

24. Programmgesteuertes Zeichnen (Canvas)

Bisher haben wir nur fertige Bilder benutzt, z.B. auf einen *BitBtn* oder in ein *Image* gesetzt. Jetzt werden wir von Delphi während der Programmausführung Grafiken zeichnen lassen.

Genau genommen zeichnet Delphi nicht direkt auf das Objekt, sondern auf dessen Zeichenfläche („Leinwand“, engl. **Canvas**). Diese Zeichenfläche ist eine **Eigenschaft**, die nur wenige Objekte besitzen: Das *Formular* selbst oder ein *Image* haben so eine Zeichenfläche, fast alle anderen Objekte nicht.

Auf der Canvas lassen sich Linien und Flächen zeichnen sowie Text ausgeben:

- x **Canvas.MoveTo (x,y)** setzt den Cursor auf die Koordinaten (x,y). *Wie üblich: links oben ist (0,0) ...*
- x **Canvas.LineTo (x,y)** zeichnet eine gerade Linie von der aktuellen Cursorposition bis zur neuen Cursorposition (x,y). *Mitgedacht?! Der Cursor steht also danach am Linienende (x,y) und wartet dort!*

Achtung: Die Koordinaten beziehen sich immer auf das Objekt, dessen Canvas angesprochen ist. Zum Beisp.

```
Form1.Canvas.MoveTo (0,0);  
Form1.Canvas.LineTo (Form1.Width, Form1.Height);  
  
... zeichnet eine Diagonale über das gesamte  
Formular.
```

```
Image22.Canvas.MoveTo (0,0);  
Image22.Canvas.LineTo (Image22.Width, Image22.Height);  
  
... zeichnet eine Diagonale über das angegebene  
Image, egal wo auf dem Formular es sich befindet.
```

- x **Canvas.Rectangle (x1,y1, x2,y2)** zeichnet ein Rechteck. Angegeben werden die Koordinaten von zwei diagonal gegenüberliegenden Punkten (x1,y1) und (x2,y2) des Rechtecks.
- x **Canvas.Ellipse (x1,y1, x2,y2)** zeichnet eine Ellipse in einem imaginären Rechteck mit den angegebenen Koordinaten. Ist das Rechteck quadratisch, so wird ein Kreis gezeichnet.
- x **Canvas.Textout (x,y, 'BlaBlaBla')** gibt den in Häkchen stehenden Text an der Position (x,y) aus.
- x **Canvas.Pixels[x,y] := clGreen** setzt einen einzelnen grünen Pixel an die Position (x,y). Achtung: Die Klammern müssen wirklich eckig sein, genau wie bei den Zellen eines StringGrids!
- x **Canvas.FloodFill (x,y, color, fsBorder)** füllt ausgehend vom Punkt (x,y) einen mit der angegebenen Farbe *color* umrandeten Bereich des Bildes mit der aktuellen Brush-Farbe. Wenn die Fläche nicht lückenlos berandet ist, wird die gesamte Canvas „geflutet“ – dumm gelaufen...

Weitere Methoden musst du nicht wissen, für Neugierige folgt aber noch eine Auflistung einiger weiterer Zeichenbefehle. Bei Bedarf kannst du dir die notwendigen Informationen aus der Hilfedatei beschaffen.

- Arc, BrushCopy, Chord, CopyRect, Draw, DrawFocusRect, FillRect, FrameRect, Pie, Polygon, PolyLine, RoundRect, StretchDraw, TextHeight, TextRect, TextWidth.

Farben und Strichstärken sind natürlich einstellbar, dabei bezieht sich bekanntlich die Eigenschaft **Pen** immer auf die (Umrandungs-)linie, die Eigenschaft **Brush** auf den inneren Füllbereich (also die Fläche):

- x **Canvas.Pen.Color** := clRed oder Canvas.Pen.Color := RGB(111,222,155) legt die Farbe fest, mit der Linien gezeichnet werden.
- x **Canvas.Pen.Width** := 5 legt die Breite des Stiftes auf 5 Pixel fest. Beim Programmstart ist die Strichbreite=1.
- x **Canvas.Brush.Color** := clNavy oder Canvas.Brush.Color := RGB(0,8,15) legt die Farbe fest, mit der Flächen gefüllt werden.
- x **Canvas.Brush.Style** regelt das Flächenfüllmuster. Die Normaleinstellung ist *bsSolid*, möglich sind aber auch *bsClear*, *bsHorizontal*, *bsVertical*, *bsFDiagonal*, *bsBDiagonal*, *bsCross*, *bsDiagCross*.

25. Der WITH-Befehl

Insbesondere bei Zeichenbefehlen ist es lästig, immer den gleichen „Vorspann“ zu notieren. Dies lässt sich mit dem **WITH ... DO** - Befehl einsparen. Damit sieht das Listing doch gleich etwas übersichtlicher aus...

Beispiel: **WITH** Image1.Canvas **DO**

BEGIN

```
Brush.Color := clRed;  
Brush.Style := bsDiagCross;  
Ellipse (5,5, Image1.Width,234);  
Font.Color := clGreen;  
Font.Size := 44;  
Font.Name := 'Arial';  
Textout (20,20, 'Es lebe der Bolzewismus');
```

END;