Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind für Amateur- und Lehrzwecke bestimmt.

Alle technischen Angaben und Programme in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Wir weisen darauf hin, daß die im Buch verwendeten Soft- und Hardwarebezeichnungen und Markennamen der jeweiligen Firmen im allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Die wiedergegebenen Produktbezeichnungen sind für die jeweiligen Rechteinhaber markenrechtlich geschützt.

Copyright	© 1998 by DATA BECKER GmbH & Co.KG Merowingerstr. 30 40223 Düsseldorf
	1. Auflage 1998 el
Reihenkonzept	Peter Meisner
Lektorat	Peter Meisner
Schlußredaktion	Sibylle Feldmann
Umschlag	Inhouse-Agentur DATA BECKER
Buchinnengestaltung	DTP Studios Herten/**Marl** DTP Studios@BusinessNet.de

ISBN 3-8158-1525-8

Inhaltsverzeichnis

Mit Beginn der Windows-Ära zu Anfang dieses Jahrzehnts erlebte die Sprache jedoch in der Gestalt von Visual Basic, das indes mit seinem Uhrahn nur noch wenige Gemeinsamkeiten aufweist, eine kaum für möglich gehaltene neue Blüte. Die Methode der visuellen Programmierung befähigte in Verbindung mit dem Einsatz vorgefertigter Komponenten auch den Hobbyprogrammierer, schnell und problemlos Windows-Software zu erstellen. Mit jeder neuen Version wurde Visual Basic zwar professioneller, ist aber seiner Linie treu geblieben, dem Programmierer nahezu alle Facetten des komplexen Betriebssystems Windows zugänglich zu machen, ohne selbst kompliziert zu sein. Inzwischen ist es zu einem ausgereiften Produkt avanciert, das wegen seines auch im Vergleich zu Konkurrenzerzeugnissen wie C++ oder Delphi stattlichen Leistungspotentials selbst bei Programmierprofis in hohem Ansehen steht.

Trotz des mit den neuen Möglichkeiten wiederum gewachsenen Umfangs bleibt Visual Basic ein leicht zu erlernendes, einfach zu bedienendes und zugleich universelles Entwicklungswerkzeug. Die Vielseitigkeit dieser Sprache wird nicht zuletzt dadurch verdeutlicht, daß sie in abgespeckten Varianten als VBA (Visual Basic for Applications) zur Automatisierung und Anpassung der Microsoft Office Applikationen und als Skriptsprache VBScript im neuen Betriebssystem Windows 98 zum Einsatz kommt. Ziel dieses Buchs ist es, Ihnen als Einsteiger, Umsteiger oder Aufsteiger zu zeigen, wie komfortabel und zügig sich mit der Professional Edition von Visual Basic solide Windows-Anwendungen schreiben lassen.

1.2 Installieren und konfigurieren

Installationsvoraussetzungen

Der von einem Installations-Assistenten gesteuerte Setup-Prozeß geht recht bequem vonstatten. Voraussetzung ist, daß Ihr Computer folgende Mindestanforderungen erfüllt:

- Pentium-PC mit 90 MHz
- 24 MByte RAM
- CD-ROM-Laufwerk
- SVGA-Monitor
- Maus oder kompatibles Zeigegerät
- Windows 95, 98 oder NT 4.0 (SP 3) als Betriebssystem

Außerdem sollten Sie über freien Plattenspeicher verfügen, der – je nachdem, wie viele Bestandteile von Visual Basic und welche Zusatzprodukte Sie installieren möchten – in der Größenordnung von 150 bis 850 MByte liegen sollte. Ferner sollten Sie sich etwas Muße gönnen, denn solche Datenmengen von den Liefer-CDs auf Ihren PC zu transferieren und zu entpacken sowie die Komponenten einzurichten, braucht seine Zeit.

Ablauf der Installation

Nach Einlegen der ersten CD startet der Installations-Assistent, es sei denn, für das CD-ROM-Laufwerk ist der automatische Start (durch Deaktivieren der Option *Automatische Benachrichtigung beim Wechsel* im Geräte-Manager von Windows) abgeschaltet. In diesem Fall wählen Sie aus dem *Start*-Menü den Befehl *Ausführen*, geben in das Textfeld des Dialogfensters *x:\setup* ein, wobei Sie x durch den Buchstaben Ihres CD-ROM-Laufwerks ersetzen, und klicken die Schaltfläche *OK* an.

Wenn Sie im ersten Fenster des Setup-Assistenten auf *Info-Datei anzeigen* klicken, werden Sie unter Umständen von dem Ergebnis verblüfft sein: Der Installations-Assistent teilt Ihnen mit, daß als erstes der Internet Explorer 4.01 (SP 1) installiert oder ein Update auf diese Version durchgeführt werden muß. Der Grund liegt darin, daß Microsoft die Online-Dokumentation (Info-, Hilfe- und Handbuchdateien) zu allen Produkten der Visual Studio-Reihe ab Version 6.0 im HTML-Format veröffentlicht.

Ganz easy: HTML, WWW

HTML ist das Kürzel für **H**yper**T**ext **M**arkup **L**anguage und stellt den Sprachstandard für die Darstellung von Seiten im WWW (**W**orld **W**ide **W**eb) dar, dem bekanntesten Internet-Dienst.

Beim Setup des Internet Explorer können Sie zwischen *Standardinstallation* und *Vollständiger Installation* wählen. Erstere bringt neben dem Web-Browser das E-Mail-Programm Outlook Express und eine Reihe von Multimedia-Erweiterungen auf Ihren PC und erfordert etwa 40 MByte an Speicherplatz, die andere Alternative benötigt rund 20 MByte mehr für Internet-Zubehör wie NetMeeting oder NetShow. Wenn Sie nicht über eine Internet-Verbindung verfügen oder auf die Zusätze verzichten können, genügt die Standardinstallation. Die in einem separaten Fenster angebotene Option *Aktualisierung des Windows Desktops* verleiht der Oberfläche von Windows 95 das Look & Feel von Windows 98, ist aber für Visual Basic nicht notwendig.

