PAINTING AND DRAWING

Windows interacts to the user through a Graphical User Interface (GUI). There are many aspects involved in implementing a GUI, but the most obvious to the user is that information is presented graphically, in charts, pictures, and drawings. Windows provides over 120 functions to assist applications in presenting graphics to the user. In addition, Windows provides a device-independent analogy that hides the details of the presentation device from the application.

You already have been exposed to the concept of a device context. The device context is the basis for all graphics in the Windows environment–it is used by Windows to describe the physical aspects of a graphics device. By using a device context, the application can create graphics presentations in a device-independent manner. Windows uses the device context to convert the graphics into the commands and data required by the specific device.

Pens, Brushes, and Other Logical Tools

Drawing on a device requires that the application specify information such as the width of lines, the colors of lines, the color and pattern of the background, etc. All of these things must be specified in a device-independent manner, and Windows converts the information into data that is specific to the device. One way of implementing this would be to require the application to specify the information every time it wished to draw to a device. This would require Windows to perform the conversion during every graphics request. A more practical approach, and a more efficient one, is to allow the application to specify common information, and to apply that common information to all subsequent graphics. This is the approach that Windows uses.

A device context contains the common information that Windows uses to draw to the device. Windows separates this information into functional groups and has each functional group represented by a tool. Windows provides functions that an application can use to create a new tool from one of the functional groups. The tool is considered a logical tool because it is not specific to any device or device context. To use the tool, the application must select the tool into a device context. It is at this point that Windows converts the device-independent information represented by the tool into the information required by the specific device. Refer to Table 15-1 for a list of the available tools and their use.

Table 15-1. Windows Drawing Tools

Tool Type	Description
Brush	The brush tool defines the pattern and color that Windows uses to fill the interiors of drawn figures.
Font	The font tool identifies the font that Windows will use when drawing text.
Palette	The palette tool specifies the colors that Windows can use to draw on a given device. Windows applications can specify up to 16,777,216 different colors. Since most devices cannot represent this many different colors, Windows allows an application to specify a subset of these colors for use when drawing.
Pen	The pen tool defines the width, color, and style of lines that Windows uses to outline drawn figures.
Region	A region defines an area on a device. Regions can be used to create complex figures and to control clipping.

Specifying an Area for Drawing

There are many ways to specify where you wish to draw on a device. At times you will specify specific points, such as when drawing a straight line. Other times you will specify that a figure is to be drawn within a bounding rectangle. In this case, you will typically specify either the X and Y points of the upper-left and lower-right corners of the rectangle, or you will create a RECT structure that contains these coordinates.

Often, more complex drawing areas must be defined. In these cases, Windows provides regions and paths. You can create a region based on a rectangle, ellipse, or polygon, and you can combine multiple regions to create new regions. You create a path using the BeginPath() function, after which you may specify various drawing functions that will be used in creating the path. Once the path is complete, use the EndPath() function to close the path. Paths let you create very complex areas.

Invalidation, Clipping, and the WM_PAINT Message

When you draw into a device context that is associated with a window on the user's screen, there is the possibility that portion of the window will be obscured by other windows. Windows handles this situation by clipping the drawing that your application does. Windows creates a clipping region that defines the portion of the window that is visible to the user. Windows does not draw to any area of the window that is not currently visible. Your application may draw anywhere it wants, but those areas that are not visible will not appear. Windows handles this using a technique called clipping. A device context includes a clipping region. Drawing that is done outside of the clipping region is not drawn. There are many functions that you can use that affect the clipping region.

What happens when a portion of your window that has been obscured becomes uncovered? In this case, that portion of the window that is no longer obscured needs to be redrawn. Windows handles this by using an invalidation region, and the WM_PAINT message. Windows maintains an invalidation region for every window that defines that portion of the window that is not currently valid. This can be because a portion of the window was previously obscured and has been made visible because your application has used one of the functions that affects the invalidation region, such as InvalidateRect(), or for other reasons. Any time the invalidation region for any window is not empty, Windows will send a WM_PAINT message to your application. When your application receives a WM_PAINT message, it must redraw that portion of the window that is invalid. You must use the BeginPaint() and EndPaint() functions to coordinate the handling of the invalidation region. When you use BeginPaint(), Windows will fill in a PAINTSTRUCT structure that defines the current invalidation region. In addition, Windows will not allow your application to draw to any area that is not within the invalidation region. You may attempt to draw to the entire window, but Windows will clip your drawing to the invalidation region. After using the BeginPaint() function, Windows assumes that the invalidation region will be redrawn. Refer to the description of the BeginPaint() function for an example.

Painting and Drawing Function Summary

Table 15-2 summarizes the functions used in painting and drawing. A detailed description of each function follows the table.

Table 15-2. Painting and Drawing Function Summary

Function	**Purpose**
AbortPath	Closes and discards the current path.
Arc	Draws an arc.
ArcTo	Draws an arc. The current position is moved to the end of the arc.
BeginPaint	Prepares for painting by allocating a device context and returning painting information.
BeginPath	Starts a path bracket.
Chord	Draws a chord.
CloseFigure	Closes an open figure in a path.
CombineRgn	Combines two regions, creating a third region.
CopyRect	Copies one rectangle to another.
CreateBrushIndirect	Creates a brush given a LOGBRUSH structure.
CreateDIBPatternBrushPt	Creates a brush given a device-independent bitmap.
CreateEllipticRgn	Creates an elliptical region.
CreateEllipticRgnIndirect	Creates an elliptical region given a RECT structure.
CreateHatchBrush	Creates a hatched brush.
CreatePatternBrush	Creates a brush given a handle to a bitmap.
CreatePen	Creates a logical pen.
CreatePenIndirect	Creates a logical pen given a LOGPEN structure.
CreatePolygonRgn	Creates a polygonal region.
CreatePolyPolygonRgn	Creates a region based on multiple polygons.
CreateRectRgn	Creates a rectangular region.
CreateRectRgnIndirect	Creates a rectangular region given a RECT structure.
CreateRoundRectRgn	Creates a rectangular region with rounded corners.
CreateSolidBrush	Creates a logical brush.
DeleteObject	Deletes a GDI object (pen, brush, font, region, bitmap, or palette).
DrawAnimatedRects	Animates the opening or closing of a rectangle.
DrawCaption	Draws a window caption.
DrawEdge	Draws edges of a rectangle using a 3D style.
DrawFocusRect	Draws a rectangle in the “focus” style.
DrawFrameControl	Draws a frame control, such as a button, menu, or scrollbar.
DrawIcon	Draws an icon.
DrawIconEx	Draws an icon or cursor, stretching and applying a given raster operation.
DrawState	Draws the given object, such as an icon or bitmap, using normal, disabled, or other effects.
DrawText	Draws text within the specified rectangle.
DrawTextEx	Draws text within the specified rectangle.
Ellipse	Draws an ellipse.
EndPaint	Indicates the end of the painting process.
EndPath	Closes the current path and selects it into the device context.
EnumObjects	Enumerates the available brushes or pens.
EqualRect	Determines if two rectangles are equal.
EqualRgn	Determines if two regions are equal.
ExcludeClipRect	Excludes the given rectangle from the current clipping area.
ExcludeUpdateRgn	Prevents drawing within invalid regions by removing the update region from the clipping region.
ExtCreatePen	Creates a logical cosmetic or geometric pen.
ExtCreateRegion	Creates a region, applying the given transformation.
ExtFloodFill	Fills an area of the window with the current brush.
ExtSelectClipRgn	Adds the given region to the current clipping region.

FillPath	Fills the current paths, interior.
FillRect	Fills the specified rectangle.
FillRgn	Fills the specified region.
FlattenPath	Converts curves within the current path into a sequence of straight lines.
FrameRect	Draws a border around the given rectangle.
FrameRgn	Draws a border around the given region.
GetArcDirection	Returns the current arc direction, used by arc and rectangle functions.
GetAspectRatioFilterEx	Returns the current aspect ratio filter, which is based on the width and height of pixels on the given device.
GetBkColor	Returns the current background color.
GetBkMode	Returns the current background color mode.
GetBoundsRect	Returns the current bounding rectangle of a device context.
GetBrushOrgEx	Returns the current brush origin.
GetClipBox	Returns the dimensions of the smallest rectangle that encloses the visible area of a device context.
GetClipRgn	Returns a handle to the current clip region.
GetCurrentPositionEx	Returns the current position in logical coordinates.
GetMiterLimit	Returns the miter limit for the device context.
GetObject	Returns information about a GDI object.
GetPath	Returns the endpoints, control points, and vertex information for the current path.
GetPixel	Returns the RGB value of the specified pixel in the device context.
GetPolyFillMode	Returns the current polygon fill mode.
GetRegionData	Returns information about the specified region.
GetRgnBox	Returns the bounding rectangle of the given region.
GetROP2	Returns the current foreground raster operation code.
GetStockObject	Returns a handle to one of various pre-defined objects.
GetUpdateRect	Returns the smallest rectangle that surrounds the current update region.
GetUpdateRgn	Returns the current update region.
InflateRect	Enlarges or shrinks the given rectangle.
IntersectClipRect	Creates a new clipping region that is the intersection of the current clipping region and the given rectangle.
IntersectRect	Returns the intersection of two rectangles.
InvalidateRect	Adds the given rectangle to the current update region.
InvalidateRgn	Adds the given region to the current update region.
InvertRect	Inverts all pixels within the given rectangle.
InvertRgn	Inverts all pixels within the given region.
IsRectEmpty	Determines whether a rectangle is empty or not.
LineDDA	Computes the points within the line, and calls a user function with each point.
LineTo	Draws a line from the current location to the given location.
LockWindowUpdate	Disables or enables drawing within a window.
MoveToEx	Moves the current location within the device context.
OffsetClipRgn	Moves the clipping region by the given amount.
OffsetRect	Moves the specified rectangle by the given amount.
OffsetRgn	Moves the specified region by the given amount.
PaintRgn	Paints the given region.
PathToRegion	Creates a region from the current path.
Pie	Draws a pie-shaped wedge.
PolyBezier	Draws one or more bezier curves.
PolyBezierTo	Draws one or more bezier curves, moving the current position.
Polygon	Draws one or more lines.
Polyline	Draws one or more lines.

PolylineTo	Draws one or more lines, moving the current position.
PolyPolygon	Draws one or more closed polygons.
PolyPolyline	Draws one or more line series.
PtInRect	Determines if the given point is within the given rectangle.
PtInRegion	Determines if the given point is within the given region.
Rectangle	Draws a filled rectangle.
RectInRegion	Determines if any part of the given rectangle is within the given region.
RedrawWindow	Redraws the specified portion of the window.
RoundRect	Draws a filled rectangle with rounded corners.
SelectClipPath	Selects the current path as the clipping region.
SelectClipRgn	Selects the given region as the clipping region.
SetArcDirection	Selects the drawing direction for arcs and rectangles.
SetBkColor	Sets the current background color.
SetBkMode	Sets the current background color mode.
SetBoundsRect	Sets the mode that Windows uses to accumulate bounding rectangles.
SetBrushOrgEx	Sets the origin for the next brush selected into the device context.
SetMiterLimit	Sets the limit for the length of miter joints.
SetPixel	Sets the color of the given pixel and returns the old pixel color.
SetPixelV	Sets the color of the given pixel.
SetPolyFillMode	Sets the mode Windows uses to fill polygons.
SetRect	Sets the specified rectangle coordinates.
SetRectEmpty	Sets the rectangle to an empty rectangle.
SetRectRgn	Converts the given region into a rectangular region.
SetROP2	Sets the foreground mix raster operation mode.
StrokeAndFillPath	Draws the outline of the current path and fills its interior.
StrokePath	Draws the outline of the current path.
SubtractRect	Returns the rectangle obtained by subtracting two given rectangles.
UnionRect	Returns the rectangle that is the union of the two given rectangles.
UnrealizeObject	Unrealizes the logical palette, so that the next RealizePalette() will remap the palette.
UpdateWindow	Sends a WM_PAINT message to the given window if there is an update region.
ValidateRect	Removes the given rectangle from the update region.
ValidateRgn	Removes the given region from the update region.
WidenPath	Widens the current path based on the current pen.

ABORTPATH WINDOWS 95 WINDOWSNT

Description	AbortPath() discards the current path for the specified device context. If a path is currently being defined, it is closed.
Syntax	BOOL AbortPath(HDC hDC)
Parameters	
hDC	HDC: Specifies the target device context.
Returns	BOOL: Returns TRUE if successful: otherwise, returns FALSE.
See Also	BeginPath(), EndPath()
Example	This example creates a path on the client area of the window when the user selects the Test! menu item. If the window size is too small for the path, the path is aborted with the AbortPath() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
```

```
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rect;
                           HDC  hDC = GetDC( hWnd );

                           GetClientRect( hWnd, &rect );

                           if( BeginPath( hDC ) )
                           {
                              try
                              {
                                 MoveToEx( hDC, 10, 10, NULL );

                                 // Check if the coordinates
                                 // are within the rect.
                                 //.........................
                                 if ( rect.right < 200 )
                                    RaiseException( 1, 0, 0, NULL );

                                 LineTo( hDC, 200, 10 );

                                 if ( rect.bottom < 200 )
                                    RaiseException( 1, 0, 0, NULL );

                                 LineTo( hDC, 200, 200 );

                                 // close final figure
                                 // and end the path.
                                 //...................
                                 CloseFigure( hDC );
                                 EndPath( hDC );

                                 // Outline the path so it is visible.
                                 //...................................
                                 StrokePath( hDC );
                              }
                              except( EXCEPTION_EXECUTE_HANDLER )
                              {
                                 AbortPath( hDC );
                              }
                           }

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
hDChDChDChDC
```

ARC   WIN32S   WINDOWS 95   WINDOWS NT

Description	Arc() draws an elliptical arc contained within the given rectangle, starting and ending at the given points. The arc is drawn in the current arc direction set with SetArcDirection() using the current pen, but is not filled. The start and endpoints do not have to lie on the arc. Windows calculates a line from the specified point to the center of the bounding rectangle and uses the arc's intersection with this line.
Syntax	BOOL Arc(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nStartX, int nStartY, int nEndX, int nEndY);
Parameters	

hDC	HDC: Specifies the device context that the arc is to be drawn in.
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
nStartX	int: Specifies the x coordinate of the starting point for the arc.
nStartY	int: Specifies the y coordinate of the starting point for the arc.
nEndX	int: Specifies the x coordinate of the ending point for the arc.
nEndY	int: Specifies the y coordinate of the ending point for the arc.
Returns	BOOL: Returns TRUE if successful. Otherwise, FALSE is returned.
Example	The following example draws the upper quarter of a circle, as shown in Figure 15-1, defined by the specified rectangle when the user selects the Test! menu item.

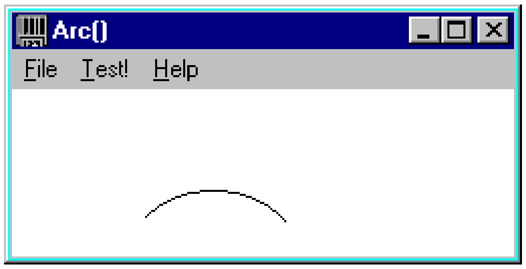

Figure 15-1 Arc() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC = GetDC( hWnd );

                           Arc( hDC, 50, 50, 150, 150, 150, 50, 50, 50 );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
hDC
```

ARCTO WINDOWS 95 WINDOWS NT

Description	ArcTo() draws an elliptical arc contained within the given rectangle, starting and ending at the given points. The arc is drawn in the current arc direction set with SetArcDirection() using the current pen, but is not filled. The start and endpoints do not have to lie on the arc. Windows calculates a line from the specified point to the center of the bounding rectangle and uses the arc's intersection with this line. The current position is moved to the end of the arc.
Syntax	BOOL ArcTo(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nStartX, int nStartY, int nEndX, int nEndY)
Parameters	
hDC	HDC: Specifies the device context that the arc is to be drawn in.

nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
nStartX	int: Specifies the x coordinate of the starting point for the arc.
nStartY	int: Specifies the y coordinate of the starting point for the arc.
nEndX	int: Specifies the x coordinate of the ending point for the arc.
nEndY	int: Specifies the y coordinate of the ending point for the arc.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
Example	This function is identical to the Arc() function, except that when ArcTo() is used, the current position is moved to the end of the arc. See Arc() for an example of this function.

BEGINPAINT

WIN32S WINDOWS 95 WINDOWS NT

Description	BeginPaint() initiates the window painting process by allocating a device context and returning information about the update region. This function is used in response to the WM_PAINT message. When BeginPaint() is called, Windows assumes that the update region has been repainted and sets the update region to an empty region.
Syntax	HDC BeginPaint(HWND hWnd, LPPAINTSTRUCT lpPaintStruct)
Parameters	
hWnd	HWND: Window handle of the window to be painted.
lpPaintStruct	LPPAINTSTRUCT: A pointer to a PAINTSTRUCT structure. Windows will fill in this structure with information about the update region of the specified window. See the defintion of the PAINTSTRUCT structure below.
Returns	HDC: If successful, returns a handle to a device context that should be using in painting the window; otherwise, NULL is returned.
See Also	EndPaint()
Related Messages	WM_PAINT

PAINTSTRUCT Definition

```
typedef struct tagPAINTSTRUCT
{
    HDC  hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[32];
} PAINTSTRUCT;
```

hdc	HDC: The device context to use for painting.
fErase	BOOL: A non-zero value if the background must be erased. The application is responsible for erasing the background if the window was created without a background brush.
rcPaint	RECT: The brounding rectangle of the area that is requested to be updated.
fRestore	BOOL: Reserved.
fIndUpdate	BOOL: Reserved.
rgbReserved	BYTE[32]: Reserved.
Example	The following example demonstrates the use of the BeginPaint() and EndPaint() functions by painting a rectangle on the client area of the window when the WM_PAINT message is received.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;

                 // Begin Painting.
                 //................
                 BeginPaint( hWnd, &ps );

                 // Paint the client area.
                 //......................
                 Rectangle( ps.hdc, 10, 10, 100, 100 );

                 // End Painting.
                 //..............
                 EndPaint( hWnd, &ps );
              }
              break;
          .
          .
          .
hDChDChDChDC
```

BEGINPATH WINDOWS 95 WINDOWS NT

Description	BeginPath() begins a path definition. The path is terminated by calling either EndPath() or AbortPath(). Once a path definition has been started, the GDI functions listed in the See Also belo are considered part of the path.
Syntax	BOOL BeginPath(HDC hDC)
Parameters	
hDC	HDC: Specifies the device context that the path is to be used in.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	AngleArc(), Arc(), ArcTo(), Chord(), CloseFigure(), Ellipse(), EndPath(), ExtTextOut(), LineTo(), MoveToEx(), Pie(), PolyBezier(), PolyBezierTo(), Polygon(), PolyLine(), PolyLineTo(), PolyPolygon(), PolyPolyline(), Rectangle(), RoundRect(), TextOut()
Example	See AbortPath() for an example of this function.

CHORD WIN32S WINDOWS 95 WINDOWS NT

Description	Chord() draws a chord, using the current pen. The chord is filled in with the current brush. A chord is the region bounded by an ellipse and a line segment. The ellipse is specified by using a rectangle.
Syntax	BOOL Chord(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nXRadial1, int nYRadial1, int nXRadial2, int nYRadial2)
Parameters	
hDC	HDC: Specifies the device context that the chord is to be drawn in.
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
nXRadial1	int: Specifies the x coordinate of the first radial’s endpoint.
nYRadial1	int: Specifies the y coordinate of the first radial’s endpoint.
nXRadial2	int: Specifies the x coordinate of the second radial’s endpoint.
nYRadial2	int: Specifies the y coordinate of the second radial’s endpoint.
Returns	BOOL: Returns TRUE is successful; otherwise, FALSE is returned.

Example The following example draws the bottom half of a circle, as shown in Figure 15-2, with the Chord() function when the user selects the Test! menu item.

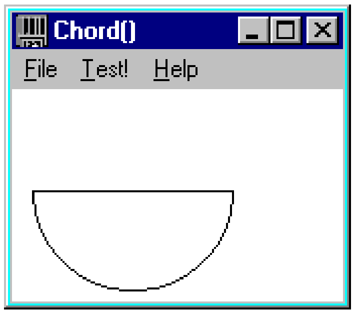

Figure 15-2 Chord() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC = GetDC( hWnd );

                           Chord( hDC, 10, 0, 110, 100, // The bounding rectangle.
                                       10, 50,          // The endpoint of the start of the
arc
                                       110, 50 );       // The endpoint of the end of the arc

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
hDC
```

CLOSEFIGURE WINDOWS 95 WINDOWS NT

Description	The CloseFigure() function closes the figure in the currently open path by drawing a line from the current position to the first point of the figure. The end of the line is joined to the figure using the current line join style.
Syntax	BOOL CloseFigure(HDC hDC)
Parameters	
hDC	HDC: Specifies the device context of the figure that is to be closed.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	AbortPath(), BeginPath(), EndPath()
Example	See AbortPath() for an example of this function.

COMBINERGN WIN32S WINDOWS 95 WINDOWS NT

Description The Combine Rgn() function combines two existing regions to produce a third region. The handle of the resulting region must already have been created, and the region it represents is replaced by the new region.

Syntax	int CombineRgn(HRGN hDest, HRGN hSource1, HRGN hSource2, int nMode)
Parameters	
hDest	HRGN: A handle to a region. If this function is successful, this region will be set to the combined regions of hSource1 and hSource2.
hSource1	HRGN: A handle to a region. This region is combined with hSource2 to create hDest.
hSource2	HRGN: A handle to a region. This region is combined with hSource1 to create hDest.
nMode	int: Specifies the combine mode, which can be one of the values listed in Table 15-3.

Table 15-3. Mode Values for CombineRgn()

Mode	**Description**
RGN_AND	The destination region is the intersection of the two regions.
RGN_COPY	The destination region is a copy of region hSource1.
GN_DIFF	The destination region is that part of hSource1 that is not also part of hSource2.
RGN_OR	The destination region is the union of the two regions.
RGN_XOR	The destination region is the union of the two regions, excluding any portion where the two regions overlap.

Returns int: The return value can be one of the values specified in Table 15-4.

Table 15-4. Return Values for CombineRgn()

Return Value	**Description**
NULLREGION	The region is empty.
SIMPLEREGION	The region is a simple rectangle.
COMPLEXREGION	The region is more than a simple rectangle.
ERROR	No region was created.

See Also	CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetRectRgn()
Example	The following example creates three regions. The first region is rectangular, the second is elliptic, and the third is a polygon. The rectangular and elliptic regions are combined to create a new region. The polygon region is then combined with the results to form a region that contains all three regions. Finally, the resulting region is filled with a hatched-brush, and all objects are deleted. The resulting figure is shown in Figure 15-3.

