Quick-Info in Visual-Basic

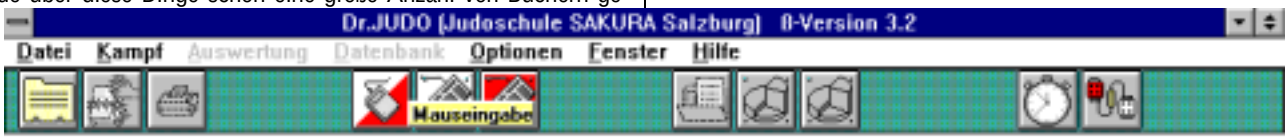
Sven Schweiger

Seit unsere PCs immer leistungsfähiger und die Anwenderprogramme immer umfangreicher werden, wurde auch der Ruf nach neuen Sprachen laut: Von Windows oder OS/2 ist man auch am PC an eine grafische Benutzeroberfläche gewöhnt, deshalb erscheint ein in C oder Pascal geschriebenes Programm unter DOS für jeden Anwender antiquiert, auch wenn es sehr gut ist. Microsoft begann daraufhin, Visual Basic zu entwickeln, wobei der Grundgedanke war, daß dem Programmierer schon fertige grafische Objekte zur Verfügung stehen, die er dann nach seinen Vorstellungen verwenden kann!

In der Version VB 3.0 professional (Version 4.0 wird für Windows 95 herauskommen), die derzeit aktuell ist, läßt sich durch die Objektorientiertheit der Sprache sehr schnell ein optisch perfektes und anwenderfreundliches Programm erstellen, das von der Komplexität her nur durch die Ausführungsgeschwindigkeit begrenzt ist. Visual C++ ist in dieser Hinsicht um Klassen schneller, aber auch um den selben Faktor unangenehmer zu programmieren! Objektorientiert bedeutet dabei, daß es kein Hauptprogramm im herkömmlichen Sinn mehr gibt, wo eine Schleife permanent durchlaufen wird und nach deren Abbruch auch das Programm stoppt: In VB 3.0 wird zuerst die grafische Oberfläche gezeichnet (Toolbox mit fast allen „Windows-Bauteilen“ wie Buttons etc. ist vorhanden!). Die einzelnen „Programmblöcke“ werden dann ausgeführt, wenn das gewünschte Ereignis (z.B. Mausklick o.ä.) am betreffenden Objekt ausgelöst wird! Man nennt dies deshalb auch ereignisorientiertes Programmieren.

Wenn es sich nicht um besonders zeitkritische Anwendungen (Animationen in Spielen...) handelt, so handelt es sich bei VB 3.0 um ein interessantes Werkzeug in der Programmierwelt!

Es wurde über diese Dinge schon eine große Anzahl von Büchern geschrieben, deshalb möchte ich in der Folge auf ein spezielles Programmierproblem eingehen. Besonders empfehlenswert sind übrigens die VB-Wälzer von Addison-Wesley (gut zum Nachschlagen bei Problemen) und Data Becker (klassischer Stil) sowie das sehr „beispielstarke“ Buch von Sybex!



Wer schon Winword 6.0 oder andere moderne Programme verwendet hat, der weiß jene kleinen gelben Kästchen zu schätzen, die mit einer Kurzhilfe erscheinen, wenn man mit der Maus länger auf einem Button stehen bleibt! In meinem Freundeskreis wurde die Programmierung eines solchen Hilfesystems in VB eifrigst diskutiert, da sich die Geschichte als gar nicht so einfach herausstellte. Deshalb möchte ich hier meine Lösungsvariante für alle VB-Begeisterten erklären:

Das Problem liegt unangenehmerweise genau da, wo die Vorteile von Visual-Basic beginnen, nämlich in der Ereignisorientiertheit der Sprache! Zuerst dachten wir, daß wir nur im Ereignis `MouseMove` des betreffenden Buttons ein Textfeld mit dem gewünschten Hilfetext `Visible` schalten müßten... So weit, so gut, aber wie und vor allem wann soll das Feld wieder unsichtbar gemacht werden, damit der Text wieder verschwindet? Das Ereignis `MouseMove` tritt nur auf, solange die Maus über dem Button bewegt wird, nicht aber beim Verlassen dieses Bereiches (und auch nicht beim Stillstand am Button)! Die Ereignisse `LostFocus` oder `MouseClick` sind auch unbrauchbar, weil der Button vielleicht gar nicht gedrückt wird. Nach vielen Versuchen mit diversen VB-Ereignissen und Tricks mußten wir schon bald erkennen, daß VB für ein solches Problem keine vernünftigen Lösungsmöglichkeiten bereitstellt, die dann auch zuverlässig funktionieren und nicht durch schnelle Mausbewegungen usw. auszutricksen sind!

