during each WM\_PAINT message and deleting them after calling EndPaint. This approach is more flexible, as it allows you to create pens as needed, but it requires careful handling to avoid deleting the pen currently selected in the device context.



Combining CreatePen and SelectObject

You can combine the CreatePen and SelectObject calls into a single statement to create a pen on the fly and select it into the device context. This is a concise and efficient approach, but it requires you to handle the deletion of the pen carefully.



Retrieving Pen Information

You can use the GetObject function to obtain the values of the LOGPEN structure fields for a given pen handle. This can be useful for inspecting the properties of a pen.



Getting the Currently Selected Pen

You can use the GetCurrentObject function to retrieve the handle to the pen that is currently selected into the device context. This can be useful when you need to switch between pens without explicitly saving their handles.



ExtCreatePen

The ExtCreatePen function is another method for creating pens. It provides more flexibility than the CreatePen function, allowing you to specify additional pen attributes such as the join style, end cap style, and miter limit. This function is discussed in more detail in Chapter 17 of the reference material.

Filling in the Gaps with Dotted and Dashed Pens

When using dotted or dashed pens, a question arises: what happens to the gaps between the dots or dashes?

By default, Windows fills in these gaps with the background color of the device context. This behavior aligns with the common practice of using a white background brush to erase the window background.

Background Mode and Background Color

The background mode and background color of the device context determine how the gaps between dots and dashes are treated.

Background Mode:

* OPAQUE: The default mode, where Windows fills in the gaps with the background color.
* TRANSPARENT: Windows ignores the background color and leaves the gaps transparent.

Background Color: The color that Windows uses to fill in the gaps when the background mode is OPAQUE.

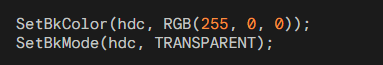
Modifying Background Color and Mode

You can modify the background color and mode using the following functions:

* SetBkColor(hdc, crColor): Sets the background color of the device context.
* GetBkColor(hdc): Retrieves the current background color of the device context.
* SetBkMode(hdc, TRANSPARENT): Sets the background mode to TRANSPARENT, preventing Windows from filling in the gaps.
* GetBkMode(hdc): Retrieves the current background mode of the device context.

Example

The following code snippet sets the background color to red and the background mode to TRANSPARENT:



DRAWING MODES AND RASTER OPERATIONS(ROPs)

In Windows graphics programming, the appearance of lines drawn on the screen is influenced by the drawing mode defined in the device context.

The drawing mode determines how the color of the pen interacts with the color of the underlying display surface. This allows for a variety of effects, such as drawing lines that appear to change color based on the background color.

Bitwise Boolean Operations for Raster Operations

When drawing a line, Windows performs a bitwise Boolean operation between the pixels of the pen and the pixels of the destination display surface.

Each pixel has a value that represents its color, and the Boolean operation determines the new color value for each pixel based on the values of the corresponding pen pixel and destination pixel.

Binary Raster Operations (ROP2) for Line Drawing

Since line drawing involves only two pixel patterns (the pen and the destination), the Boolean operation used is called a "binary raster operation" or "ROP2." Windows provides 16 different ROP2 codes that specify how to combine the pen pixels and the destination pixels.