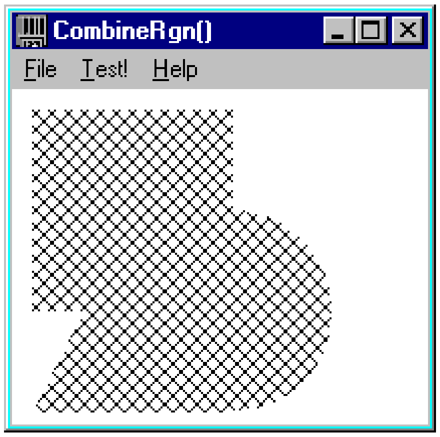

Figure 15-3 CombineRgn() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           POINT    ptArray[3];
                           HRGN     Rgn1, Rgn2, Rgn3, Rgn4;
                           LOGBRUSH lb;
                           HBRUSH   hBrush;
                           HDC      hDC = GetDC( hWnd );

                           // Create a rectangular region.
                           //..............................
                           Rgn1 = CreateRectRgn( 10, 10, 110, 110 );

                           // Create an elliptic region.
                           //............................
                           Rgn2 = CreateEllipticRgn( 60, 60, 160, 160 );

                           // Create a polygonal region consisting of three points.
                           //.......................................................
                           ptArray[0].x = 10;  ptArray[0].y = 160;
                           ptArray[1].x = 60;  ptArray[1].y = 60;
                           ptArray[2].x = 110; ptArray[2].y = 160;
                           Rgn3 = CreatePolygonRgn( ptArray,
                                                    sizeof(ptArray)/sizeof(POINT), ALTERNATE
);

                           // Create a dummy region to combine into.
                           //......................................
                           Rgn4 = CreateRectRgn( 0, 0, 10, 10 );

                           // Combine the elliptic and rectangular regions.
                           //..............................................
                           CombineRgn( Rgn4, Rgn2, Rgn1, RGN_OR );

                           // Combine the results with the polygon region.
                           //.............................................
                           CombineRgn( Rgn4, Rgn4, Rgn3, RGN_OR );

                           // Now create a hatched brush and draw the resulting region.
                           //...........................................................
                           lb.lbStyle = BS_HATCHED;
                           lb.lbColor = RGB(0, 0, 0);
                           lb.lbHatch = HS_DIAGCROSS;
                           hBrush = CreateBrushIndirect( &lb );
```

```
                              FillRgn( hDC, Rgn4, hBrush );

                              //  Finally, delete all created GDI objects.
                              //.........................................
                              DeleteObject( hBrush );
                              DeleteObject( Rgn1 );
                              DeleteObject( Rgn2 );
                              DeleteObject( Rgn3 );
                              DeleteObject( Rgn4 );

                              ReleaseDC( hWnd, hDC );
                           }
                           break;
         .
         .
         .
```

HDCCOPYRECT   WIN32S WINDOW95 WINDOWNT

Description	CopyRect() copies the coordinates of a source rectangle to a destination rectangle.
Syntax	BOOL CopyRect(LPRECT lpDest, LPRECT lpSource)
Parameters	
lpDest	LPRECT: A pointer to a RECT structure that will receive the coordinates of the source rectangle.
lpSource	LPRECT: A pointer to a RECT structure that will provide the coordinates for the destination rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise FALSE is returned.
See Also	InflateRect(), InvertRect(), SetRect(), SetRectEmpty(), SubtractRect(), UnionRect()
Example	The following example creates a rectangle, and then copies that rectangle. The copy of the rectangle is then expanded and offset to a new location with the InflateRect() and OffsetRect() functions.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT Rect1, Rect2;
                           HDC  hDC = GetDC( hWnd );

                           // Create first rectangle.
                           //.......................
                           SetRect( &Rect1, 10, 10, 110, 110 );

                           // Create second rectangle.
                           //........................
                           CopyRect( &Rect2, &Rect1 );
                           InflateRect( &Rect2, 10, -10 );
                           OffsetRect( &Rect2, 20, 10 );

                           FillRect( hDC, &Rect1, GetStockObject( LTGRAY_BRUSH ) );
                           FillRect( hDC, &Rect2, GetStockObject( DKGRAY_BRUSH ) );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

CREATEBRUSHINDIRECT WIN32S WINDOWS 95 WINDOWS NT

Description CreateBrushIndirect() creates a logical brush, given a LOGBRUSH structure that specifies the style, color, etc.

Syntax HBRUSH CreateBrushIndirect(LOGBRUSH *lpLogBrush)

Parameters

lpLogBrush LOGBRUSH *: A pointer to a LOGBRUSH structure. See the definition of the LOGBRUSH structure below.

Returns HBRUSH: Returns the handle of the created brush if successful; otherwise, NULL is returned.

See Also CreateDIBPatternBrush(), CreateDIBPatternBrushPt(), CreateHatchBrush(), CreatePatternBrush(), CreateSolidBrush(), GetStockObject()

LOGBRUSH Definition

```
typedef struct tagLOGBRUSH
{
    UINT     lbStyle;
    COLORREF lbColor;
    LONG     lbHatch;
} LOGBRUSH;
```

lbStyle UINT: The brush style. This must be one of the values listed below in Table 15-5.

Table 15-5. LOGBRUSH lbStyle Values

Value	Description
BS_DIBPATTERN	A pattern brush defined by a device-independent bitmap (DIB) specification. If lbStyle is BS_DIBPATTERN, the lbHatch member contains a handle to a DIB. Windows 95 does not support creating brushes from bitmaps larger than 8x8 pixels.
BS_DIBPATTERN8X8	Same as BS_DIBPATTERN.
BS_DIBPATTERNPT	A pattern brush defined by a device-independent bitmap (DIB) specification. The lbHatch member contains a pointer to a DIB.
BS_HATCHED	Hatched brush.
BS_HOLLOW	Hollow brush.
BS_NULL	Same as BS_HOLLOW.
BS_PATTERN	Pattern brush defined by a memory bitmap.
BS_PATTERN8X8	Same as BS_PATTERN.
BS_SOLID	Solid brush.

lbColor COLORREF: The color of the brush. If lbStyle is set to BS_HOLLOW or BS_PATTERN, this parameter is ignored. If lbStyle is set to BS_DIBPATTERN or BS_DIBPATTERNPT, the low-order word of lbColor is set to DIB_PAL_COLORS or DIB_RGB_COLORS. DIB_PAL_COLORS indicates that the the DIB contains an array of 16-bit indicies into the currently realized logical palette. DIB_RGB_COLORS indicates the DIB contains literal RGB color values.

lbHatch LONG: The hatch style of the brush. If lbStyle is set to BS_DIBPATTERN, this member is set to the handle to the DIB. If lbStyle is set to BS_DIBPATTERNPT, this member is set to the pointer of the DIB. If lbStyle is set to BS_PATTERN, this member is set to the handle of the pattern bitmap. If the lbStyle is set to BS_HATCHED, this member can be one of the values listed in Table 15-5.

Example See CombineRgn() for an example of this function.

CREATEDIBPATTERNBRUSH WIN32S WINDOWS 95 WINDOWS NT

Description CreateDIBPatternBrush() creates a logical brush that is built from the given device-independent bitmap. Note that CreateDIBPatternBrush() uses a handle to a global memory object, whereas CreateDIBPatternBrushPt() uses a pointer to a memory location.

Syntax	HBRUSH CreateDIBPatternBrush(HGLOBAL hDIBData, UINT ColorSpec)
Parameters	
hDIBData	HGLOBAL: The handle of a global memory block. This memory block should contain a BITMAPINFO structure, immediately followed by the pixel color data.
ColorSpec	UINT: Contains either DIB_PAL_COLORS or DIB_RGB_COLORS. DIB_PAL_COLORS specifies that the pixel color data in the bmiColors table contains indexes into the currently selected palette. DIB_RGB_COLORS specifies that the pixel color data in the bmiColors table contains RGB color information.
Returns	HBRUSH: If successful, the handle of the new brush is returned; otherwise, NULL is returned.
See Also	CreateBrushIndirect(), CreateDIBPatternBrushPt(), CreateHatchBrush(), CreatePatternBrush(), CreateSolidBrush(), GetStockObject()
Example	The following example illustrates the use of the CreateDIBPatternBrush() function. Assume that the global memory block identified by the parameter hGlobal contains a valid device-independent bitmap.

```
HGLOBAL hGlobal;
HBRUSH aBrush = CreateDIBPatternBrush(hGlobal, DIB_RGB_COLORS);
```

CREATEELLIPTICRGN WINDOWS 95 WINDOWS NT

Description	CreateEllipticRgn() creates a region based on an ellipse defined by the given rectangle.
Syntax	HRGN CreateEllipticRgn(int nLeft, int nRight, int nTop, int nBottom)
Parameters	
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
Returns	HRGN: If successful, the handle of the created region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetRectRgn()
Example	See CombineRgn() for an example of this function.

CREATEELLIPTICRGNINDIRECT WIN32S WINDOW95 WINDOWNT

Description	CreateEllipticRgnIndirect() creates a region based on the ellipse defined by the given rectangle.
Syntax	HRGN CreateEllipticRgnIndirect(LPRECT lpRect)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure that defines the bounding rectangle.
Returns	HRGN: If successful, the handle of the created region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn(), CreatePolygonRgn(), CreatePolyPolygonRgn(), soCreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetRectRgn()
Example	The following example creates a rectangle, using the SetRect() function, creates an elliptic and rectangular region based on this rectangle, and fills the regions when the user selects the Test! menu item.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           RECT rect;
                           HRGN hElRgn, hRectRgn;

                           hDC = GetDC( hWnd );

                           // Create a regions from a RECT structure.
                           //.....................................
                           SetRect( &rect, 10, 10, 110, 110 );
                           hElRgn   = CreateEllipticRgnIndirect( &rect );
                           hRectRgn = CreateRectRgnIndirect( &rect );

                           FillRgn( hDC, hRectRgn, GetStockObject( BLACK_BRUSH ) );
                           FillRgn( hDC, hElRgn, GetStockObject( GRAY_BRUSH ) );

                           DeleteObject( hElRgn );
                           DeleteObject( hRectRgn );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

CREATEHATCHBRUSH   WIN32S   WINDOWS 95   WINDOWS NT

Description	CreateHatchBrush() creates a logical brush using the given color and hatch pattern specifications.
Syntax	HBRUSH CreateHatchBrush(int nHatchPattern, COLORREF crColor)
Parameters	
nHatchPattern	int: Specifies the hatch pattern. This value can be one of the pattern specification values listed in Table 15-6.
crColor	COLORREF: Defines the color of the resulting brush.

Table 15-6. Pattern Specification Values for CreateHatchBrush()

Pattern	Description
HS_BDIAGONAL	Specifies a hatch pattern running from the upper-left to the lower-right at a 45-degree angle.
HS_CROSS	Specifies a hatch pattern consisting of both horizontal and vertical lines.
HS_DIAGCROSS	Specifies a hatch pattern similar to HS_CROSS but rotated 45 degrees.
HS_FDIAGONAL	Specifies a hatch pattern running from the lower-left to the upper-right at a 45-degree angle.
HS_HORIZONTAL	Specifies a hatch pattern consisting of horizontal lines.
HS_VERTICAL	Specifies a hatch pattern consisting of vertical lines.

Returns	HBRUSH: If successful, the handle of the new brush is returned. Otherwise, NULL is returned.
See Also	CreateBrushIndirect(), CreateDIBPatternBrush(), CreateDIBPatternBrushPt(), CreatePatternBrush(), CreateSolidBrush(), GetStockObject()
Example	The following example creates a hatched brush with the CreateHatchBrush() function that is identical to the brush created in the example shown in Figure 15-3 under the discussion of CombineRgn(). A rectangle is filled with the new brush.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC    hDC;
                           HBRUSH hBrush;
                           RECT   rect;

                           hDC = GetDC( hWnd );

                           hBrush = CreateHatchBrush( HS_DIAGCROSS, RGB( 0, 0, 0 ) );
                           SetRect( &rect, 10, 10, 110, 110 );
                           FillRect( hDC, &rect, hBrush );

                           DeleteObject( hBrush );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

CREATEPATTERNBRUSH

WIN32S WINDOWS 95 WINDOWS NT

Description	CreatePatternBrush() creates a logical brush based on the given bitmap. The bitmap may not have been created using CreateDIBSection().
Syntax	HBRUSH CreatePatternBrush(HBITMAP hBitmap)
Parameters	
hBitmap	HBITMAP: The handle of a bitmap that specifies the pattern to be used in creating the logical brush.
Returns	HBRUSH: If successful, the handle of the new brush is returned; otherwise, NULL is returned.
See Also	CreateBrushIndirect(), CreateDIBPatternBrush(), CreateDIBPatternBrushPt(), CreateHatchBrush(), CreateSolidBrush(), GetStockObject()
Example	The following example creates a brush based on the bitmap identified by the handle hBitmap. A red dashed pen is created, and the pen and brush are selected into a device context and used to draw a rectangle.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC     hDC;
                           HBRUSH  hBrush;
                           HBITMAP hBitmap, hOldBrush;
                           HPEN    hPen, hOldPen;

                           hDC     = GetDC( hWnd );

                           // Create the brush and pen.
                           //.........................
                           hBitmap = LoadBitmap( hInst, "BRUSH" );
                           hBrush  = CreatePatternBrush( hBitmap );
                           hPen    = CreatePen( PS_DASH, 1, RGB( 255, 0, 0 ) );
                           // Paint a rectangle with the new brush and pen.
                           //.............................................
                           hOldBrush = SelectObject( hDC, hBrush );
                           hOldPen   = SelectObject( hDC, hPen );

                           Rectangle( hDC, 10, 10, 110, 110 );

                           SelectObject( hDC, hOldBrush );
                           SelectObject( hDC, hOldPen );

                           DeleteObject( hBrush );
                           DeleteObject( hPen );
                           DeleteObject( hBitmap );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCHDCHDCHDCCREATEPEN WIN32S WINDOW95 WINDOWNT

Description	The CreatePen() function creates a logical pen using the specified width, style, and color. The pen can be used when drawing lines and curves.
Syntax	HPEN CreatePen(int nStyle, int nWidth, COLORREF crColor)
Parameters	
nStyle	int: Specifies a pen style. Use one of the values listed in Table 15-7.
nWidth	int: Specifies the width of the resulting pen, in logical units.
crColor	COLORREF: Specifies the color of the resulting pen.

Table 15-7. Pen Styles for CreatePen()

Pen Style	**Description**
PS_SOLID	The line or curve drawn with this pen will be solid.
PS_DASH	The line or curve drawn with this pen will be a dashed line. You must specify a pen width that is one or less device units when using this style.
PS_DOT	The line or curve drawn with this pen will be a dotted line. You must specify a pen width that is one or less device units when using this style.
PS_DASHDOT	The line or curve drawn with this pen will consist of alternating dashes and dots. You must specify a pen width that is one or less device units when using this style.
PS_DASHDOTDOT	The line or curve drawn with this pen will consist of alternating dashes and double dots. You must specify a pen width that is one or less device units when using this style.
PS_NULL	The line or curve drawn with this pen will be invisible.
PS_INSIDEFRAME	The line or curve drawn with this pen will be solid. If the GDI function used to draw with this pen uses a bounding rectangle as one of its parameters, the figure will be shrunk so that it fits entirely within the bounding rectangle after taking into account the width of the pen. This behaviour applies only to geometric pens.

Returns	HPEN: If successful, the handle of the pen is returned; otherwise, NULL is returned.
See Also	CreatePenIndirect(), GetStockObject()
Example	See CreatePatternBrush() for an example of this function.

CREATEPENINDIRECT WIN32S WINDOWS 95 WINDOWS NT

Description	CreatePenIndirect() creates a logical pen based on the LOGPEN structure specified.
Syntax	HPEN CreatePenIndirect(LOGPEN *lpLogPen)
Parameters	
lpLogPen	LOGPEN *: A pointer to a LOGPEN structure that specifies the width and style of the pen. See the definition of the LOGPEN structure below.
Returns	HPEN: If successful, the handle of the pen is returned; otherwise, NULL is returned.
See Also	CreatePen(), GetStockObject()

LOGPEN Definition

```
typedef struct tagLOGPEN
{
    UINT     lopnStyle;
    POINT    lopnWidth;
    COLORREF lopnColor;
} LOGPEN;
```

lopnStyle	UINT: The style of the pen to create. This must be one of the values listed in Table 15-6.
lopnWidth	POINT: A POINT structure that specifies the width of the pen in logical units.
lopnColor	COLORREF: The color of the pen.
Example	The following code segment creates a red dashed pen identical to the pen created in the CreatePatternBrush() example.

```
LOGPEN lp;
HPEN   hPen;

lp.lopnStyle = PS_DASH;
lp.lopnWidth = 1;
lp.lopnColor = RGB(255, 0, 0);

hPen = CreatePenIndirect( &lp );
```

CREATEPOLYGONRGN WIN32S WINDOWS 95 WINDOWS NT

Description	CreatePolygonRgn() creates a polygon-shaped region based on the set of points specified.
Syntax	HRGN CreatePolygonRgn(CONST POINT *lpPoints, int nPoints, int nFillMode)
Parameters	
lpPoints	CONST POINT *: Points to an array of POINT structures, each of which defines one point in the polygon.
nPoints	int: Contains the number of points in the lpPoints array.
nFillMode	int: Specifies the method Windows uses to determine which points are included in the interior of the region. Refer to Table 15-8 for details. These are the same modes used when drawing a polygon on a device context.

Table 15-8. Polygon Fill Mode Values

Fill Mode	Description
WINDING	GDI fills any region that has a nonzero winding value. This value is defined as the number of times a pen used to draw the polygon would go around the region. The direction of each edge of the polygon is important.
ALTERNATE	GDI fills the area between odd-numbered and even-numbered polygon sides on each scan line – that is GDI fills the area between the first and second side, between the third and fourth side, and so on.

Returns	HRGN: If successful, the handle of the region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn (), CreateEllipticRgnIndirect(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetPolyFillMode(), SetRectRgn()
Example	See CombineRgn() for an example of this function.

CREATEPOLYPOLYGONRGN WIN32S WINDOWS 95 WINDOWS NT

Description	CreatePolyPolygonRgn() creates a region based on a series of polygons defined by the set of points specified.
Syntax	HRGN CreatePolyPolygonRgn(CONST POINT *lpPoints, int *lpPolyPoints, int nPoints, int nFillMode)
Parameters	
lpPoints	CONST POINT *: Points to an array of POINT structures, each of which defines one point in the polygon.
lpPolyPoints	int *: Points to an array of integers. Each integer in the array defines the number of points in lpPoints used for each of the polygons.
nPoints	int: Contains the number of integers in the lpPolyPoints array.
nFillMode	int: Specifies the method Windows uses to determine which points are included in the interior of the region. Refer to Table 15-7 for details. These are the same modes used when drawing a polygon on a device context.
Returns	HRGN: If successful, the handle of the region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn (), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetPolyFillMode(), SetRectRgn()
Example	The following example creates a region that consists of two polygons, the first a triangle and the second a square. Figure 15-4 shows how this region appears when the user selects the Test! menu item and the region is filled.

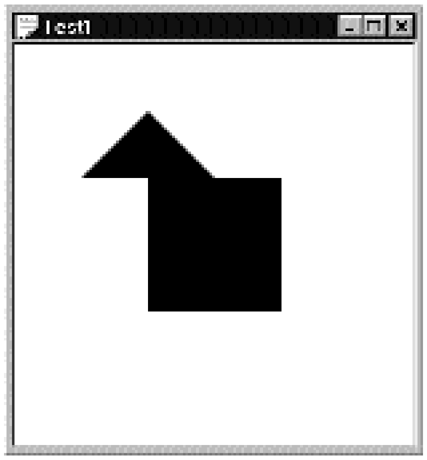

Figure 15-4 CreatePolyPolygonRgn() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC   hDC;
                           HRGN  hRgn;
                           POINT ptList[7];
                           int   nVertexCount[2];

                           hDC     = GetDC( hWnd );

                           // Initialize values.
                           //....................
                           ptList[0].x = 60;  ptList[0].y = 10;
                           ptList[1].x = 10;  ptList[1].y = 60;
                           ptList[2].x = 110; ptList[2].y = 60;
                           ptList[3].x = 160; ptList[3].y = 60;
                           ptList[4].x = 160; ptList[4].y = 160;
                           ptList[5].x = 60;  ptList[5].y = 160;
                           ptList[6].x = 60;  ptList[6].y = 60;

                           nVertexCount[0] = 3;
                           nVertexCount[1] = 4;

                           // Create region and fill it.
                           //...........................
                           hRgn = CreatePolyPolygonRgn( ptList, nVertexCount, 2, WINDING );
                           FillRgn( hDC, hRgn, (HBRUSH)GetStockObject(BLACK_BRUSH) );

                           DeleteObject( hRgn );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCCREATERECTRGN   WIN32S   WINDOWS 95   WINDOWS NT

Description	CreateRectRgn() creates a region based on the given rectangle.
Syntax	HRGN CreateRectRgn(int nLeft, int nRight, int nTop, int nBottom)
Parameters	
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
Returns	HRGN: If successful, the handle of the created region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetRectRgn()
Example	Refer to the discussion of CombineRgn() for an example of this function.

CREATERECTRGNINDIRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	CreateRectRgnIndirect() creates a region based on the given rectangle.
Syntax	HRGN CreateRectRgnIndirect(LPRECT lpRect)
Parameters	

lpRect	LPRECT: A pointer to a RECT structure that defines the bounding rectangle.
Returns	HRGN: If successful, the handle of the created region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRoundRectRgn(), ExtCreateRegion(), InvertRgn(), SetRectRgn()
Example	Refer to the example for the CreateElipticRgnIndirect() fucntion for an example of this function.

CREATEROUNDRECTRGN WIN32S WINDOWS 95 WINDOWS NT

Description	CreateRoundRectRgn() creates a region based on the given rectangle. The corners of the region are rounded.
Syntax	HRGN CreateRoundRectRgn(int nLeft, int nRight, int nTop, int nBottom, int nEllipseWidth, int nEllipseHeight)
Parameters	
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
nEllipseWidth	int: Defines the width of the ellipse used to create the rounded corners.
nEllipseHeight	int: Defines the height of the ellipse used to create the rounded corners.
Returns	HRGN: If successful, the handle of the created region is returned; otherwise, NULL is returned.
See Also	CombineRgn(), CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), ExtCreateRegion(), InvertRgn(), RoundRect(), SetRectRgn()
Example	The following example, shown in Figure 15-5, creates a rectangular region with rounded corners and fills the region with a green solid brush when the user selects the Test! menu item.

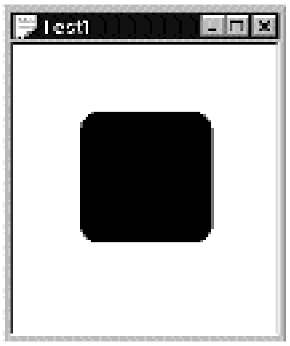

Figure 15-5 CreateRoundRectRgn() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC    hDC;
                           HRGN   hRgn;
                           HBRUSH hBrush;
                           hDC    = GetDC( hWnd );
                           hBrush = CreateSolidBrush( RGB( 0, 255, 0 ) );
                           hRgn   = CreateRoundRectRgn( 10, 10, 110, 110, 30, 30 );

                           FillRgn( hDC, hRgn, hBrush );

                           DeleteObject( hRgn );
                           DeleteObject( hBrush );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCCREATESOLIDBRUSH WIN32S WINDOWS 95 WINDOWS NT

Description	CreateSolidBrush() creates a solid logical brush that has the specified color.
Syntax	HBRUSH CreateSolidBrush(COLORREF crColor)
Parameters	
crColor	COLORREF: Specifies the RGB values for the brush color.
Returns	HBRUSH: If successful, the handle of the logical brush is returned; otherwise, NULL is returned.
See Also	CreateBrushIndirect(), CreateDIBPatternBrush(), CreateHatchBrush(), CreatePatternBrush(), GetStockObject()
Example	See CreateRoundRectRgn() for an example of this function.

DELETEOBJECT WIN32S WINDOWS 95 WINDOWS NT

Description	DeleteObject() deletes a logical GDI object. There are many different functions used to create logical GDI objects. All these objects–except those created with GetStockObject()–must be deleted using this function. The objects that should be deleted include pens, bitmaps, brushes, regions, palettes, and fonts. Don't use DeleteObject() on an object that is currently selected into a device context. Windows 95 will delete an object that is currently selected into a device context, creating a zombie object. Windows NT will fail this call on an object that is currently selected into a device context. If the object being deleted is a pattern brush, the bitmap used to create the brush is not deleted. You must delete the object yourself.
Syntax	BOOL DeleteObject(HGDIOBJ hGdiObject)
Parameters	
hGdiObject	HGDIOBJ: The handle of a GDI object. This must specify the handle of a pen, bitmap, brush, region, palette, or font. The object must not be currently selected into a device context.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CreateStockObject()
Example	See CombineRgn() for an example of this function.

DRAWANIMATEDRECTS WINDOWS 95

Description	DrawAnimatedRect() draws a series of rectangles, giving the appearance of a rectangle either opening or closing.
Syntax	BOOL WINAPI DrawAnimatedRects(HWND hWndClip, int nAniMode, LPCRECT lpRectClosed, LPCRECT lpRectOpen)
Parameters	

hWndClip	HWND: Specifies the handle of a window that will be used to clip the animated rectangles. If this parameter is NULL, then the desktop is used.
nAniMode	int: Specifies the animation mode. The value IDANI_OPEN specifies that the animation moves from closed to open. The value IDANI_CLOSE specifies that the animation moves from open to closed. The value IDANI_CAPTION specifies that the animation rect should include a caption bar.
lpRectClosed	LPCRECT: Points to a rectangle that defines the closed state. This is the starting point when IDANI_OPEN is used and the ending state when IDANI_CLOSE is used.
lpRectOpen	LPCRECT: Points to a rectangle that defines the open state. This is the starting point when IDANI_CLOSE is used, and the ending state when IDANI_OPEN is used.
Returns	BOOL: If successful, returns TRUE. Otherwise, FALSE is returned.
Example	This example animates the closing of a window by drawing successively smaller rectangles and captions, starting with a large rectangle in the upper-left corner of the window, and ending with a rectangle that is the size of the an icon.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rcStart;
                           RECT rcEnd;

                           GetWindowRect( hWnd, &rcStart );

                           rcEnd.left   = rcStart.right-10;
                           rcEnd.top    = rcStart.bottom-10;
                           rcEnd.right  = rcStart.right;
                           rcEnd.bottom = rcStart.bottom;

                           DrawAnimatedRects( hWnd, IDANI_CAPTION, &rcStart, &rcEnd );
                        }
                        break;
         .
         .
         .
```

DRAWCAPTION WINDOWS 95

Description	DrawCaption() draws a caption, as is found in the title bar of a normal window.
Syntax	VOID WINAPI DrawCaption(HDC hDC, LPRECT lpRect, HFONT hFont, HICON hIcon, LPSTR lpCaptionText, WORD wFlags)
Parameters	
hDC	HDC: The handle of a device context into which the drawing will occur.
lpRect	LPRECT: Identifies the bounding rectangle for the caption.
hFont	HFONT: Specifies the font to use when drawing the caption text. If NULL is specified, the normal window caption font will be used.
hIcon	HICON: Specifies the icon that appears at the left edge of the caption. If this parameter is NULL, then no icon is displayed.
lpCaptionText	LPSTR: Points to a null-terminated string that specifies the caption text.
wFlags	WORD: Specifies various drawing flags. This parameter may be set to zero or a combination of one or more of the values shown in Table 15-9.