In so einem Fall bleibt nur der Weg über die direkte Windows-Programmierung, die zwar nicht ganz einfach zu handhaben ist, aber nach einiger Übungszeit praktisch unbegrenzte Möglichkeiten in VB 3.0 bietet. Der Trick besteht darin, daß Windows eine Vielzahl fertiger, in C/C++ programmierter Funktionen (API-Funktionen) zur Verfügung stellt, die man in das eigene VB-Programm einbinden kann. Eine gute Erklärung dieser Funktionen findet sich übrigens in der WinAPI Hilfe von Borland-C++ für Windows!

Die Übergabe der Variablen von C in VB gestaltet sich nicht immer ganz problemlos, sie ist aber in allen guten VB-Büchern ausreichend erklärt, weshalb ich hier nicht näher darauf eingehen möchte!

Meine Idee zur Online-Hilfe war nun, durch das erste `MouseMove` Ereignis eine Abfrageroutine Auszulösen, die über die API-Funktion `GetCursorPos` solange die aktuelle Mausposition überprüft, bis der Button wieder verlassen wird. Dabei werden die aktuellen Mauskoordinaten mit den Buttonkoordinaten verglichen. Daraus ergibt sich, wann der Text wieder abgeschaltet wird. Weil `GetCursorPos` aber das Ergebnis in Pixel liefert, hat mir diese Funktion einige Stunden verärgerten Kampf mit der Technik beschert:

Um die Koordinaten vergleichen zu können, braucht man zuerst einmal die Koordinaten des Buttons. Dazu können die Eigenschaften `Top`, `Left`, `Height` und `Width` des jeweiligen Objektes verwendet werden. Leider sind diese Koordinaten in VB aber immer auf jenes Form (=Fenster) bezogen, in dem der Button gerade sitzt. Diese Kleinigkeit läßt sich beheben, indem man die Koordinaten der einzelnen Forms addiert, wobei aber die Einheit defaultmäßig auf Twips und nicht auf Pixel steht! Und zu aller Freude kann diese Einheit ausgerechnet bei einem MDI-Fenster nicht auf Pixel umgestellt werden. Das MDI-Fenster ist aber jenes Hauptfenster, daß man wahrscheinlich in 90% aller professionellen Programme als Hintergrund verwendet...

Deshalb müssen die Koordinaten der Buttons dann mit dem VB-Befehl (!) `TwipsPerPixel` auf absolute Koordinaten umgerechnet werden!

Mit diesen Informationen sollte es jedem geübten VB-Freak problemlos möglich sein, sein eigenes Profi-Hilfesystem aufzubauen, damit die Sache aber verständlicher wird, steht im Folgenden das Wichtigste gleich in VB 3.0:

Zuerst muß im `MouseMove` Ereignis das gewünschte Hilfefenster aktiviert und ein Überprüfungstimer gestartet werden. Dieser überprüft nun in einem gewählten Abstand, ob die Maus noch über dem Button steht:

Zuerst API-Funktion im BAS-Modul deklarieren:

```
Declare Sub GetCursorPos Lib "User" (XY&)
```

Dann aktuelle Werte holen und in 2 Integer VB-Variablen umrechnen (Longint-Variable in obere und untere Bits für X- und Y-Wert trennen):

```
GetCursorPos XY&
XX = CInt(XY& And &HFFF)
YY = CInt(XY& / &H10000)
```

Jetzt die aktuellen Pixel in Twips umrechnen:

```
ScaleX = Screen.TwipsPerPixelX
ScaleY = Screen.TwipsPerPixelY
XX = (XX * ScaleX)
YY = (YY * ScaleY)
```

`XX, YY` sind jetzt die aktuellen, absoluten Windows-Koordinaten der Maus in Twips!

Vorsicht: Der Punkt 0/0 ist ganz oben links in der Ecke. Ein MDI-Form hat 0/0 aber links unterhalb der Menüleiste. Die Höhe der Menüleiste ist also je nach Auflösung als additive Konstante zur Höhe zu berücksichtigen! (Bei 800x600 etwa 600 twips!)

