Windows CE 3.0

Programmiertipp: Die Picturebox beherrschen

Gerade der CE - Bildschirm mit seinen Farbmöglichkeiten will mit grafischen Elementen aufgelockert werden. Kleine Bilder haben oftmals eine große Aussagekraft. Wer es gewöhnt ist, mit Visual Basic 6 die Bilder mit der picturebox zu verwalten, der wird bei CE ein wenig enttäuscht sein.

Nicht alle Funktionen, die man gewöhnt ist, lassen sich nutzen. So ist es zumindest bei der fortgeschrittenen Benutzung der Control. Wer es gewohnt ist, die API - Aufrufe Bitblt und Stretchblt zu benutzen, der wird zuerst einmal enttäuscht sein, dass es nicht unmittelbar funktioniert. In diesem kleinen Artikel soll gezeigt werden, wie man die Kontrolle über die Picturebox zurückerhält.

Zunächst aber einmal zurück zum normalen VB6. Nicht jeder, so habe ich in Gesprächen erfahren, weiß über die Möglichkeiten Bescheid, die man mit einer Picturebox im Programm unternehmen kann. Deshalb hier ein paar einführende Erklärungen.

Eine Picturebox ist ein Container für eine Grafikdatei. In VB beschränkt man sich weitgehend auf BMP - Dateien. Diese Grafikdateien sind häufig vertreten und sie verbrauchen auch den meisten Speicherplatz. Dafür sind .bmp - Dateien auch immer wieder ins Original zurückzuverwandeln.

Gehen wir ganz einfach einmal von einer Schwarz-Weiß - Grafik aus. Sie soll in einer .BMP - Datei abgespeichert werden. Zunächst wird ein Bitmap- Header und ein Bitmap-Info Header vorhergesetzt, in dem alles steht, was das aufrufende Programm wissen muss: Grösse der Grafik, Farbanteile der Grafik, Paletteninformationen usw. Sie beginnt auf jeden Fall immer mit den beiden beiden Buchstaben BM, dieses steht wohl für Bitmap.

```
' Bitmap Header Definition
Type BITMAP '14 bytes
    bmType As Long
    bmWidth As Long
    bmHeight As Long
    bmWidthBytes As Long
    bmPlanes As Integer
    bmBitsPixel As Integer
    bmBits As Long
Type BITMAPINFOHEADER '40 bytes
    biSize As Long
    biWidth As Long
    biHeight As Long
    biPlanes As Integer
    biBitCount As Integer
    biCompression As Long
    biSizeImage As Long
    biXPelsPerMeter As Long
    biYPelsPerMeter As Long
    biClrUsed As Long
    biClrImportant As Long
End Type
```

Nach dem Header folgen die Daten. Je nach Farbtiefe wird dies verschieden arrangiert. Handelt es sich um eine Schwarz-Weiß - Grafik, so kann jedes Byte in der Bitmapdatei nach dem Header 8 Bildpunkte speichern, da ja nur 0 oder 1 auftreten kann. Für eine SW-Bitmapgrafik mit 100 * 100 Pixel = 10.000 Pixel ergibt sich daher eine ca. Dateigröße vo 10.000 / 8 + Header. Ein kleines Beispiel soll dies noch verdeutlichen: wir haben eine Bitmapzeile mit folgendem Inhalt:

(0=schwarz,1 gleich weiß)

01110101000100010011110010101011110001000001000011110000

Jetzt habe ich mir selbst eine Tortur angetan, indem ich einfach losgetippt habe. Das Auseinandernehmen macht jetzt ein wenig Arbeit. Wir sortieren jeweils 8 Zeichen aus und machen ein Byte daraus. Die ASCII -

Werte sind unten aufgeführt.

```
01110101 = 117 (dez)
00010001 = 17
00111100 = 60
10101011 = 171
11000100 = 196
00010000 = 16
11110000 = 240
```

Mit diesen Werten (mit den Zeichen der Werte ***chr(ASCII)***) wird der Bitmapfile gefüllt. Man beginnt seltsamerweise unten links innerhalb der Bitmap die Pixel abzuscannen, läuft dann nach rechts, dann entsprechend die nächste Zeile, bis man oben angelangt ist. Bei einer SW- Ggrafik also: 8 Pixel können in einem Byte gespeichert werden.

Mit dem Notepad oder einem anderen Editor sollten Sie sich die Struktur der Bitmap einmal ansehen.

Bei einerf 16 Farbengrafik sieht dies natürlich anders aus: Für 16 Farben zu speichern benötigt man 4 Bit: Wertigkeiten 1,2,4,8 . Damit lassen sich nur noch noch 2 Bildpixel in einem Byte speichern.

Entsprechend sieht es bei 256 Farben aus: hier benötigt jeder Bildpunkt ein Byte zur Speicherung.