Table 15-9. Flag Values for DrawCaption()

Flag Value	Description
DC_ACTIVE	Draws the caption using the colors for an active title bar.
DC_SMALLCAP	Draws a small caption.

Returns VOID: This function does not return a value.

DRAWEDGE WINDOWS 95

Description	DrawEdge() draws one or more edges of a rectangle. The edges are drawn using 3D drawing effects.
Syntax	BOOL DrawEdge(HDC hDC, LPRECT lpRect, UINT uEdgeType, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context into which the edge is drawn.
lpRect	LPRECT: Specifies the bounding rectangle that defines the edges.
uEdgeType	EDGE: Specifies the 3D effect to use on the edge. You must specify either one inner edge flag and one outer edge flag, or one edge type flag. Table 15-10 gives a list of these flags.
uFlags	UINT: Specifies additional drawing flags. Specify one or more of the values in Table 15-11.

Table 15-10. Edge Flags Used for the EdgeType Parameter in DrawEdge()

Edge Flag	Description
Inner edge flags	
BDR_RAISEDINNER	Specifies a raised inner edge.
BDR_SUNKENINNER	Specifies a sunken inner edge.
Outer edge flags	
BDR_RAISEDOUTER	Specifies a raised outer edge.
BDR_SUNKENOUTER	Specifies a sunken outer edge.
Edge type flags	
EDGE_BUMP	Equivalant to BDR_RAISEDOUTER and BDR_SUNKENINNER.
EDGE_ETCHED	Equivalant to BDR_SUNKENOUTER and BDR_RAISEDINNER.
EDGE_RAISED	Equivalant to BDR_RAISEDOUTER and BDR_RAISEDINNER.
EDGE_SUNKEN	Equivalant to BDR_SUNKENOUTER and BDR_SUNKENINNER.

Table 15-11. Flag Values Used for the uFlags Parameter in DrawEdge()

Flag	Description
BF_ADJUST	The rectangle is adjusted to leave room for the client area.
BF_BOTTOM	The bottom edge of the rectangle is drawn.
BF_BOTTOMLEFT	The bottom and left edges of the rectangle are drawn.
BF_BOTTOMRIGHT	The bottom and right edges of the rectangle are drawn.
BF_DIAGONAL	A diagonal edge is drawn.
BF_DIAGONAL_ENDBOTTOMLEFT	A diagonal edge is drawn. The starting point is the top-right corner of the rectangle, and the end point is the bottom-left corner of the rectangle.

BF_DIAGONAL_ENDBOTTOMRIGHT	A diagonal edge is drawn. The starting point is the top-left corner of the rectangle, and the end point is the bottom-right corner of the rectangle.
BF_DIAGONAL_ENDTOPLEFT	A diagonal edge is drawn. The starting point is the bottom-right corner of the rectangle, and the end point is the top-left corner of the rectangle.
BF_DIAGONAL_ENDTOPRIGHT	A diagonal edge is drawn. The starting point is the bottom-left corner of the rectangle, and the end point is the top-right corner of the rectangle.
BF_FLAT	A flat edge is drawn.
BF_LEFT	The left edge of the rectangle is drawn.
BF_MIDDLE	The interior of the rectangle is filled.
BF_MONO	A one-dimensional border is drawn.
BF_RECT	All four edges of the rectangle are drawn.
BF_RIGHT	The right edge of the rectangle is drawn.
BF_SOFT	Soft buttons instead of tiles.
BF_TOP	The top edge of the rectangle is drawn.
BF_TOPLEFT	The top and left edges of the rectangle are drawn.
BF_TOPRIGHT	The top and right edges of the rectangle are drawn.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
Example	The following example draws an edge within the client area of the window. The edge is drawn within a rectangle that is 10 pixels smaller than the client area of the window, and the edge is drawn with an etched appearance.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;
                 RECT        rcClient;

                 BeginPaint( hWnd, &ps );

                 // Draw edge
                 //..........
                 GetClientRect( hWnd, &rcClient );
                 InflateRect( &rcClient, -10, -10 );
                 DrawEdge( ps.hdc, &rcClient, EDGE_ETCHED, BF_RECT );

                 EndPaint( hWnd, &ps );
              }
              break;
         .
         .
         .
```

HDCDRAWFOCUSRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	The rectangle is drawn using an XOR function. To remove the rectangle, simply redraw it in the same position.
Syntax	BOOL DrawFocusRect(HDC hDC, LPRECT lpRect)
Parameters	
hDC	HDC: Specifies the device context into which the rectangle will be drawn.
lpRect	LPRECT: Points to a RECT structure that defines the rectangle to be drawn.
Returns	BOOL: Returns TRUE is successful; otherwise, FALSE is returned.

Example	The following example draws a focus rectangle when the user selects the Test! menu item. When the user selects the Test! menu item a second time, the focus rectangle is removed.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rcFocus;
                           HDC  hDC;

                           hDC = GetDC( hWnd );

                           SetRect( &rcFocus, 10, 10, 140, 30 );
                           DrawFocusRect( hDC, &rcFocus );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCHDCDRAWFRAMECONTROL WINDOWS 95 WINDOWS NT

Description	DrawFrame Control() draws one of several predefined frame controls. You may specify various style flags for the different types of frame controls.
Syntax	BOOL DrawFrameControl(HDC hDC, LPRECT lpRect, UINT uControlType, UINT uControlStyle)
Parameters	
hDC	HDC: Specifies the device context into which the control will be drawn.
lpRect	LPRECT: Points to a RECT structure that defines the bounding rectangle within which the control will be drawn.
uControlType	UINT: Specifies one of the predefined control types defined in Table 15-12.
uControlStyle	UINT: Specifies various style flags. The specific style flags depend on the value of uControlType. Refer to Table 15-13 for details.

Table 15-12. Control Types That May Be Specified in Function DrawFrameControl()

Control Type	Description
DFC_BUTTON	Draws a button control.
DFC_CAPTION	Draws a window caption.
DFC_MENU	Draws a menu bar.
DFC_SCROLL	Draws a scrollbar.

Table 15-13. Style Values for FunctionDrawFrameControl()

Style Flags	Description
If ControlType is DFC_BUTTON, ControlStyle can be one of the following values:	
DFCS_BUTTON3STATE	Three-state button

DFCS_BUTTONCHECK	Check box
DFCS_BUTTONPUSH	Pushbutton
DFCS_BUTTONRADIO	Radio button
DFCS_BUTTONRADIOIMAGE	Image for radio button (nonsquare needs image)
DFCS_BUTTONRADIOMASK	Mask for radio button (nonsquare needs mask)
If ControlType is DFC_CAPTION, ControlStyle can be one of the following values:	
DFCS_CAPTIONCLOSE	Close button
DFCS_CAPTIONHELP	Help button
DFCS_CAPTIONMAX	Maximize button
DFCS_CAPTIONMIN	Minimize button
DFCS_CAPTIONRESTORE	Restore button
If ControlType is DFC_MENU, ControlStyle can be one of the following values:	
DFCS_MENUARROW	Submenu arrow
DFCS_MENUBULLET	Bullet
DFCS_MENUCHECK	Checkmark
If ControlType is DFC_SCROLL, ControlStyle can be one of the following values:	
DFCS_SCROLLCOMBOBOX	Combo box scrollbar
DFCS_SCROLLDOWN	Down arrow of scrollbar
DFCS_SCROLLLEFT	Left arrow of scrollbar
DFCS_SCROLLRIGHT	Right arrow of scrollbar
DFCS_SCROLLSIZEGRIP	Size grip in bottom-right corner of window
DFCS_SCROLLUP	Up arrow of scrollbar
The following style can be used to adjust the bounding rectangle of the pushbutton:	
DFCS_ADJUSTRECT	Bounding rectangle is adjusted to exclude the surrounding edge of the pushbutton.
One or more of the following values can be used to set the state of the control to be drawn:	
DFCS_CHECKED	Button is checked.
DFCS_FLAT	Button has a flat border.
DFCS_INACTIVE	Button is inactive (grayed).
DFCS_MONO	Button has a monochrome border.
DFCS_PUSHED	Button is pushed.

Returns BOOL: Returns TRUE if successful; otherwise, FALSE is returned.

Example The following example draws a frame for a normal push button. Note that only the frame of a push button is drawn; an actual push button window is not created.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rcButton;
                           HDC  hDC;

                           hDC = GetDC( hWnd );

                           SetRect( &rcButton, 10, 10, 120, 40 );
                           DrawFrameControl( hDC, &rcButton, DFC_BUTTON, DFCS_BUTTONPUSH );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCDRAWICON WIN32S WINDOWS 95 WINDOWS NT

Description	DrawIcon() draws an icon at the specified location within the device context.
Syntax	BOOL DrawIcon(HDC hDC, int X, int Y, HICON hIcon)
Parameters	
hDC	HDC: Specifies the device context into which the control will be drawn.
X	int: Specifies the x coordinate in logical units for the upper-left corner of the icon.
Y	int: Specifies the y coordinate in logical units for the upper-left corner of the icon.
hIcon	HICON: The handle of the icon to be drawn.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	DrawIconEx()
Example	The following example loads the application's icon and draws it on the client area when the user selects the Test! menu item. The icon is drawn in its normal size using the DrawIcon() function and drawn twice its size with the DrawIconEx() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HICON hIcon;
                           HDC   hDC;

                           hDC = GetDC( hWnd );

                           hIcon = LoadIcon( hInst, lpszAppName );
                           DrawIcon( hDC, 10, 10, hIcon );
                           DrawIconEx( hDC, 50, 10, hIcon, 64, 64, 0, NULL, DI_NORMAL );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCDRAWICONEX WINDOWS 95

Description	DrawIconEx() draws an icon at the specified location within the device context. Additionally, the icon may be stretched or compressed, and various raster operations may be applied.
Syntax	BOOL DrawIconEx(HDC hDC, int X, int Y, HICON hIcon, int cX, int cY, DWORD dwRop, HBRUSH hBrush, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context into which the control will be drawn.
X	int: Specifies the x coordinate in logical units for the upper-left corner of the icon.
Y	int: Specifies the y coordinate in logical units for the upper-left corner of the icon.

hIcon	HICON: The handle of the icon to be drawn.
cX	int: Specifies the resulting width of the icon. The icon will be stretched or compressed to this width.
cY	int: Specifies the resulting height of the icon. The icon will be stretched or compressed to this height.
dwRop	DWORD: Specifies the raster operation to perform. The raster operation is used to determine how GDI combines the bitmap of the icon with the colors currently on the device context. If Flags does not include DI_MASK or DI_IMAGE, then this parameter is ignored.
hBrush	HBRUSH: If one of the raster operations specified by dwRop requires a brush, the brush identified is used.
uFlags	UINT: Specifies various flags that control the operation of this function. Refer to Table 15-14 for details.

Table 15-14. Flag Values for DrawIconEx()

Flag	Description
DI_COMPAT	The icon is drawn using the system-defined image, rather than the user-supplied image.
DI_DEFAULTSIZE	The cX and cY parameters are ignored. The icon is drawn in its original size.
DI_IMAGE	Performs the raster operation defined by dwRop on the icon image.
DI_MASK	Performs the raster operation defined by dwRop on the icon mask.
DI_NORMAL	This flag is a combination of DI_MASK and DI_IMAGE.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	DrawIcon()
Example	See the example for the DrawIcon() function.

HDCDRAWSTATE WINDOWS 95

Description	DrawState() draws an image using various state flags to control the rendering process. In addition, a user-defined callback function can be used to control the drawing of the image.
Syntax	BOOL WINAPI DrawState(HDC hDC, HBRUSH hBrush, DRAWSTATEPROC fnDrawFunction, LPARAM lParam, WPARAM wParam, int X, int Y, int cX, int cY, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context into which the control will be drawn.
hBrush	HBRUSH: If the Flags parameter includes DSS_MONO, the brush identified by this parameter is used.
fnDrawFunction	DRAWSTATEPROC: Identifies a callback function that may be used to draw the image. If uFlags specifies DSS_COMPLEX, then this parameter must be specified. If uFlags specifies DSS_TEXT, then this parameter is optional, and may be NULL. In all other cases this parameter is ignored. See the definition of the callback function below.
lParam	LPARAM: Additional data that depends on the value specified in uFlags. Table 15-15 gives details.
wParam	WPARAM: Additional data that depends on the value specified in uFlags. Table 15-15 gives details.
X	int: Specifies the x coordinate in logical units for the upper-left corner of the image.
Y	int: Specifies the y coordinate in logical units for the upper-left corner of the image.
cX	int: Specifies the width of the image.
cY	int: Specifies the height of the image.

uFlags	UINT: Specifies various flags that control the operation of this function. Refer to Table 15-15 for details. You must specify one image type flag, and one image state flag.

Table 15-15. Type and State Flags for DrawState()

Flag	Description
Image type flags	
DST_BITMAP	The image to be drawn is a bitmap. The low word of lParam contains the bitmap handle.
DST_COMPLEX	This image is a complex image. To render the image, DrawState() calls the defined callback function. Refer to the example for more details on using this type.
DST_ICON	The image to be drawn is an icon. The low word of lParam contains the icon handle.
DST_TEXT	The image to be drawn is a text string. lParam contains a pointer to the text string, and wParam identifies the length of the string. If the length is specified as zero, then the string is assumed to be null terminated.
DST_PREFIXTEXT	Identical to DST_TEXT, except that any ampersand (&) character is interpreted as the standard mnemonic indicator. The character following the ampersand is drawn underlined.
Image state flags	
DSS_NORMAL	The image is drawn without any modification.
DSS_UNION	The image is dithered.
DSS_DISABLED	The image is embossed.
DSS_DEFAULT	The image is drawn bold.
DSS_MONO	The image is drawn using the brush identified by the hBrush parameter.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
Callback Syntax	BOOL CALLBACK **DrawStateProc**(HDC *hdc*, LPARAM *lData*, WPARAM *wData*, int *cX*, int *cY*)
Callback Parameters	
hDC	HDC: The device context in which to draw. This device context is a memory device context with a bitmap selected with dimensions at least as great as those specified by the *cx* and *cy* parameters.
lData	LPARAM: The value passed in the *lParam* parameter of the DrawState() function.
wData	WPARAM: The value passed in the *wParam* parameter of the DrawState() function.
cX	int: The *cX* value passed to the DrawState() function which specifies the width, in device units, of the image.
cY	int: The *cY* value passed to the DrawState() function which specifies the height, in device units, of the image.
Callback Returns	BOOL: Return TRUE if the function succeeds, otherwise, return FALSE.
Example	See the example for the DrawText() function.

HDCDRAWTEXT

WIN32S WINDOWS 95 WINDOWS NT

Description	DrawText() draws the specified text, formatting the output if desired.
Syntax	int DrawText(HDC hDC, LPSTR lpTextString, int nLength, LPRECT lpRect, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context into which the control will be drawn.
lpTextString	LPSTR: A pointer to the string to be drawn.
nLength	int: Specifies the number of characters to draw. If this values is -1, then the string is assumed to be null terminated.
lpRect	LPRECT: Defines the bounding rectangle. The text is formatted within this rectangle.

uFlags	UINT: Specifies various flags that control the formatting process. This parameter may be any combination of the values shown in Table 15-16.

Table 15-16. Flags for DrawText()

Flag	Description
DT_BOTTOM	The text is aligned along the bottom of the specified rectangle. DT_SINGLELINE must also be specified.
DT_CALCRECT	The text is not drawn. Instead, DrawText() calculates the required size of the rectangle. If the specified text contains multiple lines, then the width of the specified rectangle is used, and the height of the rectangle is increased to accommodate the last line of the text. If there is only one line of text, then the right side of the rectangle is expanded to contain the entire string.
DT_CENTER	Each line of text is centered horizontally within the bounding rectangle.
DT_EXPANDTABS	Indicates that tab characters within the text string are to be expanded. The default setting for tabs is eight spaces.
DT_EXTERNALLEADING	Includes the font external leading space when calculating line height. Normally, the line height does not include the external leading space.
DT_LEFT	Each line of text is left-justified.
DT_NOCLIP	Draws the text without clipping. This allows the operation of DrawText() to be somewhat faster.
DT_NOPREFIX	Indicates that ampersand (&) characters within the text are not to be interpreted as prefix characters. Normally, the occurrence of an ampersand causes DrawText() to skip the ampersand and underline the character that follows it.
DT_RIGHT	Each line of text is right-justified.
DT_SINGLELINE	Indicates that carriage returns and linefeeds are to be ignored. The text is drawn in a single line.
DT_TABSTOP	Allows changing the default tab setting. Normally, tabs are set at every eight spaces. Using this value, you may change this spacing. The new tab spacing should be placed in the upper eight bits, bits 15 - 8, of the uFlags parameter.
DT_TOP	Indicates that the text is to be aligned along the top edge of the bounding rectangle. DT_SINGLELINE must be specified when using this flag.
DT_VCENTER	Indicates that the text is to be centered vertically within the bounding rectangle. DT_SINGLELINE must be specified when using this flag.
DT_WORDBREAK	Indicates that lines should be broken at the end of the last word that fits horizontally within the rectangle. The text string is continued with the next word on the next line of the rectangle. Carriage returns and linefeeds also break the line.

Returns	int: The return value is the height of the formatted text.
See Also	DrawTextEx(), TextOut(), ExtTextOut()
Example	The following example draws a string of text centered within the window and a disabled text string when the user selects the Test! menu item.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           RECT rcClient;

                           hDC = GetDC( hWnd );

                           GetClientRect( hWnd, &rcClient );

                           // Draw Normal Text.
                           //..................
                           DrawText( hDC, "Normal String", 13, &rcClient,
                                     DT_SINGLELINE | DT_VCENTER | DT_CENTER );
                           // Draw Disabled Text.
                           //....................
                           DrawState( hDC, NULL, NULL, (LPARAM)"Disabled String", 0,
                                      10, 10, 300, 20, DST_TEXT | DSS_DISABLED );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCDRAWTEXTEX WINDOWS 95

Description DrawTextEx() draws the specified text, formatting the output if desired.

Syntax int DrawTextEx(HDC hDC, LPSTR lpTextString, int nLength, LPRECT lpRect, UINT uFlags, LPDRAWTEXTPARAMS lpDrawTextParams)

Parameters

hDC HDC: Specifies the device context into which the control will be drawn.

lpTextString LPSTR: A pointer to the string to be drawn.

nLength int: Specifies the number of characters to draw. If this parameter is -1, then the text string is assumed to be null terminated.

lpRect LPRECT: Defines the bounding rectangle. The text is formatted within this rectangle.

uFlags UINT: Specifies various flags that control the formatting process. This parameter may be any combination of the values shown in Table 15-16 or Table 15-17.

lpDrawTextParams LPDRAWTEXTPARAMS: A pointer to a DRAWTEXTPARAMS structure that contains additional formatting information and return values. This parameter may be NULL. See the definition of the DRAWTEXTPARAMS structure below.

Table 15-17. Additional Flags for DrawTextEx()

Flag	Description
DT_EDITCONTROL	Formats and displays text as in an edit control. Any partial line that does not fit within the rectangle is not displayed.
DT_END_ELLIPSIS	If the given string does not fit within the rectangle, ellipsis (...) will be used at the end of the string. If DT_MODIFYSTRING is specified, the modified string will be returned in the buffer pointed to by lpTextString. You may not specify both this flag and DT_PATH_ELLIPSIS.
DT_PATH_ELLIPSIS	If the given string does not fit within the rectangle, ellipsis (...) will be used in the middle of the string. If DT_MODIFYSTRING is specified, the modified string will be returned in the buffer pointed to by lpTextString. You may not specify both this flag and DT_END_ELLIPSIS. If the string contains embedded backslash (\) characters, indicating a path name, then as much of the tail of the string (presumably a file name) will be maintained as possible.
DT_MODIFYSTRING	When used with either DT_PATH_ELLIPSIS or DT_END_ELLIPSIS, DT_MODIFYSTRING instructs DrawTextEx() to return the modified string at the buffer pointed to by lpTextString.
DT_RTLREADING	If the font specified in the device context is Arabic or Hebrew, then the text is rendered in a right-to-left order.
DT_TABSTOP	This flag is similar to DT_TABSTOP used with DrawText(), except that the tab stops are set based on the value specified in the structure pointed to by lpDrawTextParams.

Returns int: The return value is the height of the drawn text. Otherwise, zero is returned.

See Also DrawText(), TextOut(), ExtTextOut()

DRAWTEXTPARAMS Definition

```
typedef struct
{
```

```
                        UINT cbSize;
                        int  iTabLength;
                        int  iLeftMargin;
                        int  iRightMargin;
                        UINT uiLengthDrawn;
                    } DRAWTEXTPARAMS, *LPDRAWTEXTPARAMS;
```

cbSize	UINT: The size, in bytes, of the structure.
iTabLength	int: The size of each tab stop, in units equal to the average character width.
iLeftMargin	int: The left margin, in units equal to the average character width.
iRightMargin	int: The right margin, in units equal to the average character width.
uiLengthDrawn	UINT: Receives the number of characters processed by DrawTextEx(), including white-space characters. The number can be the length of the string or the index of the first line that falls below the drawing area. DrawTextEx() always processes the entire string if the DT_NOCLIP formatting flag is specified.
Example	This example demonstrates the DrawTextEx() functions capability to trancate a long string and place ellipsis (...) at the end of the displayable portion of the string. When the user selects the Test! menu item, the long string is displayed on the client area of the window with a bounding rectangle that does not allow the whole string to be displayed.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           RECT rcBound;

                           hDC = GetDC( hWnd );

                           SetRect( &rcBound, 10, 10, 100, 30 );
                           DrawTextEx( hDC, "This is a long sample text string.", -1,
&rcBound,
                                       DT_END_ELLIPSIS | DT_SINGLELINE | DT_VCENTER, NULL );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCELLIPSE

WIN32S WINDOWS 95 WINDOWS NT

Description	Ellipse() draws an ellipse that is bounded by the given rectangle. If the rectangle is a perfect square, then the result will be a circle. The ellipse is outlined with the current pen, and filled with the current brush.
Syntax	BOOL Ellipse(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
hDC	HDC: Specifies the device context that the ellipse is to be drawn in.
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.

Example	The following example draws an ellipse that fills the entire client area of the window. Notice that the direction the ellipse is drawn is changed to clockwise from the default with the SetArcDirection() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;
                 RECT        clRect;
                 int         nArcDir;

                 BeginPaint( hWnd, &ps );

                 GetClientRect( hWnd, &clRect );

                 // Get the current arc direction.
                 //...............................
                 nArcDir = GetArcDirection( ps.hdc );

                 // Set the arc direction to clockwise.
                 //....................................
                 SetArcDirection( ps.hdc, AD_CLOCKWISE );

                 // Draw the ellipse.
                 //..................
                 Ellipse( ps.hdc, clRect.left, clRect.top,
                                  clRect.right, clRect.bottom );

                 // Restore the arc direction.
                 //...........................
                 SetArcDirection( ps.hdc, nArcDir );

                 EndPaint( hWnd, &ps );
              }
              break;
         .
         .
         .
```

HDCHDCENDPAINT   WIN32S   WINDOWS 95   WINDOWS NT

Description	EndPaint() indicates the end of a painting procedure. It must be used after a BeginPaint(), when all painting has been completed. If the caret or cursor was hidden by BeginPaint(), then EndPaint() will restore it.
Syntax	BOOL EndPaint(HWND hWnd, LPPAINTSTRUCT lpPaintStruct)
Parameters	
hWnd	HWND: Window handle of the window to be painted.
lpPaintStruct	LPPAINTSTRUCT: A pointer to a PAINTSTRUCT structure. This should be the same structure that was filled in by the call to BeginPaint().
Returns	BOOL: This function always returns TRUE.
See Also	BeginPaint()
Related Messages	WM_PAINT
Example	See BeginPaint() or Ellipse() for an example of this function.

