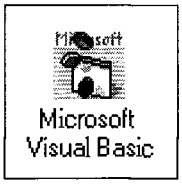
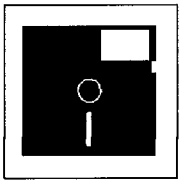
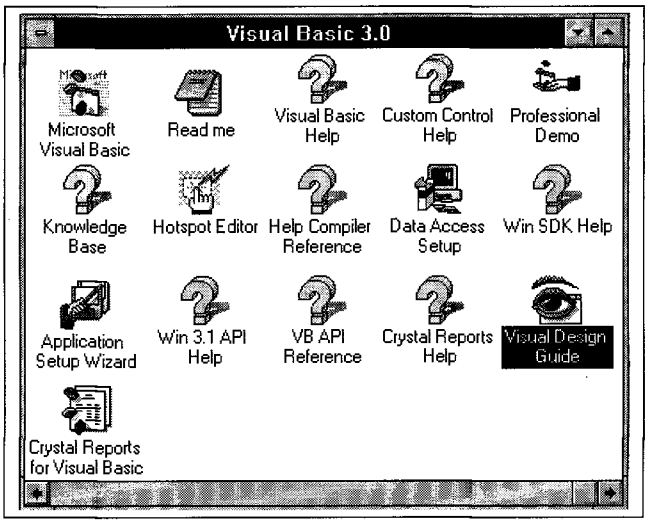






Programmieren für Juristen: Visual Basic – Lektion I

Maximilian Herberger

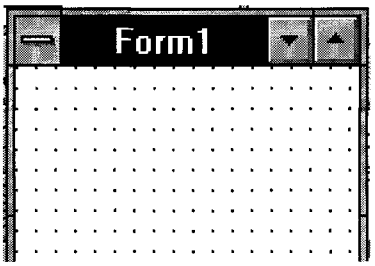


Abb. 1:
Das leere Formular "Form1"

Wer den Überlegungen im Editorial dieses Hefts zustimmt und demgemäß Visual Basic Professional auf seinem Rechner installiert hat, kann den Kurs absolvieren, den jur-pc beginnend mit diesem Heft in lockerer Folge anbieten wird. Starten wir also mit einem Doppelklick auf das Icon "Microsoft Visual Basic".
Das sich ergebende Bild ist nur am Anfang verwirrend: Verschiedene Visual Basic-Komponenten lagern sich über die gerade vorhandene Bildschirmoberfläche. Um welche Komponenten handelt es sich?
Im Mittelpunkt der Programmentwicklung steht das (am Anfang leere) Formular (Abb. 1).

Objektorientierung

Visual Basic folgt der objektorientierten Architektur und stuft dieses Formular (wie auch die weiteren im folgenden zu betrachtenden Gestaltungselemente) als Objekt ein, dem Eigenschaften ("properties") zugeordnet sind. Um welche Eigenschaften es sich handelt, ist im Fenster "Properties" zu sehen. (Sollte dieses Fenster nicht sichtbar sein, kann man es – wie auch die anderen Fenster im Pulldown-Menu "Windows" aktivieren.) Abb. 2 zeigt einen Ausschnitt des "Property"-Fensters zum beim Start vorhandenen leeren Formular Form1.

Objekt-Eigenschaften

Man kann die Eigenschaften des Formulars über das "Property"-Fenster verändern. Falls etwa die Überschrift nicht mehr "Form1" heißen soll, geht man in die Zeile "Caption" und überschreibt dort den Eintrag "Form1" mit dem gewünschten Eintrag. Das Ergebnis ist sofort in der Formularüberschrift zu sehen. Die weiteren Eigenschaften müssen hier nicht alle erläutert werden. Zum einen verstehen sich zahlreiche Bezeichnungen von selbst. Zum anderen kann man durch Verändern der Eigenschaften die Auswirkungen erkennen. Und wenn man gar nicht mehr weiter weiß, präsentiert ein Druck auf F1 die Hilfe zu der gerade aktiven (blau hinterlegten) "Property"-Zeile.

