VISUAL-BASIC 3.0 für Windows

Wolfgang Nigischer DSK-405,406,407,(399)

Vorstellung von Visual Basic 3.0

Die Arbeit mit Windows ist einfach - die Erstellung von Windows Programmen ist eher schwierig. Die Entwickler mußten nicht nur die Sprache C beherrschen und anwenden, sondern auch das teure SDK von Microsoft erwerben und sich durch eine gewaltige Menge an Dokumentation durcharbeiten. Der Trend zu einfacheren Programmiersystemen kommt nicht von ungefähr, da die Komplexität moderner APIs zunimmt und zudem ständig Neuerungen stattfinden (OLE 1.0 → OLE 2.0; Windows 3.0 auf 3.1; zusätzliche API-Funktionen...).

Mit Visual-Basic hat sich diese Erkenntnis grundlegend geändert. Die Windows-Programmierung hat ihren Schrecken verloren. Man findet ein Entwicklungssystem vor, das eine schnelle und leichte Entwicklung von Windows-Programmen ermöglicht. Es wird dabei die grafische Oberfläche von Windows mit der Programmiersprache von Basic verbunden. *Basic ist die ideale Sprache für den Einsteiger, die aber auch professionelle Lösungen unter Windows liefert*. Allerdings nicht mit dem GWBASIC, das vielleicht viele noch kennen. Mit Visual-Basic stehen alle Eigenschaften von Windows 3.1, wie Multimedia, Netzwerkfähigkeit, Drag&Drop und OLE 2.0(!) zur Verfügung.

Wenn man von VB spricht, sollte man zwei Dinge unterscheiden:

1.) VB, die Sprache.

Visual-Basic ist ein moderner BASIC-Dialekt und ist wie C oder Pascal eine strukturierte Programmiersprache. Der Grundwortschatz von Basic wurde zu diesem Zweck um einige Kontrollstrukturen erweitert (`Do Loop, While-Wend,....`). Hinsichtlich der Fuktionen wurde VB speziell um solche erweitert, die ein Zusammenarbeiten mit Windows erst ermöglichen. Auf Grund der Windows-Fähigkeit von VB mußte auch eine gravierende Änderung der Programmabarbeitung durchgeführt werden. Bei Standard-Basic wird das Programm Zeile für Zeile von Anfang bis zum Ende abgearbeitet. Generell gilt für die Windows-Programmierung, daß die Programme ereignisgesteuert ausgeführt werden (dazu mehr in den folgenden Seiten). Irgendwie wird auch eine gewisse Ordnung des eigenen Codes erzwungen (durch die verschiedenen Ebenen des Codefensters) (siehe auch **Bild 3** und **zweites Beispiel**). Während man in herkömmlichen Programmiersprachen jede Aktion des Anwenders selbst abfangen mußte, wird dem Programmierer mit VB diese Arbeit komplett abgenommen.

2.) Visual-Basic, die Entwicklungsumgebung.

VB ist ein modernes Entwicklungssystem, welches fast keine Wünsche offen läßt. So wird z.B. ein Werkzeugkasten (Toolbox) zur Erstellung von Button´s, Textfeldern,..... zur Verfügung gestellt, wobei für diese Elemente anschließend der Code automatisch generiert wird. Durch einfaches Anklicken der Elemente ist es möglich, den für dieses Element notwendigen Code (Programmsequenz) zu erstellen, wobei auch hier wiederum die Grundstrukturen des Unterprogramms `Sub` vordefiniert sind.

Mit Visual Basic erhalten Sie ein vollständiges Entwicklungssystem, bei dem Programmiersprache und Oberflächengenerator eine Einheit bilden und man zur Umsetzung eines VB Programms keine weiteren Hilfsmittel benötigt.

Die Grenzen von Visual-Basic

Ja, leider, es gibt sie. Denn die einfache Bedienung des Visual-Basic geht gleichzeitig auf Kosten der Geschwindigkeit. Eines sollte von vornherein geklärt sein: Aufwendige Programme, die mit sehr vielen Formularen[1] gleichzeitig arbeiten und auch die grafische Ausgabe des Visual-Basic recht intensiv nutzen, sollten extrem geschwindigkeitsoptimiert sein, am besten sollten sie direkt auf die Windows Systemebene zurückgreifen und die C-Routinen der sogenannten `DLL` (Dynamic Link Library) nutzen.

Auch ist es mit VB alleine nicht möglich hardwarenahe zu programmieren. Hierzu sind zusätzliche Produkte notwendig (siehe: Erweiterung von VB weiter unten).

Doch eines ist ganz klar: So schnell und ohne Probleme wie in Visual-Basic lassen sich sicherlich mit keiner anderen Windows-Programmiersprache Applikationen entwickeln. Trotzdem stehen alle Windows-Funktionen zur Verfügung und im Vergleich zu C kommt man schneller zu einem Ergebnis.[2]

Und wenn Sie nicht gerade beabsichtigen, ein Konkurrenzprodukt zu "CorelDraw" oder zur Textverarbeitung "Word für Windows" zu entwickeln.... -ja, dann liegen Sie mit dem VB richtig. Mit Visual-Basic steht nun die jüngste Entwicklungsstufe von Basic zur Verfügung. Diese Programmiersprache ermöglicht es auf sehr einfache und vor allem bequeme Weise, Programme zu erstellen, die alle Möglichkeiten nutzen, welche Windows dem Anwender zur Verfügung stellt.