ENDPATH WINDOWS 95 WINDOWS NT

Description	EndPath() closes the path started with BeginPath() and selects that path as the current path in a device context.
Syntax	BOOL EndPath(HDC hDC)
Parameters	
hDC	HDC: Specifies the device context that contains the path to be closed.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	AbortPath(), BeginPath()
Example	See AbortPath() for an example of this function.

ENUMOBJECTS WIN32S WINDOWS 95 WINDOWS NT

Description	EnumObjects() enumerates the pens or brushes available for the given device context. For each pen or brush, the specified user-defined callback routine is called. The callback routine should return zero to terminate the enumeration and nonzero to continue the enumeration.
Syntax	int EnumObjects(HDC hDC, int nObjectType, GOBJENUMPROC fnEnumProc, LPARAM lParam)
Parameters	
hDC	HDC: Specifies the device context of the objects to be enumerated.
nObjectType	int: Specifies whether pens or brushes are enumerated. This parameter may be either OBJ_PEN or OBJ_BRUSH.
fnEnumProc	GOBJENUMPROC: A pointer to a user-defined callback procedure. See the definition of the callback function below.
lParam	LPARAM: This value is an application-defined 32-bit value. It is passed to the procedure defined by fnEnumProc.
Returns	int: If successful, the last value returned by the callback procedure is returned. Otherwise, -1 is returned.
Callback Syntax	VOID CALLBACK EnumObjectsProc(LPVOID *lpLogObject*, LPARAM *lpData*)
Callback Parameters	
lpLogObject	LPVOID: A pointer to a LOGPEN or LOGBRUSH structure depending on the value of the ObjectType parameter. See the definition of the LOGBRUSH structure under the CreateBrushIndirect() function. See the definition of the LOGPEN structure under the CreatePenIndirect() function.
lpData	LPARAM: The application-defined 32-bit value passed to the EnumObjects() function.
Example	The following example enumerates the logical brushes available in the device context when the user selects the Test! menu item. When the callback is called with a logical brush, it creates a brush of that type and fills a rectangle with the brush.

```
int nHorzPos;

VOID CALLBACK EnumBrushesProc( LPVOID lpObject, LPARAM lParam )
{
   RECT   rcFill;
   HBRUSH hBrush = CreateBrushIndirect( (LPLOGBRUSH)lpObject );

   nHorzPos += 10;

   SetRect( &rcFill, nHorzPos-10, 10, nHorzPos, 50 );
   FillRect( (HDC)lParam, &rcFill, hBrush );

   DeleteObject( hBrush );
}
```

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                          {
                             HDC  hDC;

                             hDC = GetDC( hWnd );

                             nHorzPos = 0;
                             EnumObjects( hDC, OBJ_BRUSH,
                                          (GOBJENUMPROC)EnumBrushesProc, hDC );

                             ReleaseDC( hWnd, hDC );
                          }
                          break;
         .
         .
         .
```

HDCEQUALRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	EqualRect() determines if two rectangles are equal by comparing the four coordinates of each rectangle.
Syntax	BOOL EqualRect(LPRECT lpRect1, LPRECT lpRect2)
Parameters	
lpRect1	LPRECT: Specifies a pointer to one of the rectangles.
lpRect2	LPRECT: Specifies a pointer to the other rectangle.
Returns	BOOL: Returns TRUE if the rectangles are equal; otherwise, FALSE is returned.
Example	The following code segment checks the two existing rectangles to determine if they are equal and performs different operations, depending on the result.

```
RECT Rect1;
RECT Rect2;

if( !IsRectEmpty(&Rect1) && !IsRectEmpty(&Rect2) )
{
    if(EqualRect(&Rect1, &Rect2))
    {
    //  Rectangles are equal.
    }
    else
    {
    // Rectangles are not equal.
    }
}
else
{
//   One of both of the rectangles is empty.
}
```

EQUALRGN   WIN32S   WINDOWS 95   WINDOWS NT

Description	EqualRgn() determines if the two given regions are equal. Regions are considered equal if they have the same size and shape.
Syntax	BOOL EqualRgn(HRGN hRegion1, HRGN hRegion2)

Parameters	
hRegion1	HRGN: Specifies the handle of one of the regions to be compared.
hRegion2	HRGN: Specifies the handle of the other region to be compared.
Returns	BOOL: Returns TRUE if the regions are equal; otherwise, FALSE is returned.
Example	The following code segment checks two existing regions to see if they are equal. Different operations are performed based on the result.

```
HRGN hRegion1;
HRGN hRegion2;

if( EqualRgn(hRegion1, hRegion2) )
{
//  Regions are equal
}
else
{
//  Regions are not equal
}
```

EXCLUDECLIPRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	ExcludeClipRect() excludes the specified rectangle from the current clipping region.
Syntax	int ExcludeClipRect(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
hDC	HDC: Specifies the device context of the clipping region.
nLeft	int: Specifies the x coordinate of the upper-left corner of the bounding rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the bounding rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the bounding rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the bounding rectangle.
Returns	int: This function returns one of the values specified in Table 15-4.
See Also	ExtSelectClipRgn(), GetClipBox(), GetClipRgn(), IntersectClipRect(), OffsetClipRgn(), SelectClipPath(), SelectClipRgn()
Example	See the example for the ExcludeUpdateRgn() function below.

HDCHDCEXCLUDEUPDATERGN   WIN32S   WINDOWS 95   WINDOWS NT

Description	ExcludeUpdateRgn() removes the current update region from the clipping area of the specified window. This has the effect of preventing future drawing in the invalid area on the device. This function is typically used to exclude the current update region from painting that occurs outside the WM_PAINT message.
Syntax	int ExcludeUpdateRgn(HDC hDC, HWND hWindow)
Parameters	
hDC	HDC: Specifies the device context of the clipping region.
hWindow	HWND: Specifies the window handle of the window to operate on.
Returns	int: This function returns one of the values specified in Table 15-4.
See Also	GetUpdateRect(), GetUpdateRgn()
Example	This example uses the ExcludeClipRect() function to exclude a portion of the client area during painting. Also, when the user selects the Test! menu item, an area of the client area is invalidated which is protected with the ExcludeUpdateRgn() when an intersecting rectangle is painted.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
```

```
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;
                 RECT        clRect;

                 BeginPaint( hWnd, &ps );

                 GetClientRect( hWnd, &clRect );

                 // Exclude a small rectangle from the paint.
                 //.........................................
                 ExcludeClipRect( ps.hdc, 0, 0, 50, 50 );
                 FillRect( ps.hdc, &clRect, GetStockObject( LTGRAY_BRUSH ) );

                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                         {
                            RECT rect1, rect2;
                            HRGN hRgn;
                            HDC  hDC;

                            // Invalidate a region of the window.
                            //....................................
                            SetRect( &rect1, 100, 100, 150, 150 );
                            InvalidateRect( hWnd, &rect1, TRUE );

                            // Invalidate an intersecting
                            // elliptic region of the window.
                            //..............................
                            hRgn = CreateEllipticRgn( 80, 80, 130, 130 );
                            InvalidateRgn( hWnd, hRgn, TRUE );
                            DeleteObject( hRgn );

                            hDC = GetDC( hWnd );

                            // Exclude the invalidated region of the window.
                            //..............................................
                            ExcludeUpdateRgn( hDC, hWnd );

                            // Draw an intersecting rectangle with
                            // the invalid region of the window.
                            //...................................
                            SetRect( &rect2, 75, 75, 125, 125 );
                            FillRect( hDC, &rect2, GetStockObject( GRAY_BRUSH ) );

                            ReleaseDC( hWnd, hDC );
                         }
                         break;
         .
         .
         .
```

HDCHDCEXTCREATEPEN   WINDOWS 95   WINDOWS NT

Description ExtCreatePen() creates either a cosmetic or geometric logical pen for which you specify the style. With geometric pens, you may specify width, join, and endcap styles. The cosmetic pen is always one device unit wide and does not support endcap and join styles. Drawing with a cosmetic pen is generally faster than using a geometric pen.

Syntax	HPEN ExtCreatePen(DWORD dwPenStyle, DWORD dwPenWidth, CONST LOGBRUSH *lpLogBrush, DWORD dwStyleCount, DWORD *lpdwStyleArray)

Parameters

dwPenStyle	DWORD: Specifies various type, style, endcap, and join attributes. Combine one flag from each of the categories using an OR operation. The valid flags are shown in Table 15-19.
dwPenWidth	DWORD: Specifies the width of the logical pen. If a geometric pen is used, the width is specified in logical units. If a cosmetic pen is used, the width must be one.
lpLogBrush	CONST LOGBRUSH *: Points to a LOGBRUSH structure. If a geometric pen is used, the attributes from this structure will be used to create the pen. If a cosmetic pen is used, the lbColor member specifies the color of the pen, and the lbStyle member must be set to BS_BOLD. All other members are ignored for cosmetic pens.
dwStyleCount	DWORD: If dwPenStyle does not include PS_USERSTYLE, then this parameter must be zero. If dwPenStyle includes PS_USERSTYLE, then this parameter specifies the number of DWORD entries that exist in the lpStyleArray array.
lpdwStyleArray	DWORD *: A pointer to an array of DWORD style values. This parameter must be NULL if dwPenStyle does not include PS_USERSTYLE. Otherwise, the values in the array are interpreted as follows: The first entry is the length of the first dash, the second entry is the length of the first space, the third entry is the length of the second dash, and so on.

Table 15-19. dwPenStyle Values for ExtCreatePen()

Flag	Description
The following flags specify the pen type	
PS_GEOMETRIC	Creates a geometric pen.
PS_COSMETIC	Create a cosmetic pen.
The following flags specify the pen style	
PS_ALTERNATE	The line or curve drawn with this pen will have every other pixel set. This style is not supported in Windows 95 and is only valid for cosmic pens.
PS_SOLID	The line or curve drawn with this pen will be solid.
PS_DASH	The line or curve drawn with this pen will be a dashed line. You must specify a pen width that is one or less device units when using this style. In Windows 95, this style is not supported for geometric lines.
PS_DOT	The line or curve drawn with this pen will be a dotted line. You must specify a pen width that is one or less device units when using this style. In Windows 95, this style is not supported for geometric lines.
PS_DASHDOT	The line or curve drawn with this pen will consist of alternating dashes and dots. You must specify a pen width that is one or less device units when using this style. In Windows 95, this style is not supported for geometric lines.
PS_DASHDOTDOT	The line or curve drawn with this pen will consist of alternating dashes and double dots. You must specify a pen width that is one or less device units when using this style. In Windows 95, this style is not supported for geometric lines.
PS_NULL	The line or curve drawn with this pen will be invisible.
PS_USERSTYLE	The line or curve drawn with this pen is based on the values specified in the lpStyleArray parameter. This style is not supported in Windows 95.
PS_INSIDEFRAME	The line or curve drawn with this pen will be solid. If the GDI function used to draw with this pens uses a bounding rectangle as one of its parameters, the figure will be shrunk so that it fits entirely within the bounding rectangle after taking into account the width of the pen. This behaviour applies only to geometric pens.
The following flags specify the desired endcap style. These may be used only if PS_GEOMETRIC is also used.	
PS_ENDCAP_ROUND	Endcaps are round.
PS_ENDCAP_SQUARE	Endcaps are square.
PS_ENDCAP_FLAT	Endcaps are flat.

The following flags specify the desired join style. These may be used only if PS_GEOMETRIC is also used.	
PS_JOIN_BEVEL	Joins are beveled.
PS_JOIN_MITER	Joins are mitered if the miter value falls within the value specified by SetMiterLimit(). Otherwise, the joins are beveled.
PS_JOIN_ROUND	Joins are rounded.

Returns	HPEN: If successful, the handle of the logical pen is returned; otherwise, NULL is returned.
See Also	CreatePen(), CreatePenIndirect(), GetStockObject()
Example	The following example creates a gemoetric pen, using rounded endcaps. The pen draws a solid line, using a red solid brush.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC      hDC;
                           HPEN     hPen, hOldPen;
                           LOGBRUSH lb;

                           hDC = GetDC( hWnd );

                           // Create a pen.
                           //...............
                           lb.lbStyle = BS_SOLID;
                           lb.lbColor = RGB( 255, 0, 0 );
                           hPen = ExtCreatePen( PS_GEOMETRIC | PS_SOLID |
                                                PS_ENDCAP_ROUND,
                                                12, &lb, 0, NULL );

                           // Select the pen and draw a line.
                           //................................
                           hOldPen = SelectObject( hDC, hPen );
                           MoveToEx( hDC, 10, 10, NULL );
                           LineTo( hDC, 150, 10 );
                           SelectObject( hDC, hOldPen );

                           DeleteObject( hPen );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

EXTCREATEREGION WINDOWS 95 WINDOWS NT

Description	ExtCreateRegion() creates a region using the given region data and transformation information.
Syntax	HRGN ExtCreateRegion(XFORM *lpXform, DWORD dwRgnCount, RGNDATA *lpRgnData)
Parameters	
lpXform	XFORM*: A pointer to an XFORM structure that defines the desired transformation. If this parameter is NULL, then the identity transform is used. See the definition of the XFORM structure below.
dwRgnCount	DWORD: Specifies the number of bytes pointed to by lpRgnData.
lpRgnData	RGNDATA*: A pointer to a RGNDATA structure. This structure contains information that defines how the region is created. See the definition of the RGNDATA structure below.
Returns	HRGN: Returns the handle to a region if successful; otherwise, NULL is returned.

See Also	CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), InvertRgn(), SetRectRgn()

XFORM Definition

```
typedef struct _XFORM
{
    FLOAT eM11;
    FLOAT eM12;
    FLOAT eM21;
    FLOAT eM22;
    FLOAT eDx;
    FLOAT eDy;
} XFORM;
```

eM11	FLOAT: The horizontal scaling component, the cosine rotation angle, or the horizontal component for reflections.
eM12	FLOAT: The sine of the rotation angle or the horizontal proportionality constant for shear operations.
eM21	FLOAT: The negative sine of the rotation angle or the vertical proportionality constant for shear operations.
eM22	FLOAT: The vertical scaling compenent, the cosine rotation angle, or the vertical reflection component.
eDX	FLOAT: The horizontal translation component.
eDY	FLOAT: The vertical translation component.

RGNDATA Definition

```
typedef struct _RGNDATA
{
    RGNDATAHEADER rdh;
    char          Buffer[1];
} RGNDATA;
```

rch	RGNDATAHEADER: The members of this structure define the properties of the region. See the definition of the RGNDATAHEADER structure below.
Buffer	char[1]: A buffer that contains the RECT structures that make up region.

RGNDATAHEADER Definition

```
typedef struct _RGNDATAHEADER
{
    DWORD dwSize;
    DWORD iType;
    DWORD nCount;
    DWORD nRgnSize;
    RECT  rcBound;
} RGNDATAHEADER;
```

dwSize	DWORD: The size, in bytes, of the header.
iType	DWORD: The type of region. This must be RDH_RECTANGLES.
nCount	DWORD: The number of rectangles that make up the region.
nRgnSize	DWORD: The size of the buffer required to receive the RECT structures that specify the coordinates of the rectangles that make up the region. If the size is not known, this member can be zero.
rcBound	RECT: The bounding rectangle of the region.
Example	The following example creates a rectangular region and fills it with a black brush. A new region that is 50% of the first region is then created with the ExtCreateRegion() function and filled with a gray brush.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           LPRGNDATA lpRgn;
                           XFORM     xForm;
                           HDC       hDC;
                           DWORD     dwSize;
                           HRGN      hRgn, hExtRgn;

                           hDC     = GetDC( hWnd );

                           hRgn = CreateRectRgn( 10, 10, 110, 110 );
                           FillRgn( hDC, hRgn, (HBRUSH)GetStockObject(BLACK_BRUSH) );

                           // Get the region data.
                           //.....................
                           dwSize = GetRegionData( hRgn, 0, NULL );
                           lpRgn = HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, dwSize );
                           GetRegionData( hRgn, dwSize, lpRgn );

                           // Scale the region by 50% and fill it with a gray brush.
                           //.......................................................
                           xForm.eM11 = 0.5F;
                           xForm.eM12 = 0.0F;
                           xForm.eM21 = 0.0F;
                           xForm.eM22 = 0.5F;
                           xForm.eDx  = 1.0F;
                           xForm.eDy  = 1.0F;

                           hExtRgn = ExtCreateRegion( &xForm, dwSize, lpRgn );
                           FillRgn( hDC, hExtRgn, (HBRUSH)GetStockObject(GRAY_BRUSH) );

                           DeleteObject( hRgn );
                           DeleteObject( hExtRgn );
                           HeapFree( GetProcessHeap(), 0, lpRgn );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

EXTFLOODFILL — WIN32S WINDOWS 95 WINDOWS NT

Description ExtFloodFill() fills an area of the given device context using the current brush.

Syntax BOOL ExtFloodFill(HDC hDC, int X, int Y, COLORREF crColor, UINT uFillType)

Parameters

hDC	HDC: A handle to the device context where the fill is to occur.
X	int: Specifies the starting x coordinate, in logical units.
Y	int: Specifies the starting y coordinate, in logical units.
crColor	COLORREF: Specifies the color that defines the area to be filled. The interpretation of this parameter is dependent on the uFillType parameter.
uFillType	UINT: This parameter is used to define the area to be filled. Table 15-19 gives the available values for this parameter.

Table 15-19. Values for the uFillType Parameter for ExtFloodFill()

Fill Type	Description
FLOODFILLBORDER	The crColor parameter specifies a border color. The area within this border is filled. This is identical to the FloodFill() function.
FLOODFILLSURFACE	The crColor parameter specifies a surface color. Filling continues outward in all directions as long as this color is encountered.

Returns BOOL: Returns TRUE if successful; otherwise, FALSE is returned.

See Also FillRect(), FloodFill()

Example The following example draws a rectangle on the client area. When the user clicks the mouse on the client area, the area is filled with a gray brush. If the area within the rectangle is clicked, only the rectangle is filled.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;

                 BeginPaint( hWnd, &ps );
                 Rectangle( ps.hdc, 10, 10, 110, 110 );
                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_LBUTTONDOWN :
              {
                 HBRUSH hOldBrush;
                 HDC    hDC = GetDC( hWnd );

                 hOldBrush = SelectObject( hDC, GetStockObject(GRAY_BRUSH) );

                 ExtFloodFill( hDC, LOWORD(lParam), HIWORD(lParam),
                                   RGB(255,255,255), FLOODFILLSURFACE );

                 SelectObject( hDC, hOldBrush );

                 ReleaseDC( hWnd, hDC );
              }
              break;
         .
         .
         .
```

HDCHDCHDCHDCEXTSELECTCLIPRGN WINDOWS 95 WINDOWS NT

Description ExtSelectClipRgn() creates a new clipping region by combining the current clipping region with the region specified. In addition, you can control the manner in which GDI combines the two regions.

Syntax int ExtSelectClipRgn(HDC hDC, HRGN hRegion, int nCombineMode)

Parameters

hDC HDC: Identifies the device context.

hRegion HRGN: The handle to a region. This region is combined with the current clipping region using the nCombineMode parameter.

nCombineMode int: Specifies the method that GDI is to use when combining the given region with the current clipping region. You may specify any of the mode values shown in Table 15-3.

Returns int: The return value is one of the values specified in Table 15-4.

See Also ExcludeClipRect(), GetClipBox(), GetClipRgn(), IntersectClipRect(), OffsetClipRgn(), SelectClipPath(), SelectClipRgn()

Example The following example creates an elliptic region and rectangular region and sets them as the clipping regions for the device context. The client area of the window is then filled with a gray brush. Only the areas defined by the regions are actually filled.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           HRGN hRgn1, hRgn2;
                           RECT rcClient;

                           hDC     = GetDC( hWnd );

                           // Create a elliptic and rectangular region and
                           // select them as the selected clipping regions.
                           //...............................................
                           hRgn1 = CreateEllipticRgn( 10, 10, 110, 110 );
                           ExtSelectClipRgn( hDC, hRgn1, RGN_COPY );

                           hRgn2 = CreateRectRgn( 40, 40, 150, 150 );
                           ExtSelectClipRgn( hDC, hRgn2, RGN_OR );

                           // Fill the entire client area with gray,
                           // only the regions will be filled in.
                           //....................................
                           GetClientRect( hWnd, &rcClient );
                           FillRect( hDC, &rcClient, GetStockObject(GRAY_BRUSH) );

                           DeleteObject( hRgn1 );
                           DeleteObject( hRgn2 );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
      .
      .
      .
```

HDCHDCFILLPATH WINDOWS 95 WINDOWS NT

Description	FillPath() closes any open path, then fills the path using the current brush and polygon fill mode.
Syntax	BOOL FillPath(HDC hDC)
Parameters	
hDC	HDC: Identifies the device context that contains the desired path.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	BeginPath(), EndPath(), SetPolyFillMode()
Example	The following example fills the current path of the device context using a solid red brush.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HBRUSH hBrush, hOldBrush;
                           HPEN   hPen;
                           int    iROP2;
                           HDC    hDC = GetDC( hWnd );

                           if( BeginPath( hDC ) )
                           {
                              MoveToEx( hDC, 10, 10, NULL );
                              LineTo( hDC, 200, 10 );
```

```
                     LineTo( hDC, 200, 200 );

                     // close final figure
                     // and end the path.
                     //..................
                     CloseFigure( hDC );
                     EndPath( hDC );

                     // Fill the path with a red brush.
                     //...............................
                     iROP2 = SetROP2(hDC, R2_MERGEPENNOT);
                     hPen = SelectObject(hDC, GetStockObject(NULL_PEN));
                     hBrush = CreateSolidBrush( RGB(255,0,0) );
                     hOldBrush = SelectObject(hDC, hBrush );

                     FillPath(hDC);

                     // Restore the DC to its previous state
                     //.....................................
                     SetROP2(hDC, iROP2);
                     SelectObject(hDC, hPen);
                     SelectObject(hDC, hOldBrush);