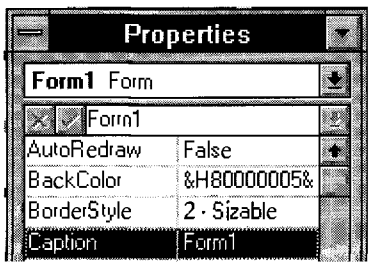


Abb. 2:
Eigenschaften des Objekts "Formular"

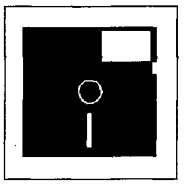
Nicht zu verwechseln mit der Überschrift und für das anschließende Programmieren wichtiger ist der Name, den das Formular hat. Über diesen Namen wird es nämlich (wie die anderen Objekte auch) im folgenden beim Programmieren angesprochen. Den Namen des ersten Formulars zu ändern dürfte nicht nötig sein, da "Form1" als Bezeichnung einprägsam genug ist. Bei den anderen Objekten sollte man es sich aber zur Regel machen, sogleich einen sprechenden und gut zu behaltenden Namen statt des von Visual Basic zugeordneten Standardnamens zu vergeben.

Objekt-Ereignisse

Von den Eigenschaften, die Zustände eines Objekts sind, unterscheidet der objektorientierte Ansatz die Vorgänge, die sich bezogen auf ein Objekt ereignen können. Da man sich solche Vorgänge auch als Verfahren vorstellen kann, die auf ein Objekt angewendet werden, findet sich des öfteren die Bezeichnung "procedures". Welche "Prozeduren" das beim Objekt "Formular" sein können, zeigt ein Doppelklick auf das Formular (vgl. Abb. 3).

Abb. 3:
"Prozeduren" für das Objekt "Formular"

Der erste und elementare auf ein Formular bezogene Vorgang ist der des Ladens, der deshalb konsequenterweise am Anfang der Liste steht. Man spricht bezogen auf die Art der



Programmierung, die im folgenden erläutert wird, auch von "ereignisorientierter" (oder "ereignisgesteuerter") Programmierung. Die "Vorgänge" bzw. "Prozeduren" sind nichts anderes als derartige Ereignisse, auf die das Programm reagiert. Deshalb ist dem Ereignis, daß das Formular geladen wird, im Programm-Code sogleich eine (noch leere) Unterroutine zugeordnet, die durch "sub" und "end sub" eingeklammert ist. Das spätere (ernsthafte) Programmieren wird sich an dieser Stelle abspielen. Zuvor bietet die visuelle Programmierumgebung aber noch einige Hilfestellungen an, die den Entwurf der Programm-Oberfläche in nützlicher Weise unterstützen. Um das verstehen zu können, muß man sich mit dem "Werkzeugkasten", der "toolbox" vertraut machen (vgl. Abb. 4).

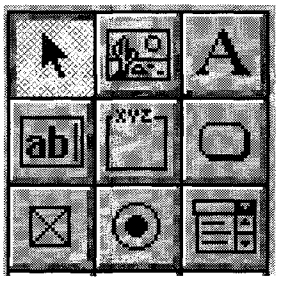


Abb. 4:
Der "Werkzeugkasten" (oberer Teil)

Ein "Werkzeug" ist die Textbox, die durch den Knopf mit dem Eintrag "ab" repräsentiert wird. Wenn wir auf diesen Knopf klicken und dann mit der Mouse auf dem Formular einen Kasten aufziehen, entsteht das aus Abb. 5 ersichtliche Formular mit dem darin plazierten Objekt "Textbox", das von Visual Basic den Namen "text1" bekommen hat. Wir belassen es für die Zwecke dieser ersten Übung bei diesem Namen, merken uns aber vor, daß bei einem wirklichen Programmierprojekt zunächst dieser Name geändert werden sollte.
Damit nun ein erstes kleines Programm entsteht, muß auf Grund eines Ereignisses eine Wirkung entstehen. Das Ereignis soll das Laden des Formulars sein und die darauffolgende Wirkung Das Erscheinen des Textes "Willkommen!" in der soeben kreirten Textbox. Zu diesem Zweck müssen wir innerhalb der oben beschriebenen Unterroutine "Form Load" ein wenig "programmieren", und zwar folgendermaßen:

Programm "Willkommen!"

```
Sub Form_Load ()
Text1.Text = "Willkommen!"
End Sub
```