Erweiterbarkeit von Visual-Basic

Will man die zuvor erwähnten Beschränkungen umgehen, besteht die Möglichkeit VB zu erweitern (siehe **Bild 1**), zusätzliche Elemente (Custom Controls[3]) meistens in Form von `*.VBX`,[4] oder `*.DLL`-Dateien wie beispielsweise: `Cmdialog`[5], `Gauge-Control` (analoge Meßwertanzeige, wie Tacho oder Thermometer), `Grid-Control` (Erstellt eine Tabelle, die mit Werten gefüllt werden kann), oder `MSComm` (Regelt den Austausch von Daten über die serielle Schnittstelle), `Graph-Control` (bietet die Möglichkeit der Darstellung von numerischen Daten, wie Meßwerten, oder Geschäftsgrafiken), `3D Button Animated Button` (erlaubt Buttons mit Bitmaps, Icons oder Bilddateien im Metafile - Format zu belegen und diese zu animieren), um nur einige zu nennen.

Mit vielen dieser Zusatz-Controls erhält man auch gleichzeitig die Erlaubnis, diese in compilierter Form, zusammen mit dem eigenen Anwendungs-Programm, lizenzfrei weitergeben zu dürfen.

Mit anderen Worten: Wenn etwas mit VB alleine nicht durchführbar ist, kann man sich unter den mittlerweile sehr zahlreichen Dritt-Anbietern nach einer geeigneten Erweiterung umsehen.

6

[1] Die Anzahl der Formen ist bei der aktuellen Version 3.0 auf 230 limitiert, max 80 davon dürfen geladen sein (bei Verwendung von Windows 3.1)

[2] Ich kenne Firmen, die oftmals Prototypen in VB anfertigen, damit sich der Kunde von dem, was ihm bevorsteht, schneller ein Bild machen kann. Diese "Hüllen" aus VB werden anschließend perfektioniert; sowohl in Hinsicht auf den Basic-Code, als auch durch eine Verknüpfung mit in C geschriebenen Programmen. Die Schnittstelle zum Anwender ist somit von Anfang an in VB!

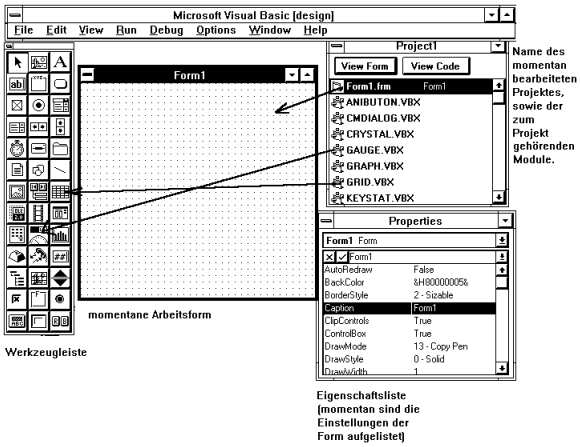
[3] Diese werden mit Hilfe des CDK erstellt.

[4] Diese Erweiterungen werden in der Toolbox sichtbar (siehe Bild 1). Die hier Aufgezählten werden in der professional Edition mit geliefert.

[5] Common dialog. Eine kurze Erklärung finden Sie unter: Die neuen Features von VB am Ende des Artikels. Ab VB 3.0 auch in der Standardversion enthalten..

[6] Um die Verwirrung der Erweiterbarkeit zu steigern:
Es ist mittlerweile möglich durch ein entsprechendes Zusatzprogramm den Windows - Nachrichtenstrom eines beliebigen VB Forms oder Controls abzufangen. Diese als Subclassing bekannte Technik ermöglicht einer Anwendung die Ermittlung von Ereignissen, die von der verwendeten Sprache normalerweise nicht unterstützt werden. Und das ohne daß der Anwendungs - Entwickler auf C oder sonstige DLL - Programmierung zurückgreifen muß.
Auch läßt sich bereits auf solche Windows - API Funktionen zugreifen, die unter VB normalerweise nicht verwendet werden können. (SpyWorks für VB, von Desaware).

Beim Start von Visual-Basic erscheint folgendes Bild:[7]



Wahlweise mit Symbolleiste:
Hierbei werden die am häufigsten benötigten Befehle des Menüs grafisch dargestellt. Rechts werden zusätzlich noch die X/Y Koordinaten des jeweils selektierten Controls in Bezug zur linken oberen Ecke der "Arbeitsform" angegeben.

Bild 1:
Startbildschirm von Visual-Basic mit Symbolleiste

Programmierung[8]

Schritt 1

Erstellen einer Form (also des Windows-Fensters, das der Anwender vom Programm zu sehen bekommt), zeichnen der benötigten Schaltflächen, Rahmen, Listboxen und dgl.[9]. Das heißt, schon beim Programmieren hat man unmittelbar mit jenen Elementen zu tun, die der Anwender später auf dem Bildschirm vorfindet. Es erinnert fast an "malen nach Zahlen" (Programmieren wie gemalt). Man wählt das entsprechende Objekt mit der Maus aus der Toolbox (**Bild 2 links**) aus. Anschließend erscheint über der "Arbeitsform" ein Fadenkreuz, ähnlich wie in Paintbrush, mit dem dann das jeweilige Objekt an der gewünschten Stelle in der Form gezeichnet werden kann. Man kann bereits jetzt in der Entwicklungszeit wichtige Voreinstellungen der Eigenschaften im "Properties"-Fenster[10] vornehmen[11].

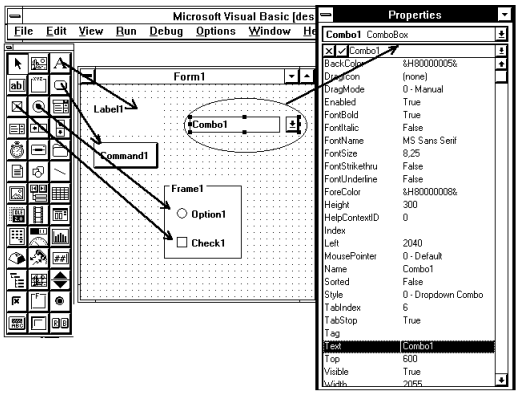


Bild 2: Erstellen einer "Arbeits" - Form

Der nächste Schritt:

Schreiben des Programmcodes[12]. Im **Bild 3** wurde durch Doppel-Klicken der Command-Schaltfläche das Codefenster für dieses Objekt geöffnet. Oben links erscheinen die momentan bereits gezeichneten Objekte, rechts davon die möglichen Ereignisse. So steht `Click` beispielsweise für ein Ereignis, mit dem der "End-User" mit der linken Maustaste auf dieses Objekt klickt. Als Reaktion auf ein solches Ereignis ruft das betroffene Objekt jene Prozedur auf, die der Programmierer für dieses Ereignis geschrieben und vorgesehen hat. Die bisher bekannte lineare Programmierung weicht der ereignisorientierten[13].

Visual-Basic = Quick-Basic + Ereignisprozeduren.

Diese Ereignisse werden in Ereignisprozeduren verarbeitet. Die Ereignisprozeduren sind `SUB/END SUB`-Prozeduren die von Visual-Basic aufgerufen werden [14].

Nach deren Abarbeitung wird der Prozeß wieder an Windows zurückgegeben und so eine gewisse "Multi tasking Fähigkeit" erreicht.

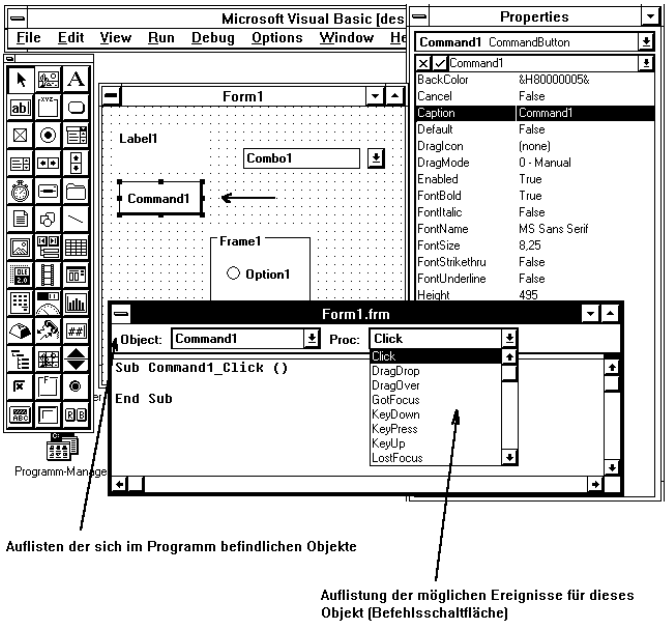


Bild 3: Kodefenster für Command1-Schaltfläche

Genug der grauen Theorie:

Erstes Beispiel

Da das "Hallo World" meines Erachtens ziemlich abgedroschen ist, habe ich mir eine andere Aufgabenstellung ausgedacht: Es soll ein Programm erstellt werden, das die Fläche eines Rechtecks berechnet, und das Ergebnis in einer formatierten Ausgabe ausgibt. Damit das ganze ohne VB-Umgebung läuft, wird anschließend eine EXE-Datei erstellt.

---

[7] Visual Basic Professional Version 3.0

[8] Hierbei gehe ich von einem kleinen Beispiel-Projekt aus. Bei größeren Projekten ist es ratsam, vorher einige Überlegungen im Bezug auf Aussehen, Benutzerfreundlichkeit u.s.w. anzustellen. Entsprechende "Richtlinien" und Vorschläge gibt es ja mittlerweile zu genüge...

[9] Visual Development

[10] In der deutschen Standard-Version ist das das "Eigenschafts-Fenster"

[11] Zu den Eigenschaften eines Objekts gehören beispielsweise sein Name, Inhalt, Größe und Farbe,... aber auch die ab der Vers. 2.0 eingeführte `HelpContexID` Nummer, die beim Hilfeaufruf des eigenen Programms die richtige Seite des Help-Files aufruft. (**Bild 2**)

[12] Sinnvollerweise sollte man vorher den gezeichneten Objekten aussagekräftigere Namen geben. Die Bezeichnung (Name) Eigenschaft (im Properties - Fenster angegeben) eines Objektes muß mit einem Buchstaben beginnen und darf max. 40 Zeichen enthalten, einschließlich Zahlen und Unterstreichungen.

[13] Das sind unter anderem: Bewegen der Maus, Klicken mit der Maus, eine Taste drücken, oder Systemereignisse, wie beispielsweise DEE-Nachrichten oder Nachrichten des Timers. Es gibt keine aktive Programmzeile, so lange kein Ereignis auftritt.

[14] Eine klassische Basic-Unterroutine, die durch `GoSub` aufgerufen und durch `Return` beendet wird, gibt es in Visual Basic zwar, ist jedoch völlig überflüssig. Statt dessen ist es viel übersichtlicher, einfacher und flexibler die sog. "Subs" des VB zu verwenden. Eine `Sub` ist nichts anderes als ein BASIC-Unterprogramm, wird im Programmtext jedoch gesondert plaziert und läßt sich wie jeder normaler BASIC-Befehl aufrufen. Wobei VB ein weiteres Codefenster öffnet und Ihnen das Gerüst der Unterroutine (oder Funktion) vorgibt (**Bild 3**)

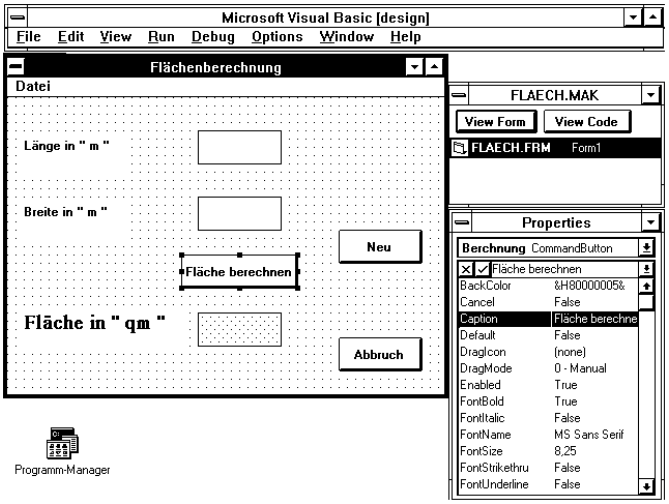


Bild 4: Erstellung der Form mit den Objekten

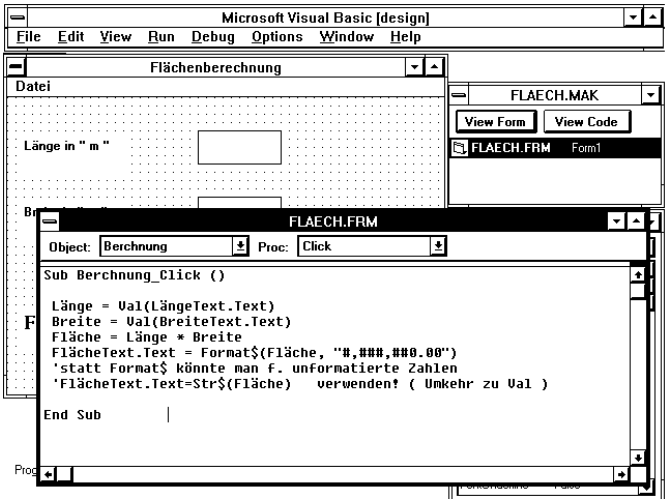


Bild 5: Den Code in der Schaltfläche **`Flächenberechnung`** unterbringen

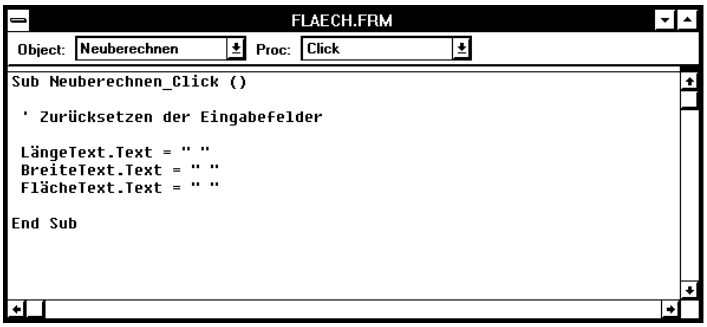


Bild 6: Die Schaltfläche `Neu` mit Leben erfüllen

Dasselbe gilt für die `Abbruch` Schaltfläche:

```
Sub Befehl2_Click ()
    End
   ' Programm-Ende
End Sub
```

Nach dem Austesten des Programms[15] geht es an die Erstellung eines EXE - Files:

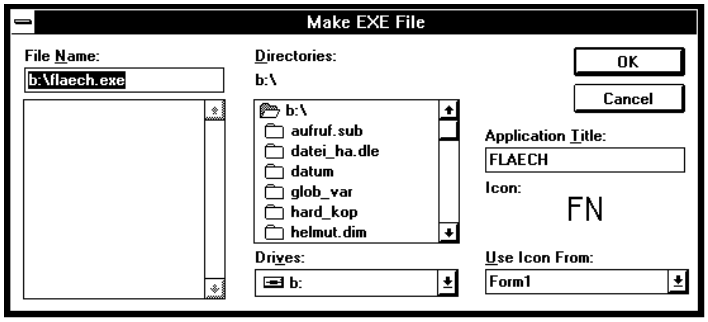


Bild 7: Erstellen eines EXE - Files

Ein Bild sagt mehr als tausend Worte; für diejenigen, die VB nicht kennen, eine kurze Erklärung zu `Icon` und zum `Application Title`, der Rest des obigen Bildes ist glaube ich klar: Unter `Applications-Title` wird die Überschrift des Programms eingetragen (das, was im Windows Programm-Manager unter dem Icon steht). Das gewünschte Icon kreiert man zuerst selbst[16] (ein Icon-Editor ist im Lieferumfang von VB enthalten - mit Source-Code!) und weist dann anschließend der Startform in der Eigenschaftsliste das gewünschte Icon zu. Zu guter letzt erhalten Sie eine EXE-Datei. Um das Programm weitergeben zu können, ist es noch notwendig, eine sog. Laufzeitbibliothek mit auf die Diskette zu kopieren (`VBRUN300.DLL`[17]). Es sind also mindestens zwei Dateien notwendig um ein mit VB erstelltes Programm zum laufen zu bringen. Nämlich die eigentliche EXE-Datei und diese Laufzeitbibliothek.

Visual Basic ist also kein reiner Compiler. Trotzdem findet bei dem Erstellen eines *.EXE-Files eine Aufbereitung der Programmzeilen statt, somit kann man aber auch eigentlich nicht mehr von einen Interpreter (wie GWBASIC) sprechen.

Wo werden diese Dateien bei der Installation hin kopiert? Nun ich würde sagen die eigene Programmdatei gehört in ein eigenes Verzeichnis. Die `*.DLL` und `*.VBX`-Dateien gehören in das Windows\System-Verzeichnis. Diese Dateien kommen für alle mit VB erstellten Programme zum Einsatz, das heißt, sie befinden sich nur einmal im Speicher, auch wenn mehrere mit VB erstellten `EXE`-Files aktiv sind[18]. Das bringt zwar eine Platzersparnis auf der Festplatte, birgt aber die Gefahr in sich, daß es bei Verwendung von mehreren VB-Programmen, die mit verschiedenen Versionen erstellt wurden, zu Problemen kommen kann. Ein entsprechendes Installationsprogramm ist ab VB 3.0 in der Standardversion und vorher in der Professional-Version enthalten[19]. Dieses Installationsprogramm eruiert unter anderem das richtige Windows-Verzeichnis, in das diese `VBX`- und `DLL`-Dateien hingehören, kreiert eine eigene Programm-Gruppe im Programm-Manager und ein eigenes Verzeichnis für Ihre Programm- bzw. Datendateien.

---

[15] Es kann bereits während der Entwicklungszeit das Programm getestet werden, ohne, das man eine EXE - Datei erstellt.

[16] oder sucht sich eines von den vielen mit gelieferten Icons aus.

[17] Oder je nach Version mit dem das Programm compilliert wurde: `VBRUN100.DLL, VBRUN200.DLL`. Werden third-Party-Objekte in das eigene Programm mit eingebunden, so sind diese ebenfalls mitzuliefern (meistens in Form von `DLL`- oder `VBX`-Dateien, z.B.: `GRID.VBX`, wenn eine Tabelle programmiert wurde.

[18] Bei fast allen Zusatzcontrolls sind auch im VB Buch die entsprechenden Benützungshinweise für Visual C++ angeführt. Das heißt, das alle diese Controls ohne Änderung von beiden Programmiersprachen verwendet werden können!

[19] In der Proffesional - Version sind auch zusätzlich Informations - Files über die Programmierung (unter anderem API's, so z.B. eine komplette VB API Referenz und API Deklarationen in Form von Help - Files) und zur Programmgestalltung enthalten.
Weiters: Programm - Beispiele, sowie der Windows Help - Compiler Version 3.1 enthalten.
Allerdings ist die Prifessional - Edition in englisch. Nur die Standart - Ausgabe ist in Deutsch erhältlich.
Bitte besorgen Sie sich einen Prospekt von Microsoft über VB, da stehen mehr Details drinnen, ich habe den Prospekt nur von einer alten Version

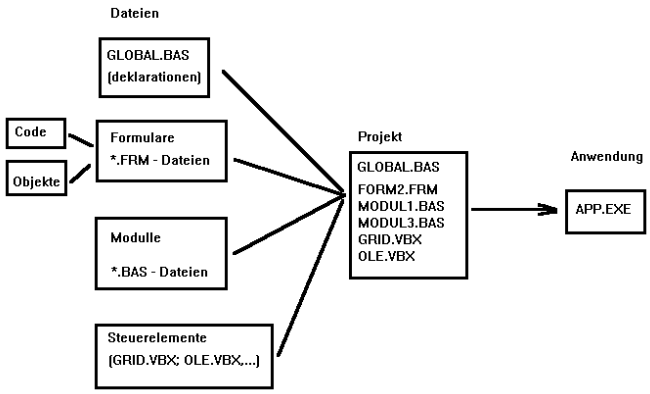


Bild 8: Dateien eines Projekts

Zum **Bild 8**: Eine Form besteht also aus verschiedenen Objekten, die mit entsprechenden Code versehen sind. Zusätzlich kann es noch eine Datei mit dem Namen `GLOBAL.BAS`, die generelle Deklarationen, sowie Makros enthält, und durch diese Definitionen auch die Lesbarkeit der Programme erhöhen, geben. Der Inhalt dieser Datei setzt sich häufig aus einem Teil des Inhaltes der `KONSTANT.TXT` und `WINAPI.TXT` zusammen.

Beispiele aus `Constant.TXT`

```
Global Const KEY_LEFT = &H25
Global Const KEY_UP = &H26
Global Const KEY_RIGHT = &H27
Global Const KEY_DOWN = &H28
Global Const KEY_SELECT = &H29
Global Const KEY_PRINT = &H2A
Global Const KEY_EXECUTE = &H2B
Global Const KEY_SNAPSHOT = &H2C
Global Const KEY_INSERT = &H2D
Global Const KEY_DELETE = &H2E
Global Const KEY_HELP = &H2F
```

Beispiele aus `WINAPI.TXT`

```
'  Hilfe-Bereich.
'  Befehle für WinHelp()
Global Const HELP_CONTEXT = &H1    'Zeige Thema in ulTopic an
Global Const HELP_QUIT = &H2       'Beende Hilfe
Global Const HELP_INDEX = &H3      'Zeige Index an
Global Const HELP_HELPONHELP = &H4 'Hilfetexte für Hilfe
anzeigen
Global Const HELP_SETINDEX = &H5   'Setze den aktuellen Index
                                   'für Mehrfachindex-Hilfen
Global Const HELP_KEY = &H101      'Zeige Thema für
                                   'Schlüsselwort in offabData
Global Const HELP_MULTIKEY = &H201
Declare Function WinHelp Lib "User" (ByVal hWnd As Integer,
ByVal lpHelpFile As String,
ByVal wCommand As Integer, dwData As Any) As Integer
Type MULTIKEYHELP
   mkSize As Integer
   mkKeylist As String * 1
   szKeyphrase As String * 253     'Array-Länge ist beliebig,
                                   'kann geändert werden
End Type
```

`Module` sind reine Codeteile eines Programms, ohne dazugehörender Form. Das könnten beispielsweise immer wiederkehrende Eingabeüberprüfungs - Funktionen sein, die man sich der Übersicht halber in so einer `BAS`-Datei abspeichert. Man kann sich also mit dieser Methode eine eigene Bibliothek häufig benötigter Codes zurecht legen und diese dann von einer Form aus aufrufen.

Visual Basic unterstützt also einen modularen Programmaufbau.

Modulare Programme bieten eine Reihe von Vorteilen: Sie sind leicht zu verstehen, leicht zu erweitern, und vor allem, die Fehlersuche wird erheblich erleichtert, da sich die Suche nach einem Fehler auf ein einzelnes Modul beschränken kann.