                     DeleteObject( hBrush );
                  }

                  ReleaseDC( hWnd, hDC );
               }
               break;
.
.
.
```

HDCHDCHDCFILLRECT Win32s Windows 95 Windows NT

Description	FillRect() fills the specified rectangle using the given brush.
Syntax	BOOL FillRect(HDC hDC, LPRECT lpRect, HBRUSH hBrush)
Parameters	
hDC	HDC: Specifies the device context to fill.
lpRect	LPRECT: A pointer to a RECT structure that defines the area to be filled.
hBrush	HBRUSH: The handle of a logical brush that is used to fill the rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ExtFloodFill(), FloodFill(), Rectangle()
Example	See ExcludeClipRect() for an example of this function.

HDCHDCFILLRGN Win32s Windows 95 Windows NT

Description	FillRgn() fills the specified region using the given brush.
Syntax	BOOL FillRect(HDC hDC, HRGN hRegion, HBRUSH hBrush)
Parameters	
hDC	HDC: Specifies the device context to fill.
hRegion	HRGN: Specifies the handle to a region to be filled.
hBrush	HBRUSH: The handle of a logical brush that is used to fill the region.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ExtFloodFill(), FloodFill()
Example	See ExtCreateRegion()for an example of this function.

FLATTENPATH WIN32S WINDOWS 95 WINDOWS NT

Description	FlattenPath() flattens the path in the given device context by converting any curves into line sequences.
Syntax	BOOL FlattenPath(HDC hDC)
Parameters	
hDC	HDC: Identifies the device context that contains the desired path.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	BeginPath(), EndPath()
Example	The following example converts the curves within a bezier curve to lines with the FlattenPath() function when the user selects the Test! menu item.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           POINT  pts[4];
                           int    i;
                           HDC    hDC = GetDC( hWnd );

                           pts[0].x = 10;  pts[0].y = 50;
                           pts[1].x = 30;  pts[1].y = 0;
                           pts[2].x = 60;  pts[2].y = 150;
                           pts[3].x = 120; pts[3].y = 50;

                           // Draw normal bezier.
                           //....................
                           if( BeginPath( hDC ) )
                           {
                              PolyBezier( hDC, pts, 4 );
                              EndPath( hDC );
                              StrokePath( hDC );
                           }

                           // Adjust coordinates.
                           //....................
                           for( i=0; i<4; i++ )
                              pts[i].y += 80;

                           // Draw bezier converted to lines.
                           //...............................
                           if( BeginPath( hDC ) )
                           {
                              PolyBezier( hDC, pts, 4 );
                              EndPath( hDC );

                              FlattenPath( hDC );

                              StrokePath( hDC );
                           }

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCFRAMERECT WIN32S WINDOWS 95 WINDOWS NT

Description	FrameRect() draws the border of the specified rectangle. The interior is not filled. Note that, unlike FrameRgn(), this function always assumes that the border has a width and height of one logical unit.
Syntax	BOOL FrameRect(HDC hDC, LPRECT lpRect, HBRUSH hBrush)
Parameters	
hDC	HDC: Identifies the device context.
lpRect	LPRECT: A pointer to a RECT structure that defines the rectangle.
hBrush	HBRUSH: The handle of a brush that is used to draw the border of the rectangle. The rectangle is always drawn with a width of one logical unit.
Returns	int: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	FrameRgn(), Rectangle(), RoundRect()
Example	See InvertRect() for an example of this function.

FRAMERGN WIN32S WINDOWS 95 WINDOWS NT

Description	FrameRgn() draws the border of the specified region. The interior is not filled.
Syntax	int FrameRgn(HDC hDC, HRGN hRegion, HBRUSH hBrush, int nWidth, int nHeight)
Parameters	
hDC	HDC: Identifies the device context.
hRegion	HRGN: The handle of the region that is to be framed.
hBrush	HBRUSH: The handle of a brush that is used to draw the border of the region.
nWidth	int: Specifies the width of the border, in logical units.
nHeight	int: Specifies the height of the border, in logical units.
Returns	int: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	FrameRect(), PaintRgn()
Example	See InvertRect() for an example of this function.

GETARCDIRECTION WINDOWS 95 WINDOWS NT

Description	GetArcDirection() returns the direction that GDI draws arcs and rectangles.
Syntax	int GetArcDirection(HDC hDC)
Parameters	
hDC	HDC: Specifies the desired device context.
Returns	int: The return value is either AD_CLOCKWISE or AD_COUNTERCLOCKWISE if successful. Otherwise, zero is returned.
See Also	SetArcDirection()
Example	See the example for the Ellipse() function.

HDCHDCHDCGETASPECTRATIOFILTEREX WIN32S WINDOWS 95 WINDOWS NT

Description	GetAspectRatioFilterEx() returns the aspect ratio of the identified device context. The aspect ratio is defined as the ratio between the horizontal and vertical size of a physical pixel on the device.
Syntax	BOOL GetAspectRatioFilterEx(HDC hDC, LPSIZE lpAspectRatio)

Parameters	
hDC	HDC: Identifies the desired device context.
lpAspectRatio	LPSIZE: A pointer to a SIZE structure. GDI will fill in this structure with the aspect ratio filter value.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
Example	The following example retrieves the physical pixel aspect ratio for the specified device and computes a floating-point compensation value that will allow the application to draw perfect circles on the given device.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           SIZE   szAspect;
                           double dRatio = 1.0;
                           HDC    hDC    = GetDC( hWnd );

                           if ( GetAspectRatioFilterEx( hDC, &szAspect ) &&
                                szAspect.cx != szAspect.cy )
                           {
                              dRatio = (double)szAspect.cx / (double)szAspect.cy;
                           }

                           Ellipse( hDC, 10, (int)(10.0*dRatio), 110, (int)(110.0*dRatio) );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

HDCHDCGETBKCOLOR   WIN32S   WINDOWS 95   WINDOWS NT

Description	GetBkColor() returns the background color of the specified device context.
Syntax	COLORREF GetBkColor(HDC hDC)
Parameters	
hDC	HDC: Identifies the desired device context.
Returns	COLORREF: If successful, the return value is the background color of the specified device context. Otherwise, CLR_INVALID is returned.
See Also	SetBkColor()
Example	This example draws a string on the client area of the window with the current background color and mode. The user can change the background color and mode from the Options menu. When the application receives the WM_INITMENU message, the Options menu is initialized with the current background color and mode. Note that this application uses the CS_OWNDC style to create a window with a private device context.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;

                 BeginPaint( hWnd, &ps );
```

```
               TextOut( ps.hdc, 10, 10, "Sample Text", 11 );

               EndPaint( hWnd, &ps );
            }
            break;

    case WM_INITMENU :
            {
               HDC hDC = GetDC( hWnd );

               if ( GetBkColor( hDC ) == RGB( 255, 255, 255 ) )
                  CheckMenuRadioItem( (HMENU)wParam, IDM_WHITE, IDM_BLACK,
                                                     IDM_WHITE, MF_BYCOMMAND );
               else if ( GetBkColor( hDC ) == RGB( 192, 192, 192 ) )
                  CheckMenuRadioItem( (HMENU)wParam, IDM_WHITE, IDM_BLACK,
                                                     IDM_GRAY, MF_BYCOMMAND );
               else if ( GetBkColor( hDC ) == RGB( 0, 0, 0 ) )
                  CheckMenuRadioItem( (HMENU)wParam, IDM_WHITE, IDM_BLACK,
                                                     IDM_BLACK, MF_BYCOMMAND );

               if ( GetBkMode( hDC ) == OPAQUE )
                  CheckMenuRadioItem( (HMENU)wParam, IDM_OPAQUE, IDM_TRANSPARENT,
                                                     IDM_OPAQUE, MF_BYCOMMAND );
               else
                  CheckMenuRadioItem( (HMENU)wParam, IDM_OPAQUE, IDM_TRANSPARENT,
                                                     IDM_TRANSPARENT, MF_BYCOMMAND );
               ReleaseDC( hWnd, hDC );
            }
            break;

    case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
               case IDM_WHITE :
               case IDM_GRAY :
               case IDM_BLACK :
               case IDM_OPAQUE :
               case IDM_TRANSPARENT :
                  {
                     HDC hDC = GetDC( hWnd );

                     switch( LOWORD( wParam ) )
                     {
                        case IDM_WHITE : SetBkColor( hDC, RGB( 255, 255, 255 ) ); break;
                        case IDM_GRAY  : SetBkColor( hDC, RGB( 192, 192, 192 ) ); break;
                        case IDM_BLACK : SetBkColor( hDC, RGB( 0, 0, 0 ) );       break;

                        case IDM_OPAQUE      : SetBkMode( hDC, OPAQUE );      break;
                        case IDM_TRANSPARENT : SetBkMode( hDC, TRANSPARENT ); break;
                     }

                     ReleaseDC( hWnd, hDC );
                     InvalidateRect( hWnd, NULL, TRUE );
                  }
                  break;
       .
       .
       .
```

HDCHDCHDCGETBKMODE WIN32S WINDOWS 95 WINDOWS NT

Description	GetBkMode() returns the current background mode of the specified device context. The background mode determines how the background color is used.
Syntax	int GetBkMode(HDC hDC)
Parameters	
hDC	HDC: Identifies the desired device context.

Returns	int: If successful, the return value is the background mode of the desired device context. This value will be either OPAQUE or TRANSPARENT. Otherwise, zero is returned.
See Also	SetBkMode()
Example	See GetBkColor() for an example of this function.

GETBOUNDSRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	Windows maintains an accumulated bounding rectangle for a device context. When enabled, this bounding rectangle is enlarged any time a drawing function extends beyond the current bounding rectangle. The current bounding rectangle thus defines the smallest rectangle that contains all drawing within the device context. This routine returns the current bounding rectangle.
Syntax	UINT GetBoundsRect(HDC hDC, LPRECT lpRect, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context.
lpRect	LPRECT: A pointer to a RECT structure. This structure will be filled in upon return from this function.
uFlags	UINT: Specifies various flags that define how GetBoundsRect() operates. Table 15-20 gives details.

Table 15-20. Flag Values for the uFlags Parameter of GetBoundsRect()

Flag	Description
DCB_RESET	If this flag is specified, the current accumulated bounding rectangle will be cleared. If this flag is not set, the current accumulated bounding rectangle is not cleared.

Returns	UINT: The return values specifies various conditions that may have occurred during the processing of this function. One of the values in Table 15-21 may be returned.

Table 15-21. Return Values for GetBoundsRect()

Return Value	Description
0	The indicated device context is invalid.
DCB_DISABLE	Boundary accumulation has been disabled.
DCB_ENABLE	Boundary accumulation has been enabled.
DCB_RESET	The bounding rectangle is currently empty.
DCB_SET	The bounding rectangle is currently not empty.

See Also	SetBoundsRect()
Example	This example clears the bounds rectangle and adds two overlapping rectangles to the bounding rectangle by accumulating them with the SetBoundsRect() function when the user selects the Test! menu item. The resulting bounding rectangle is retrieved with the GetBroundsRect() function and framed.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rect;
                           HDC  hDC = GetDC( hWnd );

                           // Clear the bounding rectangle.
                           //..............................
                           SetBoundsRect( hDC, NULL, DCB_RESET | DCB_ENABLE );

                           // Add first rectangle to bounds.
                           //...............................
                           SetRect( &rect, 10, 10, 50, 50 );
                           SetBoundsRect( hDC, &rect, DCB_ACCUMULATE );
                           FrameRect( hDC, &rect, GetStockObject( LTGRAY_BRUSH ) );

                           // Add second rectangle to bounds.
                           //................................
                           SetRect( &rect, 40, 40, 110, 110 );
                           SetBoundsRect( hDC, &rect, DCB_ACCUMULATE );
                           FrameRect( hDC, &rect, GetStockObject( LTGRAY_BRUSH ) );

                           // Retrieve the accumulated bounds rectangle and frame it.
                           //........................................
                           GetBoundsRect( hDC, &rect, 0 );
                           FrameRect( hDC, &rect, GetStockObject( BLACK_BRUSH ) );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

GETBRUSHORGEX WIN32S WINDOWS95 WINDOWSNT

Description	GDI maintains a brush origin that defines how a brush is aligned before it is used for painting. The GetBrushOrgEx() function returns the current setting of the brush origin.
Syntax	BOOL GetBrushOrgEx(HDC hDC, LPPOINT lpPoint)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to a POINT structure. This structure will receive the current brush origin.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	SetBrushOrgEx()
Example	The following code segment retrieves and saves the current brush origin. It then sets the brush origin, performs some drawing function, and resets the brush origin to its original state.

```
POINT ptOldOrigin;
HDC   hDC;

GetBrushOrgEx( hDC, &ptOldOrigin );
SetBrushOrgEx( hDC, 3, 3, NULL );
.
. //Draw with the brush.
.
SetBrushOrgEx( hDC, ptOldOrigin.x, ptOldOrigin.y, NULL );
```

GETCLIPBOX

WIN32S WINDOWS 95 WINDOWS NT

Description	GetClipBox() returns the smallest rectangle that completely surrounds the currently visible portion of the window. This is based on the current clipping region as well as any overlapping windows. (GetClipRgn() may be more accurate.)
Syntax	int GetClipBox(HDC hDC, LPRECT lpRect)
Parameters	
hDC	HDC: Specifies the device context.
lpRect	LPRECT: A pointer to a RECT structure. This structure will receive the rectangle coordinates.
Returns	int: The return value will be one of the values specified in Table 15-3.
See Also	ExcludeClipRect(), ExtSelectClipRgn(), GetClipRgn(), IntersectClipRect(), OffsetClipRgn(), SelectClipPath(), SelectClipRgn()
Example	The following example creates a region and selects it as the clipping region. The clipping box is then retrieved with the GetClipBox() function which is then framed to make it visible.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                      {
                         HRGN hRgn;
                         RECT rect;
                         HDC  hDC = GetDC( hWnd );

                         // Create a region and select it as the clipping region.
                         //..................................................
                         hRgn = CreateRectRgn( 10, 10, 110, 110 );
                         SelectClipRgn( hDC, hRgn );

                         // Get the clipping box and frame it.
                         //...................................
                         GetClipBox( hDC, &rect );
                         FrameRect( hDC, &rect, GetStockObject( BLACK_BRUSH ) );

                         DeleteObject( hRgn );
                         ReleaseDC( hWnd, hDC );
                      }
                      break;
         .
         .
         .
```

GETCLIPRGN

WINDOWS 95 WINDOWS NT

Description	GetClipRgn() returns a copy of the current clipping region. This function is similar to the GetClipBox() function except that it returns a more accurate bound for the clipping area.
Syntax	int GetClipRgn(HDC hDC, HRGN hRegion)
Parameters	
hDC	HDC: Specifies the device context.
hRegion	HRGN: Specifies a valid handle to an existing region. Upon return from this function, this handle will represent a copy of the clipping region of the specified device context.
Returns	int: This function will return -1 on errors, 1 if successful, and 0 if there is no clipping region for the given device context.

See Also ExcludeClipRect(), ExtSelectClipRgn(), GetClipBox(), IntersectClipRect(), OffsetClipRgn(), SelectClipPath(), SelectClipRgn()

Example This example demonstrates the accuracy of the GetClipRgn() function by creating a path and selecting it as the clipping area then the application retrieves the clipping region which is then filled.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HRGN hRgn;
                           HDC  hDC = GetDC( hWnd );

                           // Build triangular path.
                           //......................
                           BeginPath( hDC );
                           MoveToEx( hDC, 10, 10, NULL );
                           LineTo( hDC, 110, 10 );
                           LineTo( hDC, 110, 110 );
                           CloseFigure( hDC );
                           EndPath( hDC );

                           // Select path as clipping area.
                           //.............................
                           SelectClipPath( hDC, RGN_COPY );

                           // Create a dummy region.
                           //......................
                           hRgn = CreateRectRgn( 10, 10, 11, 11 );

                           // Get the clipping region and fill it.
                           //.....................................
                           GetClipRgn( hDC, hRgn );
                           FillRgn( hDC, hRgn, GetStockObject( BLACK_BRUSH ) );

                           DeleteObject( hRgn );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

GETCURRENTPOSITIONEX WIN32S WINDOWS 95 WINDOWS NT

Description GetCurrentPositionEx() returns the current position within the device context. The current position can be set with MoveToEx() and changed, using a variety of drawing functions, such as LineTo().

Syntax BOOL GetCurrentPositionEx(HDC hDC, LPPOINT lpPoint)

Parameters

hDC HDC: Specifies the device context.

lpPoint LPPOINT: A pointer to a POINT structure. This structure will receive the coordinates of the current position within the specified device context.

Returns BOOL: Returns TRUE if successful; otherwise, FALSE is returned.

See Also MoveToEx()

Example The following code segment retrieves and saves the current position of the specified device context. It then draws a line, which alters the current position. The current position is then set back to its original value.

```
POINT CurrentPoint;
HDC hDC;

if( GetCurrentPositionEx( hDC, &CurrentPoint ) )
{
    LineTo( hDC, 100, 100 );
    MoveToEx( hDC, CurrentPoint.x, CurrentPoint.y, NULL );
}
```

GETMITERLIMIT WINDOWS 95 WINDOWS NT

Description GetMiterLimit() retrieves the miter limit for the specified device context. GDI will draw miter joins only if they fall within the current miter limit. Joins that are outside the current miter limit are drawn as a bevel join.

Syntax BOOL GetMiterLimit(HDC hDC, PFLOAT pfLimit)

Parameters

hDC HDC: Specifies a device context.

pfLimit PFLOAT: A pointer to a floating-point value. This value will receive the current miter limit.

Returns BOOL: Returns TRUE if successful; otherwise, FALSE is returned.

See Also SetMiterLimit()

Example This example demonstrates the miter limit when stroking a path with a mitered pen. Each time the user selects the Test! menu item, the miter limit is increased by 1 until the miter limit reaches 3 where it is reset. The GetMiterLimit() function is used to retrieve the current miter limit so it can be restored after the path is stroked. Another alternative would be to use the third parameter of the SetMiterLimit() function to return the current miter limit when the new one is set.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
static FLOAT fMiterLimit = 0.0f;

   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           LOGBRUSH lb;
                           HPEN     hPen, hOldPen;
                           FLOAT    fOldMiter;
                           HDC      hDC = GetDC( hWnd );

                           // Build triangular path.
                           //.......................
                           BeginPath( hDC );
                           MoveToEx( hDC, 20, 20, NULL );
                           LineTo( hDC, 120, 20 );
                           LineTo( hDC, 120, 120 );
                           CloseFigure( hDC );
                           EndPath( hDC );

                           // Retrieve the current miter limit.
                           //..................................
                           GetMiterLimit( hDC, &fOldMiter );

                           // Reset the miter limit.
                           //.......................
```

```
                  if ( fMiterLimit == 3.0f )
                     fMiterLimit = 0.0f;

                  fMiterLimit += 1.0f;

                  lb.lbStyle = BS_SOLID;
                  lb.lbColor = RGB( 255, 0, 0 );
                  hPen = ExtCreatePen( PS_GEOMETRIC | PS_SOLID |
                                       PS_ENDCAP_ROUND | PS_JOIN_MITER,
                                       10, &lb, 0, NULL );

                  hOldPen = SelectObject( hDC, hPen );

                  // Set the miter limit, stroke the path,
                  // and restore the miter limit.
                  //....................................
                  SetMiterLimit( hDC, fMiterLimit, NULL );
                  StrokePath( hDC );
                  SetMiterLimit( hDC, fOldMiter, NULL );

                  SelectObject( hDC, hOldPen );

                  DeleteObject( hPen );
                  ReleaseDC( hWnd, hDC );
               }
               break;
      .
      .
      .
```

GETOBJECT — WIN32S WINDOWS 95 WINDOWS NT

Description GetObject() returns information about a bitmap, palette, logical pen, brush, font, or DIB section.

Syntax int GetObject(HGDIOBJ hGdiObj, int nBuffer, LPVOID lpBuffer)

Parameters

hGdiObj HGDIOBJ: Specifies the handle of a GDI object. This should be the handle of a logical bitmap, DIB section (created with CreateDIBSection()), pen, palette, brush, or font.

nBuffer int: The size, in bytes, of the buffer pointed to by lpBuffer.

lpBuffer LPVOID: A pointer to a buffer. The data returned is dependent on the value of hGdiObj. Refer to Table 15-22 for details. This parameter may be NULL.

Table 15-22. Data Types for Function GetObject()

Type of Object	Buffer Contents
Logical Bitmap	The buffer will contain a BITMAP structure. Only the height, width, and color format information is valid.
DIB Section	The buffer will contain a DIBSECTION structure.
Logical Pen	The buffer will contain either an EXTLOGPEN structure or a LOGPEN structure, depending on the type of pen.
Logical Brush	The buffer will contain a LOGBRUSH structure.
Logical Font	The buffer will contain a LOGFONT structure.
Palette	The buffer will contain a WORD structure specifying the number of palette entries that exist within the palette. You can use GetSystemPaletteEntries() to retrieve the values.

Returns int: If successful, GetObject() returns the number of bytes copied into the buffer. If lpBuffer is NULL, the return value is the required size of the buffer. In the case of an error, the return value is zero.

Example The following example retrieves the BITMAP information for a bitmap and displays the height and width of the bitmap in a message box..

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           TCHAR   szMsg[64];
                           BITMAP  bmp;
                           HBITMAP hBmp = LoadBitmap( hInst, "TestBitmap" );

                           GetObject( hBmp, sizeof( BITMAP ), &bmp );

                           wsprintf( szMsg, "Bitmap Height = %d, Width = %d",
                                            bmp.bmHeight, bmp.bmWidth );

                           MessageBox( hWnd, szMsg, lpszAppName, MB_OK );
                        }
                        break;
         .
         .
         .
```

GETPATH WINDOWS 95 WINDOWS NT

Description	GetPath() returns line endpoints and curve control points describing the path currently defined in the specified device context.
Syntax	int GetPath(HDC hDC, LPPOINT lpPoints, LPBYTE lpTypeArray, int nCount)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoints	LPPOINT: A pointer to an array of POINT structures. This structure will receive the coordinates of points that define the path. Each point in this array is accompanied by one entry in the lpTypeArray. The values in the lpTypeArray specify how these points should be interpreted.
lpTypeArray	LPBYTE: A pointer to an array of BYTE values. Each entry in this array corresponds to one entry in the lpPoints array. The values in the lpPoints array are interpreted based on the corresponding value in the lpTypeArray. Table 15-23 shows these values and their interpretation.
nCount	int: Specifies the number of entries in the lpPoints and lpTypeArray arrays. These two arrays must be the same size.

Table 15-23. Type Values Returned by GetPath()

Type Flag	Description
PT_MOVETO	Indicates that the associated point in the lpPoints array is the first point of a line or curve.
PT_LINETO	Indicates that the associated point in the lpPoints array is the second point in a line. The previous point in the lpPoints array is the first point in the line.
PT_BEZIERTO	Indicates that the associated point in the lpPoints array is part of a bezier curve. PT_BEZIERTO points always occur in groups of three. The point before the PT_BEZIERTO points defines the start of the curve. The first two PT_BEZIERTO points define the control points, and the final PT_BEZIERTO point defines the endpoint.
PT_CLOSEFIGURE	This value, when combined with PT_LINETO or PT_BEZIERTO, indicates that the figure should be closed after drawing the line or curve. The figure is closed by drawing a line from the last point in the figure to the point specified by the most recent PT_MOVETO.

Returns	int: If nCount is zero, the return value is the number of entries in the arrays that need to be allocated to contain all data. If nCount is not zero, then the return value will be the number of items copied into the arrays, if there is sufficient space. If there is not sufficient space, the return value will be -1.
See Also	BeginPath(), EndPath()
Example	The following function determines the amount of data necessary to contain the current path information, allocates the proper space, and then obtains the information for the path.