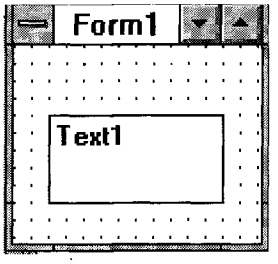


Abb. 5:
Formular mit Textbox

Das bedeutet folgendes: Wenn das Eröffnungsformular geladen wird, soll der Textbox namens "text1" der Text "Willkommen!" zugeordnet werden. Um zu sehen, ob sich das Programm wie erwartet verhält, können wir es sofort ablaufen lassen (Visual Basic ist zunächst einmal ein Interpreter). Zu diesem Zwecke wählt man "Run" und "Start" oder betätigt einfach die Taste F5: Voilä – Ergebnis wie erwartet. "Run" und "End" beendet dann diesen kleinen Probelauf.
Nur um mit der Grundidee der werkzeuggestützten graphischen Gestaltung einer Oberfläche noch einen Schritt weiter vertraut zu werden (nicht um die Illusion zu erwecken, es werde schon "richtig" programmiert), sei noch eine kleine Übung angeschlossen. Ziel der Übung ist es, durch Betätigung eines Befehlsknopfes im Textfeld das Systemdatum erscheinen zu lassen. Zu diesem Zweck benötigen wir neben dem bereits vorhandenen Textfeld einen Befehlsknopf. Der Befehlsknopf (vgl. Abb. 6) wird – wie eben beim Textfeld beschrieben – auf dem Formular plaziert. Das Ergebnis ist in Abb. 7 zu sehen.

Programm "Systemdatum"

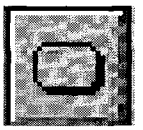


Abb. 6:
Der Befehlsknopf

Nun muß dem Befehlsbutton für das Ereignis "Click" noch Programmcode hinterlegt werden, um auf den Mouse-Klick hin das Datum im Textfeld erscheinen zu lassen. Dieser Code sieht wie folgt aus:

```
Sub Command1_Click ()
datum$ = Format$(Now, "dd.mm.yyyy")
Text1.Text = datum$
End Sub
```

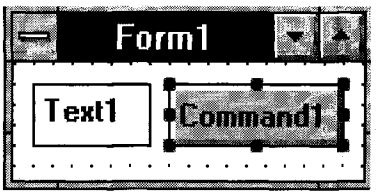
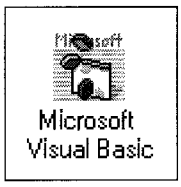
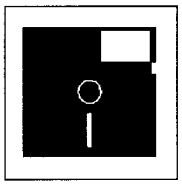


Abb. 7:
Formular mit Befehls-Button

Falls man nun den Befehlsknopf "drückt", erscheint das Systemdatum im Textfeld. (Die Verwendung der Funktion Format$ sorgt für die Formatierung des Datums, nachzulesen im Handbuch "Language Reference", S. 118, oder in der Online-Hilfe.)
Entgegen einer naheliegenden Vermutung sind wir jetzt schon in der Lage, Nützliches zu programmieren. Zum Beweis soll ein Programm geschrieben werden, das es erlaubt, den Inhalt des Clipboards "additiv" in einen Text zu übernehmen. (Bekanntlich kann das Clipboard selbst nicht als solch eine "Textschublade" dienen, da der frühere Inhalt bei jeder Schreibaktion überschrieben wird.)
Zuerst sind einige Operationen am vorhandenen Textfeld erforderlich.
Da das Textfeld längere Textstücke aufnehmen soll, vergrößern wir es etwas: Klick auf das Objekt, "Ziehen" an den Begrenzungsquadraten auf den Umrißlinien, ein Windows-Standardvorgang, vom Zeichnen her vertraut.
Nun gilt es, aus Gründen der gefälligen Präsentation einige Eigenschaften des Objekts "Textbox" zu verändern. Wir möchten den darin stehenden Text "Text1" nicht beibehalten.

Programm "Textschublade"





Also wird die Eigenschaft "Text" im "Properties"-Fenster auf leer gesetzt (= Löschen des dortigen Eintrags "Text1"). Dann soll, da wir uns im Text bewegen wollen, eine vertikale Scrollbar vorhanden sein, also: Eigenschaft "ScrollBars" auf "2-Vertical" setzen. Und schließlich soll der Text umbrochen werden, also: Eigenschaft "MultiLine" auf "True" setzen.
Schließlich muß dafür gesorgt werden, daß der Befehlsbutton das Gewünschte tut. Das bewirkt der folgende Code:

```
Sub Command1_Click ()
notiz$ = Text1.Text + ClipBoard.GetText()
Text1.Text = notiz$
End Sub
```