Zweites Beispiel

Hierbei möchte ich hauptsächlich zeigen, wie der Inhalt einer Variablen weiter gereicht wird, da Variablen je nach Deklaration verschiedene Geltungsbereiche haben.

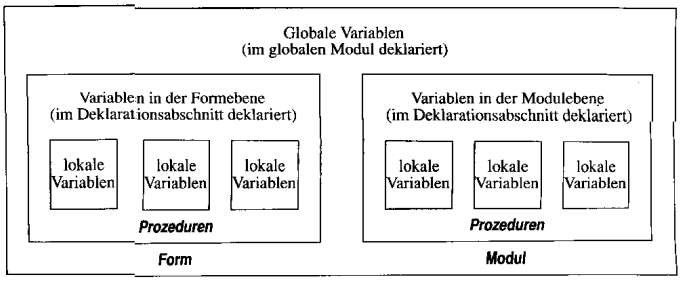


Bild 9: Geltungsbereich der Variablen

Wiederum wird die Fläche eines Rechtecks berechnet. Diesmal wird aber das Ergebnis in einer anderen Form dargestellt. Die Berechnung wird über die Befehlsschaltfläche der ersten Form aufgerufen. Man erkennt im **Bild 10** deutlich in der fokussierten Form die Dimensionierung der beiden Variablen, sowie ein eigenes Unterprogramm, das der Deklaration-Ebene untergeordnet ist.

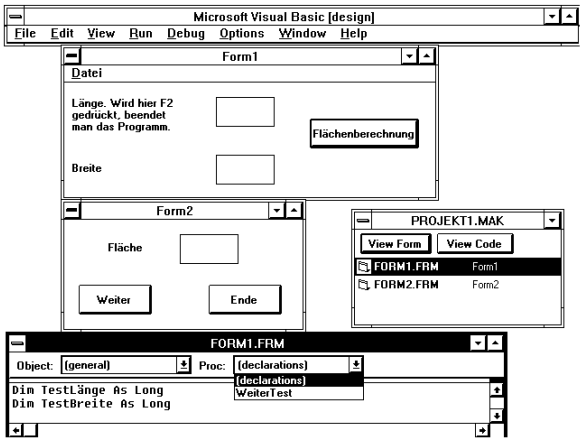


Bild 10: Alternative Form für eine Rechteck - Flächenberechnung

Als weitere Facette kommt in diesem Beispiel auch noch eine Funktion `KeyDown` vor, die bei Betätigen der F2-Taste das Programm vorzeitig beendet und ein kleines Menü dazu (siehe **Bild 11**).

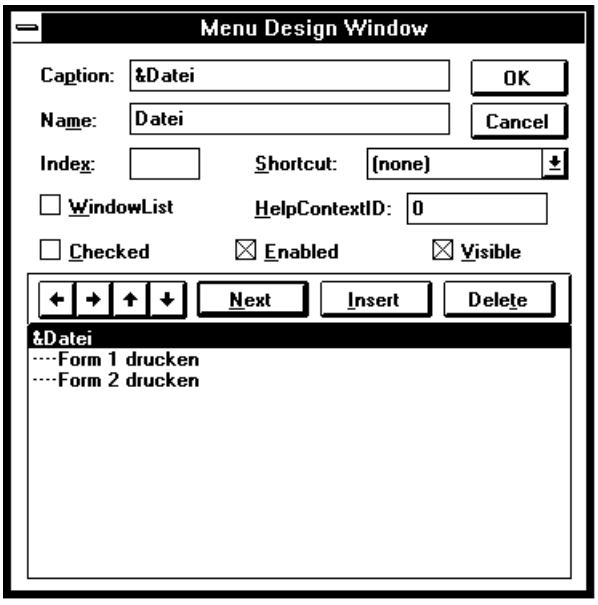


Bild 11: Menüentwurfsfenster

Code der ersten Form[20]

```
VERSION 2.00
Begin Form Form1
   Caption         =   "Form1"
   ClientHeight    =   1905
   ClientLeft      =   1035
   ClientTop       =   1305
   ClientWidth     =   6165
   Height          =   2595
   Left            =   975
   LinkMode        =   1  'Source
   LinkTopic       =   "Form1"
   ScaleHeight     =   1905
   ScaleWidth      =   6165
   Top             =   675
   Width           =   6285
   Begin TextBox Breite
      Height          =   495
      Left            =   2520
      TabIndex        =   3
      Top             =   1200
      Width           =   975
   End
   Begin CommandButton Berechnung
      Caption         =   "Flächenberechnung"
      Default         =   -1  'True
      Height          =   495
      Left            =   4080
      TabIndex        =   4
      Top             =   600
      Width           =   1815
   End
   Begin TextBox Länge
      Height          =   495
      Left            =   2520
      TabIndex        =   2
      Top             =   240
      Width           =   975
   End
   Begin Label Bezeichnung2
      Caption         =   "Breite"
      Height          =   255
      Left            =   120
      TabIndex        =   1
      Top             =   1320
      Width           =   735
   End
   Begin Label Bezeichnung1
      Caption         =   "Länge. Wird hier F2 gedrückt, beendet man das
Programm."
      Height          =   615
      Left            =   120
      TabIndex        =   0
      Top             =   240
      Width           =   1815
   End
   Begin Menu Datei                                          '
Dateimenü
      Caption         =   "&Datei"
      Begin Menu PrintForm1
         Caption         =   "Form 1 drucken"
      End
      Begin Menu PrintForm2
         Caption         =   "Form 2 drucken"
      End
   End
End
Option Explicit                 'Deklarationsebene der Form 1 !!!!
Dim TestLänge As Long
Dim TestBreite As Long
Dim Fläche As Long
Sub Berechnung_Click ()
    TestLänge = Val(Länge.Text)
    TestBreite = Val(Breite.Text)
      Form2.Show
        WeiterTest      ' Weiter zur Unterroutiene

End Sub
Sub Länge_KeyDown (Tastencode As Integer, Umschalten As Integer)
    Const FTASTE_F2 = &H71
     If Tastencode = FTASTE_F2 Then End
End Sub
Sub PrintForm1_Click ()
    Form1.PrintForm
End Sub
Sub WeiterTest ()
    Fläche = TestLänge * TestBreite

     Form2.FlächenAusgabe.Text = Str$(Fläche)
End Sub
```