```
int GetPathInfo( HDC hDC, LPPOINT* plpPointAry, LPBYTE* plpByteAry )
{
   int nPointCount;

   // Retrieve the number of points in path.
   //.....................................
   nPointCount = GetPath( hDC, NULL, NULL, 0 );

   // Allocate memory, the calling application is responsible fore freeing this memory.
   //....................................................................................
   *plpPointAry = HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, nPointCount*sizeof(POINT) );
   *plpByteAry = HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, nPointCount );

   // Retrieve the path information.
   //..............................
   GetPath( hDC, *plpPointAry, *plpByteAry, nPointCount );

   return( nPointCount );
}
```

GETPIXEL WIN32S WINDOWS 95 WINDOWS NT

Description	GetPixel() returns the RGB value of the pixel specified. The specified pixel must be within the current clipping region.
Syntax	COLORREF GetPixel(HDC hDC, int X, int Y)
Parameters	
hDC	HDC: Specifies the desired device context.
X	int: Identifies the x coordinate of the desired pixel.
Y	int: Identifies the y coordinate of the desired pixel.
Returns	COLORREF: The return value is the RGB value of the specified pixel if successful. Otherwise, the return value is CLR_INVALID.
See Also	SetPixelV()
Example	See SetPixelV() for an example of this function.

GETPOLYFILLMODE WIN32S WINDOWS 95 WINDOWS NT

Description	GetPolyFillMode() returns the polygon fill mode for the specified device context. The polygon fill mode determines how GDI fills polygons.
Syntax	int GetPolyFillMode(HDC hDC)
Parameters	
hDC	HDC: Specifies the desired device context.
Returns	int: The return value is one of the valid polygon fill modes. Valid values are defined in Table 15-8.
See Also	CreatePolygonRgn(), SetPolyFillMode()
Example	This example paints a five pointed star, as shown in Figure 15-6, using the Polygon() function and fills the polygon with the current polygon fill mode. The user can change the fill mode with the menu from winding to alterternate and back again.

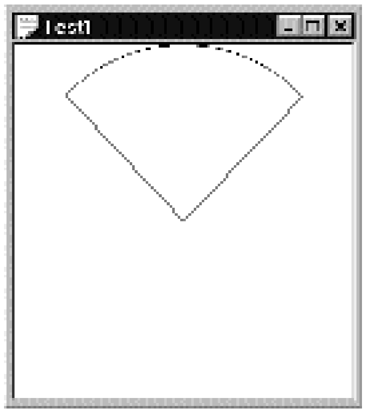

Figure 15-6 GetPolyFillMode() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
static POINT pt[6];

   switch( uMsg )
   {
      case WM_CREATE :
              pt[0].x = 60;  pt[0].y = 10;
              pt[1].x = 20;  pt[1].y = 100;
              pt[2].x = 115; pt[2].y = 42;
              pt[3].x = 5;   pt[3].y = 42;
              pt[4].x = 100; pt[4].y = 100;
              pt[5].x = 60;  pt[5].y = 10;

              SelectObject( GetDC( hWnd ), GetStockObject( GRAY_BRUSH ) );
              break;

      case WM_PAINT :
              {
                 PAINTSTRUCT ps;

                 BeginPaint( hWnd, &ps );
                 Polygon( ps.hdc, pt, 6 );
                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_INITMENU :
              if ( GetPolyFillMode( GetDC( hWnd ) ) == WINDING )
                 CheckMenuRadioItem( (HMENU)wParam, IDM_WINDING, IDM_ALTERNATE,
                                              IDM_WINDING, MF_BYCOMMAND );
              else
                 CheckMenuRadioItem( (HMENU)wParam, IDM_WINDING, IDM_ALTERNATE,
                                              IDM_ALTERNATE, MF_BYCOMMAND );
              break;

      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_ALTERNATE :
                 case IDM_WINDING :
                       {
                          HDC hDC = GetDC( hWnd );
                          SetPolyFillMode( GetDC( hWnd ), LOWORD( wParam ) == IDM_WINDING ?
                                           WINDING : ALTERNATE );

                          // Re-paint window.
                          //..................
                          RedrawWindow( hWnd, NULL, NULL, RDW_ERASE | RDW_INVALIDATE );
                       }
```

```
                    break;
         .
         .
         .
```

GETREGIONDATA WINDOWS 95 WINDOWS NT

Description	GetRegionData() returns information that describes the specified region.
Syntax	DWORD GetRegionData(HRGN hRegion, DWORD dwCount, RGNDATA *lpRgnData)
Parameters	
hRegion	HRGN: Specifies the handle to a valid region.
dwCount	DWORD: Specifies the size, in bytes, of the buffer pointed to by lpRgnData.
lpRgnData	RGNDATA *: A pointer to a RGNDATA structure. This parameter may be NULL. See the definition of the RGNDATA structure under the ExtCreateRegion() function.
Returns	DWORD: If lpRgnData is NULL, or if dwCount specifies a size that is not sufficient to contain all the data, then the return value is the size, in bytes, of the required buffer. Otherwise, the return value is 1 if the function is successful, and 0 if an error occurs.
See Also	ExtCreateRegion(), GetRgnBox()
Example	See ExtCreateRegion() for an example of this function.

GETRGNBOX WIN32S WINDOWS 95 WINDOWS NT

Description	GetRgnBox() obtains the smallest rectangle that completely surrounds the given region.
Syntax	int GetRgnBox(HRGN hRegion, LPRECT lpRect)
Parameters	
hRegion	HRGN: Specifies a handle to the desired region.
lpRect	LPRECT: A pointer to a RECT structure. This structure will receive the coordinates of the bounding rectangle.
Returns	int: The return value can be one of the region complexity values specified in Table 15-4.
See Also	GetRegionData()
Example	The following example creates an elliptic region and retrieves the bounding rectangle. The elliptic region is then filled and the bounding rectangle framed.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           RECT rect;
                           HRGN hElRgn;

                           hDC = GetDC( hWnd );

                           hElRgn = CreateEllipticRgn( 10, 10, 110, 110 );
                           GetRgnBox( hElRgn, &rect );

                           FillRgn( hDC, hElRgn, GetStockObject( GRAY_BRUSH ) );
                           FrameRect( hDC, &rect, GetStockObject( BLACK_BRUSH ) );

                           DeleteObject( hElRgn );
                           ReleaseDC( hWnd, hDC );
```

```
                    }
                    break;
         .
         .
         .
```

GETROP2 WIN32S WINDOWS 95 WINDOWS NT

Description	GetROP2() returns the current raster operation that defines how GDI combines foreground colors.
Syntax	int GetROP2(HDC hDC)
Parameters	
hDC	HDC: Specifies the desired device context.
Returns	int: The return value is one of the raster operation codes listed in Table 15-24.

Table 15-24. Return Values for GetROP2()

Raster Operation Code	Description
R2_BLACK	Pixel is always 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
R2_MERGEPEN	Pixel is a combination of the pen color and the screen color.
R2_MERGEPENNOT	Pixel is a combination of the pen color and the inverse of the screen color.
R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the screen color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen.

See Also SetROP2()

Example The following example fills an elliptic region with a gray brush and draws an overlapping rectangle with a hollow brush. The current raster operation setting is saved while it is changed to R2_NOT while drawing the rectangle. The raster operation setting is restored after drawing has completed. Note that the current raster operation could be saved using the return value of the SetROP2() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HBRUSH hOldBrush;
                           HDC    hDC;
                           HRGN   hElRgn;
```

```
                    int    nOldROP;

                    hDC     = GetDC( hWnd );
                    nOldROP = GetROP2( hDC );
                    hElRgn  = CreateEllipticRgn( 10, 10, 110, 110 );

                    FillRgn( hDC, hElRgn, GetStockObject( GRAY_BRUSH ) );

                    // Draw hollow rectangle with ROP2 = R2_NOT.
                    //........................................
                    hOldBrush = SelectObject( hDC, GetStockObject( HOLLOW_BRUSH  ) );
                    SetROP2( hDC, R2_NOT );
                    Rectangle( hDC, 30, 30, 130, 130 );

                    // Clean up.
                    //..........
                    SelectObject( hDC, hOldBrush );
                    SetROP2( hDC, nOldROP );
                    DeleteObject( hElRgn );
                    ReleaseDC( hWnd, hDC );
                 }
                 break;
     .
     .
     .
```

GETSTOCKOBJECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	GetStockObject() returns the handle to one of the pre-defined system stock objects. These objects exist within the system at all times, and you don't need extra resources to create them. You do not have to delete objects whose handles are obtained using this function.
Syntax	HGDIOBJ GetStockObject(int nObjectType)
Parameters	
nObjectType	int: Specifies one of the valid stock objects. This parameter can be one of the values listed in Table 15-25.

Table 15-25. Object Types Specified with GetStockObject()

Object Type	Description
BLACK_BRUSH	Black brush.
DKGRAY_BRUSH	Dark gray brush. This should be used only in windows that have CS_VREDRAW and CS_HREDRAW class styles.
GRAY_BRUSH	Gray brush. This should be used only in windows that have CS_VREDRAW and CS_HREDRAW class styles.
HOLLOW_BRUSH	Hollow brush (equivalent to NULL_BRUSH).
LTGRAY_BRUSH	Light gray brush. This should be used only in windows that have CS_VREDRAW and CS_HREDRAW class styles.
NULL_BRUSH	Null brush (equivalent to HOLLOW_BRUSH).
WHITE_BRUSH	White brush.
BLACK_PEN	Black pen.
NULL_PEN	Null pen.
WHITE_PEN	White pen.
ANSI_FIXED_FONT	Windows fixed-pitch (monospace) system font.
ANSI_VAR_FONT	Windows variable-pitch (proportional space) system font.
DEVICE_DEFAULT_FONT	Device-dependent font. This is only valid for Windows NT
DEFAULT_GUI_FONT	Default font for user interface objects such as menus and dialog boxes. This object may change at any time due to user preference settings. This is only valid in Windows 95.
OEM_FIXED_FONT	Original equipment manufacturer (OEM) dependent fixed-pitch (monospace) font.

SYSTEM_FONT	System font. By default, Windows uses the system font to draw menus, dialog box controls, and text. In Windows versions 3.0 and later, the system font is a proportionally spaced font; earlier versions of Windows used a monospace system font.
SYSTEM_FIXED_FONT	Fixed-pitch (monospace) system font used in Windows versions earlier than 3.0. This stock object is available for compatibility with earlier versions of Windows.
DEFAULT_PALETTE	Default palette. This palette consists of the static colors in the system palette.

Returns	HGDIOBJ: The return value is a handle to the desired stock object.
See Also	CreateBrushIndirect(), CreateDIBPatternBrush(), CreateDIBPatternBrushPt(), CreateHatchBrush(), CreatePatternBrush(), CreatePen(), CreatePenIndirect(), CreateSolidBrush(), ExtCreatePen()
Example	See GetRgnBox() and GetROP2() for examples of this function.

GETUPDATERECT WIN32S WINDOWS 95 WINDOWS NT

Description	GetUpdateRect() retrieves the smallest rectangle that completely contains the current update region of the specified window. The coordinates are returned in client coordinates, unless the window was created with the CS_OWNDC style and the mapping mode is not MM_TEXT.
Syntax	BOOL GetUpdateRect(HWND hWnd, LPRECT lpRect, BOOL bErase)
Parameters	
hWnd	HWND: Specifies the window handle of the desired window.
lpRect	LPRECT: A pointer to a RECT structure. This parameter may be NULL.
bErase	BOOL: If this parameter is TRUE and an update region for the window exists, then a WM_ERASEBKGND will be sent to the window.
Returns	BOOL: Returns TRUE if the update region is not empty. Returns FALSE otherwise.
See Also	ExcludeUpdateRgn(), GetUpdateRgn()
Example	The following function determines if a given point is within the update rectangle of a window.

```
BOOL IsPointInUpdateRect( HWND hWnd, POINT pt )
{
   RECT  rcUpdate;

   if( GetUpdateRect( hWnd, &rcUpdate, FALSE ) )
      return( PtInRect( &rcUpdate, pt ) );

   return( FALSE );
}
```

GETUPDATERGN WIN32S WINDOWS 95 WINDOWS NT

Description	GetUpdateRgn() retrieves the update region for the specified window. The region is relative to the upper-left corner of the window and is defined in client coordinates.
Syntax	int GetUpdateRgn(HWND hWnd, HRGN hRegion, BOOL bErase)
Parameters	
hWnd	HWND: Specifies the window handle of the desired window.
hRegion	HRGN: Specifies the handle of an existing region.
bErase	BOOL: If this parameter is TRUE and an update region for the window exists, then a WM_ERASEBKGND will be sent to the window.
Returns	int: The return value indicates the type of region returned and may be one of the values specified in Table 15-4.
See Also	ExdudeUpdateRgn(), GetUpdateRect()
Example	This example is similar to the one for the GetUpdateRect() function with the exception that it works with the update region which is a more accurate bounds for the update area of a window.

```
BOOL IsPointInUpdateRgn( HWND hWnd, POINT pt )
{
   HRGN hRgnUpdate;
   BOOL bReturn = FALSE;

   // Create a dummy region.
   //......................
   hRgnUpdate = CreateRectRgn( 0, 0, 1, 1 );

   if( GetUpdateRgn( hWnd, hRgnUpdate, FALSE ) )
      bReturn = PtInRgn( hRgnUpdate, pt.x, pt.y ) );

   // Clean up before returning.
   //...........................
   DeleteObject( hRgnUpdate );

   return( bReturn );
}
```

INFLATERECT WIN32S WINDOWS 95 WINDOWS NT

Description	InflateRect() increases or decreases the height and width of the specified rectangle. Positive values increase the height or width, and negative values decrease the height or width.
Syntax	BOOL InflateRect(LPRECT lpRect, int nWidthAmount, int nHeightAmount)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure that contains an existing rectangle.
nWidthAmount	int: Specifies the amount that the width of the rectangle should be increased or decreased.
nHeightAmount	int: Specifies the amount that the height of the rectangle should be increased or decreased.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CopyRect(), InvertRect(), SetRect(), SetRectEmpty(), SubtractRect(), UnionRect()
Example	See CopyRect() for an example of this function.

INTERSECTCLIPRECT WIN32S WINDOWS 95 WINDOWS NT

Description	IntersectClipRect() modifies the current clipping region. The new clipping region will be the intersection of the existing clipping region and the specified rectangle coordinates.
Syntax	int IntersectClipRect(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
hDC	HDC: Specifies the device context.
nLeft	int: Specifies the coordinate of the left edge of the rectangle.
nTop	int: Specifies the coordinate of the top edge of the rectangle.
nRight	int: Specifies the coordinate of the right edge of the rectangle.
nBottom	int: Specifies the coordinate of the bottom edge of the rectangle.
Returns	int: The return values identifies the complexity of the resulting clipping region. Refer to Table 15-4 for details.
See Also	ExcludeClipRect(), ExtSelectClipRgn(), GetClipBox(), GetClipRgn(), OffsetClipRgn(), SelectClipPath(), SelectClipRgn()
Example	The following example demonstrates the use of the IntersectClipRect() function by creating an elliptic clipping region and intersecting it with a rectangle. The resulting clipping area is painted by filling the entire client area with a gray brush.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           HRGN hElRgn;
                           RECT rect;

                           hDC     = GetDC( hWnd );
                           hElRgn  = CreateEllipticRgn( 10, 10, 110, 110 );

                           // Select a clipping region.
                           //..........................
                           SelectClipRgn( hDC, hElRgn );

                           // Intersect a rectangle with the clipping region.
                           //...............................................
                           IntersectClipRect( hDC, 70, 40, 150, 150 );

                           // Fill the entire client area which will only show
                           // the resulting clipping area.
                           //.................................................
                           GetClientRect( hWnd, &rect );
                           FillRect( hDC, &rect, GetStockObject( GRAY_BRUSH ) );

                           DeleteObject( hElRgn );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

INTERSECTRECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	IntersectRect() computes the intersection rectangle of two source rectangles.
Syntax	BOOL IntersectRect(LPRECT lpDestination, LPRECT lpRect1, LPRECT lpRect2)
Parameters	
lpDestination	LPRECT: A pointer to the destination rectangle. This rectangle will receive the coordinates of the resulting rectangle.
lpRect1	LPRECT: Points to one of the two source rectangles.
lpRect2	LPRECT: Points to one of the two source rectangles.
Returns	BOOL: Returns TRUE if the two rectangles intersect. Otherwise, FALSE is returned, and lpDestination is set to an empty rectangle.
See Also	SubtractRect(), UnionRect()
Example	The following example draws two intersecting rectangles and fills the intersection area of the two rectangles. The intersection is computed with the IntersectRect() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC  hDC;
                           RECT rect1, rect2, rect3;

                           hDC = GetDC( hWnd );
```

```
                              SetRect( &rect1, 10, 10, 80, 80 );
                              SetRect( &rect2, 50, 50, 110, 110 );

                              // Intersect the rectangles.
                              //..........................
                              IntersectRect( &rect3, &rect1, &rect2 );

                              // Draw the rectangles and fill the intersection area.
                              //..................................................
                              FrameRect( hDC, &rect1, GetStockObject( BLACK_BRUSH ) );
                              FrameRect( hDC, &rect2, GetStockObject( BLACK_BRUSH ) );
                              FillRect( hDC, &rect3, GetStockObject( LTGRAY_BRUSH ) );

                              ReleaseDC( hWnd, hDC );
                           }
                           break;
         .
         .
         .
```

INVALIDATERECT — WIN32S WINDOWS 95 WINDOWS NT

Description	InvalidateRect() adds the specified rectangle to the current update region of the given window. The update region defines that portion of the window that must be redrawn.
Syntax	BOOL InvalidateRect(HWND hWnd, LPRECT lpRect, BOOL bErase)
Parameters	
hWnd	HWND: Specifies the window handle of the desired window.
lpRect	LPRECT: A pointer to a RECT structure.
bErase	BOOL: If this parameter is TRUE and an update region for the window exists, then a WM_ERASEBKGND will be sent to the window.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	InvalidateRgn()
Example	See GetBkColor() for an example.

INVALIDATERGN — WIN32S WINDOWS 95 WINDOWS NT

Description	InvalidateRgn() adds the specified region to the current update region of the given window. The update region defines that portion of the window that must be redrawn.
Syntax	BOOL InvalidateRgn(HWND hWnd, HRGN hRegion, BOOL bErase)
Parameters	
hWnd	HWND: Specifies the window handle of the desired window.
hRegion	HRGN: Specifies the handle of an existing region.
bErase	BOOL: If this parameter is TRUE and an update region for the window exists, then a WM_ERASEBKGND will be sent to the window.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	InvalidateRect()
Example	See ExcludeUpdateRgn() for an example of this function.

INVERTRECT — WIN32S WINDOWS 95 WINDOWS NT

Description	InvertRect() inverts all pixels within the specified rectangle. A logical NOT operation is performed on all pixels.
Syntax	BOOL InvertRect(HDC hDC, LPRECT lpRect)
Parameters	

hDC	HDC: Specifies the device context.
lpRect	LPRECT: A pointer to a RECT structure that defines the coordinates of the rectangle to be inverted.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CopyRect(), InflateRect(), SetRect(), SetRectEmpty(), SubtractRect(), UnionRect()
Example	This example demonstrates two simple push buttons. When the example starts, there is a rectangular and an elliptic button drawn on the client area of the window. When the user presses them with the left mouse button, they invert to show a selected state.

```
#define NO_BUTTON      0
#define BUTTON_RECT    1
#define BUTTON_RGN     2

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
static POINT pt;
static RECT  rcButton;
static HRGN  hRgnButton     = NULL;
static BOOL  bHighlite      = FALSE;
static int   nButtonPressed = NO_BUTTON;

   switch( uMsg )
   {
      case WM_CREATE :
              SetRect( &rcButton, 10, 10, 130, 40 );
              hRgnButton = CreateEllipticRgn( 150, 10, 200, 60 );
              break;

      case WM_PAINT : // Draw the buttons.
              {
                 PAINTSTRUCT ps;

                 BeginPaint( hWnd, &ps );

                 FrameRect( ps.hdc, &rcButton, GetStockObject( BLACK_BRUSH ) );
                 FrameRgn( ps.hdc, hRgnButton, GetStockObject( BLACK_BRUSH ), 1, 1 );
                 TextOut( ps.hdc, 20, 16, "Button A", 8 );
                 TextOut( ps.hdc, 170, 27, "B", 1 );

                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_LBUTTONDOWN :
              pt.x = LOWORD( lParam );
              pt.y = HIWORD( lParam );

              // Check if button A was pressed.
              //..............................
              if ( PtInRect( &rcButton, pt ) )
              {
                 HDC hDC = GetDC( hWnd );

                 // Select the button.
                 //...................
                 InvertRect( hDC, &rcButton );
                 ReleaseDC( hWnd, hDC );

                 bHighlite = TRUE;
                 nButtonPressed = BUTTON_RECT;
                 SetCapture( hWnd );
              }

              // Check if button B was pressed.
              //..............................
              if ( PtInRegion( hRgnButton, LOWORD( lParam ), HIWORD( lParam ) ) )
              {
                 HDC hDC = GetDC( hWnd );
```

```
            // Select the button.
            //...................
            InvertRgn( hDC, hRgnButton );
            ReleaseDC( hWnd, hDC );

            bHighlite = TRUE;
            nButtonPressed = BUTTON_RGN;
            SetCapture( hWnd );
         }
         break;


         // If the user moves the mouse cursor out of the
         // button, the button is un-highlited, otherwise,
         // the button is highlited.
         //..........................................
case WM_MOUSEMOVE :
         if ( nButtonPressed != NO_BUTTON )
         {
            HDC hDC = GetDC( hWnd );

            pt.x = LOWORD( lParam );
            pt.y = HIWORD( lParam );

            switch ( nButtonPressed )
            {
               case BUTTON_RECT :
                     if ( PtInRect( &rcButton, pt ) && !bHighlite )
                     {
                        InvertRect( hDC, &rcButton );
                        bHighlite = TRUE;
                     }
                     if ( !PtInRect( &rcButton, pt) && bHighlite )
                     {
                        InvertRect( hDC, &rcButton );
                        bHighlite = FALSE;
                     }
                     break;

               case BUTTON_RGN :
                     if ( PtInRegion( hRgnButton, pt.x, pt.y ) && !bHighlite )
                     {
                        InvertRgn( hDC, hRgnButton );
                        bHighlite = TRUE;
                     }
                     if ( !PtInRegion( hRgnButton, pt.x, pt.y) && bHighlite )
                     {
                        InvertRgn( hDC, hRgnButton );
                        bHighlite = FALSE;
                     }
                     break;
            }
            ReleaseDC( hWnd, hDC );
         }
         break;

         // The user let off the mouse button, un-highlite the button.
         //...........................................................
case WM_LBUTTONUP :
      if ( nButtonPressed != NO_BUTTON )
      {
         if ( bHighlite )
         {
            HDC hDC = GetDC( hWnd );

            switch( nButtonPressed )
            {
               case BUTTON_RECT : InvertRect( hDC, &rcButton ); break;
               case BUTTON_RGN  : InvertRgn( hDC, hRgnButton ); break;
            }
            ReleaseDC( hWnd, hDC );
```

```
                  bHighlite = FALSE;
               }

               nButtonPressed = NO_BUTTON;
               ReleaseCapture();
            }
            break;
         .
         .
         .
```

INVERTRGN WIN32S WINDOWS 95 WINDOWS NT

Description	InvertRgn() inverts all pixels within the specified region. A logical NOT operation is performed on all pixels.
Syntax	BOOL InvertRgn(HDC hDC, HRGN hRegion)
Parameters	
hDC	HDC: Specifies the device context.
hRegion	HRGN: The handle of the region to be inverted.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CombineRgn(), CreateEllipticRgn(), CreateEllipticRgnIndirect(), CreatePolygonRgn(), CreatePolyPolygonRgn(), CreateRectRgn(), CreateRectRgnIndirect(), CreateRoundRectRgn(), ExtCreateRegion(), SetRectRgn()
Example	See InvertRect() for an example of this function.

ISRECTEMPTY WIN32S WINDOWS 95 WINDOWS NT

Description	IsRectEmpty() checks the given rectangle to determine if it is empty. An empty rectangle is defined as one that has no area, for example, a rectangle whose right boundary is one unit to the left of the left boundary.
Syntax	BOOL IsRectEmpty(LPRECT lpRect)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure that defines the given rectangle.
Returns	BOOL: Returns TRUE if the rectangle is empty; otherwise, FALSE is returned.
See Also	SetRectEmpty()
Example	See EqualRect() for an example of this function.

LINEDDA WIN32S WINDOWS 95 WINDOWS NT

Description	LineDDA() is similar to the LineTo() function, but instead of drawing pixels into a device context, this function calls a user-supplied function for each pixel. This is useful for drawing lines of a unique style or for animating movement of objects in a straight line.
Syntax	BOOL LineDDA(int nXStart, int nYStart, int nXEnd, int nYEnd, LINEDDAPROC lpLineProc, LPARAM lParam)
Parameters	
nXStart	int: Specifies the x coordinate of the start of the line.
nYStart	int: Specifies the y coordinate of the start of the line.
nXEnd	int: Specifies the x coordinate of the end of the line.
nYEnd	int: Specifies the y coordinate of the end of the line.
lpLineProc	LINEDDAPROC: The address of a callback routine. See the callback syntax below.

lPram	LPARAM: Application-defined data. This parameter is passed to the callback routine and can be used to provide additional information.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	LineTo()
Callback Syntax	VOID CALLBACK **LineDDAProc**(int *X*, int *Y*, LPARAM *lpData*)
Callback Parameters	
X	int: The x-coordinate of the current point.
Y	int: The y-coordinate of the current point.
lpData	LPARAM: The application defined 32-bit value passed in the *lParam* parameter of the LineDDA() function.
Example	The following example draws a series of small rectangles along the line from (15, 15) to (135, 300). The points in the line are computed using the LineDDA() function. In this example, the device context identifier is passed to the callback function in the *lParam* parameter. The callback function draws rectangles every fifth point.