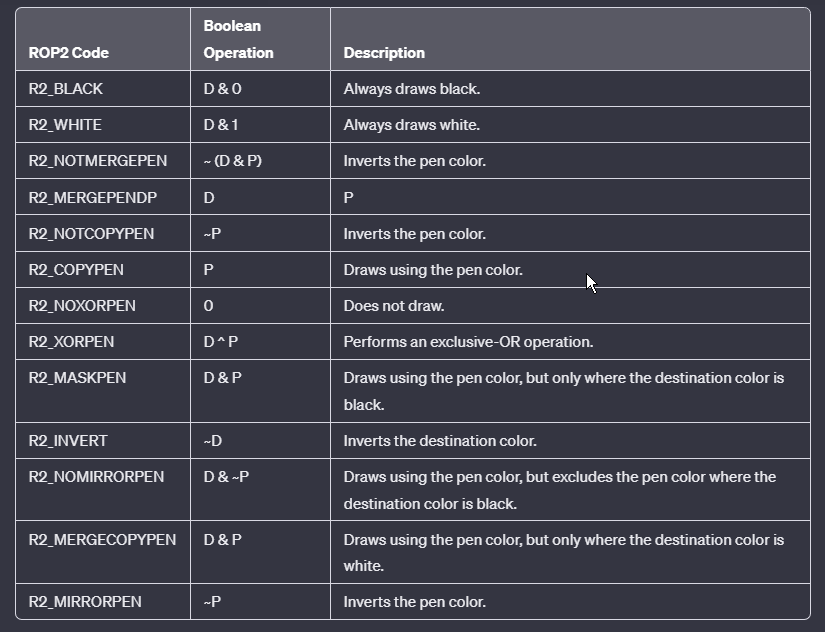
Default Drawing Mode: R2\_COPYPEN

The default drawing mode in the device context is R2\_COPYPEN. This mode simply copies the pixels of the pen to the destination, resulting in the expected behavior of drawing lines using the pen's color.

Other ROP2 Codes and Their Effects

The 15 other ROP2 codes provide various ways to combine pen and destination colors. Some examples include:

The table below summarizes the 16 ROP2 drawing modes and their corresponding Boolean operations:



Explanation:

ROP2 Code: The numerical code used to represent the raster operation.

Boolean Operation: The boolean operation or combination of operations represented by the ROP2 code.

Description: A brief description of the effect or operation achieved by the specified boolean operation.

Please note that the symbols used in the Boolean operations are as follows:

D: Destination color.

P: Pen color.

**~:** NOT operation.

**&:** AND operation.

**|:** OR operation.

**^:** XOR operation.

Setting and Getting Drawing Modes

The drawing mode of a device context determines how the color of the pen interacts with the color of the underlying display surface when drawing lines or shapes. To set the drawing mode, use the SetROP2 function:



The iDrawMode argument specifies the desired drawing mode, which is one of the 16 ROP2 codes defined by Windows. To retrieve the current drawing mode, use the GetROP2 function:



*Default Drawing Mode (R2\_COPYPEN)*

The default drawing mode is R2\_COPYPEN, which simply copies the pen color to the destination. This means that lines or shapes drawn using this mode will appear in the same color as the pen.

*R2\_NOTCOPYPEN Mode*

The R2\_NOTCOPYPEN mode inverts the color of the pen before drawing. As a result, lines or shapes drawn with a black pen will appear as white, and lines or shapes drawn with a white pen will appear as black.

*R2\_BLACK and R2\_WHITE Modes*

The R2\_BLACK mode always draws lines or shapes as black, regardless of the pen color or background color. Similarly, the R2\_WHITE mode always draws lines or shapes as white.

*R2\_NOP Mode*

The R2\_NOP mode, also known as the "no operation" mode, does not draw anything. It leaves the destination unchanged, essentially erasing any previous drawing.

ROP2 Codes on Color Systems

While the previous examples focused on monochrome systems, most modern systems use color displays.

On color systems, Windows applies the bitwise operation defined by the ROP2 code to each color bit of the pen and destination pixels.

This allows for a wider range of visual effects when drawing lines or shapes.

*R2\_NOT Drawing Mode*

The R2\_NOT drawing mode inverts the destination color to determine the color of the line, regardless of the pen color. For instance, drawing a line with a black pen on a cyan destination will result in a magenta line. This mode always produces a visible line except when drawing with a black pen on a medium gray background.

Practical Applications of ROP2 Codes

The ROP2 codes provide various ways to control the appearance of lines and shapes drawn using pens.

For example, the R2\_NOT mode can be used to create contrasting lines on colored backgrounds, while the R2\_MERGEPEN mode can be used to blend lines with the background.

These drawing modes and their corresponding ROP2 codes offer flexibility in creating various visual effects and enhancing the appearance of graphical elements in Windows applications.

DRAWING FILLED AREAS

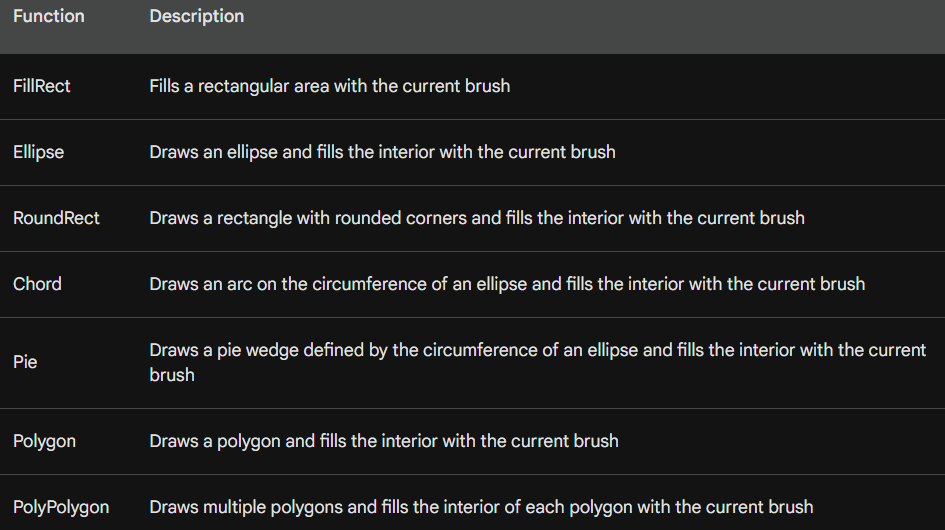
Drawing Filled Areas with Borders

Drawing filled areas with borders involves utilizing a combination of pens and brushes to define the outline and interior of the desired shape.

Windows provides several functions for drawing various filled shapes, including rectangles, ellipses, rounded rectangles, chords, pies, polygons, and poly-polygons.

Functions for Drawing Filled Shapes

The following table summarizes the functions for drawing filled shapes:



Using Pens and Brushes

The outline of the filled shape is drawn using the current pen selected in the device context. The selected pen determines the color, width, and style of the outline. To select a pen, use the SelectObject function:



where hdc is the device context and hPen is the handle to the pen.

The interior of the filled shape is filled using the current brush selected in the device context. The selected brush determines the color and pattern of the interior. To select a brush, use the SelectObject function:

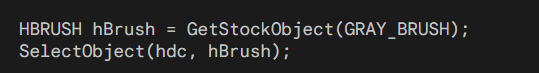


where hdc is the device context and hBrush is the handle to the brush.

Default Brushes

By default, Windows uses the WHITE\_BRUSH for filling the interior of shapes. You can change the default brush by selecting a different brush into the device context.

Windows defines several stock brushes, such as LTGRAY\_BRUSH, GRAY\_BRUSH, DKGRAY\_BRUSH, BLACK\_BRUSH, and NULL\_BRUSH. To select a stock brush, use the GetStockObject function:



Drawing without a Border

To draw a filled shape without a border, select the NULL\_PEN into the device context:



This will prevent Windows from drawing the outline of the shape.

Drawing the Outline without Filling

To draw the outline of a shape without filling the interior, select the NULL\_BRUSH into the device context:



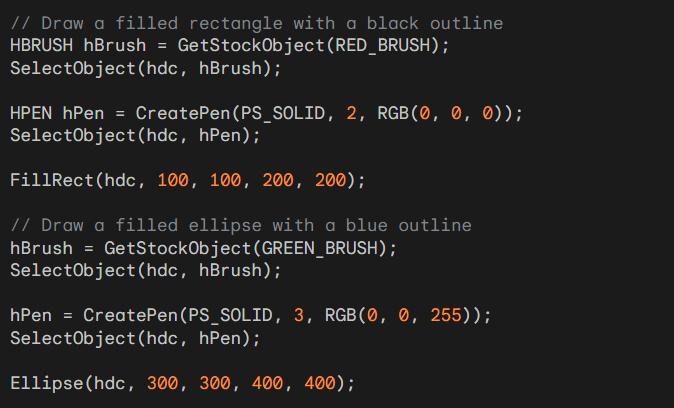
This will draw the outline of the shape using the current pen color, but the interior will remain transparent.