Bei größeren Farbauflösungen muss dann entsprechend mit mehreren Bytes pro Pixel gespeichert werden. Bei 65535 Farben sind es zwei Bytes, ein Highbyte und ein Lowbyte.

Soweit der Ausflug in die Theorie.

Will man die CE- Picturebox benutzen, so stellt man fest, dass man das gewünschte Bild zur Entwurfzeit nicht direkt einladen kann. Es muss dynamisch eingelesen werden. Dazu der einfache Befehl:

```
Picture1.picture="abc.bmp"
```

oder besser mit vollständigem Pfad:

```
Picture1.picture=app.path+"\abc.bmp".
```

Das Löschen der Bitmap ist ebenfalls recht einfach. Man übergibt ein leeres Bild:

```
Picture1.picture=""
```

Soweit klappt es ja ganz gut und man kann die Programme mit netten Grafiken anreichern.
Wenn man dann jedoch in die Spezialitäten gehen will, so muss man noch etwas Praxis gewinnen.

Will man z.B. eine Picturebox mit ihren Pixeln auslesen, so versagt das normale CE. Zumindest habe ich keinen Befehl gefunden, der dem V6B6 -Befehl `point(x,y)` entspricht. Mit `farbe = point(x,y)` wird die Farbe des gewählten Pixels zurückgegeben.
Will man so etwas in CE benutzen, so ist die API zu bemühen.

Hier gibt es zwei Funktionen, die diese Manipulation zulassen.

Declare Function GetPixel Lib "Coredll" Alias "GetPixel" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long) As Long

Declare Function SetPixel Lib "Coredll" Alias "SetPixel" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal crColor As Long) As Long

Bei der ersten Funktion ist es klar: Mit der Übergabe4 von x und y als Koordinatenwerte gibt die Funktion den Farbwert des ausgesuchten Pixels zurück.

```
dim r
r = getpixel(200,200)
```

Liefert r z.B. den Wert 255, so ist ein roter Pixel an der Stelle 200,200 im Bild.

Doch ein unschöner Aufruf ist noch in der Funktion versteckt - zumindest für den Anfang. Es muss der *hdc* - der Devicecontext der Picturebox übergeben werden.
Bei VB6 ist dies kein Problem, da dort die Picturebox den *hdc* als Eigenschaft mit sich führt. Hier kann also der Wert *picture1.hdc* abgefragt und eingesetzt werden. Bei VBCE ist es etwas umständlicher. Der *hdc* muss eigens ermittelt werden.

Auch hierbei hilft wieder die API. Zunächst aber muss der Handle (hwnd) her, der von Windows jeweils als eigene Nummer zur Kennzeichnung an eine Control vergeben wird. Bei VBCE hat zum Beispiel die Form einen abfragbaren *handle* und ebenso einen *hdc*.

Zunächst benötigen wir aus der WINCEAPI.TXT die beiden Funktionen:

```
Declare Function GetFocus Lib "Coredll" Alias "GetFocus" () As Long

Declare Function GetDC Lib "Coredll" Alias "GetDC" (ByVal hwnd As Long) As Long
```

Zunächst muss man die Control (hier die Picturebox) aktivieren, damit der *hdc* und der *handle* dieser Control bestimmt werden kann.

```
Picture1.SetFocus
```

Damit ist die *Picturebox1* markiert.

```
Dim phandle
phandle = getfocus()
```

Der gewonnene *phandle* wird jetzt zur Bestimmung des Kontextes *phdc* eingesetzt:
phdc = getdc (phandle)

Jetzt ist es geschafft *phandle* kann jetzt in die beiden ersten API - Aufrufe *setpixel* und *getpixel* eingesetzt werden.

Jetzt wird ess noch etwas komplizierter. Es sollen jetzt Teile einer Picturebox kopiert w rden und diese in eine andere Picturebox eingefügt werden. Auch hierzu benötigt man den Devicekontext. Später soll dann noch das kopierte Teilstück noch vergrößert oder verkleinert (gestretcht) werden.

Die beiden Funktionenr *bitblt* und *stretchblt* sind mächtige Werkzeuge, die man für viele grafische Manipulationen einsetzen kann.

In dem untereren Bild sollen die einzelnen kleinen Bilder (aus einem Memoryspiel) kopiert werden und in eine zweite Picturebox eingefügt werden.

Man soll auf das gewünschte Bild klicken und die Kopie landet in der zweiten Picturebox mit den Maßen 32 *32 Pixel .

Zur Realisierung genügt die erste (kürzere) Funktion: *bitblt*

```
Declare Function BitBlt Lib "coredll" (ByVal hDestDC As Long, ByVal x As Long,
ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As
Long,ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long
```

Vielleicht sieht die obige Funktion auf den ersten Blick etwas wirr aus. Sie ist aber schnell erklärt. Wir benötigen zwei Kontexte (*hdc*) der beiden benutzten Pictureboxen. Zum einen den dest=destination=Ziel - *hdc*, zum anderen den src=source=Quelle - *hdc*.