```
int nPointSkip;

VOID CALLBACK LineDDACallBack( int X, int Y, LPARAM lParam )
{
   HDC hDC = (HDC)lParam;

   nPointSkip++;
   if( nPointSkip < 5 )
      return;

   nPointSkip = 0;
   Rectangle( hDC, X, Y, X+10, Y+10 );
}

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC hDC = GetDC( hWnd );

                           nPointSkip = 0;
                           LineDDA( 15, 15, 135, 300, (LINEDDAPROC)LineDDACallBack,
(LPARAM)hDC );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

LINETO

WIN32S WINDOWS 95 WINDOWS NT

Description	The LineTo() function draws a line from the current position to the indicated position. The line is drawn using the current pen.
Syntax	BOOL LineTo(HDC hDC, int X, int Y)
Parameters	
hDC	HDC: Specifies the desired device context.
X	int: Specifies the x coordinate of the end of the line in logical units.
Y	int: Specifies the y coordinate of the end of the line in logical units.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	LineDDA()
Example	See GetMiterLimit() for an example of this function.

LOCKWINDOWUPDATE   WIN32S   WINDOWS 95   WINDOWS NT

Description	LockWindowUpdate() temporarily disables updates within the specified window. Only one window can be locked. While a window is locked, Windows discards any drawing done in a device context associated with the window. Windows does accumulate a bounding rectangle of any drawing that is done. Once the window is unlocked, Windows invalidates the accumulated rectangle, which will eventually cause a WM_PAINT message to be sent to the window. To unlock the window, specify a value of NULL for the window handle.
Syntax	BOOL LockWindowUpdate(HWND hWindow)
Parameters	
hWindow	HWND: The window handle of the window to be locked. NULL unlocks the currently locked window.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	RedrawWindow(), UpdateWindow()
Example	The following example locks the current window, draws a rectangle, and then unlocks the rectangle. Windows will not draw the rectangle but will invalidate that portion of the window and will issue a WM_PAINT message after the window has been unlocked.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC hDC = GetDC( hWnd );

                           LockWindowUpdate( hWnd );

                           Rectangle( hDC, 10, 10, 110, 110 );
                           ReleaseDC( hWnd, hDC );

                           MessageBox( hWnd, "The rectangle did not paint.",
                                       lpszAppName, MB_OK );

                           LockWindowUpdate( NULL );

                        }
                        break;
          .
          .
          .
```

MOVETOEX   WIN32S   WINDOWS 95   WINDOWS NT

Description	MoveToEx() moves the current position associated with the specified device context. The current position is used with various drawing functions such as LineTo().
Syntax	BOOL MoveToEx(HDC hDC, int X, int Y, LPPOINT lpPoint)
Parameters	
hDC	HDC: Specifies the desired device context.
X	int: Specifies the x coordinate of the target point. This is specified in logical units.

Y	int: Specifies the y coordinate of the target point. This is specified in logical units.
lpPoint	LPPOINT: A pointer to a POINT structure. This parameter may be NULL. If specified, MoveToEx() places the previous current position in this structure.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	GetCurrentPositionEx()
Example	See GetMiterLimit() for an example of this function.

OFFSETCLIPRGN WIN32S WINDOWS 95 WINDOWS NT

Description	OffsetClipRgn() moves the current clipping region of the specified device context.
Syntax	int OffsetClipRgn(HDC hDC, int X, int Y)
Parameters	
hDC	HDC: Specifies a device context.
X	int: Specifies the amount that the clipping region should be moved in the x, or horizontal, direction. This value is specified in logical units.
Y	int: Specifies the amount that the clipping region should be moved in the y, or vertical, direction. This value is specified in logical units.
Returns	int: The complexity of the current clipping region is returned. It may be one of the values shown in Table 15-4.
See Also	ExcludeClipRect(), ExtSelectClipRgn(), GetClipBox(), GetClipRgn(), IntersectClipRect(), SelectClipPath(), SelectClipRgn()
Example	See SelectClipRgn() for an example of this function.

OFFSETRECT WIN32S WINDOWS 95 WINDOWS NT

Description	OffsetRect() moves the coordinates of the specified rectangle.
Syntax	BOOL OffsetRect(LPRECT lpRect, int X, int Y)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure that defines the rectangle.
X	int: Specifies the amount that the rectangle should be moved in the x, or horizontal, direction. This value is specified in logical units.
Y	int: Specifies the amount that the rectangle should be moved in the y, or vertical, direction. This value is specified in logical units.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	Rectangle(), RoundRect(), OffsetRgn()
Example	See CopyRect() for an example of this function.

OFFSETRGN WIN32S WINDOWS 95 WINDOWS NT

Description	OffsetRgn() moves the coordinates of the specified region.
Syntax	int OffsetRgn(HRGN hRegion, int X, int Y)
Parameters	
hRegion	HRGN: The handle to an existing region.
X	int: Specifies the amount that the region should be moved in the x, or horizontal, direction. This value is specified in logical units.
Y	int: Specifies the amount that the region should be moved in the y, or vertical, direction. This value is specified in logical units.

Returns	int: The complexity of the current clipping region is returned. It may be one of the values shown in Table 15-4.
See Also	OffsetRect()
Example	The following example paints the given region, moves the specified region by 10 logical units in the x direction and 15 logical units in the y direction, and paints it again.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HRGN hRgn = CreateEllipticRgn( 10, 10, 110, 110 );
                           HDC  hDC  = GetDC( hWnd );

                           // Paint shadow of image.
                           //......................
                           FillRgn( hDC, hRgn, GetStockObject( GRAY_BRUSH ) );

                           // Move the region to a new location.
                           //..................................
                           OffsetRgn( hRgn, -5, -5 );

                           // Paint the image.
                           //................
                           PaintRgn( hDC, hRgn );
                           FrameRgn( hDC, hRgn, GetStockObject( BLACK_BRUSH ), 1, 1 );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

PAINTRGN — WIN32S WINDOWS 95 WINDOWS NT

Description	PaintRgn() draws the specified region into the given device context. The region is drawn using the current brush.
Syntax	BOOL PaintRgn(HDC hDC, HRGN hRegion)
Parameters	
hDC	HDC: Specifies a device context.
hRegion	HRGN: Specifies the region to be drawn.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	FrameRect(), FrameRgn()
Example	See OffsetRgn() for an example of this function.

PATHTOREGION — WIN32S WINDOWS 95 WINDOWS NT

Description	PathToRegion() converts the current path within the device context into a region, and returns the handle of that region. Once converted, the path is discarded.
Syntax	HRGN PathToRegion(HDC hDC)
Parameters	
hDC	HDC: Specifies the desired device context.
Returns	HRGN: Returns the handle to a region if successful; otherwise, NULL is returned.

See Also	BeginPath(), EndPath()
Example	The following example creates a path, then converts that path to a region and fills it. Since a device context can contain only one path, this is a good method of creating multiple complex areas.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC hDC = GetDC( hWnd );

                           if( BeginPath( hDC ) )
                           {
                              HRGN hRgn;

                              // Build the path.
                              //................
                              MoveToEx( hDC, 10, 10, NULL );
                              LineTo( hDC, 200, 10 );
                              LineTo( hDC, 200, 200 );
                              CloseFigure( hDC );
                              EndPath( hDC );

                              // Convert the path to a region.
                              //.............................
                              hRgn = PathToRegion( hDC );

                              // Fill the region.
                              //.................
                              FillRgn( hDC, hRgn, GetStockObject( GRAY_BRUSH ) );

                              DeleteObject( hRgn );
                           }

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

PIE

WIN32S WINDOWS 95 WINDOWS NT

Description	Pie() draws a portion of an ellipse and connects the two endpoints to the center of the ellipse, resulting in a pie-shaped wedge. The figure is drawn with the current pen and filled with the current brush. The endpoints do not have to fall on the circle. Windows will compute the intersection of a line drawn from the center of the circle to the specified endpoint and will use the intersection point.
Syntax	BOOL Pie(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int xRadial1, int yRadial1, int xRadial2, int yRadial2)
Parameters	
hDC	HDC: Specifies the desired device context.
nLeft	int: Specifies the x coordinate of the left edge of the bounding rectangle that defines the ellipse.
nTop	int: Specifies the y coordinate of the top edge of the bounding rectangle that defines the ellipse.
nRight	int: Specifies the x coordinate of the right edge of the bounding rectangle that defines the ellipse.
nBottom	int: Specifies the y coordinate of the bottom edge of the bounding rectangle that defines the ellipse.
xRadial1	int: Specifies the x coordinate of the start point of the partial ellipse.

yRadial1	int: Specifies the y coordinate of the start point of the partial ellipse.
xRadial2	int: Specifies the x coordinate of the endpoint of the partial ellipse.
yRadial2	int: Specifies the y coordinate of the endpoint of the partial ellipse.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	Arc()
Example	The following example draws the upper quarter of a circle as a pie wedge when the user selects the Test! menu item. Figure 15-7 shows the result.

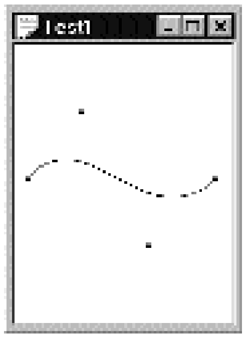

Figure 15-7 Pie() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rcClient;
                           HDC  hDC = GetDC( hWnd );

                           GetClientRect( hWnd, &rcClient );

                           Pie( hDC, rcClient.left,  rcClient.top,
                                     rcClient.right, rcClient.bottom,
                                     rcClient.right, rcClient.top,
                                     rcClient.left,  rcClient.top );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

POLYBEZIER — WINDOWS 95 WINDOWS NT

Description	PolyBezier() draws one or more bezier curves. A bezier curve is defined by four points, a start point, two control points, and an endpoint. For multiple bezier curves, you must specify four points for the first curve and three additional points for each additional curve, as the endpoint of one curve serves as the start point of the next curve. The device context's current position is not changed.
Syntax	BOOL PolyBezier(HDC hDC, LPPOINT lpPoint, DWORD dwPoints)

Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each point structure defines the coordinates of one point.
dwPoints	DWORD: The number of points in the lpPoint array. There must be three points per curve plus the initial start point.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezierTo(), PolyLine(), PolyLineTo(), PolyPolygon(), PolyPolyLine()

```
Example   See FlattenPath() for an example of this function.
```

POLYBEZIERTO WINDOWS 95 WINDOWS NT

Description	PolyBezierTo() draws one or more bezier curves. A bezier curve is defined by four points, a start point, two control points, and an endpoint. For multiple bezier curves, you must specify four points for the first curve and three additional points for each additional curve, as the endpoint of one curve serves as the start point of the next curve. Unlike the function PolyBezier(), this function sets the device context's current position to the last point in the curve.
Syntax	BOOL PolyBezierTo(HDC hDC, LPPOINT lpPoint, DWORD dwPoints)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each point structure defines the coordinates of one point.
dwPoints	DWORD: The number of points in the lpPoint array. There must be three points per curve. The current position is used as the starting point.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezier(), PolyLine(), PolyLineTo(), PolyPolygon(), PolyPolyLine()
Example	This example draws a bezier curve, as shown in Figure 15-8, when the user selects the Test! menu item. Unlike the PolyBezier() function, the starting position of the curve is the current position which is set with the MoveToEx() function. A straight line is painted at the end of the bezier curve to demonstrate how additional painting functions can continue to paint with the updated position.

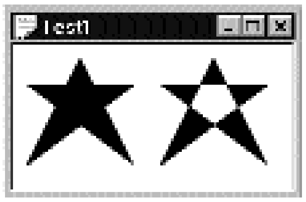

Figure 15-8 PolyBezierTo() Example.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
```

```
                      POINT ptCurve[4];
                      int   i;
                      HDC   hDC = GetDC( hWnd );

                      ptCurve[0].x = 10;  ptCurve[0].y = 100;
                      ptCurve[1].x = 50;  ptCurve[1].y = 50;
                      ptCurve[2].x = 100; ptCurve[2].y = 150;
                      ptCurve[3].x = 150; ptCurve[3].y = 100;

                      // Display point rects.
                      //....................
                      for( i=0; i < 4; i++ )
                         Rectangle( hDC, ptCurve[i].x-2, ptCurve[i].y-2,
                                         ptCurve[i].x+2, ptCurve[i].y+2 );

                      // Position starting point for the curve.
                      //......................................
                      MoveToEx( hDC, ptCurve[0].x, ptCurve[0].y, NULL );

                      // Paint the bezier curve.
                      //.......................
                      PolyBezierTo( hDC, &ptCurve[1], 3 );

                      // Continue with a line.
                      //.....................
                      LineTo( hDC, 200, 100 );

                      ReleaseDC( hWnd, hDC );
                   }
                   break;
         .
         .
         .
```

POLYGON — WIN32S WINDOWS 95 WINDOWS NT

Description	Polygon() draws a series of lines using the specified points. In addition, a line is draw from the last point to the first point, closing the figure. The polygon is drawn using the current pen and is filled using the current brush and polygon fill mode.
Syntax	BOOL Polygon(HDC hDC, LPPOINT lpPoint, int nPoints)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each point structure defines the coordinates of one point. There must be at least two points in the array.
nPoints	int: The number of points in the lpPoint array. There must be at least two points in the array.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	SetPolyFillMode()
Example	See GetPolyFillMode() for an example of this function.

POLYLINE — WIN32S WINDOWS 95 WINDOWS NT

Description	Polyline() draws a series of lines using the specified points and the current pen. Unlike Polygon(), this function does not close or fill the drawn figure.
Syntax	BOOL Polyline(HDC hDC, LPPOINT lpPoint, int nPoints)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each point structure defines the coordinates of one point. There must be at least two points in the array.
nPoints	int: The number of points in the lpPoint array. There must be at least two points in the array.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezier(), PolyBezierTo(), PolyLineTo(), PolyPolygon(), PolyPolyLine()
Example	The following example draws an hourglass-shaped figure. Unlike Polygon(), the first point must be repeated in order to close the figure.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           POINT ptLine[5];
                           HDC   hDC = GetDC( hWnd );

                           ptLine[0].x = 10;  ptLine[0].y = 10;
                           ptLine[1].x = 100; ptLine[1].y = 100;
                           ptLine[2].x = 10;  ptLine[2].y = 100;
                           ptLine[3].x = 100; ptLine[3].y = 10;
                           ptLine[4].x = 10;  ptLine[4].y = 10;

                           Polyline( hDC, ptLine, 5 );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

POLYLINETO WINDOWS 95 WINDOWS NT

Description	PolylineTo() draws a series of lines using the specified points and the current pen. Unlike Polygon(), this function does not close or fill the drawn figure. The current position of the specified device context is set to the last point in the array of points.
Syntax	BOOL PolylineTo(HDC hDC, LPPOINT lpPoint, DWORD dwPoints)
Parameters	
hDC	HDC: Specifies the desired device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each point structure defines the coordinates of one point. The first line is drawn between the current position and the first point in the *lpPoint* array.
dwPoints	DWORD: The number of points in the lpPoint array.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezier(), PolyBezierTo(), PolyLine(), PolyPolygon(), PolyPolyLine()
Example	This example is the same as the example for the Polyline() function except that the starting position is set with the MoveToEx() function. The Polyline() example uses the first point in the array as the starting point.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
```

```
                        POINT ptLine[4];
                        HDC   hDC = GetDC( hWnd );

                        ptLine[0].x = 100; ptLine[0].y = 100;
                        ptLine[1].x = 10;  ptLine[1].y = 100;
                        ptLine[2].x = 100; ptLine[2].y = 10;
                        ptLine[3].x = 10;  ptLine[3].y = 10;

                        // Set the starting position.
                        //..........................
                        MoveToEx( hDC, 10, 10, NULL );

                        PolylineTo( hDC, ptLine, 4 );

                        ReleaseDC( hWnd, hDC );
                     }
                     break;
         .
         .
         .
```

POLYPOLYGON — WIN32S WINDOWS 95 WINDOWS NT

Description	The PolyPolygon() function draws a series of polygons, each of which is drawn using the current pen, and filled using the current brush and polygon fill mode. The polygons may overlap.
Syntax	BOOL PolyPolygon(HDC hDC, LPPOINT lpPoint, LPINT lpVertices, int nPolygons)
Parameters	
hDC	HDC: Specifies a device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each entry in this array defines one point.
lpVertices	LPINT: A pointer to an array of integers. Each entry in this array corresponds to one polygon. The value of each entry specifies the number of points in the lpPoint array that define the corresponding polygon. This value must be at least two.
nPolygons	int: Specifies the number of entries in the lpVertices array.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezier(), PolyBezierTo(), PolyLine(), PolyLineTo(), PolyPolyLine()
Example	See CreatePolyPolygonRgn() for example of this function.

POLYPOLYLINE — WINDOWS 95 WINDOWS NT

Description	PolyPolyline() draws a series of line sequences using the current pen. This is similar to the Polyline() function, except that multiple line sequences can be drawn in one function call.
Syntax	BOOL PolyPolyline(HDC hDC, LPPOINT lpPoint, LPDWORD lpVertices, DWORD dwPolylines)
Parameters	
hDC	HDC: Specifies a device context.
lpPoint	LPPOINT: A pointer to an array of POINT structures. Each entry in this array defines one point.
lpVertices	LPDWORD: A pointer to an array of DWORD values. Each entry in this array corresponds to one line sequence. The value of each entry specifies the number of points in the lpPoint array that define the corresponding line sequence. This value must be at least two.
dwPolylines	DWORD: Specifies the number of entries in the lpVertices array.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	PolyBezier(), PolyBezierTo(), PolyLine(), PolyLineTo(), PolyPolygon()
Example	This example demonstrates the PolyPolyline() function by drawing a rectangle and a triangle when the user selects the Test! menu item.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           POINT ptLine[9];
                           DWORD dwVert[2];
                           HDC   hDC = GetDC( hWnd );

                           // rectangle.
                           //..................
                           dwVert[0] = 5;
                           ptLine[0].x = 10;  ptLine[0].y = 10;
                           ptLine[1].x = 100; ptLine[1].y = 10;
                           ptLine[2].x = 100; ptLine[2].y = 100;
                           ptLine[3].x = 10;  ptLine[3].y = 100;
                           ptLine[4].x = 10;  ptLine[4].y = 10;

                           // triangle.
                           //..........
                           dwVert[1] = 4;
                           ptLine[5].x = 80;  ptLine[5].y = 50;
                           ptLine[6].x = 140; ptLine[6].y = 110;
                           ptLine[7].x = 20;  ptLine[7].y = 110;
                           ptLine[8].x = 80;  ptLine[8].y = 50;

                           PolyPolyline( hDC, ptLine, dwVert, 2 );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

PTINRECT   WIN32S WINDOWS 95 WINDOWS NT

Description	PtInRect() determines if the specified point lies within the boundaries of the given rectangle. Note that if the point falls on the top or left edge, it is considered to be inside the rectangle, and if the point falls on the right or bottom edge, it is considered to be outside the rectangle.
Syntax	BOOL PtInRect(LPRECT lpRect, POINT ptPoint)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure that defines the boundaries of the rectangle.
ptPoint	POINT: A POINT structure that defines the point. Note that the actual POINT structure is passed to this function, not a pointer to a POINT structure.
Returns	BOOL: Returns TRUE if the point lies within the rectangle or on the top of left edges. Otherwise, FALSE is returned.
See Also	PtInRegion(), RectInRegion()
Example	See InvertRect() for an example of this function..

PTINREGION   WIN32S WINDOWS 95 WINDOWS NT

Description	PtInRegion() determines if the specified point lies within the indicated region.
Syntax	BOOL PtInRegion(HRGN hRegion, int X, int Y)
Parameters	
hRegion	HRGN: The handle of the desired region.

X	int: Specifies the x coordinate of the desired point.
Y	int: Specifies the y coordinate of the desired point.
Returns	BOOL: Returns TRUE if the point lies within the region; otherwise, FALSE is returned.
See Also	PtInRect(), RectInRegion()
Example	See InvertRect() for an example o fthis function.

RECTANGLE WIN32S WINDOWS 95 WINDOWS NT

Description	Rectangle() draws a rectangle defined by the given coordinates. The rectangle is drawn using the current pen and filled using the current brush.
Syntax	BOOL Rectangle(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
hDC	HDC: Specifies the device context that the arc is to be drawn in.
nLeft	int: Specifies the x coordinate of the upper-left corner of the rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CreateRoundRectRgn(), OffsetRect(), RoundRect()
Example	See CreatePatternBrush() for an example of this function.

RECTINREGION WIN32S WINDOWS 95 WINDOWS NT

Description	RectInRegion() determines if any part of the specified rectangle lies within the given region.
Syntax	BOOL RectInRegion(HRGN hRegion, LPRECT lpRect)
Parameters	
hRegion	HRGN: Specifies the handle of a valid region.
lpRect	LPRECT: A pointer to a RECT structure.
Returns	BOOL: If any portion of the rectangle, except the right and bottom edges, lies within the region, the return value is TRUE; otherwise, it is FALSE.
See Also	PtInRect(), PtInRegion()
Example	The following code segment determines if any portion of the specified rectangle exists within the given region.