Customizing Brushes

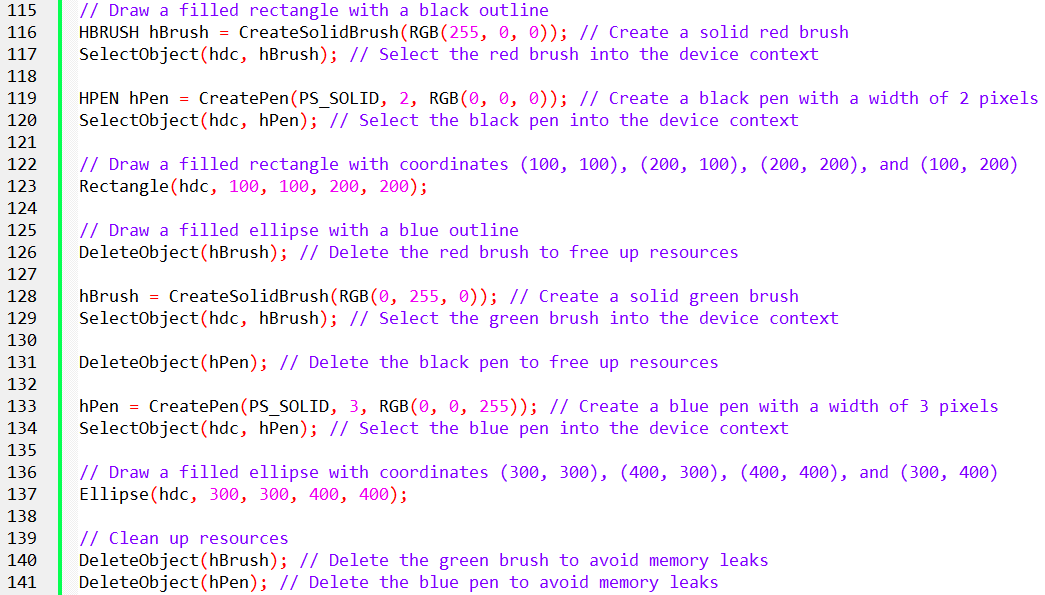
In addition to using stock brushes, you can also create customized brushes. This allows you to define more complex brush patterns. Creating customized brushes is covered in more detail in subsequent chapters.

Code Examples

Here are some code examples for drawing filled areas with borders:

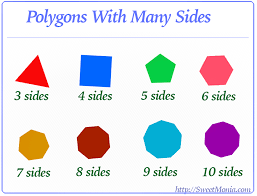


These examples demonstrate the use of FillRect and Ellipse to draw filled shapes with borders. The code first selects the desired brush and pen into the device context, and then calls the respective drawing function to create the shape.