Die Quelle ist Picturebox1, das Ziel ist Picturebox2. Daher müssen wir zwei verschiedene *hdc´*s ermitteln.

In der Funktion müssen weitere Parameter eingegeben werden. X, y, nwidth, nheight gehören zum Ziel. Die Kopie wird an der linken oberen Ecke x,y begonnen und nwidth breit und nheight hoch eingefügt. Es folgen die Eingaben für die Quellbitmap. Zunächst ist es wieder der hdc, dann die Koordinaten für den linken oberen Punt xsrc und ysrc, an dem die Kopie beginnt.

In unserem Beispiel ist die Zielbox 32*32 Pixel groß,so dass der erste Teil der Funktion so aussieht:

```
r=BitBlt (DestDC ,0, 0, 32, 32, SrcDC , ???,???, srccopy)
```

Die Fragezeichen müssen noch aufgefüllt werden. Das Quellbild besteht aus 16 kleinen Bildern,die jeweils 32*32 Pixel groß sind. Sedtzt man die Abfrage in den MouseDown Bereich des Qellbildes, so kann man den angewählten Punkt auf dem Bild mit x und y bestimmen.

```
Option Explicit

Declare Function StretchBlt Lib "coredll" _
(ByVal hdc As Long, _
ByVal x As Long, _
ByVal y As Long, _
ByVal nWidth As Long, _
ByVal nHeight As Long, _
ByVal hSrcDC As Long, _
ByVal xSrc As Long, _
ByVal ySrc As Long, _
ByVal nSrcWidth As Long, _
ByVal nSrcHeight As Long, _
ByVal dwRop As Long) As Long

Declare Function GetFocus Lib "coredll" () As Long
Declare Function GetDC Lib "coredll" (ByVal hwnd As Long) As Long
Declare Function SetPixel Lib "coredll" _
    (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, _
    ByVal crColor As Long) As Long
Declare Function GetPixel Lib "coredll" _
    (ByVal hdc As Long, ByVal x As Long, ByVal y As Long) As Long

Const SRCCOPY = &HCC0020 ' (DWORD) dest = source
Const SRCPAINT = &HEE0086        ' (DWORD) dest = source OR dest
Const SRCAND = &H8800C6  ' (DWORD) dest = source AND dest
```

```
Const SRCINVERT = &H660046       ' (DWORD) dest = source XOR dest
Const SRCERASE = &H440328        ' (DWORD) dest = source AND (NOT dest )
Const NOTSRCCOPY = &H330008      ' (DWORD) dest = (NOT source)
Const NOTSRCERASE = &H1100A6     ' (DWORD) dest = (NOT src) AND (NOT dest)
Const MERGECOPY = &HC000CA       ' (DWORD) dest = (source AND pattern)
Const MERGEPAINT = &HBB0226      ' (DWORD) dest = (NOT source) OR dest
Const PATCOPY = &HF00021 ' (DWORD) dest = pattern
Const PATPAINT = &HFB0A09        ' (DWORD) dest = DPSnoo
Const PATINVERT = &H5A0049       ' (DWORD) dest = pattern XOR dest
Const DSTINVERT = &H550009       ' (DWORD) dest = (NOT dest)
Const BLACKNESS = &H42 ' (DWORD) dest = BLACK
Const WHITENESS = &HFF0062       ' (DWORD) dest = WHITE


Private Sub Combo1_Change()

End Sub

Private Sub Combo1_Click()
Dim desthwnd, desthdc, srchwnd, srchdc, r
pb2.ZOrder 0
pb2.SetFocus
desthwnd = GetFocus()
desthdc = GetDC(desthwnd)
pb1.SetFocus
srchwnd = GetFocus()
srchdc = GetDC(srchwnd)
r = BitBlt(desthdc, 0, 0, 32, 32, srchdc, _
(CInt(Combo1.List(Combo1.ListIndex)) - 1) * 32, 0, SRCCOPY)

pb3.SetFocus
desthwnd = GetFocus()
desthdc = GetDC(desthwnd)
pb2.SetFocus
srchwnd = GetFocus()
srchdc = GetDC(srchwnd)

r = StretchBlt(desthdc, 0, 0, pb3.Width, pb3.Height, srchdc, 0, 0, pb2.Width, pb2.Height,
MERGECOPY)
End Sub

Private Sub Command1_Click()
App.End
End Sub

Private Sub Command2_Click()
pb2.Cls
pb3.Cls
End Sub

Private Sub Form_Load()
Dim i
For i = 1 To 16
Combo1.AddItem CStr(i)
Next i
pb1.Picture = App.Path + "\memory.bmp"
End Sub

Private Sub pb1_Click()

End Sub

Private Sub VScroll1_Change()
pb1.Top = VScroll1.Value
End Sub
```