MFC GDI

Inhaltsverzeichnis

1.1 Gerätekonzept

statische Variable

```
CPaintDC dc(this); // Gerätekontext für Zeichnen
```

dynamische Variable

```
PAINSTRUCT ps
CDC *pDC
pDC = BeginPaint(&ps);
// Zeichenoperationen
EndPaint(&ps);
```

// Dialog

```
void CGrafik1Dlg::OnPaint()
{
    CPaintDC dc(this); // Gerätekontext für Zeichnen
    dc.MoveTo (int x, int y);
    dc.LineTo (int x, int y);

}
```

// SDI Fenster

```
void CLabor1View::OnDraw(CDC* pDC)
{
    // dynamischer Gerätekonzept
    pDC->MoveTo (int x, int y);
    pDC->LineTo (int x, int y);
}
```

1.2 Fensterbereich

```
CPaintDC dc(this); // Gerätekontext für Zeichnen
// Hole die Abmessungen
CRect rect;
GetClientRect(&rect);
dc.MoveTo(10,10);
dc.LineTo(rect.Width()-10, rect.Height()-10 );
```

1.3 Linien-Operationen

- **MoveTo** (int x, int y);

- **LineTo** (int x, int y);

- **Polyline** (LPPOINT lpPoints, int nCount);
- **PolylineTo**(LPPOINT lpPoints, int nCount);
  POINT p[5] = { 100,100, 300,100, 300,300, 100,300, 100,100};
  Polyline(p,5);

- **Rectangle** (int x1, int y1, int x2, int y2);

- **RoundRect** (int x1, int y1, int x2, int y2);

- **Arc**
- **ArcTo** (Startwert ist der letzte Punkt von LineTo)
  BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
  BOOL Arc(LPCRECT lpRect, POINT ptStart, POINT ptEnd);

  x1: LEFT
  y1: TOP
  x2: RIGHT
  y2: BOTTOM
  x3: Specifies the x-coordinate of the point that defines the arc's starting point (in logical units). This point does not have to lie exactly on the arc.
  y3: Specifies the y-coordinate of the point that defines the arc's starting point (in logical units). This point does not have to lie exactly on the arc.
  x4: Specifies the x-coordinate of the point that defines the arc's endpoint (in logical units). This point does not have to lie exactly on the arc.
  y4: Specifies the y-coordinate of the point that defines the arc's endpoint (in logical units). This point does not have to lie exactly on the arc.

- **PolyDraw** (Punktfeld, Typfeld, Anzahl)
  BOOL **PolyDraw**(const POINT* lpPoints, const BYTE* lpTypes, int nCount);

  lpPoints: Points to an array of POINT data structures that contains the endpoints for each line segment and the endpoints and control points for each Bézier spline.
  lpTypes: Points to an array that specifies how each point in the lpPoints array is used. Values can be one of the following:

  Konstante der Typdefinitionen:
  - PT_MOVETO
  - PT_LINETO
  - PT_BEZIERTO
  - PT_CLOSEFIGURE

- BOOL **PolyBezier**(const POINT* lpPoints, int nCount);
- BOOL **PolyBezierTo**(const POINT* lpPoints, int nCount);
  (Startwert ist der letzte Punkt von LineTo)

  lpPoints Points to an array of POINT data structures that contain the endpoints and control points of the spline(s).

  nCount: Specifies the number of points in the lpPoints array. This value must be one more than three times the number of splines to be drawn, because each Bézier spline requires two control points and an endpoint, and the initial spline requires an additional starting point.

- BOOL **AngleArc**(int x, int y, int nRadius, float fStartAngle, float fSweepAngle);

  x: Specifies the logical x-coordinate of the center of the circle.
  y: Specifies the logical y-coordinate of the center of the circle.
  nRadius: Specifies the radius of the circle in logical units. This value must be positive.
  fStartAngle: Specifies the starting angle in degrees relative to the x-axis.
  fSweepAngle: Specifies the sweep angle in degrees relative to the starting angle.