ClipBoard.GetText() ist eine fertige Visual Basic-Funktion, die im Clipboard vorhandenen Text einem String zuweist. Dabei wird der vorherige Inhalt der Textbox als Text1.Text mit übernommen (daher die "additive" Wirkung) und dann der Textbox ergänzt um den Inhalt des Clipboards wieder zugewiesen.
Ein erster kleiner Zwischentest zeigt: Wir können der Reihe nach Texte aus dem Clipboard in die Textbox übernehmen. Dabei funktioniert die Textbox wie ein kleiner Editor: Man kann sich "scrollend" im Text bewegen, man kann Text löschen, man kann Text einfügen.
Allerdings können wir mit dem bisherigen Ergebnis noch nicht zufrieden sein. Denn schließlich wollen wir am Ende unserer Arbeit den gesammelten Text auch abspeichern können. Das ist bisher nicht möglich. Der kunstgerechte Weg bestünde darin, den Text in eine Textdatei zu schreiben. Das würde aber dem Gang der Dinge in unserem Kurs vorgreifen, da es die Einführung der Dateibehandlung bereits an dieser Stelle erfordern würde. Um dem zu entgehen und den Text trotzdem speichern zu können, wählen wir einen zwar etwas umständlicheren, aber im Ergebnis an dieser Stelle tauglichen Weg: Wir schreiben am Ende über einen zweiten Befehlsknopf den akkumulierten Text in das Clipboard zurück. Von dort können wir ihn dann über Bearbeiten-Einfügen in eine Textverarbeitung übernehmen. Der nötige Code für diesen zweiten Befehlsbutton lautet:

```
Sub Command2_Click ()
ClipBoard.SetText Text1.Text
Text1.Text = ""
End Sub
```

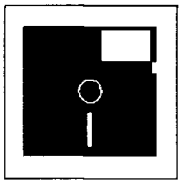
ClipBoard.SetText ist das Gegenstück zu ClipBoard.GetText und schreibt Text in das Clipboard, hier den in der Textbox angesammelten. Die darauffolgende Zeile löscht den Inhalt der Textbox durch Zuweisung eines leeren Strings.
Und damit wir schließlich das Programm auch außerhalb des Visual Basic-Interpreters beenden können, sehen wir noch einen dritten Befehlsbutton mit folgendem Code vor:

```
Sub Command3_Click ()
ClipBoard.SetText Text1.Text
End
End Sub
```

Um den Benutzer davor zu schützen, das Programm zu beenden, ohne vorher den Gesamttext in das Clipboard zurückgeschrieben zu haben, ist die Zeile "ClipBoard.SetText Text1.Text" (die bereits in Command2_Click anzutreffen war) hier nochmals verwendet worden.
Selbst wenn unser Programm zugegebenermaßen auf der Stufe "Überlistung des Clipboards" stehenbleibt, demonstriert es doch ansatzweise die Möglichkeit, mit Visual Basic in unaufwendiger Weise auf Windows-Funktionalität zuzugreifen. Im weiteren Verlauf dieses Kurses wird sich zeigen, daß darin eine der entscheidenden Stärken von Visual Basic liegt, die auch einem wenig erfahrenen Programmierer zugänglich ist.
Benutzen wir das letzte Programmbeispiel noch dazu, einen Blick auf weitere Arbeitsabläufe bei der Programmerstellung zu werfen.

Das Projektfenster

Das "Projektfenster" (stets erreichbar über "Window-Project") gibt eine Liste der im Projekt verwendeten Dateien (vgl. Abb. 8 auf der folgenden Seite).



Visual Basic lädt dabei eine Anzahl von Dateien mit der Extension .VBX, die man nicht unbedingt alle benötigt. Es handelt sich bei diesen .VBX(Visual Basic Executable)-Dateien, von denen später noch genauer die Rede sein wird, um ausführbare Dateien, die Visual Basic in der Form von Objekten eine zusätzliche Funktionalität zur Verfügung stellen. In der Terminologie des Handbuchs werden diese Funktionserweiterungen als "custom controls" bezeichnet.

Uns interessiert hier vorerst nur die Zeile zu Form1.frm (das ist unser Programm-Formular) mit den beiden Varianten "View Form" und "View Code".

Mit "View Form" wird das Formular samt den darauf plazierten Objekten angezeigt. Man kann dann diese Objekte bearbeiten, neue hinzufügen etc.

Mit "View Code" ist der im Programm geschriebene Code erreichbar. Das ist neben dem Doppelklick auf die Objekte, der gleichfalls den Code anzeigt, ein zweiter Weg zum Betrachten des Codes. Für den Code, der unabhängig von Objekten geschrieben wird (von ihm wird später noch die Rede sein), ist man auf diese Möglichkeit angewiesen.

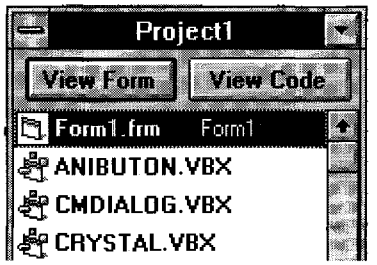


Abb. 8:
Das Projektfenster

Am Ende der Arbeit interessiert man sich dafür, das Projekt abzuspeichern, einen lesbaren Quelltext zu erhalten und eine .EXE-Datei herzustellen. Man bewerkstelligt all das im Pulldown-Menu "File".