In gewohnter Windows-Manier schließt die Installation eines Softwareprodukts mit einem Neustart des Systems ab. Unterbrechen Sie diesen Ablauf bitte nicht, der Installations-Assistent meldet sich automatisch zurück!

Tip: Unterbrochene Installation fortsetzen

Sollte der Installations-Assistent (aus welchen Gründen auch immer) seine Tätigkeit nicht von selbst fortsetzen, so können Sie ihn durch Aufruf der Datei *Setup.exe* im Stammordner der ersten CD dazu bringen, an der richtigen Stelle weiterzumachen.

Falls auf Ihrem PC der Internet Explorer nebst Komponenten in der verlangten Version (inklusive Service Pack 1, das einige Programmfehler korrigiert) vorhanden ist, entfällt natürlich dieser erste Installationsabschnitt.

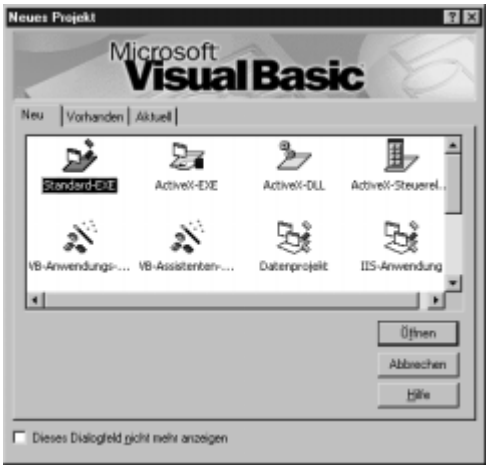
Der Setup-Assistent fährt auf jeden Fall in der beschriebenen Weise fort, die Existenz benötigter Systemkomponenten zu überprüfen, diese gegebenenfalls zu installieren oder zu aktualisieren und einen Neustart durchzuführen.

Als Installationsart für das eigentliche Visual Basic-Paket können Sie *Benutzerdefiniert* wählen. Auch wenn Sie an den voreingestellten Optionen nichts ändern, erhalten Sie so einen guten Überblick über Art und Umfang der installierbaren Komponenten. Zunächst nicht berücksichtigte Teile nachträglich zu installieren oder nicht benötigte später wieder zu entfernen, stellt kein Problem dar: Das mit *Start/Einstellungen/Systemsteuerung/Software* zu öffnende Dialogfenster gibt Ihnen auf der ersten Registerkarte die Möglichkeit, den Installationsumfang des in der Liste markierten Produkts über die Schaltfläche *Hinzufügen/Entfernen* zu modifizieren.

Zum Schluß sollten Sie unter Verwendung der mitgelieferten MSDN-Library-CDs (Microsoft Developer Network) die Online-Referenz auf Ihren PC einrichten. Dabei empfiehlt es sich, die teilweise recht informativen Programmbeispiele ebenfalls auf die Festplatte übertragen zu lassen. Sie werden im Ordner *…\MSDN98\…\Samples\VB98* abgelegt. Um sich beim Zugriff aus der Entwicklungsumgebung heraus die lästige „Klickerei“ durch diese Ordnerstruktur zu ersparen, können sie mit dem Windows-Explorer in einen unterhalb des Visual Basic-Ordners erstellten Ordner (etwa mit dem Namen *Beispiele*) verschoben werden.

Entwicklungsumgebung konfigurieren

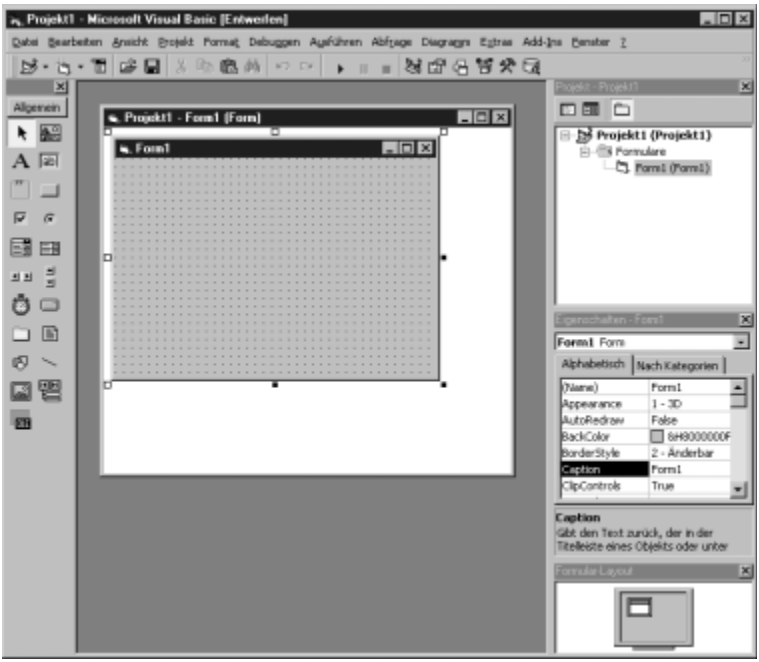
Beim erstmaligen Start (*Start/Programme/Microsoft Visual Studio 6.0/Microsoft Visual Basic 6.0*) zeigt Visual Basic das Dialogfeld *Neues Projekt* an.



Das Dialogfeld Neues Projekt

Das Fenster enthält die drei Registerkarten *Neu*, *Vorhanden* und *Aktuell*; auf der ersten können Sie den Typ eines neu zu entwickelnden Projekts bestimmen, die zweite läßt Sie ein bereits existierendes Projekt laden und die letzte offeriert eine Liste der bisher geöffneten Projekte. Wenn Sie auf der Registerkarte *Neu* den ersten Eintrag *Standard-EXE* für ein „normales“ Visual Basic-Programm durch einen Klick auf die Schaltfläche *Öffnen* bestätigen – diese Wahl erfolgt bei künftigen Starts automatisch, falls das Kontrollkästchen links unten aktiviert wird –, präsentiert sich die Entwicklungsumgebung von Visual Basic in ihrer Grundkonfiguration. Innerhalb des Hauptfensters, das die Windows-Standardelemente Titel-, Menü- und Symbolleiste enthält, sind fünf untergeordnete Fenster geöffnet:

- Werkzeugsammlung (links)
- Formular-Designer (Mitte)
- Projekt-Explorer (rechts oben)
- Eigenschaftenfenster (rechts Mitte)
- Formular-Layout-Fenster (rechts unten)



Die Entwicklungsumgebung nach dem Start

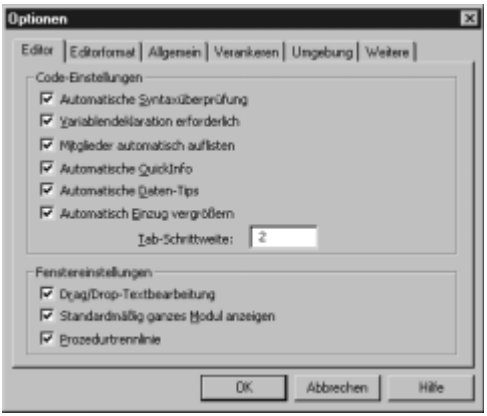
Mehr Infos: ➨ **Handhabung und Verwendung sämtlicher Elemente der Entwicklungsumgebung sind Thema des 3. Kapitels**

Die integrierte Entwicklungsumgebung von Visual Basic (IDE, **I**ntegrated **D**evelopment **E**nvironment) läßt sich hinsichtlich des Erscheinungsbilds und der Funktionsweise individuell einrichten.

Es ist ratsam, einige Einstellungen vorzunehmen, bevor Sie Ihr erstes Projekt in Angriff nehmen. Dazu rufen Sie im *Extras*-Menü den Befehl *Optionen* auf.

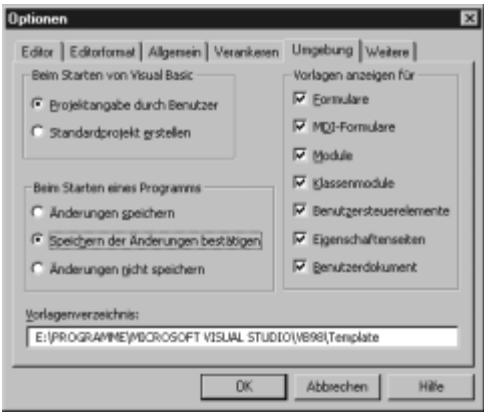
1. Auf der Registerkarte *Editor* tragen Sie durch das Markieren des Kontrollkästchens *Variablendeklaration erforderlich* dazu bei, gewisse Programmierfehler von vornherein zu vermeiden.

2. Ferner können Sie die *Tab-Schrittweite*, die festlegt, um wieviel Leerzeichen Programmzeilen beim Drücken der Tab-Taste eingerückt werden, ohne weiteres auf zwei Stellen reduzieren.



Optionen der Registerkarte Editor

3. Auf der Registerkarte *Umgebung* sollten Sie in der *Kategorie Beim Starten eines Programms* eine der beiden ersten Optionen wählen.



Optionen der Registerkarte Umgebung

Auf weitere sinnvolle Einstellungen wird an späterer Stelle noch einzugehen sein.

1.3 Ihr erstes Visual Basic-Projekt: Einen Windows-Wecker erstellen

Bei der Erstellung Ihres ersten Projekts, eines „Weckers“, der Sie an einen Termin erinnern soll, gehen Sie in vier Schritten vor.

6.2 Eigene Symbolleisten erstellen

Den raschesten Zugriff auf Befehle erhält der Benutzer über die Schaltflächen einer Symbolleiste, die in den meisten Windows-Anwendungen zum Ausstattungsstandard gehört. In der Entwicklungsumgebung von Visual Basic lassen sich bis zu fünf Symbolleisten einblenden.

Mehr Infos: Kurze Wege: Die Symbolleisten ➡ Kapitel 3.3

Mit dem *Symbolleiste*-Steuerelement steht eine gut ausgestattete ActiveX-Komponente zur Verfügung, die das Generieren von Symbolleisten leichtmacht.

Die Schaltflächen einer Symbolleiste geben eine Grafik und/oder Text wieder. Die Grafiken, entweder Bitmap- (*.bmp*) oder Symboldateien (*.ico*), stammen aus einem *Abbildungsliste*-Steuerelement, das über die *ImageList*-Eigenschaft der Symbolleiste mit dieser verknüpft wird.

Die einzelnen Abbildungen sind Elemente einer Auflistung von *ListImage*-Objekten, die durch einen ganzzahligen Index (*Index*-Eigenschaft) oder einen aus einer Zeichenfolge bestehenden Schlüssel (*Key*-Eigenschaft) unterschieden werden.

Die Bildgröße wird von dem *Abbildungsliste*-Steuerelement normiert, entweder auf die den Eigenschaften *ImageHeight* und *ImageWidth* vor dem Hinzufügen der ersten Grafik zugewiesenen Werte oder auf die Maße der ersten geladenen Abbildung.

Zur Laufzeit werden Abbildungen zur *ListImages*-Auflistung mit der *Add*-Methode hinzugefügt und mit der *Remove*-Methode aus ihr entfernt. Im Entwurfsmodus verwendet man dafür das Dialogfeld *Eigenschaftenseiten*, das im Eigenschaftenfenster über den Eintrag *Benutzerdefiniert* geöffnet wird.

Entsprechendes gilt auch für die Schaltflächen des *Symbolleiste*-Steuerelements, die Bestandteile einer Auflistung von *Button*-Objekten sind. Verhalten und Erscheinungsbild der Schaltflächen sind durch ihre *Style*-Eigenschaft bestimmt, die einen von fünf Werten besitzt.

Style-Eigenschaft	Beschreibung
tbrButtonGroup	Eine Schaltfläche der Gruppe kann gedrückt sein, das Drücken einer anderen bringt sie wieder in den Ausgangszustand (ähnlich Optionsfeld-Gruppe).
tbrCheck	Umschalter (ein/aus), Zustand durch Stellung (eingedrückt/nicht eingedrückt) angezeigt
tbrDefault	Normale Befehlsschaltfläche (Voreinstellung)
tbrDropdown	Platzhalter für Dropdown-Menü, das in Form von *ButtonMenu*-Objekten eingefügt wird
tbrPlaceholder	Platzhalter variabler Breite für ein anderes Steuerelement (z. B. Kombinationsfeld)
tbrSeparator	Unsichtbarer Abstandhalter mit fester Breite (8 Pixel)

Die *Value*-Eigenschaft des *Button*-Objekts legt fest, ob die Schaltfläche momentan gedrückt (*tbrPressed*) oder nicht gedrückt (*tbrUnPressed*) ist.

Zur Laufzeit kann die Symbolleiste vom Benutzer angepaßt werden (Schaltflächen entfernen oder umordnen), und zwar mit Hilfe eines Dialogfelds, das per Doppelklick auf einen freien Bereich der Symbolleiste geöffnet wird, vorausgesetzt, daß die *AllowCustomize*-Eigenschaft des Steuerelements auf *True* gesetzt ist.

Im Programmcode läßt sich dies mit der *Customize*-Methode erreichen, mit den Methoden *SaveToolbar* und *RestoreToolbar* kann der aktuelle Zustand einer Symbolleiste in der Windows-Registrierung abgespeichert und wiederhergestellt werden.

Das Erstellen einer Symbolleiste in Visual Basic ist nicht schwierig, aber es sind doch eine Reihe von Details zu beachten. Am besten macht man sich mit ihnen anhand eines konkreten Beispiels vertraut.

Übung 16 Symbolleiste erstellen

Das Projekt aus der vorigen Übung soll mit einer Symbolleiste versehen werden, die den Zugriff auf alle Befehle ermöglicht, die in den Menüs verfügbar sind. Als erstes müssen geeignete Abbildungen für die Schaltflächen der Symbolleiste bereitgestellt werden. Um Grafikdateien für die drei Schaltflächen zur Einstellung der Hintergrundfarbe des Textfelds zu erzeugen, gehen Sie folgendermaßen vor.

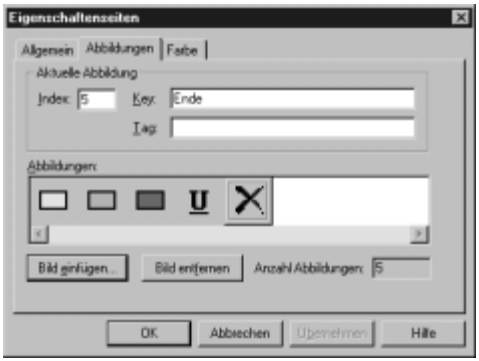
1. Starten Sie die als Zubehör von Windows mitgelieferte Anwendung Paint oder ein anderes Pixel-Grafikprogramm.
2. Öffnen Sie die Datei *Rectangl.bmp* aus dem Ordner *Graphics\Bitmaps\Tlbr_w95*.
3. Mit dem Füllwerkzeug färben Sie das Innere des Rechtecks gelb.
4. Durch den Befehl *Datei/Speichern unter* legen Sie die Grafik in der Datei *Gelb.bmp* im Ordner *_Übungen* ab.
5. Wiederholen Sie die beiden letzten Schritte, wobei Sie die Farben in Cyan und Magenta und die Dateinamen entsprechend in *Cyan.bmp* und *Magenta.bmp* abändern.

Zeichnen Sie nun ein *Abbildungsliste*-Steuerelement in das Formular. Da es zur Laufzeit nicht sichtbar ist, kommt es auf die Position nicht an.

Hinweis: Abbildungsliste in Werkzeugsammlung einfügen

Falls das Steuerelement in der Werkzeugsammlung fehlt, fügen Sie es mit Hilfe der Registerkarte *Steuerelemente* des Dialogfelds *Komponenten* (Befehl *Projekt/Komponenten*) hinzu, indem Sie den Eintrag *Microsoft Windows Common Controls 6.0* auswählen. Bestandteil dieser Gruppe von ActiveX-Komponenten (Datei *Mscomctl.ocx*) ist auch das *Symbolleiste*-Steuerelement.

Klicken Sie im Eigenschaftenfenster für die Abbildungsliste auf die Schaltfläche neben *Benutzerdefiniert*, um das Dialogfeld *Eigenschaftenseiten* einzublenden.



Eigenschaftenseiten-Dialog der Abbildungsliste

Mit Hilfe der Schaltfläche *Bild einfügen* auf der Registerkarte *Abbildungen* werden dann fünf Bilddateien der Abbildungsliste in dieser Reihenfolge hinzugefügt.

Bilder der Abbildungsliste	Key
..._Übungen\Gelb.bmp	
..._Übungen\Cyan.bmp	
..._Übungen\Magenta.bmp	
...\Graphics\Bitmaps\Tlbr_w95\Undrln.bmp	Unterstrichen
...\Graphics\Bitmaps\Tlbr_w95\Delete.bmp	Ende

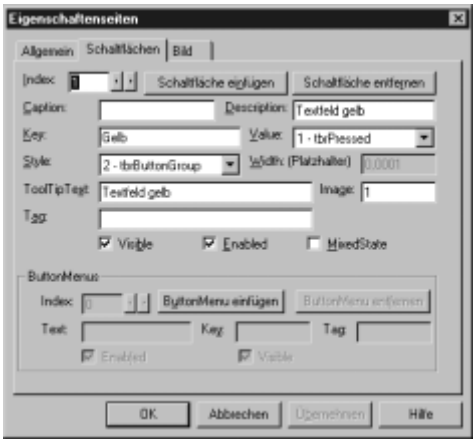
Der Index beginnt für Auflistungen bei 1 und wird mit jedem hinzukommenden Bild automatisch um 1 erhöht. Die *Key*-Eigenschaft der drei ersten *ListImage*-Objekte bleibt leer, sie wird nur für die beiden letzten auf die angegebenen Zeichenfolgen gesetzt.

Als nächstes wird das Formular mit einem *Symbolleiste*-Steuerelement versehen. Die veränderten Eigenschaftswerte des Formulars und die Namen der hinzugefügten Objekte sind:

Objekt	Eigenschaft	Einstellung
Formular	Name	frmÜbung16
	Caption	Symbolleiste
Abbildungsliste	Name	ilsSymbolleiste
Symbolleiste	Name	tlbMenüs

Wie zuvor bei der Abbildungsliste öffnen Sie nun das Dialogfeld *Eigenschaftenseiten* der Symbolleiste. Auf der Registerkarte *Allgemein* wählen Sie für die *ImageList*-Eigenschaft aus dem Dropdown-Kombinationsfeld den Eintrag *ilsSymbolleiste* und stellen damit die Verknüpfung zwischen Symbolleiste und Abbildungsliste her.

Mit Hilfe von *Schaltfläche einfügen* auf der Registerkarte *Schaltflächen* werden der Symbolleiste fünf *Button*-Objekte hinzugefügt.



Eigenschaftenseiten-Dialog der Symbolleiste

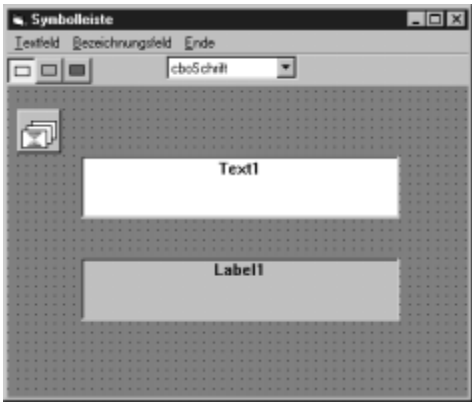
Die *Index*-Eigenschaft wird wieder automatisch verwaltet, die übrigen Eigenschaften versehen Sie mit folgenden Werten:

Description	Key	Value	Style	Width	Image
Textfeld gelb	Gelb	tbrPressed	tbrButtonGroup		1
Textfeld cyan	Cyan		tbrButtonGroup		2
Textfeld magenta	Magenta		tbrButtonGroup		3
Trennfläche1			tbrSeparator		
Schriftart	Schriftart		tbrPlaceholder	1800	

Der *ToolTipText*-Eigenschaft der ersten drei Schaltflächen werden dieselben Zeichenfolgen wie der *Description*-Eigenschaft zugewiesen. Wenn Sie jetzt noch dieses Kombinationsfeld in die Symbolleiste einfügen,

Objekt	Eigenschaft	Einstellung
Kombinationsfeld	Name	cboSchrift
	Style	2 - Dropdown-Liste
	ToolTipText	Schriftart Bezeichnungsfeld

ist die Entwurfsphase abgeschlossen, und die Oberfläche hat ungefähr folgendes Aussehen:



Die Oberfläche von Übung16 nach Beenden der Entwurfsphase

Eine Symbolleiste erwartet der Benutzer im allgemeinen direkt unterhalb der Menüleiste. Dies ist mit dem Wert von *vbAlignTop* auch die Voreinstellung der dafür zuständigen *Align*-Eigenschaft. Durch Wahl anderer Werte läßt sich aber die Symbolleiste auch an einem der übrigen Formularränder verankern oder in Größe und Position frei festlegen.

Der Programmcode aus *Übung15* ist um einige Zeilen in der *Load*-Prozedur des Formulars zu erweitern. Dort werden zuerst weitere Schaltflächen durch Anwendung der *Add*-Methode zur *Buttons*-Auflistung hinzugefügt.

```
Objekt.Add(Index, Key, Caption, Style, Image)
```

Die Argumente der Methode definieren fünf der Eigenschaften eines *Button*-Objekts. Sie sind alle optional und stellen benannte Argumente dar. Die so erstellte Objektinstanz wird mit der *Set*-Anweisung einer Objektvariablen vom Typ *Button* zugewiesen:

```
Dim btnX As Button

Set btnX = tlbMenüs.Buttons.Add(Key:="Unterstrichen", _
  Style:=tbrCheck, Image:="Unterstrichen")
```

Da kein Wert für *Index* angegeben ist, wird das *Button*-Objekt an das Ende der Auflistung gesetzt. Der Schlüssel (*Key*-Eigenschaft) lautet *"Unterstrichen"*, und die Schaltfläche zeigt keinen Text an (leere *Caption*-Eigenschaft). Die Schaltfläche stellt einen Umschalter dar (*Style = tbrCheck*) und gibt die Grafik der zugeordneten Abbildungsliste wieder, deren Schlüssel den Wert *"Unterstrichen"* besitzt.

Die restlichen Eigenschaften der Schaltflächen (wie *Description* oder *ToolTipText*) müssen gesondert festgelegt werden. Übrig bleibt noch die Aufgabe, das Kombinationsfeld zu füllen und an die vorgesehene Position zu bringen. Ersteres wird in gewohnter Weise durch die *AddItem*-Methode erledigt. Zum Verschieben und Ändern der Abmessungen von Formularen und Steuerelementen bietet sich die *Move*-Methode an:

```
[Objekt.]Move links [, oben[, Breite[, Höhe] ] ]
```

Nur das erste der Argumente (*links*) ist erforderlich, nicht aufgeführte behalten ihren Wert. Durch die drei Zeilen

```
With tlbMenüs.Buttons("Schriftart")
  cboSchrift.Move .Left, .Top, .Width
End With
```

wird das Kombinationsfeld so über der Platzhalter-Schaltfläche (mit dem Schlüssel *Schriftart*) positioniert, daß es links oben bündig mit ihr abschließt und gleichzeitig seine Breite an die Schaltflächenbreite angepaßt wird.

Hinweis: Height-Eigenschaft schreibgeschützt

Die Höhe von Dropdown-Kombinationsfeldern und -Listen kann nicht verändert werden, ihre *Height*-Eigenschaft ist schreibgeschützt.

Der Code der *Form_Load*-Prozedur lautet dann:

```
Private Sub Form_Load()
  Dim btnX As Button

  ' Schaltflächen hinzufügen
  Set btnX = tlbMenüs.Buttons.Add(Key:="Trennfläche2", _
    Style:=tbrSeparator)
  Set btnX = tlbMenüs.Buttons.Add(Key:="Unterstrichen", _
    Style:=tbrCheck, Image:="Unterstrichen")
  Set btnX = tlbMenüs.Buttons.Add(Key:="Trennfläche3", _
    Style:=tbrSeparator)
  Set btnX = tlbMenüs.Buttons.Add(Key:="Ende", _
    Style:=tbrDefault, Image:="Ende")

  ' Eigenschaften der Schaltflächen
  tlbMenüs.Buttons("Trennfläche2").Description = _
    "Trennfläche2"
  tlbMenüs.Buttons("Trennfläche3").Description = _
    "Trennfläche3"
```

```
  With tlbMenüs.Buttons("Unterstrichen")
    .Description = "Unterstrichen"
    .ToolTipText = "Text unterstreichen"
  End With
  With tlbMenüs.Buttons("Ende")
    .Description = "Ende"
    .ToolTipText = "Programmende"
  End With

  ' Kombinationsfeld füllen
  cboSchrift.AddItem "Courier New"
  cboSchrift.AddItem "Symbol"
  cboSchrift.AddItem "Times New Roman"
  cboSchrift.Text = cboSchrift.List(0)

  ' und an die richtige Position bringen
  With tlbMenüs.Buttons("Schriftart")
    cboSchrift.Move .Left, .Top, .Width
  End With

  txtMenü.Text = "Textfeld:" & vbCrLf & _
    "Die Hintergrundfarbe" & vbCrLf & _
    "kann verändert werden."
  txtMenü.BackColor = vbYellow

  lblMenü.Caption = "Bezeichnungsfeld:" & vbCrLf & _
    "Schriftart und Schriftschnitt" & vbCrLf & _
    "sind variabel."
  lblMenü.Font.Name = "Courier New"
End Sub
```

Wenn Sie das Programm probeweise starten, sehen Sie, daß die Abbildungsliste nicht mehr angezeigt wird und die Symbolleiste sämtliche Bedienungselemente aufweist.

Die Oberfläche von Übung16 zur Laufzeit

Damit die Schaltflächen auf der Symbolleiste die ihnen zugedachten Funktionen ausführen, bedarf es noch zweier Prozeduren. Die im Kombinationsfeld gewählte Schriftart wird durch deren *ListIndex*-Eigenschaft identifiziert, sie wird beim Aufruf der *Menü*-Prozedur *mnuBFeldSchrift_Click* als Argument übergeben.

```
Private Sub cboSchrift_Click()
  mnuBFeldSchrift_Click cboSchrift.ListIndex
End Sub
```

Damit ist gleichzeitig sichergestellt, daß die aktuelle Schrift auch im Menü *Bezeichnungsfeld* korrekt angezeigt wird. Um zu erreichen, daß umgekehrt die im Menü vorgenommene Schriftauswahl im Kombinationsfeld erscheint, fügen Sie am Ende der *If*-Abfrage der Prozedur *mnuBFeld Schrift_Click* (siehe S. 290) folgendes ein:

```
    ' Kombinationsfeld synchronisieren
    cboSchrift.ListIndex = Index
```

Den Code zur Programmierung der Schaltflächen schreiben Sie in die Prozedur des *ButtonClick*-Ereignisses der Symbolleiste. Es tritt ein, wenn auf eine „echte“ Schaltfläche (kein Abstand- oder Platzhalter) angeklickt wird.

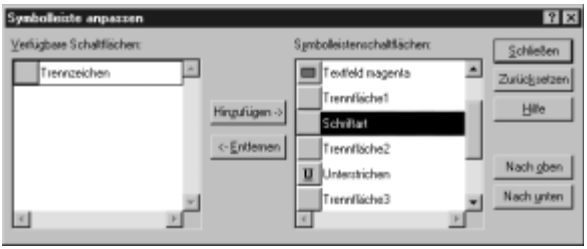
In einer *Select Case*-Anweisung wird anhand des Schlüssels abgefragt, welche Schaltfläche angeklickt wurde, und dann der entsprechende Menübefehl aufgerufen.

```
Private Sub tlbMenüs_ButtonClick(ByVal _
  Button As MSComctlLib.Button)

  Select Case Button.Key
  Case "Gelb"
    mnuTFeldFarbe_Click (0)
  Case "Cyan"
    mnuTFeldFarbe_Click (1)
  Case "Magenta"
    mnuTFeldFarbe_Click (2)
  Case "Unterstrichen"
    mnuBFeldSchrift_Click (4)
  Case "Ende"
    mnuEnde_Click
  End Select
End Sub
```

Hinweis: Zweifache Verwendung von Button

Lassen Sie sich nicht dadurch verwirren, daß im Argumentteil dieser Prozedur zweimal das Wort *Button* auftaucht. Zum einen wird damit der Variablenname, zum anderen der Objekttyp bezeichnet, dem allerdings mit *MSComctlLib* der Name der zugehörigen Klassenbibliothek vorangestellt ist, was der Klarstellung dient. Selbstverständlich können Sie aber die Variable auch umbenennen.

Die Anwendung ist jetzt fertiggestellt und kann gestartet werden. Wenn Sie auf eine freie Stelle in der Symbolleiste doppelklicken, wird das Dialogfeld *Symbolleiste anpassen* eingeblendet.



Das Dialogfeld Symbolleiste anpassen

Hier können Sie mit den entsprechenden Befehlsschaltflächen die Symbolleiste umgestalten. In der Liste der vorhandenen Schaltflächen erscheint neben jeder Schaltfläche der Text, der zuvor als *Description*-Eigenschaft festgelegt worden ist.

Weil sich die Schaltflächen der Symbolleiste neu arrangieren lassen (falls die *AllowCustomize*-Eigenschaft auf ihrem voreingestellten Wert *True* belassen wird), kann deren Reihenfolge, die sich in den *Index*-Werten der *Button*-Objekte widerspiegelt, verändert werden. Deshalb folgende Empfehlung, nach der in der *ButtonClick*-Prozedur verfahren worden ist:

Tip: Key-Eigenschaft zur Identifizierung verwenden

Zur Identifizierung der Objekte in einer Auflistung sollten, wenn es vorkommen kann, daß die Reihenfolge geändert wird, die konstant bleibenden Werte der *Key*-Eigenschaft anstelle der veränderlichen *Index*-Werte benutzt werden.

Zu den Verbesserungen am *Symbolleiste*-Steuerelement in der neuen Version von Visual Basic gehört das *ButtonMenu*-Objekt, mit dem auf einer Schaltfläche Dropdown-Menüs erstellt werden können. Um dieses Feature auszuprobieren, können Sie das letzte Anwendungsbeispiel durch wenige Änderungen wie folgt umgestalten.

Übung 17 Symbolleiste mit Dropdown-Menü

Ziel ist es, die Funktionalität der Menüleiste in die Symbolleiste zu kopieren. Dabei sollen die Befehle des *Textfeld*-Menüs durch das neue *ImageCombo*-Steuerelement, die des Menüs *Bezeichnungsfeld* durch ein Dropdown-Menü und der Befehl zum Beenden der Anwendung durch eine normale Schaltfläche realisiert werden.

Löschen Sie zuerst durch mehrmaliges Anklicken von *Schaltfläche entfernen* auf der Karte *Schaltflächen* des *Eigenschaftenseiten*-Dialogs der Symbolleiste (siehe S. 297) alle vorhandenen Schaltflächen und fügen Sie dann vier Schaltflächen gemäß dieser Tabelle hinzu:

Caption	Key	Style	Width	ToolTipText	Image
	Farbe	tbrPlaceholder	1500		
Schrift		tbrDropdown		Schriftart/Schriftstil	
		tbrSeparator			
	Ende	tbrDefault		Programm beenden	5

Die zweite Schaltfläche mit dem Stil *tbrDropdown* versehen Sie unter Verwendung von *ButtonMenu einfügen* im unteren Teil der Registerkarte mit folgenden Menüpunkten, wobei wie gewohnt zur Erzeugung einer Trennlinie ein einzelner Bindestrich genügt:

Index	Text
1	Courier
2	Symbol
3	Times New Roman
4	-
5	Unterstrichen

Zum Schluß zeichnen Sie ein *ImageCombo*-Steuerelement in die Symbolleiste und setzen seine Eigenschaften und die der anderen betroffenen Objekte auf die angegebenen Werte.

Objekt	Eigenschaft	Einstellung
Formular	Name	frmÜbung17
	Caption	Symbolleiste mit Menü
ImageCombo	Name	icboFarbe
	ImageList	ilsSymbolleiste
	Locked	True
	ToolTipText	Textfeld-Farbe
Symbolleiste	Name	tlbMenüs
	AllowCustomize	False
	Style	1 – tbrFlat
	TextAlignment	1 – tbrTextAlignRight

Der Eigenschaft *ImageList* des *ImageCombo*-Steuerelements, die sich nur im Dialogfeld *Eigenschaftenseiten* einstellen läßt, wird die schon für die Symbolleiste verwendete Abbildungsliste zugewiesen.

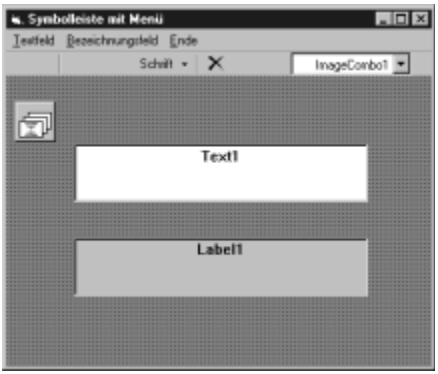
Im Gegensatz zur normalen Dropdown-Liste, die automatisch für Eingaben gesperrt ist, muß hier die Sperrung explizit erfolgen (*Locked* = *True*).

Um die Symbolleiste möglichst schmal zu halten, wird durch den Wert der *TextAlignment*-Eigenschaft der Schaltflächentext nicht unter einem eventuell vorhandenen Bild, sondern rechts von ihm positioniert.

Die flach dargestellten Schaltflächen (*Style*) können zur Laufzeit nicht neu arrangiert werden (*AllowCustomize*), weil das hier nicht erforderlich ist, und zudem – das soll nicht verschwiegen werden – im Dialogfeld *Symbolleiste anpassen* weder die Darstellung noch das Hinzufügen oder Entfernen von Schaltflächen einwandfrei funktionieren.

Der Grund für dieses Verhalten (vielleicht das Dropdown-Menü?) konnte bisher nicht geklärt werden.

Die Oberfläche der Anwendung hat nun dieses Aussehen:



Die Oberfläche von Übung17

In der *Load*-Prozedur des Formulars wird die Liste des *ImageCombo*-Steuerelements gefüllt, indem der *ComboItems*-Auflistung neue Objekte mit der *Add*-Methode hinzugefügt werden. Von den insgesamt sechs benannten Argumenten diese Methode kommen zwei zum Einsatz: *Text* zur Festlegung der Zeichenfolge und *Image* zur Kennzeichnung des zugehörigen Bilds aus der Abbildungsliste (*Index*- oder *Key*-Wert).

Hinweis: Englische Argumentnamen verwenden

Bei dem im Grunde lobenswerten Unterfangen, die Online-Hilfe ins Deutsche zu übertragen, ist dem Übersetzer auf der Seite, die die *Add*-Methode für die *ComboItems*-Auflistung erläutert, ein Fauxpas unterlaufen: Er hat des Gutgemeinten zuviel getan und auch die Argumentnamen übersetzt. Die Verwendung dieser deutschen Bezeichnungen als benannte Argumente führt zu einem Kompilierungsfehler. Die korrekten Namen finden Sie im Objektkatalog (Details zum Mitglied *Add* der Klasse *ComboItems*).

Vor dem Plazieren des *ImageCombo*-Steuerelements auf der ersten Schaltfläche der Symbolleiste wird noch das erste Listenelement über die *Selected*-Eigenschaft der Auflistung im Textfeldbereich des Steuerelements angezeigt. Die Prozedur enthält somit folgenden Code:

```
Private Sub Form_Load()
  Dim ciX As ComboItem

  ' ImageCombo füllen, 1. Element auswählen
  Set ciX = icboFarbe.ComboItems.Add(Text:="gelb", _
    Image:=1)
  Set ciX = icboFarbe.ComboItems.Add(Text:="cyan", _
    Image:=2)
  Set ciX = icboFarbe.ComboItems.Add(Text:="magenta", _
    Image:=3)
  icboFarbe.ComboItems(1).Selected = True
```

```
  ' und an die richtige Position bringen
  With tlbMenüs.Buttons("Farbe")
    icboFarbe.Move .Left, .Top, .Width
  End With

  txtMenü.Text = "Textfeld:" & vbCrLf & _
    "Die Hintergrundfarbe" & vbCrLf & _
    "kann verändert werden."
  txtMenü.BackColor = vbYellow

  lblMenü.Caption = "Bezeichnungsfeld:" & vbCrLf & _
    "Schriftart und Schriftschnitt" & vbCrLf & _
    "sind variabel."
  lblMenü.Font.Name = "Courier New"
End Sub
```

An die Stelle der *Click*-Prozedur des Kombinationsfelds tritt jetzt die Prozedur *icboFarbe_Click*, die allerdings nicht mit nur einer Anweisung auskommt:

```
Private Sub icboFarbe_Click()
  Dim icX As ComboItem

  For Each icX In icboFarbe.ComboItems
    If icX.Selected Then
      mnuTFeldFarbe_Click icX.Index - 1
      Exit For
    End If
  Next
End Sub
```

Mit einer *For Each*-Schleife werden alle Elemente der *ComboItems*-Auflistung durchlaufen, bis das ausgewählte Element (*Selected = True*) gefunden ist. Sein *Index*-Wert wird dann benutzt, um den entsprechenden Befehl des *Textfeld*-Menüs aufzurufen, jedoch erst nach Korrektur dieses Werts, da die Zählung bei Auflistungen generell mit 1, beim *Menü*-Steuerelementefeld aber mit 0 anfängt.

Die zur Synchronisation von Menü und Kombinationsfeld in die Prozedur *mnuBFeldSchrift_Click* eingefügte Anweisung wird gelöscht oder auskommentiert. Um die Anzeigen in *Textfeld*-Menü und *ImageCombo*-Steuerelement zu synchronisieren, wird zu Beginn der *mnuTFeldFarbe_Click*-Prozedur folgende Anweisung eingefügt:

```
  'ImageCombo synchronisieren
  icboFarbe.ComboItems(Index + 1).Selected = True
```

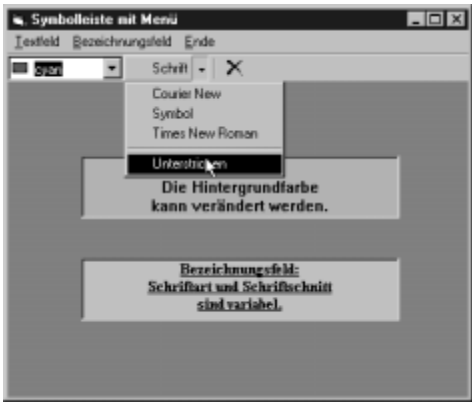
Bei Auswahl eines Befehls im Symbolleisten-Menü *Schrift* tritt das *ButtonMenuClick*-Ereignis ein. In der zugehörigen Prozedur werden Sie auf die Befehle des Menüs *Bezeichnungsfeld* umgeleitet:

```
Private Sub tlbMenüs_ButtonMenuClick(ByVal _
  ButtonMenu As MSComctlLib.ButtonMenu)

  mnuBFeldSchrift_Click ButtonMenu.Index - 1
End Sub
```

In der *ButtonClick*-Ereignisprozedur braucht dann nur noch die Schaltfläche zum Beenden des Programms behandelt zu werden:

```
Private Sub tlbMenüs_ButtonClick(ByVal _
  Button As MSComctlLib.Button)

  If Button.Key = "Ende" Then
    mnuEnde_Click
  End If
End Sub
```

Nach dem Starten der Anwendung können Sie das Menü *Schrift* durch Anklicken der kleinen Pfeil-Schaltfläche rechts von der Beschriftung aufklappen. Sie werden feststellen, daß alle Befehle der normalen Menüs in gleicher Weise auf der Symbolleiste zugänglich sind. Häkchen zur Anzeige der momentanen Einstellung stehen allerdings in Symbolleisten-Menüs nicht zur Verfügung, weil den *ButtonMenu*-Objekten im Unterschied zum *Menü*-Steuerelement neben anderen die *Checked*-Eigenschaft fehlt.



Übung17 zur Laufzeit