Projekt: Hello World

Du kennst sicher auch das berühmte Hello World-Programm, dass ein jeder Programmiersprachenneuling als erstes Programm schreibt (falls nicht: Es wird der Text "Hello World" auf irgendeine Weise ausgegeben), doch ich werde dieses Beispiel etwas in seinem Wesen verändern. Man soll in einem Fenster in ein Textfeld seinen Namen eingeben können, sodass das Programm nicht nur die Welt sondern auch den Benutzer grüßt.

VB-Projekt erstellen

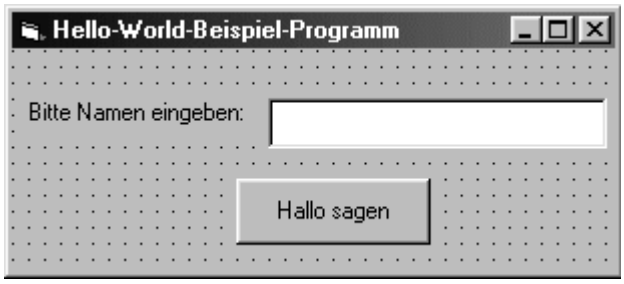
Also, zuerst startest du VB (Visual Basic), ein Fenster erscheint, in dem du auswählen kannst, ob du eine "Standart-EXE" erstellen willst, den "VB-Anwendungsassistenten" ausführen willst oder dir die "VB-Einsteiger-Steuerelemente" anschauen willst (Ich gehe davon aus, dass du die Einsteiger Version von VB 6.0 hast. Bei der Professional und Enterprise Edition steht mehr Punkte zur Auswahl). Die "VB-Einsteiger-Steuerelemente" können wir bei unserem Vorhaben vernachlässigen. Den "VB-Anwendungsassistenten" kannst du verwenden, wenn du größere Projekte wie eine Textverarbeitung erstellen willst, der Assistent erstellt dann Menüs und mehrere vorgefertigte Fenster (ich habe für Vocabulaire den Anwendungsassistenten verwendet). Für unsere Anwendung reicht die "Standart-EXE", wenn du darauf doppelklickst, wird ein Fenster mit einem Formular darin erstellt. In dieses Formular werden wir jetzt einige Steuerelemente einfügen.

Formulardesign

Gehe in die Werkzeugleiste, dort wählst du das Bezeichnungsfeld () durch einen Klick darauf aus, und ziehst bei gedrückter Maustaste auf dem Formular das Feld auf. Die Größe ist erst einmal egal, die kann später noch verändert werden. Dann gehst du in die Eigenschaften, dort suchst du die Eigenschaft "*Caption*", die normalerweise standardmäßig ausgewählt ist, trage in das Feld dahinter "*Bitte Namen eingeben:*" ein. Wenn du jetzt auf dein Formular blickst, hat sich die Beschriftung "*Label1*" des Bezeichnungsfelds in "*Bitte Namen eingeben:*" verwandelt. Wenn du den Text nicht ganz lesen kannst, dann vergrößere das Feld, indem du das blaue Quadrat am Rand des Feldes nach rechts ziehst. Dann platzierst du das Feld noch richtig, in dem du auf das Feld klickst und es bei gedrückter Maustaste an die richtige Stelle verschiebst.

Erstelle dann ein Textfeld () rechts neben dem Bezeichnungsfeld. Gehe bei der Erstellung wie bei dem Bezeichnungsfeld vor. Gehe dann auf die Eigenschaft "*Text*" und lösche den Text in dem Feld dahinter. Wenn du nun auf dein Formular blickst, ist die Beschriftung in dem Textfeld verschwunden. Dimensioniere und platziere das Textfeld noch, wie du es beim Bezeichnungsfeld getan hast. Sollte das Formular nicht groß genug sein, dass das Bezeichnungsfeld und das Textfeld hineinpassen, dann vergrößere es wie bei den Steuerelementen. Wenn das Formular noch markiert ist, kannst auch hier noch die Beschriftung verändern. Verändere dazu die Caption-Eigenschaft des Formulars, in den gewünschten Namen, z.B "Hello-World-Beispiel-Programm". Jetzt fügst du noch eine Befehlsschaltfläche () ein. Dies funktioniert wie bei den anderen Steuerelementen. Gebe dann unter Caption: "*Hallo sagen*" ein. Der Knopf hat jetzt die Beschriftung "Hallo sagen". Verschiebe es bei Bedarf noch an die richtige Stelle.