"Save Project" speichert die Angaben zum Projekt in eine .MAK-Datei. Lädt man später diese Projektdatei, ist die gesamte Arbeitsumgebung wieder präsent. Die im Projekt verwendeten Formulare werden bei diesem Vorgang mitgespeichert, und zwar als .FRM-Dateien.

"Save Project"

"Save Text" speichert zusammenhängend in lesbarer Form den Quelltext des Programms. Bei der Lektüre erkennt man, welchen Vorteil die visuelle Gestaltung der Oberfläche mit sich bringt. Was man mit "Point and Click" schnell gestaltet hat, füllt im Quelltext mehrere komplizierte Zeilen. Man wird denn auch den Quelltext weniger wegen dieser Einträge lesen, als vielmehr wegen des sonstigen, den Programmablauf steuernden Codes. Diesen kann man zwar auch mit der erwähnten Option "View Code" betrachten. Dort erscheint er aber in "Häppchen", was für einen Gesamtüberblick problematisch ist. Um das zu kompensieren, wurde die Option "Save Text" hinzugefügt.

"Save Text"

"Make EXE File" schließlich erzeugt – wie der Name sagt – eine .EXE-Datei. Es handelt sich dabei aber nicht um eine eigenständig lauffähige Programmdatei. Benötigt wird zusätzlich auf jeden Fall noch ein Runtime-Programm (gegenwärtig VBRUN300.DLL) zusammen mit den verwendeten .VBX- und .DLL(Dynamic Link Library)-Dateien. Bei komplexen Projekten ist deshalb eine Installationsroutine aus Gründen der Anwenderunterstützung empfehlenswert. Davon wird aus gegebenem Anlaß später noch die Rede sein. Hier genügt es, die .EXE-Datei zu erzeugen und sie zum Testen über den Dateimanager aufzurufen.

"Make EXE File"

Und so sieht dann das Programm nach dem Aufruf der .EXE-Datei in geringfügig verschönerter Form aus (vgl. Abb. 9). Die Texte für "Caption" wurden in nach den obigen Beschreibungen leicht nachvollziehbarer Weise im Formular und bei den drei Befehlsknöpfen sprechender gestaltet.

Mit den Vorkenntnissen aus dieser Lektion ausgestattet ist es bis zur Programmierung einer kleinen, aber leistungsfähigen juristischen Datenbank nur noch ein kleiner Schritt. Davon demnächst mehr.

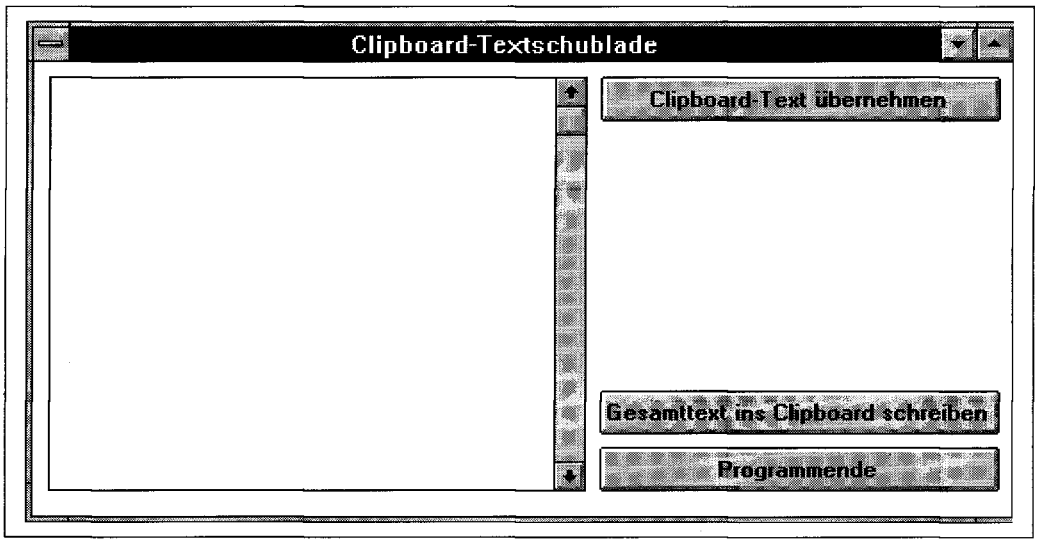


Abb. 9:
Programm "Clipboard-Textschublade"