Code der zweiten Form

```
VERSION 2.00
Begin Form Form2
   Caption         =   "Form2"
   ClientHeight    =   1815
   ClientLeft      =   1035
   ClientTop       =   3585
   ClientWidth     =   3960
   Height          =   2220
   Left            =   975
   LinkMode        =   1  'Source
   LinkTopic       =   "Form2"
   ScaleHeight     =   1815
   ScaleWidth      =   3960
   Top             =   3240
   Width           =   4080
   Begin CommandButton Ende
      Caption         =   "Ende"
      Height          =   495
      Left            =   2400
      TabIndex        =   2
      Top             =   1080
      Width           =   1215
   End
   Begin CommandButton Weiter
      Caption         =   "Weiter"
      Height          =   495
      Left            =   240
      TabIndex        =   3
      Top             =   1080
      Width           =   1215
   End
   Begin TextBox FlächenAusgabe
      Height          =   495
      Left            =   1920
      TabIndex        =   1
      Top             =   240
      Width           =   975
   End
   Begin Label Bezeichnung1
      Caption         =   "Fläche"
      Height          =   255
      Left            =   720
      TabIndex        =   0
      Top             =   360
      Width           =   735
   End
End
Sub Ende_Click ()
    Unload Form1
    End
End Sub
Sub Weiter_Click ()
  Form1.Show
  Form1.Länge.Text = ""
  Form1.Breite.Text = ""
  Form1.Länge.SetFocus  ' setzt den Corsor inner auf Länge!
  Form2.Hide
End Sub
```

Es ist auch möglich, gleichzeitig zwei verschiedene Prozeduren zu betrachten:[21]

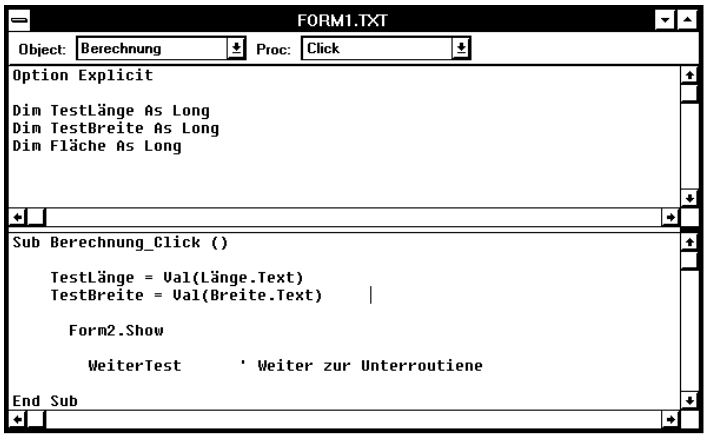


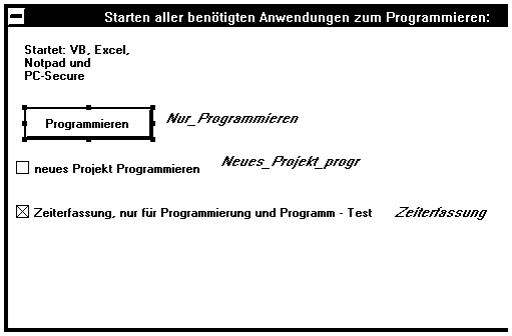
Bild 12: Gleichzeitige Darstellung verschiedener Codesequenzen

[20] Mit der kleineren Schrift am Anfang möchte ich nur den grafischen Teil gegenüber dem eigentlichen Code differenzieren. Die hier angegebenen, hauptsächlich grafischen Angaben erledigen sich zum Teil mit Schritt 1 der Programmierung, nämlich dem Zeichnen der Steuerelemente und dem "Ausfüllen" des Eigenschaftsfensters.

[21] Ähnlich, wie man z.B. in Win Word der Text teilen kann, ist das auch in VB möglich.

Drittes Beispiel

Abschließend möchte ich noch kurz zeigen, wie in VB ein anderes Programm aufgerufen wird und wie `CheckBoxen` als Entscheidungsträger verwendet werden.



Mit *Kursiv* - Schrift habe ich die im Projekt verwendeten Namen angegeben.

Bild 13: Checkboxen im Programm

Der Code (auszugsweise) dazu:

```
Sub Nur_Programmieren_Click ()
    Startprogramm = Shell("C:\EXCEL\EXCEL.EXE", 2)
Debug.Print "Startprogramm: "; Startprogramm
' Angabe des Programmes und der Startoption (2 ist auf Icon verkleinert,
1 ist Vollbilddarstellung).
    Startprogramm = Shell("C:\WINDOWS\NOTEPAD.EXE", 2)
Debug.Print "Startprogramm: "; Startprogramm
        If Neues_Projekt_progr.Value = True Then
            Startprogramm = Shell("C:\WINDOWS\WINFILE.EXE", 2)
        End If
        If Zeiterfassung.Value Then
        Startprogramm = Shell("C:\XTIMELOG\XTIMELOG.EXE", 1)
Debug.Print "Startprogramm: "; Startprogramm
        End If

    Startprogramm = Shell("C:\VB\VB.EXE", 1)
    Startprogramm = Shell("C:\PCTOOLS\PCSECURE.EXE")
' DOS-Programm aus PC-Tools 8.0
   End
End Sub
```

Nobody is perfekt...

Nicht unerwähnt lassen möchte ich die umfangreichen Debug-Möglichkeiten, die VB anbietet. Bereits während der Codeeingabe wird nach Drücken der Return-Taste die zuletzt eingegebene Codezeile überprüft. Während der Testphase können über ein Direktfenster (`watch window`) sämtliche Variablen sichtbar gemacht werden (`Debug.Print`). Zusätzlich kann im Direktfenster durch die Eingabe `Print "Variablenname";` der momentane Wert der gewünschten Variablen ermittelt werden. Wem das "`Debug.Print`"-Verfahren während der Entwicklungsarbeit zuviel Schreibarbeit bereitet, kann ab Version 2.0 bestimmte Variablen gezielt überwachen lassen (`Watch Expressions`) und zwar innerhalb der Prozedur oder im ganzen Programm. Auch werden auf Wunsch alle zu einem bestimmten Zeitpunkt (Möglichkeit des Haltepunkt setzen) aktiven Prozeduren aufgezeigt. (`Calls Dialog`)

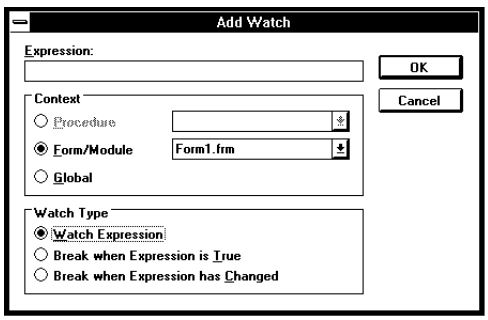


Bild 15: Beobachtung von Variablen

Um sich die Sache mit dem Direktfenster besser vorstellen zu können, hier wieder 2 Screen-Shoots aus meinem zweiten Beispiel - Programm: Das obere Fenster ist das Debug-Fenster, das sich wiederum in 2 Teile aufgliedert: Im oberen Teil wurden laut vorhergehenden Bild zwei Beobachtungswerte definiert. Der untere Teil kommt durch das `Debug.Print` (im Source-Code des darunter liegenden Fensters) zustande.

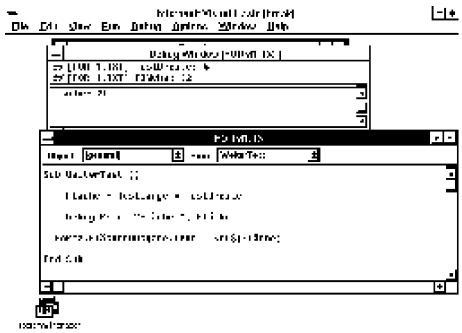


Bild 16: Debuggen in Visual-Basic

Wie vorher bereits erwähnt ist es dann noch möglich vor Erstellung einer `EXE`-Datei einen Probelauf des zu erstellenden Programms durchzuführen. Nicht zuletzt steht noch eine umfangreiche kontextsensitive On-Line-Hilfe mit Beispielen zur Verfügung. (Über 3 MB !).

Die neuen Features VB 3.0

- Access-Datenbank Zugriff und deren Daten-Editierung, ebenso auf Fox Pro Vers 2.0 und 2.5, dBASE III und IV, Paradox, Btrive, sowie ODBC, wie etwa der SQL-Server. Bei dem ganzen Vorgang bleibt die eigentliche Datenbank-Engine dem Programmierer verborgen. Der bekommt nur ein neues Custom Control (Data Control) zu sehen.
- PopupMenu-Befehl
- Common dialog (für häufig benötigte Dialog-Boxen, wie "Datei öffnen", "Datei speichern",..) [22].
- OLE 2.0 Unterstützung
- Crystal Reports (Reportgenerator zur Bildschirm- oder Drucker- Ausgabe von Datenbanken
- Outline-Control (zum Darstellen hierarchischer Strukturen, wie dem Verzeichnisbaum im Windows Dateimanager); nur Prof. Edition.
- Setup-Witzard, wie bereits im Artikel erwähnt).

Für alle, die jetzt Lust auf's Programmieren bekommen haben, hier die Systemvoraussetzungen, des Rechners, um mit VB arbeiten zu können:

- IBM PC oder Kompatibler mit 80286er Prozessor oder höher[23]
- MS Windows 3.0 oder Höher, MS-DOS 3.1 oder höher.
- 1 MB, besser 2 MB Hauptspeicher
- Festplatte (VB 3.0 Prof. ca. 30 MB!, je nach Anzahl der zusätzlich installierten Beispiele).
- Diskettenlaufwerk
- Maus
- für Pen-Computing natürlich einen Pen-Computer und Windows für PEN-Computing
- für Multimedia-Programmierung ebenfalls entsprechende Erweiterungen
- für ODBC-Programmierung entsprechende Datenbanktreiber[24]; direkte Datenbankzugriffe können ab Vers. 3.0 getätigt werden (MS Access, Fox Pro Vers 2.0 und 2.5, dBASE III und IV, Paradox, Btrive)
- MAPI-Programmierung[25]: MS Mail 3.0 oder höher

Ach übrigens: VB gibt es auch unter DOS. Im Programmpaket, das extra erworben werden muß, sind Umwandlungs-Programme mitgeliefert, die VB Windows-Source-Code in DOS-kompatiblen umwandeln. So werden spezielle Windows-Teile aufgegliedert (wie z.B. das Windows API) □

[22] Das ist jener Standarddialog, den Sie unter den mit Windows mit gelieferten Zubehörprogrammen unter dem Menüpunkt "Datei" aufrufen (sieht bei allen Programmen gleich aus).

[23] Besser ist ein 386er, mit ausreichend RAM (6 MB).

[24] Ist ebenfalls in den VB Sprachumfang integriert.

[25] Kommunikation zwischen Applikationen und Electronic-Mail über beliebige Netzwerke.