Wenn die Maus dann den Button verlassen hat, wird der Überprüfungstimer gestoppt und der Hilfetext wieder abgeschaltet!

Die oben erläuterten Cursorabfragemethoden sind auch bei der Programmierung von Bildschirmschonern oder Grafikprogrammen sehr

nützlich, da die Umrechnung Twips/Pixel unter Windows sehr häufig benötigt wird. Sie ist erstaunlicherweise auf 1 Pixel genau und funktioniert problemlos!

Zum Abschluß möchte ich allen noch-nicht VB-Freunden für die das oben Geschilderte vielleicht ein bißchen kompliziert klingen mag, sagen, daß diese Sprache es wirklich Wert ist, beachtet zu werden. Der negative Beigeschmack, den das Wort BASIC immer noch besitzt, ist puncto Vielfältigkeit und Programmierstil bei Visual Basic sicher fehl am Platz! Auch wenn man für komplexe oder zeitkritische Anwendungen auf Visual C++ zurückgreifen muß, ist VB, was Handhabung und Anwenderfreundlichkeit betrifft, besonders interessant!

Für Anfragen zum beschriebenen Problem stehe ich gerne zur Verfügung, viel Spaß beim Programmieren wünscht Sven Schweiger.

□

```
'Deklarationen im BAS-Modul:
Global Const RESX = 645 '708 für 1024, 645 für 800
Global Const RESY = 75  '60 für 768, 75 für 600
Global MLeft As Integer
Global Oben, Links, Unten, Rechts As Integer
Global Message As String
Declare Sub GetCursorPos Lib "User" (XY&)

‘Hilfe-Prozedur im BAS-Modul
Sub HelpAktiv (Message As String, ByVal Oben, ByVal Links,
ByVal Höhe, ByVal Breite)
  Unten = Oben + Höhe
  Rechts = Links + Breite
  If (DrJudoMDI.HelpPanel.Caption <> Message) Or
(DrJudoMDI.HelpPanel.Visible = False) Then
       DrJudoMDI.VerzTimer.Enabled = True 'Verzögerung!
  End If
  DrJudoMDI.HelpTimer.Enabled = True
End Sub

'Die folgende Sub gehört in das Mousemoveereignis des
betreffenden Buttons
‘Setzen der Variablen je nach den ButtonParametern!
 'NN ist der Buttonname, MM der Name des MDI Fensters
 Oben = Val(NN.Top) + RESX + Val(MM.Top)
 Links = Val(NN.Left) + RESY + Val(MM.Left)
 Höhe = Val(NN.Height)
 Breite = Val(NN.Width)
 Message = "Ich bin die Hilfe"
 Call HelpAktiv(Message, Oben, Links, Höhe, Breite)

'Die folgende Sub muß in einen Timer (Name HelpTimer) gesetzt
werden
'HelpPanel ist ein 3D-Panel zum Anzeigen des Hilfetextes
GetCursorPos XY&
XX = CInt(XY& And &HFFF)
YY = CInt(XY& / &H10000)
ScaleX = Screen.TwipsPerPixelX
ScaleY = Screen.TwipsPerPixelY
XX = (XX * ScaleX)
YY = (YY * ScaleY)
If Not ((XX > Links) And (XX < Rechts) And (YY > Oben) And (YY
< Unten) And (Message = HelpPanel.Caption)) Then
     HelpTimer.Enabled = False
     HelpPanel.Visible = False
End If

'Die folgende Sub gehört in einen Timer (Name VerzTimer)
'Er bestimmt, wie lange vor der Meldung gewartet werden soll
 GetCursorPos XY&
 XX = CInt(XY& And &HFFF)
 YY = CInt(XY& / &H10000)
  ScaleX = Screen.TwipsPerPixelX   'in Twips umrechnen
 ScaleY = Screen.TwipsPerPixelY
 XX = (XX * ScaleX)
 YY = (YY * ScaleY)
 If (XX > Links) And (XX < Rechts) And (YY > Oben) And (YY <
Unten) Then
    HelpPanel.Caption = Message
    HelpPanel.Top = 280
    HelpPanel.Left = Rechts - Val(DrJudoMDI.Left) + 50
    HelpPanel.Visible = True
 End If
 VerzTimer.Enabled = False
```