```
RECT rcRectangle;
HRGN hRgn;

if( RectInRgn( hRgn, &rcRectangle))
{
   //  Some portion of the rectangle lies within the region
}
else
{
   //  No portion of the rectangle lies within the region
}
```

REDRAWWINDOW WIN32S WINDOWS 95 WINDOWS NT

Description	RedrawWindow() causes the specified rectangle or region of the given window to be updated. Various options allow you to control the update process.
Syntax	BOOL RedrawWindow(HWND hWindow, LPRECT lpRect, HRGN hRegion, UINT uFlags)

Parameters	
hWindow	HWND: Specifies the handle of the window. If this parameter is NULL, the desktop window is updated.
lpRect	LPRECT: A pointer to a RECT structure. This rectangle contains the coordinates of the client area to be updated. This parameter may be NULL. If this parameter and hRegion are both NULL, the entire client area is assumed.
hRegion	HRGN: Specifies the handle of the region to be updated. If this parameter is not NULL, then lpRect is ignored. If this parameter and lpRect are both NULL, the entire client area is assumed.
uFlags	UINT: Specifies various flags that define how the window will be affected, how repainting will occur, and which windows are affected. Refer to Table 15-26 for details.

Table 15-26. Values for the uFlags Parameter of RedrawWindow()

Flag	Description
Invalidation flags	
RDW_ERASE	The affected windows will receive a WM_ERASEBKGND message when being redrawn. The RDW_INVALIDATE flag must also be specified.
RDW_FRAME	Causes that section of the non-client area that intersects the update region to receive a WM_NCPAINT message. The RDW_INVALIDATE flag also must be specified.
RDW_INTERNALPAINT	Forces a WM_PAINT message to be posted to the affected windows regardless of the existence of any invalid region.
RDW_INVALIDATE	Causes the area defined by hRegion or lpRect to be marked invalid. If both parameters are NULL, the entire client area is invalidated.
Validation flags	
RDW_NOERASE	Causes any pending WM_ERASEBKGND message to be ignored.
RDW_NOFRAME	Causes any pending WM_NCPAINT to be ignored. The RDW_VALIDATE message also must be specified.
RDW_NOINTERNALPAINT	Causes any pending WM_PAINT message to be ignored. WM_PAINT still will be sent if the resulting update area is not NULL.
RDW_VALIDATE	Causes the area defined by hRegion or lpRect to be marked as valid. Any pending internal WM_PAINT messages are still sent to the window.
Repainting flags	
RDW_ERASENOW	Causes any pending WM_NCPAINT and WM_ERASEBKGND messages to be completed before this function returns. WM_PAINT messages will be handled at the normal time.
RDW_UPDATENOW	Causes any pending WM_NCPAINT, WM_ERASEBKGND, and WM_PAINT messages to be completed before this function returns.
Child window control flags. Normally, child windows created without the WS_CLIPCHILDREN flag will be recursively updated. The following flags modify this behavior.	
RDW_ALLCHILDREN	Causes all child windows to be affected.
RDW_NOCHILDREN	Causes child windows to remain unaffected.

Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	LockWindowUpdate(), UpdateWindow()
Example	See GetPolyFillMode() for an example of this function.

ROUNDRECT

WIN32S WINDOWS 95 WINDOWS NT

Description	RoundRect() draws a rectangle defined by the given coordinates. The rectangle is drawn using the current pen, and filled using the current brush. The corners of the rectangle are rounded using the specified ellipse parameters.

Syntax	BOOL RoundRect (HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nWidth, int nHeight)
Parameters	
hDC	HDC: Specifies the device context that the arc is to be drawn in.
nLeft	int: Specifies the x coordinate of the upper-left corner of the rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the rectangle.
nWidth	int: Specifies the width of the ellipse that is used to draw the rounded corners.
nHeight	int: Specifies the height of the ellipse that is used to draw the rounded corners.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	CreateRoundRectRgn(), OffsetRect(), Rectangle()
Example	The following example draws a rectangle with rounded corners. For an example of how this figure would appear, refer to Figure 15-5, and the discussion of CreateRoundRectRgn().

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HDC hDC = GetDC( hWnd );
                           RoundRect( hDC, 10, 10, 110, 110, 30, 30 );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

SELECTCLIPPATH　　WIN32S　WINDOWS 95　WINDOWS NT

Description	SelectClipPath() combines the current clipping region of the specified device context with the current path.
Syntax	BOOL SelectClipPath(HDC hDC, int nCombineMode)
Parameters	
hDC	HDC: Identifies the desired device context.
nCombineMode	int: Specifies the manner that Windows will use to combine the current clipping region with the current path. This parameter may be one of the values specified in Table 15-3.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ExcludeClipRect(), ExtSelectClipRgn(), GetClipBox(), GetClipRgn(), IntersectClipRect(), OffsetClipRgn(), SelectClipRgn()
Example	See GetClipRgn() for an example of this function.

SELECTCLIPRGN　　WIN32S　WINDOWS 95　WINDOWS NT

Description	SelectClipRgn() sets the specified region as the current clipping region.
Syntax	BOOL SelectClipRgn(HDC hDC, HRGN hRegion)
Parameters	
hDC	HDC: Identifies the desired device context.

hRegion	HRGN: Specifies the handle of a valid region. This region will become the new clipping region for the device context.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ExcludeClipRect(), ExtSelectClipRgn(), GetClipBox(), GetClipRgn(), IntersectClipRect(), OffsetClipRgn(), SelectClipPath()
Example	See GetClipBox() for an example of this function.

SETARCDIRECTION — WIN32S WINDOWS 95 WINDOWS NT

Description	SetArcDirection() sets the direction that Windows uses to draw arcs and rectangles.
Syntax	int SetArcDirection(HDC hDC, int nDirection)
Parameters	
hDC	HDC: Specifies the desired device context.
nDirection	int: Specifies the new drawing direction. This parameter may be AD_CLOCKWISE or AD_COUNTERCLOCKWISE.
Returns	int: Returns the old arc direction if successful; otherwise, zero is returned.
See Also	GetArcDirection()
Example	See Ellipse() for an example of this function.

SETBKCOLOR — WIN32S WINDOWS 95 WINDOWS NT

Description	SetBkColor() sets the background color. The background color is used when drawing text, when converting bitmaps from color to monochrome, and as the gap color with pens.
Syntax	COLORREF SetBkColor(HDC hDC, COLORREF crColor)
Parameters	
hDC	HDC: Specifies the desired device context.
crColor	COLORREF: Specifies the value of the new background color. If the device cannot represent the color directly, then the closest match is used.
Returns	COLORREF: Returns the previous background color.
See Also	GetBkColor(), SetBkMode(), GetBkMode()

```
Example   See GetBkColor() for an example of this function.hDChDChDChDChDC
```

SETBKMODE — WIN32S WINDOWS 95 WINDOWS NT

Description	SetBkMode() sets the current background mode. The background mode specifies the manner that Windows uses to draw the backgrounds of text, the gaps in lines, and to convert color bitmaps to monochrome.
Syntax	int SetBkMode(HDC hDC, int nMode)
Parameters	
hDC	HDC: Specifies the desired device context.
nMode	int: Specifies the background mode, which may be either OPAQUE or TRANSPARENT.
Returns	int: The return value is the previous background mode.
See Also	GetBkColor(), GetBkMode(), SetBkColor()
Example	See GetBkColor() for an example of this function.

SETBOUNDSRECT WIN32S WINDOWS 95 WINDOWS NT

Description	SetBoundsRect() allows an application to affect the accumulation rectangle. Windows maintains an accumulation rectangle for all drawing operations.
Syntax	UINT SetBoundsRect(HDC hDC, LPRECT lpRect, UINT uFlags)
Parameters	
hDC	HDC: Specifies the device context to be affected.
lpRect	LPRECT: A pointer to a RECT structure. This parameter may be NULL.
uFlags	UINT: Specifies various actions that affect how the accumulation rectangle is modified. This may be a combination of the values shown in Table 15-27.

Table 15-27. Values for the uFlags parameter of SetBoundsRect()

Flag	Description
DCB_ACCUMULATE	Adds the specified rectangle to the accumulation rectangle. If DCB_RESET is also specified, then the accumulation rectangle is set to the specified rectangle.
DCB_DISABLE	Turns off boundary accumulation. This is the default state.
DCB_ENABLE	Turns on boundary accumulation.
DCB_RESET	The current bounding rectangle is cleared. If DCB_ACCUMULATE is also specified, DCB_RESET takes effect before the specified rectangle is added.

Returns	UINT: The return values specifies the previous state of the accumulation rectangle. It may be any of the values specified in Table 15-21 in the function GetBoundsRect().
See Also	GetBoundsRect()
Example	Refer to the example of GetBoundsRect() for an example of this function.

SETBRUSHORGEX WIN32S WINDOWS 95 WINDOWS NT

Description	SetBrushOrgEx() sets the brush origin for the specified device context. The next brush that is selected into the device context will be aligned using the brush origin.
Syntax	BOOL SetBrushOrgEx(HDC hDC, int X, int Y, LPPOINT lpPoint)
Parameters	
hDC	HDC: Specifies the device context.
X	int: Specifies the x coordinate for the new brush origin, in device units. This value must be in the range 0 - 7.
Y	int: Specifies the y coordinate for the new brush origin, in device units. This value must be in the range 0 - 7.
lpPoint	LPPOINT: A pointer to a POINT structure. This parameter may be NULL. If specified, the previous brush origin is returned in this structure.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	GetBrushOrgEx()
Example	See GetBrushOrgEx() for an example of this function.

SETMITERLIMIT WINDOWS 95 WINDOWS NT

Description	SetMiterLimit() sets the miter limit for line joins. If a line join falls within this limit, a miter join will be drawn. Otherwise, a bevel join is drawn.

Syntax	BOOL SetMiterLimit(HDC hDC, FLOAT fLimit, PFLOAT pfOldLimit)
Parameters	
hDC	HDC: Specifies a device context.
fLimit	FLOAT: Specifies the new miter limit value. The miter length is defined as the distance from the intersection of the line walls on the inside of the join to the intersection of the line walls on the outside of the join.
pfOldLimit	PFLOAT: A pointer to a FLOAT. This parameter may be NULL. If specified, the previous miter limit is returned in this parameter.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	GetMiterLimit()
Example	See GetMiterLimit() for an example of this function.

SETPIXEL WIN32S WINDOWS 95 WINDOWS NT

Description	SetPixel() sets the specified pixel to the specified color and returns the old color of the pixel. The function fails if the pixel is outside the current clipping region. Use the GetDeviceCaps() function with the RC_BITBLT value to determine if the device supports this function.
Syntax	COLORREF **SetPixelV**(HDC *hDC*, int *X*, int *Y*, COLORREF *crColor*)
Parameters	
hDC	HDC: Specifies a device context.
X	int: The x-coordinate, in logical units, of the desired pixel.
Y	int: The y-coordinate, in logical units, of the desired pixel.
crColor	COLORREF: The desired color for the pixel.
Returns	COLORREF: Returns the old color of the pixel if successful, otherwise, -1 is returned.
See Also	SetPixelV(), GetPixel()
Example	The usage of this function is similar to the SetPixelV() function and can be used in a similar manner as shown in the example under the SetPixelV() function.

SETPIXELV WINDOWS 95 WINDOWS NT

Description	SetPixelV() sets the specified pixel to the specified color. If the color cannot be reproduced exactly by the device, then the closest approximation is used. This function fails if the pixel is outside the current clipping region. Use the GetDeviceCaps() function with the RC_BITBLT value to determine if the device supports this function. This function is faster than the SetPixel() function because it does not need to return the color value of the point actually painted.
Syntax	BOOL SetPixelV(HDC hDC, int X, int Y, COLORREF crColor)
Parameters	
hDC	HDC: Specifies a device context.
X	int: Specifies the x coordinate of the desired pixel. This is in logical units.
Y	int: Specifies the y coordinate of the desired pixel. This is in logical units.
crColor	COLORREF: Specifies the desired color for the pixel.
Returns	BOOL: Returns TRUE if successful, otherwise, FALSE is returned.
See Also	SetPixel(), GetPixel()
Example	This example displays an image on the client area of the window. When the user selects the Test! menu item, all the black pixels in the image are converted to red with the GetPixel() and SetPixelV() functions.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;
                 HBITMAP     hBitmap;
                 HDC         hDC;

                 BeginPaint( hWnd, &ps );

                 // Paint the image on the client area.
                 //.................................
                 hBitmap = LoadBitmap( hInst, "TESTIMAGE" );
                 hDC     = CreateCompatibleDC( ps.hdc );

                 SelectObject( hDC, hBitmap );

                 BitBlt( ps.hdc, 10, 10, 64, 66, hDC, 0, 0, SRCCOPY );

                 DeleteDC( hDC );
                 DeleteObject( hBitmap );

                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           int x, y;
                           HDC hDC = GetDC( hWnd );

                           // Turn all black pixels into red pixels.
                           //....................................
                           for( y = 0; y < 76; y++ )
                              for( x = 0; x < 74; x++ )
                              {
                                 if ( GetPixel( hDC, x, y ) == 0 )
                                    SetPixelV( hDC, x, y, RGB( 255, 0, 0 ) );
                              }

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

SetPolyFillMode   Win32s   Windows 95   Windows NT

Description	SetPolyFillMode() sets the polygon fill mode for the specified device context. The polygon fill mode determines how Windows fills the interior of polygons.
Syntax	int SetPolyFillMode(HDC hDC, int nFillMode)
Parameters	
hDC	HDC: Specifies the device context.
nFillMode	int: Specifies the desired polygon fill mode. This may be one of the values shown in Table 15-7.
Returns	int: Returns the previous setting of the polygon fill mode.
See Also	GetPolyFillMode()
Example	See GetPolyFillMode() for an example of this function.

SETRECT WIN32S WINDOWS 95 WINDOWS NT

Description	SetRect() sets the coordinates of a RECT structure to the given values.
Syntax	BOOL SetRect(LPRECT lpRect, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure. Windows sets the coordinates of this structure to the specified values.
Left	int: Specifies the x coordinate of the upper-left corner of the rectangle.
Top	int: Specifies the y coordinate of the upper-left corner of the rectangle.
Right	int: Specifies the x coordinate of the lower-right corner of the rectangle.
Bottom	int: Specifies the y coordinate of the lower-right corner of the rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	SetRectEmpty(), SetRectRgn()
Example	See CopyRect() for an example of this function.

SETRECTEMPTY WIN32S WINDOWS 95 WINDOWS NT

Description	SetRectEmpty() sets the coordinates of a RECT structure to an empty rectangle.
Syntax	BOOL SetRectEmpty(LPRECT lpRect)
Parameters	
lpRect	LPRECT: A pointer to a RECT structure. Windows sets the coordinates of this structure to values that specify an empty rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	SetRect(), SetRectRgn()
Example	The following code segment initializes a RECT structure to an empty rectangle (0,0,0,0).

```
RECT rc;

if( SetRectEmpty( &rc ) )
{
   // The rectangle has been set to empty
}
```

SETRECTRGN WIN32S WINDOWS 95 WINDOWS NT

Description	SetRectRgn() changes the specified region to a rectangular region with the given coordinates. The region does not contain the bottom or right edges of the rectangle.
Syntax	BOOL SetRectRgn(HRGN hRegion, int nLeft, int nTop, int nRight, int nBottom)
Parameters	
hRegion	HRGN: The handle of an existing region. The previous region is replaced by the region created with this function.
nLeft	int: Specifies the x coordinate of the upper-left corner of the rectangle.
nTop	int: Specifies the y coordinate of the upper-left corner of the rectangle.
nRight	int: Specifies the x coordinate of the lower-right corner of the rectangle.
nBottom	int: Specifies the y coordinate of the lower-right corner of the rectangle.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	SetRect(), SetRectEmpty()
Example	This example uses the SetRectRgn() function to create an increasingly larger rectanlge region which is then framed with the FrameRect() function.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HRGN hRgn = CreateRectRgn( 10, 10, 30, 30 );
                           HDC  hDC  = GetDC( hWnd );
                           int  i;

                           // Draw rectangle regions.
                           //.......................
                           for( i = 0; i < 50; i += 10 )
                           {
                              SetRectRgn( hRgn, 10, 10, 30+i, 30+i );
                              FrameRgn( hDC, hRgn, GetStockObject( BLACK_BRUSH ), 1, 1 );
                           }

                           DeleteObject( hRgn );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

SETROP2 — WIN32S WINDOWS 95 WINDOWS NT

Description	SetROP2() sets the raster operation code that Windows uses when combining colors with colors that are already present on the device.
Syntax	int SetROP2(HDC hDC, int nRasterOp)
Parameters	
hDC	HDC: Specifies a device context.
nRasterOp	int: Specifies the new raster operation code. Table 15-24 defines several of the more useful ROP codes.
Returns	int: Returns the previous raster operation code.
See Also	GetROP2()
Example	See FillPath() for an example of this function.

STROKEANDFILLPATH — WINDOWS 95 WINDOWS NT

Description	StrokeAndFillPath() closes any open path in the specified device context. It then draws the outline of the path using the current pen and fills the interior using the current brush.
Syntax	BOOL StrokeAndFillPath(HDC hDC);
Parameters	
hDC	HDC: Specifies the device context.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	StrokePath()
Example	This example creates a path in the shape of a triangle when the user selects the Test! menu item. The path is widened to the width of a pen that is 8 pixels wide with the WidenPath() function. The resulting path is then filled with the StrokeAndFillPath() function. The results are shown in Figure 15-9.

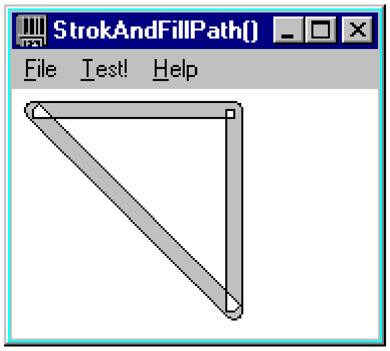

Figure 15-9 StrokeAndFillPath() Example

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           HBRUSH hOldBrush;
                           HPEN   hOldPen, hPen;
                           HDC    hDC = GetDC( hWnd );

                           // Create a path.
                           //...............
                           BeginPath( hDC );
                           MoveToEx( hDC, 10, 10, NULL );
                           LineTo( hDC, 110, 10 );
                           LineTo( hDC, 110, 110 );
                           CloseFigure( hDC );
                           EndPath( hDC );

                           // Create a wide pen.
                           hPen = CreatePen( PS_SOLID, 8, RGB( 0, 0, 0 ) );

                           hOldPen   = SelectObject( hDC, hPen );
                           hOldBrush = SelectObject( hDC, GetStockObject( LTGRAY_BRUSH ) );

                           // Redefine the path as the width of
                           // the pen surrounding the path.
                           //.................................
                           WidenPath( hDC );
                           SelectObject( hDC, hOldPen );

                           // Stroke and fill the path.
                           //..........................
                           StrokeAndFillPath( hDC );

                           DeleteObject( hPen );
                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

STROKEPATH

WINDOWS 95 WINDOWS NT

Description	StrokePath() closes any open path in the specified device context. It then draws the outline of the path using the current pen.
Syntax	BOOL StrokePath(HDC hDC)

Parameters	
hDC	HDC: Specifies the device context.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	StrokeAndFillPath()
Example	See GetMiterLimit() for an example of this function.

SUBTRACTRECT

WIN32S WINDOWS 95 WINDOWS NT

Description	SubtractRect() creates a new rectangle by subracting the two specified rectangles. The two rectangles must overlap.
Syntax	BOOL SubtractRect(LPRECT lpDest, LPRECT lpSource1, LPRECT lpSource2)
Parameters	
lpDest	LPRECT: Specifies a pointer to a RECT structure. This structure will receive the results of the subtraction.
lpSource1	LPRECT: Specifies a pointer to a RECT structure. lpSource2 is subtracted from this rectangle.
lpSource2	LPRECT: Specifies a pointer to a RECT structure. This parameter is subtracted from the rectangle specified by lpSource1.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	IntersectRect(), UnionRect()
Example	The following example illustrates the use of SubtractRect(). Two rectangles are created, and then a third rectangle is created that is the result of a subtraction of the first two rectangles. The first rectangle and the resulting rectangle are framed to show the results.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rect1, rect2, rect3;
                           HDC  hDC = GetDC( hWnd );

                           SetRect( &rect1, 10, 10, 110, 110 );
                           SetRect( &rect2, 40, 10, 180, 140 );
                           SubtractRect( &rect3, &rect1, &rect2 );

                           FrameRect( hDC, &rect1, GetStockObject( BLACK_BRUSH ) );
                           FrameRect( hDC, &rect3, GetStockObject( BLACK_BRUSH ) );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
```

UNIONRECT

WIN32S WINDOWS 95 WINDOWS NT

Description	UnionRect() creates a new rectangle by combining the two specified rectangles. The new rectangle is the smallest rectangle that contains both the specified rectangles.
Syntax	BOOL UnionRect(LPRECT lpDest, LPRECT lpSource1, LPRECT lpSource2)
Parameters	
lpDest	LPRECT: Specifies a pointer to a RECT structure. This structure will receive the results of the union.

lpSource1	LPRECT: Specifies a pointer to a RECT structure.
lpSource2	LPRECT: Specifies a pointer to a RECT structure.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	IntersectRect(), SubtractRect()
Example	The following example computes the union of two rectangles when the user selects the Test! menu item. The two source rectangles and the resulting rectangle are then framed.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rect1, rect2, rect3;
                           HDC  hDC = GetDC( hWnd );

                           SetRect( &rect1, 10, 10, 110, 110 );
                           SetRect( &rect2, 40, 10, 180, 140 );
                           UnionRect( &rect3, &rect1, &rect2 );

                           FrameRect( hDC, &rect1, GetStockObject( LTGRAY_BRUSH ) );
                           FrameRect( hDC, &rect2, GetStockObject( LTGRAY_BRUSH ) );
                           FrameRect( hDC, &rect3, GetStockObject( BLACK_BRUSH ) );

                           ReleaseDC( hWnd, hDC );
                        }
                        break;
         .
         .
         .
```

UNREALIZEOBJECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	UnrealizeObject() unrealizes the logical palette specified. When a logical palette is realized, the colors in the palette are matched against the closest approximations that the device can support. Once this is done, changes to the logical palette are not reflected in the physical palette. In order to update the physical palette, the logical palette must be unrealized and then realized again. This function should not be used on a palette created with GetStockObject().
Syntax	BOOL UnrealizeObject(HGDIOBJ hPalette)
Parameters	
hPalette	HGDIOBJ: Specifies the handle of a logical palette.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	RealizePalette(), SelectObject()
Example	The following code segment illustrates the use of this function. A logical palette is selected into the device context and realized. It is then unrealized, modified, and realized again. For more information on palettes, see Chapter 16.

```
HPALETTE hPalette;
HDC hDC;

HPALETTE OldPalette = (HPALETTE) SelectObject( hDC, hPalette );
RealizePalette( hDC, hPalette );       // The palette is realized (device palette updated).
.
.
.
UnrealizeObject( hPalette );           // The palette is unrealized so it can be modified.
.
. // Modify the palette
```

```
.
RealizePalette( hDC, hPalette );       // The palette is realized again so changes take
effect
```

UPDATEWINDOW   WIN32S   WINDOWS 95   WINDOWS NT

Description	UpdateWindow() causes Windows to send a WM_PAINT message to the specified window. The message is sent directly to the window procedure of the specified window, bypassing the message queue. If the update region is empty, no message is sent.
Syntax	BOOL UpdateWindow(HWND hWindow)
Parameters	
hWindow	HWND: Specifies the window handle of the window to be updated.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	LockWindowUpdate(), RedrawWindow()
Related Messages	WM_PAINT
Example	See InvalidateRgn() for an example of this function.

VALIDATERECT   WIN32S   WINDOWS 95   WINDOWS NT

Description	ValidateRect() removes the specified rectangle from the invalid region of the specified window.
Syntax	BOOL ValidateRect(HWND hWindow, LPRECT lpRect)
Parameters	
hWindow	HWND: Specifies the window handle of the desired window.
lpRect	LPRECT: A pointer to a RECT structure. This structure contains the coordinates of the rectangle that is to be removed from the invalid region of the specified window.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ValidateRgn()
Example	The following example removes a rectangular area and a elliptic region from the invalidation region. It then causes a paint to occur where the rectangle and ellipic region are ignored.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_PAINT :
              {
                 PAINTSTRUCT ps;
                 RECT        rect;

                 BeginPaint( hWnd, &ps );

                 // Fill the entire client area gray.
                 //..................................
                 GetClientRect( hWnd, &rect );
                 FillRect( ps.hdc, &rect, GetStockObject( LTGRAY_BRUSH ) );

                 EndPaint( hWnd, &ps );
              }
              break;

      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           RECT rect;
                           HRGN hRgn;
                           HDC  hDC = GetDC( hWnd );
                           SetRect( &rect, 10, 10, 110, 110 );
                           hRgn = CreateEllipticRgn( 120, 10, 220, 110 );

                           // Invalidate the entire client area.
                           //....................................
                           InvalidateRect( hWnd, NULL, FALSE );

                           // Paint and validate the areas
                           // we don't want the default to paint.
                           //....................................
                           FillRect( hDC, &rect, GetStockObject( WHITE_BRUSH ) );
                           FillRgn( hDC, hRgn, GetStockObject( WHITE_BRUSH ) );
                           ValidateRect( hWnd, &rect );
                           ValidateRgn( hWnd, hRgn );

                           DeleteObject( hRgn );

                           // Force the window to paint.
                           //...........................
                           UpdateWindow( hWnd );
                        }
                        break;
         .
         .
         .
```

VALIDATERGN   WIN32S WINDOWS 95 WINDOWS NT

Description	ValidateRgn() removes the specified region from the invalid region of the specified window.
Syntax	BOOL ValidateRgn(HWND hWindow, HRGN hRegion)
Parameters	
hWindow	HWND: Specifies the window handle of the desired window.
hRegion	HRGN: The handle of a valid region. This region defines the area that is to be removed from the invalid region of the specified window.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	ValidateRect()
Example	See ValidateRect() for an example of this function.

WIDENPATH   WINDOWS 95 WINDOWS NT

Description	WidenPath() redefines the current path of the specified device context. The path is redefined to be the area that would be drawn using the current pen. The pen must be either a geometric pen or a pen with a width of more than one. Any bezier curves within the path are converted into line sequences during the process.
Syntax	BOOL WidenPath(HDC hDC)
Parameters	
hDC	HDC: Specifies the device context.
Returns	BOOL: Returns TRUE if successful; otherwise, FALSE is returned.
See Also	FlattenPath()
Example	See StrokeAndFillPath() for an example of this function.