Jetzt sollte dein Formular so aussehen:



Das Coding

Dein Programm hat jetzt eine Benutzeroberfläche, doch es weiß noch nicht was es machen soll, wenn man auf den Knopf "Hallo sagen" klickt. Das muss man dem Programm jetzt beibringen und zwar mit der Programmiersprache Basic.
Doppelklicke auf den Knopf "Hallo sagen". Es erscheint ein Fenster mit bis jetzt unbekanntem Text. Es handelt sich bei diesem Text um den Anfang und das Ende eine VB-Subroutine. Der Cursor steht schon zwischen den beiden Zeilen und wartet auf deine Eingabe. Jetzt geht's ans programmieren. Ich weise nun einer Variablen näheres den Text des Textfeldes zu, dies geschieht mit: *Var = Text1.Text*. *Var* heißt die Variable, *Text1* ist der Name des Textfeldes, mit dem Punkt hinter *Text1* ruft man die Eigenschaften des Textfeldes auf. Und die Eigenschaft *Text* die bei der Ausführung leer war, ist durch die Eingabe des Namens in das Textfeld gleich dem Name des Benutzers geworden. Hier sehen sie auch ein weiteres Feature von VB 6.0: Intelligente-Sense, d.h. das Wort hinter dem Punkt wird automatisch anhand einer Liste versucht zu vervollständigen.
Nun wollen wir noch eine Meldung ausgeben, die die Begrüßung vornimmt, dies geschieht mit: *MsgBox "Hello World! Hello " & Var*. *MsgBox* ist der Befehl der eine Meldung auslöst, der Text in den Anführungszeichen ist die Beschriftung und mit dem Kaufmannsund (&) wird die Variable *Var* angehängt ausgelesen, sodass hinter "Hello World! Hello " noch der Name aus der Textbox steht. Dieses anhängen nennt sich Zeichenkettenverknüpfung.
Nun ist es an der Zeit, das Programm mal auszuprobieren, drücke dazu auf den Pfeil (▶) in der Symbolleiste. Das Programm startet, gib nun irgendeinen Namen in das Textfeld ein. Wenn du auf den Knopf "Hallo sagen" klickst wird eine Meldung, die so ähnlich wie diese, ausgegeben:

Hier war der Name im Textfeld "*Mustafa Mustermann*". Mit einem Klick auf "*OK*" wird die Meldung geschlossen und du gelangst zurück zum ersten Fenster. Klicke auf das Kreuzchen am rechten oberen Rand des Fensters, um das Fenster wieder zu schließen und zurück zu Visual Basic zu gelangen.

Selbstausführende EXE-Datei erstellen

Das Programm hat bis jetzt einen Hacken, wenn VB nicht aufgerufen ist, kann das

Programm nicht ausgeführt werden. Das werden wir jetzt ändern. Erst einmal speichern wir unser Programm. Gehe dazu ins Menü "*Datei*" und dann auf "*Projekt speichern*". Visual Basic fragt nun wo das Programm gespeichert werden soll. Wähle ein beliebiges Verzeichnis aus und gib einen Namen ein. Diese Abfrage findet für das Formular und für das Projekt statt. Also müssen zwei Namen eingegeben werden. Gehe dann auf "*Datei*" und dann auf den Menüpunkt "*Projekt1.exe erstellen*". Wähle in dem dann erscheinenden Fenster ein Verzeichnis aus, und gib dem Programm einen Namen (Hier bitte nicht .exe am Schluss vergessen). Unter "*Optionen*" kannst du noch einige Sachen in Bezug auf Copyright usw. einstellen. Klicke darauf, nun kannst du die Versionsnummer eingeben, da dies die Version 1 ist lassen wird die Einstellung unverändert. Gib daneben noch den Namen für deine Anwendung ein, wenn wir ein Icon ausgewählt hätten, könnten wir nun auch noch ein Icon wählen. Nach Belieben kannst du noch Einträge bei Copyright, Dateibeschreibung, Firmenname, Kommentare, Marken und Produktname machen. Diese Einträge erscheinen später wenn du mit der rechten Taste auf das Programm klickst und Eigenschaften in dem Kontextmenü auswählst. Wenn du deine Einträge gemacht hast, klicke auf "*OK*" um zum "*Projekt erstellen*"-Fenster zurückzukehren. Drücke auch hier auf "*OK*" und das Programm wird kompiliert (in Computersprache aus Einsen und Nullen übersetzt) und die EXE-Datei wird erstellt. Die EXE-Datei ist nun fertig und kann unabhängig von VB ausgeführt werden.