1.4 Pen

- **CPen**(int nPenStyle, int nWidth, COLORREF crColor); throw(CResourceException);

Pen-New

- **CPen**(int nPenStyle, int nWidth, const LOGBRUSH* pLogBrush, int nStyleCount = 0, const DWORD* lpStyle = NULL); throw(CResourceException);

1.4.1 Pen-Style

- PS_SOLID
- PS_DASH
- PS_DASHDOT
- PS_DASHDOTDOT
- PS_NULL
- PS_INSIDEFRAME

1.4.2 Pen-Style (new)

Verknüpft mit OR oder |

- PS_GEOMETRIC Creates a geometric pen.
- PS_COSMETIC Creates a cosmetic pen.
- PS_ALTERNATE Creates a pen that sets every other pixel.
- PS_USERSTYLE Creates a pen that uses a styling array supplied by the user.
  The end cap can be one of the following values:
- PS_ENDCAP_ROUND End caps are round.
- PS_ENDCAP_SQUARE End caps are square.
- PS_ENDCAP_FLAT End caps are flat. The join can be one of the following values:
- PS_JOIN_BEVEL Joins are beveled.
- PS_JOIN_MITER Joins are mitered when they are within the current limit set by the::SetMiterLimit function. If the join exceeds this limit, it is beveled.
  - PS_JOIN_ROUND Joins are round.

pLogBrush: Points to a LOGBRUSH structure. If nPenStyle is PS_COSMETIC, the lbColor member of the LOGBRUSH structure specifies the color of the pen and the lbStyle member of the LOGBRUSH structure must be set to BS_SOLID. If nPenStyle is PS_GEOMETRIC, all members must be used to specify the brush attributes of the pen.

nStyleCount: Specifies the length, in doubleword units, of the lpStyle array. This value must be zero if nPenStyle is not PS_USERSTYLE.

lpStyle: Points to an array of doubleword values. The first value specifies the length of the first dash in a user-defined style, the second value specifies the length of the first space, and so on. This pointer must be NULL if nPenStyle is not PS_USERSTYLE.

lbStyle:

- BS_DIBPATTERN A pattern brush defined by a device-independent bitmap (DIB) specification. If lbStyle is BS_DIBPATTERN, the lbHatch member contains a handle to a packed DIB.
- BS_DIBPATTERNPT A pattern brush defined by a device-independent bitmap (DIB) specification. If lbStyle is BS_DIBPATTERNPT, the lbHatch member contains a pointer to a packed 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_SOLID Solid brush.

lbHatch

hatch style

Wenn lbStyle = BS_DIBPATTERN, dann defniert lbHatch eine gepackte DIB

Wenn lbStyle = BS_DIBPATTERNPT, dann defniert lbHatch einen Pointer zu einer gepackte DIB.

Wenn BS_HATCHED
HS_BDIAGONAL A 45-degree upward, left-to-right hatch
HS_CROSS Horizontal and vertical crosshatch
HS_DIAGCROSS 45-degree crosshatch
HS_FDIAGONAL A 45-degree downward, left-to-right hatch
HS_HORIZONTAL Horizontal hatch
HS_VERTICAL Vertical hatch

If lbStyle is BS_PATTERN, lbHatch is a handle to the bitmap that defines the pattern. If lbStyle is BS_SOLID or BS_HOLLOW, lbHatch is ignored.

Beispiel:

```
CPen cpen(PS_SOLID, 5, RGB(255,0,255) );
dc.SelectObject(&cpen);


    LOGBRUSH LogBrush;
    LogBrush.lbColor = RGB(0,0,255);
    LogBrush.lbStyle=BS_HATCHED;
    LogBrush.lbHatch=HS_BDIAGONAL;
    //LogBrush.lbHatch=HS_CROSS;
    //LogBrush.lbHatch=HS_DIAGCROSS;
    //LogBrush.lbHatch=HS_FDIAGONAL;
    //LogBrush.lbHatch=HS_HORIZONTAL;
    //LogBrush.lbHatch=HS_VERTICAL;

    CPen cpen3(PS_GEOMETRIC, 4, &LogBrush,0,NULL );
    dc.SelectObject(&cpen3);

    POINT points1[5] = { 100,100, 300,100, 300,300, 100,300, 100,100};
    dc.Polyline(points1,5);
```

1.5 Flächen-Operationen

- **Ellipse**(int x1, int y1, int x2, int y2);
- **Polyline**(LPPOINT lpPoints, int nCount);
- **Rectangle** (int x1, int y1, int x2, int y2);
- **RoundRect** (int x1, int y1, int x2, int y2);
- **FillRect**(Crect * rect; Cbrush brush);
- **Chord** (P1 bis P4); (siehe Arc)
- **Pie**(P1 bis P4); (siehe Arc)

- **Polygon** (LPPOINT lpPoints, int nCount);
  POINT p[5] = { 100,100, 300,100, 300,300, 100,300, 100,100};
  Polygon(p,5);

- BOOL **PolyPolygon**(LPPOINT lpPoints, LPINT lpPolyCounts, int nCount);

lpPoints: Points to an array of POINT structures or CPoint objects that define the vertices of the polygons.

lpPolyCounts: Points to an array of integers, each of which specifies the number of points in one of the polygons in the lpPoints array.

nCount: The number of entries in the lpPolyCounts array. This number specifies the number of polygons to be drawn. This value must be at least 2.

Beispiel:

```
POINT points1[7] =
{
    100,100, 300,100, 300,300, 100,300,
    300,10, 330,40, 370,10,
};
int PolyCounts[2] ={ 4,3 };
dc.PolyPolygon(points1,PolyCounts, 2);
```

1.6 CBrush

1.6.1 Konstruktoren

CBrush::CBrush
CBrush( );

- **CBrush**(COLORREF crColor); throw(CResourceException);
- **CBrush**(int nIndex, COLORREF crColor); throw(CResourceException);
- **CBrush**(CBitmap* pBitmap); throw(CResourceException);

crColor: Specifies the foreground color of the brush as an RGB color. If the brush is hatched, this parameter specifies the color of the hatching.

nIndex: Specifies the hatch style of the brush. It can be any one of the following values:

Konstanten des Musters

- HS_BDIAGONAL Downward hatch (left to right) at 45 degrees
- HS_CROSS Horizontal and vertical crosshatch
- HS_DIAGCROSS Crosshatch at 45 degrees
- HS_FDIAGONAL Upward hatch (left to right) at 45 degrees
- HS_HORIZONTAL Horizontal hatch
- HS_VERTICAL Vertical hatch

pBitmap: Points to a CBitmap object that specifies a bitmap with which the brush paints.

Bemerkungen:

Has four overloaded constructors. The constructor with no arguments constructs an uninitialized CBrush object that must be initialized before it can be used.

If you use the constructor with no arguments, you must initialize the resulting CBrush object with CreateSolidBrush, CreateHatchBrush, CreateBrushIndirect, CreatePatternBrush, or CreateDIBPatternBrush. If you use one of the constructors that takes arguments, then no further initialization is necessary. The constructors with arguments can throw an exception if errors are encountered, while the constructor with no arguments will always succeed.

The constructor with a singleCOLORREF parameter constructs a solid brush with the specified color. The color specifies an RGB value and can be constructed with the RGB macro in WINDOWS.H.

The constructor with two parameters constructs a hatch brush. The nIndex parameter specifies the index of a hatched pattern. The crColor parameter specifies the color.

The constructor with a CBitmap parameter constructs a patterned brush. The parameter identifies a bitmap. The bitmap is assumed to have been created by using CBitmap::CreateBitmap, CBitmap::CreateBitmapIndirect, CBitmap::LoadBitmap, or CBitmap::CreateCompatibleBitmap. The minimum size for a bitmap to be used in a fill pattern is 8 pixels by 8 pixels.

1.6.2 Initilisierungen

- CreateSolidBrush Initializes a brush with the specified solid color.
- CreateHatchBrush Initializes a brush with the specified hatched pattern and color.
- CreateBrushIndirect Initializes a brush with the style, color, and pattern specified in aLOGBRUSH structure.
- CreatePatternBrush Initializes a brush with a pattern specified by a bitmap.
- CreateDIBPatternBrush Initializes a brush with a pattern specified by a device-independent bitmap (DIB).
- CreateSysColorBrush Creates a brush that is the default system color.

1.6.3 Beispiel 1

```
CBrush brush(   RGB(255,0,255)   );
dc.SelectObject(&brush);          // setzen des Objekts
dc.Ellipse(100,100,  500,500);    // Ellipse  muster
dc.Arc(170,360, 420,420,   220,400, 390,400);    // Ellipse
```

1.6.4 Beispiel 2

```
CBrush brush2( HS_DIAGCROSS, RGB(255,0,255) );
dc.SelectObject(&brush2);
dc.Arc(170,360, 420,420,   220,400, 390,400);    // Ellipse
```

1.6.5 Beispiel 3

```
CBrush brush3;
brush3.CreateSolidBrush( RGB(255,255,255));
dc.SelectObject(&brush3);
dc.Arc(170,360, 420,420,   220,400, 390,400);    // Ellipse
```

1.7 CFont

CFont();

- Constructs a **CFont** object.
- The resulting object must be initialized with **CreateFont**, **CreateFontIndirect**, **CreatePointFont**, or **CreatePointFontIndirect** before it can be used.

Beispiel:

```
CFont font;
font.CreatePointFont(140,"Times New Roman");
CFont* oldfont = pDC->SelectObject( &font);

dc.DrawText ( "Mein erster Text", -1, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);

dc.SetTextColor( RGB(255,0,0) );
dc.TextOut(100,550,"DrawText");

CRect rect1(0,0,200,100);
dc.DrawText ( "ABCDEFGHIJKLMNOPQRSTUVWXYZ", -1, &rect1, DT_SINGLELINE | DT_LEFT);

pDC->SelectObject( oldfont );

siehe Dokumentation Fonts
```

pDC->SetBkMode(TRANSPARENT); // Verhindern, dass ein weißer Rand um den Text gezeichnet wird.

Methoden zum Zeichnen eines Textes:

- DrawText
- TextOut
- ExTextOut
- TabbedTextOut
- GrayString

virtual int **DrawText**(LPCTSTR lpszString, int nCount, LPRECT lpRect, UINT nFormat);
int **DrawText**(const CString& str, LPRECT lpRect, UINT nFormat);

lpszString: Points to the string to be drawn. If nCount is –1, the string must be null-terminated.
nCount: Specifies the number of chars in the string. If nCount is –1, then lpszString is assumed to be a long pointer to a null-terminated string and DrawText computes the character count automatically.
lpRect: Points to a RECT-Structure
str: A Cstring-object that contains the specified characters to be drawn.
nFormat: Specifies the method of formatting the text. It can be any combination of the following values (combine using the bitwise OR operator):

Optionen:

- DT_BOTTOM: + DT_SINGLELINE.
- DT_CALCRECT: DrawText returns the height of the formatted text, but does not draw the text.
- DT_BOTTOM:
- DT_CENTER:
- DT_LEFT
- DT_NOCLIP
- DT_NOPREFIX (&& vs. &)
- DT_RIGHT
- DT_TABSTOP
- DT_TOP
- DT_VCENTER
- DT_WORKBREAK (Trennung)

- virtual BOOL **TextOut**(int x, int y, LPCTSTR lpszString, int nCount);
- BOOL **TextOut**(int x, int y, const CString& str);

x:	Specifies the logical x-coordinate of the starting point of the text.
y:	Specifies the logical y-coordinate of the starting point of the text.
lpszString:	Points to the character string to be drawn.
nCount:	Specifies the number of bytes in the string.
Str:	A CString object that contains the characters to be drawn.

Optionen:

pDC->SetTextAlign(Optionen);
- TA_BOTTOM
- TA_BASELINE
- TA_TOP
- TA_LEFT
- TA_CENTER
- TA_RIGHT
- TA_NOUPDATECP
- TA_UPDATECP

- BOOL **ExtTextOut**(int x, int y, UINT nOptions, LPCRECT lpRect, const CString& str, LPINT lpDxWidths);

x:	The logical x-coordinate of the character cell for the first character
y:	The logical y-coordinate of the top of the character cell for the first character
nOptions:	Specifies the rectangle type:
ETO_CLIPPED	Specifies that text is clipped to the rectangle.
ETO_OPAQUE	Specifies that the current background color fills the rectangle.
lpRect:	Points to a RECT structure that determines the dimensions of the rectangle. This parameter can be NULL. You can also pass a CRect object for this parameter.
lpszString:	Points to the specified character string to be drawn. nCount: Specifies the number of characters in the string.
lpDxWidths:	Points to an array of values that indicate the distance between origins of adjacent character cells. For instance, lpDxWidths[i] logical units will separate the origins of character cell i and character cell i + 1. If lpDxWidths is NULL, ExtTextOut uses the default spacing between characters.
str:	A CString object that contains the specified characters to be drawn.