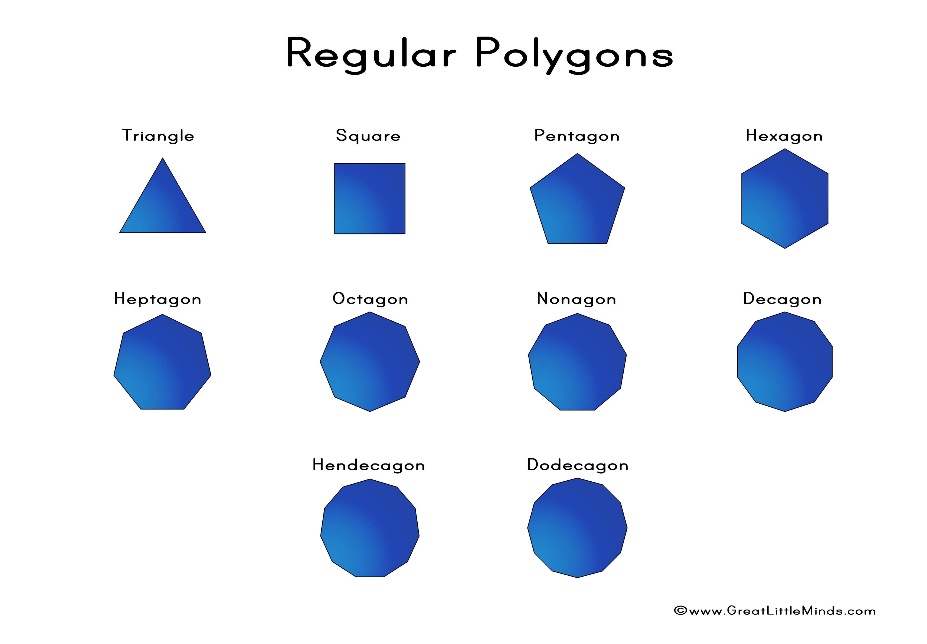


DRAWING POLYGONS

In-depth explanation of the Polygon function, the Polygon-filling mode, and the SetPolyFillMode function in Windows graphics programming.



The Polygon and PolyPolygon functions are used to draw polygons, which are multi-sided shapes composed of connected line segments. The Polygon function draws a single polygon, while the PolyPolygon function draws multiple polygons.

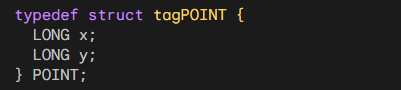


Polygon Function

The Polygon function takes three arguments:

* hdc: The device context in which to draw the polygon
* apt: An array of POINT structures that define the vertices of the polygon
* iCount: The number of vertices in the polygon

The POINT structure is defined as:



The Polygon function connects the specified vertices with lines and fills the enclosed area with the current brush. If the last vertex in the apt array is different from the first vertex, Windows automatically connects the last vertex to the first vertex, closing the polygon.

PolyPolygon Function

The PolyPolygon function takes four arguments:

* hdc: The device context in which to draw the polygons
* apt: An array of POINT structures that define the vertices of all polygons
* aiCounts: An array of integers that specify the number of vertices in each polygon
* iPolyCount: The number of polygons to draw

The PolyPolygon function draws multiple polygons based on the provided vertex data and vertex counts. It fills the enclosed areas of each polygon with the current brush, just like the Polygon function.

Polygon-Filling Mode

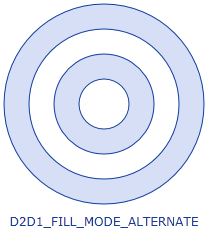
The polygon-filling mode determines how the interior of a polygon is filled. The default polygon-filling mode is ALTERNATE, but you can set it to WINDING using the SetPolyFillMode function:



The iMode parameter can be either ALTERNATE or WINDING.

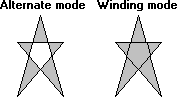
Alternate Mode

In alternate mode, an imaginary line is drawn from a point inside the enclosed area to infinity. The enclosed area is filled only if the imaginary line crosses an odd number of polygon boundary lines. This is why the points of the star in Figure 5-19 are filled but the center is not.



Winding Mode

Winding mode is more complex than alternate mode and is generally considered to be more robust. It calculates a winding number for each point inside the polygon based on the number of times the polygon boundary lines wrap around the point. The point is considered inside the polygon if the winding number is non-zero.



Choosing the Polygon-Filling Mode

In most cases, winding mode will fill all enclosed areas of a single polygon. However, for complex polygons with self-intersections or holes, alternate mode may be more appropriate.