Wenn es auf jedem System laufen soll

Jedoch nur auf Systemen auf denen VB installiert ist. Da zur Ausführung Systemdateien von VB benötigt werden. Um auch dieses Problem zu lösen müsstest du den "Verpackungs- und Weitergabe"-Assistenten, der dann für dich eine Setup-Routine schreibt die die benötigten Systemdateien beinhaltet. Da dieses Programm zu einfach ist, lasse ich diesen Schritt weg. Doch der Assistent ist selbsterklärend.

API-Programmierung

Die Fähigkeiten und Funktionen von Windows sind manchmal sehr beeindruckend, was liegt also näher, als sich ein paar von diesen Fähigkeiten auszuborgen. Dies funktioniert auch und im Gegensatz von anderen Möglichkeiten ganz legal, denn es gibt eine Schnittstelle die Win32API, die es Programmen erlaubt auf die Fähigkeiten des Betriebssystems zuzugreifen. Der Begriff Win32API steht für Windows32 Application Programmiering Interface (Windows32 steht hier für das 32bit Windows also Windows9x/Me und NT/2000/XP).
Die Fähigkeiten von Windows werden als Funktion in das Visual Basic Projekt eingebunden, die Syntax für eine solche API-Funktionseinbindung lautet allgemein so:
[Public | Private] Declare Function *name* Lib "*libname*" [Alias "*aliasname*"] [As *type*]

Wirkungsbereich der API-Funktion

Mit Public oder Private wird der "Wirkungsbereich" der Funktion festgelegt. Public ist, wie der Name schon sagt für das ganze Projekt zugänglich, Private nur in der Prozedur in der die API-Funktion eingebunden ist. *Declare* ist das Schlüsselwort, dass anzeigt, dass es sich um eine API-Deklaration handelt. *libname* ist der Name der .dll (der genaue Pfad muss nicht angegeben werden, denn Windows sucht automatisch im System-Verzeichnis). Bei den drei Hauptbibliotheken (kernel32, user32 und gdi32) muss auch das .dll nicht angehängt werden. *Alias* ist nur erforderlich, wenn der *name* und der *aliasname* nicht übereinstimmen. Einer der häufigsten Fehler, die

in Verbindung mit der Verwendung der Win32-API gemacht werden, sind dass kein Rückgabewerttyp deklariert wird. Der Standardtyp Variant führt häufig zu einem Fehler oder zu einer Speicherausnahme. Die meisten API-Funktionen haben einen Long-Datentyp als Rückgabewert. Wem das As... zu lange ist, kann im Variablennamen gleich das Typenzeichen voranstellen (% für Integer, & für Long)

- **kernel32.dll**
  Diese .dll enthält Funktionen, die mit dem Kernel (Betriebssystemkern) zusammenhängen, wie z.B. zur Speicherverwaltung oder die meisten Ein- und Ausgabefunktionen für Geräte.
  Du brauchst bei dieser Biblothek **kein** .dll anzuhängen.
- **user32.dll**
  Diese .dll enthält Funktionen, die mit der Benutzeroberfläche (Fenster,...) zusammenhängen, wie z.B. Funktionen zur Erstellung von Fenstern, Steuerelementen und Dialogfeldern.
  Du brauchst bei dieser Biblothek **kein** .dll anzuhängen.
- **gdi32.dll**
  Diese .dll enthält Funktionen, die mit der grafischen Ausgabe auf dem Bildschirm und dem Drucker zu tun haben, hierzu gehört auch die Textausgabe.
  Du brauchst bei dieser Biblothek **kein** .dll anzuhängen.
- **winmm.dll**
  Diese .dll enthält Funktionen, die mit Multimediafunktionen in Verbindung stehen, wie z.B. die Steuerung von Audio-, Videogeräten und Joysticks.
  Es ist das Anhängen von '.dll' **erforderlich**!
- **shell32.dll**
  Diese .dll enthält Funktioen, die mit der Benutzerschnittstelle auf einer hohen Ebene zu tun haben, wie z.B. Drag&Drop-Operationen für Dateien oder die Verwaltung von Verknüpfungen zwischen Dateien.
  Es ist das Anhängen von '.dll' **erforderlich**!