Beispiel:

```
CString str("Hallo Text");
int lpDxWidths[11] = {42,52,52,33,44,55,33,33,33,22};
dc.ExtTextOut(50,100,ETO_OPAQUE, NULL,    str, lpDxWidths);
```

1.8 Boolesche Operatoren beim Zeichnen

Bit-Operator

R2_NOP	color = color
R2_NOT	color = NOT color
R2_BLACK	color = BLACK
R2_WHITE	color = WHITE
R2_COPYPEN	color = NewColor
R2_NOTCOPYPEN	color = NOT NewColor
R2_MERGEPENNOT	color = (NOT color) OR NewColor
R2_MASKPENNOT	color = (NOT color) AND NewColor
R2_MERGENOTPEN	color = (NOT NewColor) OR color
R2_MASKNOTPEN	color = (NOT NewColor) AND color
R2_MERGEPEN	color = color OR NewColor
R2_NOTMERGEPEN	color = NOT (color OR NewColor)
R2_MASKPEN	color = color AND NewColor
R2_NTMASKPEN	color = NOT (color AND NewColor)
R2_XORPEN	color = color XOR NewColor
R2_NOTXORPEN	color = NOT (color XOR NewColor)

Beispiel:
dc.SetROP2(COPYPEN);

1.9 Koordinatensysteme

- Logische Koordinaten, sind Daten, die einer CDC-Methode übergeben werden, um etwas zu zeichnen.
- Physikalische Koordinaten bezeichnen die konkreten Bildpunkte des Ausgabegerätes
  - Bildschirm
  - Drucker
  - Metafile
  - Speicher
- Abbildungmodi:
  - MM_TEXT 1 Bildpunkt +x / +y
  - MM_LOMETRIC 0,1 mm +x / -y
  - MM_LOENGLISH 0,01 Zoll +x / -y
  - MM_HIENGLISH 0,001 Zoll +x / -y
  - MM_TWIPS 1/440 Zoll +x / -y
  - MM_ISOTROPIC benutzerdefiniert x/y gleichskaliert
  - MM_ANISOTROPIC benutzerdefiniert x/y-Skalierung unabhängig
- dc.SetMapMode(MM_LOMETRIC);

1.10 Fertige GDI-Objekte im Stock

NULL_PEN	Stift, der nicht zeichnet
BLACK_PEN	Stift, schwarz, solid, 1 Pixel
WHITE_PEN	Stift, weiß, solid, 1 Pixel
NULL_BRUSH	Pinsel, der nicht zeichnet
HOLLOW_BRUSH	Pinsel, der nicht zeichnet
BLACK_BRUSH	Pinsel, schwarz
DKGRAY_BRUSH	Pinsel, dunkelgrau
GRAY_BRUSH	Pinsel, grau
LTGRAY_BRUSH	Pinsel, hellgrau
WHITE_BRUSH	Pinsel, weiß
ANSI_FIXED_FONT	Ansi-Schrift, nicht proportional (à la Courier)
ANSI_VAR_FONT	Ansi-Schrift, proportional (à la Helvetica, Garamond)
SYSTEM_FONT	System-Schrift proportional
SYSTEM_FIXED_FONT	System-Schrift nicht proportional (FixedSys)

Beispiele:

```
CPen pen2(PS_NULL, 0, RGB(0,0,0));
CPen pen;
pen.CreateStockObject(NULL_PEN);
dc.SelectObject(&pen);

Cbrush brush;
brush.CreateStockObject( GRAY_BRUSH);
dc.SelectObject(&brush);
dc.Ellipse(0,0,100,200);



dc.SelectStockObject(NULL_PEN);
dc.SelectStockObject(LTGRAY_BRUSH);
dc.Ellipse(0,0,100,200);
```