Beispiel für eine API-Deklaration

Das auf ein konkretes Beispiel anzuwenden, heißt die API-Deklaration für einen Zugriff auf die gdi32-Biblothek (Graphic Device Interface (enthält Funktionen für die Grafikausgabe auf dem Bildschirm und die Druckausgabe) so:
Private Declare Function Polygon Lib "gdi32" (ByVal hdc As Long, lpPoint As POINTAPI, ByVal nCount As Long) As Long

Aus der Biblothek gdi32.dll wird die Funktion Polygon importiert, diese enthält die Variablen hdc, die vom Long-Datentyp ist, die Variable lpPoint deren Datentyp vorher Benutzerdefiniert wurde und die Long-Variable nCount. Bei den Variablen nCount und hdc wird mit dem Schlüsselwort ByVal die Variable als Wert überwiesen (Das Gegenteil wäre ByRef, bei dem aus dem Variableninhalt ein Verweis gemacht wird).
An dieser Deklaration ist gut zu sehen, dass man bei der API-Deklaration sehr auf die Datentypen achten muss. Das API-Interface besteht aus C++-Syntax und C++ nimmt es etwas enger mit Datentypen als Visual Basic.

Beispiel-Projekt Polygon

Ich denke an einem kleinen Beispiel können wir unsere trockenen Theoriekenntnisse

einsetzten. Weil die Theorie sehr trocken (aber hoffentlich nicht langweilig) war, wollen wir einmal etwas malen (lassen). Wir werden die Biblothek gdi32.dll dazu verwenden und die oben beschriebene Funktion Polygon zum Zeichnen von einem kleinen Bild verwenden.

Projekt erstellen

Starte VB und lege ein neues Projekt an, an dem dann erscheinenden Formular soll nichts verändert werden. Wir wechseln stattdessen gleich in die Quelltextansicht und geben als erstes die Definition des benutzerdefinierten Datentyps POINTAPI an:

```
Private Type POINTAPI
X As Long
Y As Long
End Type
```

API-Funktion einfügen

Nun, da VB der Datentyp bekannt ist, können wir die Funktion Polygon in unser Projekt einfügen:

```
Private Declare Function Polygon Lib "gdi32" (ByVal hdc As Long, lpPoint As POINTAPI, ByVal nCount As Long) As Long
```

Zeichnungs-Prozedur einfügen

Nun erstellen wir eine Prozedur, die das Zeichnen übernehmen soll:

```
Private Sub LoadPointArray(ByVal width As Long, ByVal Height As Long, ByVal Increment As Integer, PointArray() As POINTAPI)
Dim curidx As Integer
ReDim PointArray((Height \ Increment) + 2)
Do
curidx = curidx + 1
PointArray(curidx).X = width
PointArray(curidx).Y = Height - curidx * Increment
curidx = curidx + 1
PointArray(curidx).X = 0
PointArray(curidx).Y = curidx * Increment
Loop While curidx * Increment < Height
End Sub
```

Aufruf der Funktion einfügen

Nun noch den Aufruf des Formulars zum Zeichnen Form_Paint:

```
Dim points() As POINTAPI
LoadPointArray ScaleWidth / Screen.TwipsPerPixelX, ScaleHeight /
Screen.TwipsPerPixelY, 5, points()
Result = Polygon(hWnd, points(0), UBound(points) + 1)
Call Polygon(hdc, points(0), UBound(points) + 1)
End Sub
```

Was zu beachten ist

Hier ist es nötig, dass X (bzw. Y) erst vom Pixel in das Visual Basic interne Twips Größeformat umgewandelt werden muss, sonst erhält man nur eine vergrößerte Ecke des Bildes im Fenster.

Zum Abschluss

Und zu guter Letzt noch eine kleine Anweisung, damit dass Bild auch noch in voller Bracht erscheint, wenn die Größe des Formulars vom Benutzer bei Laufzeit verändert wird:

Private Sub Form_Resize()
Refresh
End Sub

Nun sollte es nach der Ausführung so aussehen: