3 Entwicklungsumgebung

Dieses Kapitel gibt Tips zum Umgang mit der Entwicklungsumgebung von Visual Basic, ohne allerdings auf alle Details einzugehen. Der Aufbau des Kapitels orientiert sich an den Phasen der Programmerstellung (Formulare, Code, Kompilierung).

Zwei wichtige Komponenten der Entwicklungsumgebung werden nicht in diesem Kapitel, sondern in anderen Teilen des Buchs beschrieben: das Testfenster sowie die Kommandos zur Fehlersuche in einem eigenen Kapitel zum Thema Fehlersuche / Fehlerabsicherung (Seite 365), der Menüeditor in einem Abschnitt zur Gestaltung und Programmierung von Menüs (Seite 445).

3.1 Einführung

Projekttypen

Beim Start von Visual Basic bzw. beim Einfügen eines neuen Projekts werden Sie mit einer stattlichen Auswahl von Projekttypen überrascht. Wenn Sie ein traditionelles Programm ohne Internet- und ActiveX-Spielereien entwickeln wollen, entscheiden Sie sich für STANDARD-EXE.

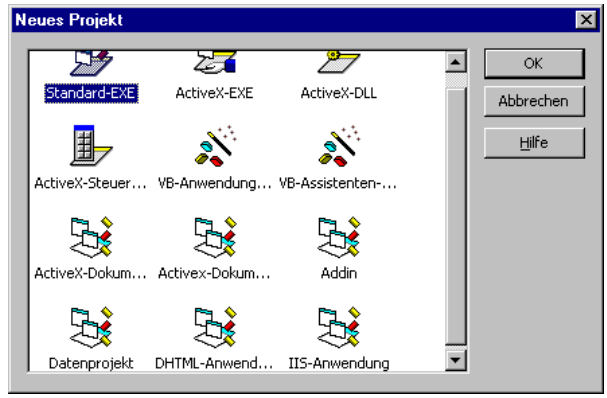

Bild 3.1: Projektauswahl in der Enterprise-Version

Die folgende Tabelle gibt einen Überblick über die restlichen Optionen:

- VB-ANWENDUNGSASSISTENT: Dieser Menüeintrag führt ebenfalls zu einem normalen Programm – allerdings ist Ihnen bei den ersten Schritten ein Assistent behilflich. Bei manchen Programmen können Sie sich damit einige Zeit für relativ monotone Arbeiten, etwa für den Menüentwurf, sparen.
- DATENPROJEKT: Das ist die dritte Variante, die zu einem gewöhnlichen Programm führt. Allerdings werden diesmal einige Datenbankfunktionen aktiviert (ADO-Bibliothek) und je ein *DataEnvironment*- und ein *DataReport*-Designer in das Projekt integriert (siehe Seite 727).
- ACTIVEX-EXE / -DLL: Dieser Eintrag leitet die Entwicklung eines Out-of-Process- bzw. In-Process-ActiveX-Server ein (kurz ActiveX-Server). Diesen Server können Sie anschließend in anderen Programmen via ActiveX Automation steuern. In Version 4 gab es das alles auch schon, allerdings sprach man damals von OLE-Servern und Object Automation (Seite 945).
- ACTIVEX-STEUERELEMENT: Damit beginnen Sie die Entwicklung eines neuen Steuerelements (Seite 991).

- ACTIVEX-DOKUMENT-EXE / -DLL: Wenn Sie Programme entwickeln möchten, die im Internet Explorer ausgeführt werden, sind Sie hier richtig (Seite 1027).
- DHTML-ANWENDUNG: Auch DHTML-Programme werden im Internet Explorer ausgeführt. Im Unterschied zu ActiveX-Dokumenten sieht die Benutzerschnittstelle aber eher wie ein Dokument aus (Seite 1059).
- IIS-ANWENDUNGEN: IIS-Anwendungen sind ActiveX-DLLs, die vom Internet Information Server ausgeführt werden (Seite 1127).
- ADD-IN / VB-ASSISTENTEN-MANAGER: Diese beiden Einträge wählen Sie, wenn Sie ein Add-In zur Visual-Basic-Entwicklungsumgebung – etwa einen Assistenten – entwickeln möchten. (Add-In-Programmierung wird in diesem Buch nicht behandelt.)

ANMERKUNG

Visual-Basic-intern gibt es nur vier unterschiedliche Projekttypen:

- STANDARD-EXE
- ACTIVEX-DLL (umfaßt auch ACTIVEX-DOKUMENT-DLL)
- ACTIVEX-EXE (umfaßt auch ACTIVEX-DOKUMENT-EXE und ADD-IN)
- ACTIVEX-STEUERELEMENT.

Die vielen Projekttypen in Bild 3.1 ergeben sich lediglich aus diversen Optionen, automatisch aktivierten Objektbibliotheken, automatisch inkludiertem Programmcode etc. Diese Einstellungen können Sie natürlich auch manuell vornehmen. Im Dialog PROJEKT|EIGENSCHAFTEN können Sie den Projekttyp auch nachträglich verändern.

Schablonen (Template-Verzeichnis)

Hinter dem gerade behandelten Projektauswahlkatalog verbirgt sich das `Template`-Verzeichnis. Dieses Verzeichnis enthält – inhaltlich gruppiert – Schablonen für neue Visual-Basic-Komponenten. Im Dialog NEUES PROJEKT wird (neben einigen vordefinierten Projekten) einfach der Inhalt des Verzeichnisses `Template/Projects` angezeigt. Die folgende Tabelle gibt einen Überblick über die `Template`-Verzeichnisse:

`Classes`	Schablonen für Klassenmodule (KLASSENMODUL HINZUFÜGEN)
`Code`	Schablenen für Module (MODUL HINZUFÜGEN)
`Controls`	Schablonen für Assistenten
`Forms`	Schablonen für Formulare (FORMULAR HINZUFÜGEN)
`MDIForms`	Schablonen für MDI-Formulare (MDI-FORMULAR HINZUFÜGEN)
`Menus`	Schablonen für Assistenten
`Projects`	Schablonen für Projekte (NEUES PROJEKT)
`PropPage`	Schablonen für Eigenschaftsdialoge zu Steuerelementen (EIGENSCHAFTSSEITE HINZUFÜGEN)

`UserCtls`	Schablonen für ActiveX-Steuerelemente (BENUTZERSTEUERELEMENT HINZUFÜGEN)
`UserDocs`	Schablonen für ActiveX-Dokumente (BENUTZERDOKUMENT HINZUFÜGEN)

Sie können problemlos eigene Schablonen zur Verfügung stellen, indem Sie Dateien aus vorhandenen Programmen in eines der Template-Verzeichnisse kopieren. Schablonen können dabei helfen, daß Formulare ein einheitliches Design haben, daß die Einstellung von Projekteigenschaften (etwa Copyright-Angaben) automatisch erfolgen etc. Immer wieder benötigte Codebausteine können im Modules-Verzeichnis abgelegt werden und ermöglichen so eine einfache Wiederverwendung einmal erstellten Codes. Wenn aus Schablonen erzeugte Komponenten gespeichert werden, muß ein neues Verzeichnis angegeben werden – es kann also nicht passieren, daß irrtümlich veränderte Komponenten im Template-Verzeichnis gespeichert werden.

Wenn die Visual-Basic-Entwicklung im Team oder innerhalb einer Firma erfolgt, kann es sinnvoll sein, allen Programmierern ein gemeinsames Template-Verzeichnis auf einem zentralen Server zugänglich zu machen. Das von der Entwicklungsumgebung verwendete Template-Verzeichnis kann dazu in EXTRAS | OPTIONEN | UMGEBUNG verändert werden.

Gleichzeitige Bearbeitung mehrerer Projekte (Projektgruppen)

Mit Visual Basic können Sie mehrere Projekte gleichzeitig bearbeiten. Das macht nicht nur die Entwicklung von ActiveX-Steuerelementen oder -Servern einfacher, sondern hilft auch beim Austausch von Programmcode oder Steuerelementen zwischen Programmen.

Zum Laden weiterer Projekte führen Sie statt DATEI | PROJEKT ÖFFNEN das Kommando - | PROJEKT HINZUFÜGEN aus. Alle geladenen Projekte werden im Projektfenster angezeigt. Wenn Sie im Projektfenster ein Projekt zusammenklappen (ausblenden), werden alle Fenster dieses Projekts geschlossen. Auf diese Weise gelingt es, selbst bei der gleichzeitigen Bearbeitung mehrerer Projekte einen Überblick über Dutzende Fenster zu bewahren.

Wenn mehrere Projekte gleichzeitig geladen sind, muß Visual Basic wissen, mit welchem Projekt es beim Programmstart beginnen soll. Dazu klicken Sie das Projekt im Fenster PROJEKTGRUPPE an und führen das Kontextmenükommando ALS STARTEINSTELLUNG FESTLEGEN aus. Dieses Projekt wird jetzt durch eine fette Schrift hervorgehoben. (Wenn Sie ActiveX-Komponenten entwickeln, muß das Programm gestartet werden, das die Komponente *einsetzt*. Das Projekt mit dem Code für die Komponente wird in der Folge ebenfalls gestartet – aber darum kümmert sich die Entwicklungsumgebung automatisch.)

Optionen

Wenn Sie nicht gerade mit einem 21-Zoll-Monitor gesegnet sind, werden Sie vor lauter verankerten Fenstern in der Visual-Basic-Entwicklungsumgebung kaum Platz für ein Formular- oder Codefenster finden. Sie können diesen Unfug abstellen, indem Sie die meisten (am besten alle) Kontrollkästchen in EXTRAS | OPTIONEN | VERANKERN deaktivieren. Anschließend können Sie alle Fenster ohne Einschränkungen verschieben und überlappend anordnen.

Standardgemäß startet die Entwicklungsumgebung jetzt als MDI-Anwendung (d.h., alle Visual-Basic-Fenster befinden sich innerhalb eines großen Hauptfensters). Unter Version 4 waren die Visual-Basic-Fenster dagegen alle eigenständig. Das hatte den Vorteil, daß das gerade ausgeführte Programm oft leichter zu finden war, weil es nicht hinter dem Hauptfenster verborgen war, während Sie Visual-Basic-Fenster zur Fehlersuche verwenden. Dieser SDI-Modus (Single Document Interface) wird weiterhin unterstützt und kann mit EXTRAS | OPTIONEN | WEITERE OPTIONEN aktiviert werden.

Symbolleisten

Visual Basic ist mit mehreren Symbolleisten ausgestattet, auch wenn normalerweise nur eine einzige angezeigt wird. Mit ANSICHT | SYMBOLLEISTE können Sie nicht nur die drei anderen Symbolleisten aktivieren, Sie können alle vorhandenen Symbolleisten, Menüs und Kontextmenüs bearbeiten sowie selbst Symbolleisten erstellen. Die Veränderung der Menüs und Symbolleisten erfolgt im wesentlichen durch *Drag and Drop* (wie in den Office-97-Programmen). Dabei werden Sie im BEFEHLE-Blatt des ANPASSEN-Dialogs auf zahlreiche Befehle stoßen, die im Visual-Basic-Standardmenü gar nicht enthalten sind! Einige Beispiele:

BEARBEITEN | BLOCK AUSKOMMENTIEREN

BEARBEITEN | AUSKOMMENTIEREN DES BLOCKS AUFHEBEN

BEARBEITEN | LESEZEICHEN SETZEN

BEARBEITEN | NÄCHSTES / VORIGES LESEZEICHEN

BEARBEITEN | LESEZEICHEN LÖSCHEN

Alle hier aufgezählten Kommandos sind nur im Codefenster von Interesse. Das Auskommentieren von Programmzeilen ist oft bei der Fehlersuche angenehm. Die Möglichkeit, im Programmcode Lesezeichen zu setzen, hilft besonders bei der Orientierung und Analyse umfangreicher Programme, die Sie nicht selbst geschrieben haben.

3.2 Gestaltung von Formularen

Das Design von Formularen ist in der Regel der erste Schritt bei der Entwicklung von Visual-Basic-Programmen. Formulare können von Ihrem Programm als Dialoge (starre Größe, der Anwender muß auf den Dialog reagieren) oder als Fenster im her-

kömmlichen Sinn (variable Größe, der Anwender kann in andere Fenster desselben Programms wechseln) eingesetzt werden.

Formularlayoutfenster

Mit dem Formularlayoutfenster können Sie die Startposition von Formularen bequem einstellen. (Wenn Sie möchten, können Sie natürlich auch die Formulareigenschaften *Left*, *Top*, *Width*, *Height* und *StartupPosition* manuell im Eigenschaftsfenster einstellen.)

Zusatzsteuerelemente

Die Toolbox (Werkzeugsammlung) stellt eine Palette der zur Zeit verfügbaren Steuerelemente dar. Wenn die Toolbar gerade nicht angezeigt wird, führen Sie das Kommando ANSICHT | WERKZEUGSAMMLUNG aus.

Wenn Sie während der Formularerstellung feststellen, daß Ihnen einzelne Zusatzsteuerelemente fehlen, klicken Sie die Toolbar mit der rechten Maustaste an und wählen den Kontextmenüeintrag KOMPONENTEN. In diesem Dialog werden wahlweise alle verfügbaren Zusatzsteuerelemente oder alle verfügbaren OLE-Objekte (etwa WordPad-Dokumente oder Excel-Tabellen) angezeigt. Durch Anklicken des jeweiligen Kontrollkästchens werden die Elemente in die Toolbar aufgenommen bzw. aus ihr entfernt. Beachten Sie, daß manche Einträge in der Liste (etwa die Data Bound List Controls) mit mehreren Steuerelementen verbunden sind.

Tip

Achten Sie darauf, daß die Toolbox keine Steuerelemente enthält, die Sie im Programm gar nicht benötigen. Dadurch verlängert sich die Zeit zum Laden des Projekts. Zudem glaubt der Installationsassistent, daß diese Steuerelemente tatsächlich benötigt werden und vergrößert damit unnötigerweise den Installationsumfang Ihres Programms.

Eine Neuerung der Toolbar besteht darin, daß Steuerelemente in mehreren Gruppen angeordnet werden können. Das ist vor allem dann praktisch, wenn Sie sehr viele Steuerelemente in einem Programm nutzen. Gruppen können ebenfalls über das Kontextmenü der Toolbar gebildet bzw. verändert werden. Steuerelemente werden immer in die gerade aktive Gruppe eingefügt. Die Standardsteuerelemente können nicht entfernt werden.

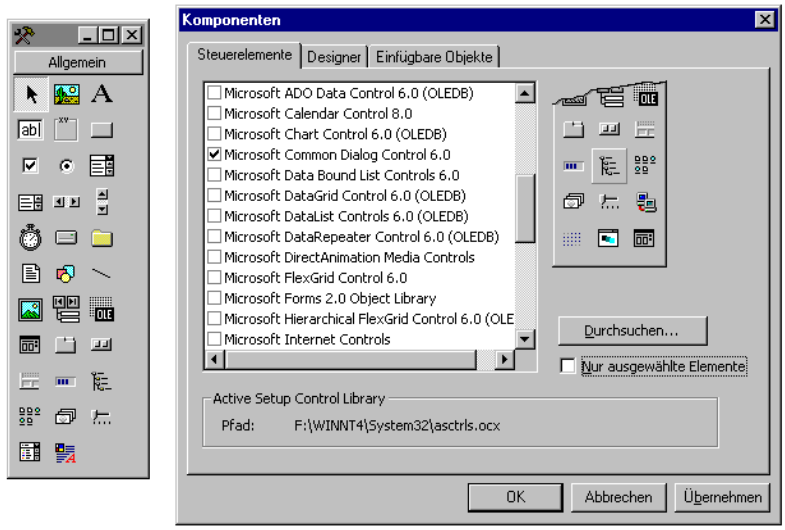

Bild 3.2: Die Toolbox mit dem Dialog zum Einfügen und Entfernen von Zusatzsteuerelementen

Steuerelemente benennen

Zur Identifikation von Steuerelementen im Programmcode ist die *Name*-Eigenschaft vorgesehen. Aber gerade diese Eigenschaft wird gerne vergessen oder gar mit der *Caption*-Eigenschaft verwechselt. (*Caption* bestimmt nur, welcher Text im Steuerelement angezeigt wird. *Caption* kann jederzeit vom Programm geändert werden. Viele Steuerelemente besitzten gar keine *Caption*-Eigenschaft, weil sie keinen Beschriftungstext anzeigen.)

Geben Sie den Steuerelementen möglichst aussagekräftige Namen. Der Name sollte sowohl etwas über die Art des Objekts als auch über seine Bedeutung für das Programm aussagen. Dabei ist es sinnvoll, im gesamten Programm eine einheitliche Konvention einzuhalten. Eine (von Microsoft favorisierte) Vorgehensweise besteht darin, die ersten drei oder vier Buchstaben dazu zu verwenden, den Typ der Steuerelemente auszudrücken. Typische Elementnamen sind dann *btnOK* (für den OK-Button), *btnAbbruch*, *lblInfo* (für ein Labelfeld), *txtName* (für ein Textfeld), *picIcon* (für ein Bild) etc.

Tip

Beachten Sie bitte, daß Sie nach der Veränderung einer *Name*-Eigenschaft bereits erstellten Code ebenfalls verändern müssen.

Steuerelementfelder

Visual Basic vergibt beim Einfügen von Steuerelementen automatisch Namen. Im Regelfall werden Sie diese Namen anschließend durch aussagekräftigere ersetzen (durch die Einstellung der *Name*-Eigenschaft im Eigenschaftsfenster).

Ein Sonderfall tritt auf, wenn zwei oder mehr Steuerelemente den gleichen Namen bekommen. Dieser Fall tritt ein, wenn Sie ein Steuerelement mehrfach über die Zwischenablage einfügen oder die *Name*-Eigenschaft eines Steuerelements so einstellen wie die eines schon vorhandenen Steuerelements. Visual Basic fragt dann, ob Sie ein Steuerelementfeld bilden möchten. Steuerelementfelder haben den Sinn, eine Gruppe gleichartiger Steuerelemente (beispielsweise vier Auswahlkästchen) durch eine gemeinsame Ereignisprozedur zu verwalten. Die Steuerelemente können noch immer durch die *Index*-Eigenschaft voneinander unterschieden werden. Steuerelementfelder sparen oft redundante und fehleranfällige Tipparbeit.

Steuerelemente in Container einfügen

Im Regelfall werden Steuerelemente direkt in das Formular eingefügt. Es gibt allerdings auch Steuerelemente, die als Container für andere Steuerelemente auftreten. Die beiden am häufigsten eingesetzten Container sind das Bild- und das Rahmenfeld. Container haben den Sinn, andere Steuerelemente optisch und logisch zu gruppieren. Besonders wichtig ist das, wenn mehrere Gruppen von Optionsfeldern gebildet werden sollen: dann sollte sich jede Gruppe in einem eigenen Container befinden (der durchaus auch unsichtbar sein darf).

Um ein neues Steuerelement nicht unmittelbar im Formular, sondern in ein Containerfeld einzufügen, muß das Containerfeld vor dem Einfügen mit der Maus markiert werden. Die so eingefügten Steuerelemente sind jetzt mit dem Containerfeld verbunden. Ein Verschieben des Containers verschiebt auch die darin enthaltenen Steuerelemente. Dasselbe gilt auch für das Löschen von Containerfeldern. Containerfelder können übrigens selbst wiederum in Containern plaziert werden. Insgesamt ist auf diese Weise eine maximal sechsstufige Hierarchie möglich. (Das Ausschöpfen dieser Grenze wird aber wohl kaum sinnvoll sein.)

Plazierung von Steuerelementen

Seit Version 5 stehen im FORMAT-Menü eine Menge Kommandos zur Verfügung, mit der Sie Gruppen von Steuerelementen ausrichten können, den Abstand zwischen diesen Steuerelementen gleichmäßig einstellen können etc. Bevor Sie sich aus langjähriger Visual-Basic-Gewohnheit damit plagen, mehrere Steuerelemente einzeln zu manipulieren, werfen Sie einen Blick auf dieses Menü und probieren Sie die Kommandos aus!

Steuerelemente können übrigens auch mit der Tastatur bewegt werden: Die vier Cursortasten zusammen mit Strg verschieben das gerade aktive Steuerelement um einen Rasterpunkt. (Der Rasterabstand kann im OPTIONEN-Dialog eingestellt werden.)

Eigenschaften

Bei den meisten Steuerelementen werden die Eigenschaften über das Eigenschaftsfenster (F4) eingestellt. Manche Zusatzsteuerelemente verfügen darüber hinaus über ein

eigenes Dialogfenster mit zusätzlichen Eigenschaftsseiten (Shift+F4). Zur raschen Auswahl des gerade aktiven Steuerelements können Sie nicht nur die Maus verwenden, sondern auch Tab oder Shift+Tab.

Im Eigenschaftsfenster kann das gerade aktuelle Steuerelement aus der Liste des Elementfelds ausgewählt werden. In den meisten Fällen ist es aber bequemer, das jeweilige Steuerelement einfach im Formularfenster anzuklicken. Beachten Sie, daß auch Formulare, Module etc. Eigenschaften haben, die ebenfalls im Eigenschaftsfenster eingestellt werden. Zur Auswahl des Formulars klicken Sie einfach an eine beliebige Stelle im Formular, die nicht durch ein Steuerelement bedeckt ist.

Zur effizienten Bedienung des Eigenschaftsfensters existieren einige wichtige Tastaturkommandos: Mit Strg+Shift+Buchstabe können Sie sehr rasch die gewünschte Eigenschaft auswählen, also z.B. Strg+Shift+C für *Caption*. Mit Shift+Tab bewegen Sie den Eingabecursor in das Objektfeld und können dort mit den Cursortasten oder durch die Eingabe des Anfangsbuchstabens das gewünschte Steuerelement auswählen.

Die Eigenschaften mehrerer Steuerelemente gleichzeitig einstellen

Im Formular ist die gleichzeitige Auswahl mehrerer Steuerelemente möglich. Dazu können Sie über die gewünschten Elemente mit der Maus einen Rahmen ziehen oder die Elemente der Reihe nach bei gedrückter Shift- oder Strg-Taste mit der Maus anklikken. Im Eigenschaftsfenster werden jetzt nur noch jene Eigenschaften angezeigt, die in allen Elementen gemeinsam verfügbar sind. Die Veränderung der Eigenschaften wirkt sich dann auf alle markierten Steuerelemente gleichzeitig aus und ermöglicht so eine sehr rationelle Formulargestaltung.

Farben

Zur Einstellung der Vorder- und Hintergrundfarben von Steuerelementen steht ein Farbpalettenfenster zur Verfügung. Zur kleinen Variante dieses Fensters kommen Sie durch einen Doppelklick auf die *BackColor*- oder *ForeColor*-Eigenschaft im Eigenschaftsfenster.

Wenn Sie die Farben mehrerer Steuerelemente ändern wollen, ist es praktischer, mit ANSICHT|FARBPALETTE ein stationäres Farbpalettenfenster zu aktivieren. Im Farbpalettenfenster werden rechts ein größeres und darin ein kleineres Quadrat angeklickt. Wenn Sie das größere Quadrat anklicken, betreffen Farbveränderungen die Hintergrundfarbe. Das kleinere Quadrat steht für die Vordergrundfarbe.

In der Defaulteinstellung verwenden die Steuerelemente Windows-Systemfarben. In der Farbpalette sind dagegen nur RGB-Farben vorgesehen. Sobald Sie eine Farbe durch die Farbpalette einstellen, ist diese Einstellung starr, also unabhängig von den Windows-Systemfarben. Das ist aber nicht immer beabsichtigt und kann auf einem Rechner, auf dem ein anderes Farbschema eingestellt ist, zu unerwarteten und oft unerwünschten Farbeffekten führen. Wenn Sie eine Farbe mit einer der Systemfarben

einstellen möchten, müssen Sie den entsprechenden hexadezimalen Wert per Tastatur eingeben. Eine Liste mit den hexadezimalen Codes der Farben finden Sie in der Online-Hilfe (Thema *Color-Konstanten*).

ANMERKUNG

Unter Visual Basic gibt es zwei Möglichkeiten, Farben zu codieren: Die eine ist der allseits bekannte RGB-Wert: Für die Farbanteile Rot, Grün und Blau kann ein Wert zwischen 0 und 255 angegeben werden. Das Ergebnis ist eine hexadezimale Zahl, deren beide erste Stellen 0 sind, also etwa *&H00FF0000* für ein intensives Rot. (Dieser hexadezimale Wert wird auch im Eigenschaftsfenster angezeigt.) Daneben existieren Windows-Systemfarben, deren Codierung mit *&H80000000* beginnt. Windows-Systemfarben können im Windows-System eingestellt werden (klicken Sie mit der rechten Maustaste auf den Windows-Desktop) und gelten für alle Windows-Programme. *&H8000000F* ist beispielsweise die Hintergrundfarbe für alle 3D-Objekte (Buttons, Dialoge etc.). Eine Auswahl aller Systemfarben gibt das Eigenschaftsfenster bei der Veränderung von Farbeinstellungen.

Zeichenreihenfolge

Steuerelemente sind generell gleichberechtigt. Allerdings wird bei der Darstellung unterschieden, in welcher Reihenfolge die Steuerelemente eingefügt wurden: Wenn sich Steuerelemente überlappen, dann überdecken die zuletzt eingefügten Steuerelemente die schon früher eingefügten. Die Zeichenreihenfolge von Steuerelementen kann über FORMAT | REIHENFOLGE | IN DEN VORDER- / HINTERGRUND nachträglich verändert werden. Die beiden Kommandos stehen auch im Kontextmenü zur Verfügung, das beim Anklicken eines Steuerelements mit der rechten Maustaste erscheint.

Tabulatorreihenfolge

Der Eingabecursor in Formularen kann sowohl beim Formularentwurf als auch im laufenden Programm mit Tab von einem Element zum nächsten bewegt werden. Da beim Entwurf von Formularen selten exakt die Reihenfolge eingehalten wird, die für Tab gelten soll, müssen Sie die *TabIndex*-Werte am Ende des Entwurfs zumeist manuell im Einstellungsfeld des Hauptfensters ändern. Visual Basic paßt nach der Änderung eines *TabIndex*-Werts automatisch alle anderen Werte an, d.h., es kann nicht vorkommen, daß mehrere Steuerelemente denselben *TabIndex* aufweisen.

Bei zusammengehörenden Kontrollkästchen ist es oft nicht sinnvoll, diese Kontrollkästchen alle mit Tab zu durchlaufen. Wenn Sie möchten, daß einmal Tab den Eingabecursor zum ersten Kontrollkästchen bewegt und nochmals Tab den Cursor zum ersten Steuerelement nach den Kontrollkästchen, dann müssen Sie bei allen Kontrollkästchen außer beim ersten die Eigenschaft *TabStop* auf *False* setzen.

Tastenkürzel

Bekanntlich können Buttons in Formularen mit Alt+Buchstabe ausgewählt werden, wenn in den Buttons ein unterstrichener Buchstabe steht. Dazu müssen Sie im *Caption*-Text vor den jeweiligen Buchstaben ein &-Zeichen stellen. Dieses Zeichen ist auch in Labelfeldern und Rahmen erlaubt. Bei Labelfeldern wird der Eingabecursor dann in ein veränderliches Steuerelement mit dem nächstgrößeren *TabIndex*-Wert bewegt, bei Rahmen wird der Eingabecursor in das erste Steuerelement im Rahmen gestellt.

Steuerelemente kopieren und einfügen

Steuerelemente können ebenso wie Textausschnitte in die Zwischenablage kopiert und anschließend wieder in das Formular eingefügt werden. Dadurch können Sie mit wenig Aufwand mehrere gleichartige Steuerelemente erzeugen: Sie stellen zuerst alle Eigenschaften eines Steuerelements ein, kopieren es dann mit Strg+Einfg (Menükommando BEARBEITEN | KOPIEREN) in die Zwischenablage und fügen es mit Shift+Einfg (Menükommando BEARBEITEN | EINFÜGEN) so oft in Ihr Formular ein, wie Sie das Element benötigen. Das Kopieren und Einfügen funktioniert auch für ganze Gruppen markierter Steuerelemente und zwischen unterschiedlichen Formularen.

Beim Einfügen eines zweiten gleichartigen Steuerelements meldet sich Visual Basic mit der Frage, ob Sie ein Steuerelementfeld bilden möchten. Wenn Sie die Frage mit JA beantworten, vergibt Visual Basic automatisch ansteigende *Index*-Werte und beläßt die *Name*-Eigenschaft unverändert. Wenn Sie dagegen NEIN wählen, gibt Visual Basic dem neuen Steuerelement automatisch eine andere *Name*-Bezeichnung (z.B. *Text2* statt *Text1*).

Steuerelemente sperren

Wenn Sie einzelne Steuerelemente vor ungewollten Veränderungen schützen möchten, können Sie die Steuerelemente sperren. Dazu markieren Sie die betroffenen Elemente mit der Maus und führen BEARBEITEN | STEUERELEMENTE SPERREN aus. Ein nochmaliges Ausführen des Kommandos deaktiviert den Schutz. Wenn Sie nicht einzelne Steuerelemente, sondern das ganze Formular auswählen, gilt der Schutz für alle Steuerelemente des Formulars.

3.3 Codeeingabe

Nach der Gestaltung der Formulare ist die Erstellung des Programmcodes der nächste Schritt zu einem funktionierenden Visual-Basic-Programm. Der Programmcode wird in eigenen Codefenstern angezeigt (je Datei eines). Um ein einfaches Kopieren von Programmteilen aus einem Unterprogramm in ein anderes zu erlauben, kann das Programmcodefenster in zwei Bereiche geteilt werden.

Definition neuer Prozeduren

Wenn Sie eine neue Prozedur schreiben möchten, bestehen dazu mehrere Möglichkeiten. Am einfachsten geht es bei den Ereignisprozeduren: Durch einen Doppelklick auf das Formular bzw. auf das Steuerelement wird das Codefenster automatisch geöffnet. Der Cursor wird in die Codeschablone mit der für das jeweilige Element wichtigsten Ereignisprozedur gestellt. Wenn Sie den Code für ein anderes Ereignis schreiben möchten, wählen Sie das betreffende Ereignis einfach im rechten Listenfeld des Codefensters aus.

Bei allgemeinen Prozeduren (Unterprogramme, Funktionen) müssen Sie zuerst das Codefenster öffnen, indem Sie die Prozedur programmieren möchten. Anschließend können Sie mit EXTRAS | PROZEDUR HINZUFÜGEN eine Schablone für eine neue Prozedur per Mausklick erstellen. (Die Bedeutung der Schlüsselwörter *Sub*, *Function*, *Property*, *Public*, *Privat* und *Static* wird ab Seite 121 genau beschrieben.)

Wenn Sie ein wenig Übung und Erfahrung mit Visual Basic haben, werden Sie die Definition einer neuen Prozedur noch schneller erledigen, indem Sie die Anweisungen *Function Name* oder *Sub Name* im Codefenster eingeben. Visual Basic vervollständigt die Prozedurdefinition automatisch durch *End Function* oder *End Sub*.

Prozedurattribute

Im Dialog EXTRAS | PROZEDURATTRIBUTE können Sie zusätzliche Attribute zu Prozeduren einstellen. Dazu zählen eine Kurzbeschreibung der Prozedur, Verweise auf Hilfedateien sowie diverse Attribute, die dann von Interesse sind, wenn Sie Eigenschaftsprozeduren für Klassenmodule bzw. für ActiveX-Komponenten programmieren.

So können Sie eine Eigenschaftsprozedur als Defaultprozedur deklarieren: PROZEDUR-ID = VOREINSTELLUNG. Etwas komplizierter ist die Definition von Aufzähleigenschaften: PROZEDUR-ID = -4 (wie intuitiv!), Option IM EIGENSCHAFTSFENSTER NICHT ANZEIGEN aktivieren (gemeint ist hier der Objektkatalog). Mehr Informationen zur Programmierung von Eigenschaftsprozeduren finden Sie ab Seite 170.

Der Dialog kann als Musterbeispiel dafür gelten, wie man eigene Dialoge möglichst nicht gestalten sollte: unübersichtlich, zweideutig beschriftet, nur mit hellseherischen Fähigkeiten zu bedienen ... Warum die Einstellungen nicht überhaupt im Eigenschaftsfenster durchgeführt werden können (was mit der restlichen Entwicklungsumgebung konsistent gewesen wäre), bleibt auch rätselhaft.

Cursorbewegung im Programmcodefenster

Der Textcursor kann innerhalb eines Unterprogramms bzw. einer Funktion wie gewohnt mit den Cursortasten bewegt werden. Bild ↑ bzw. Bild ↓ bewegen den Textcursor seitenweise durch eine Prozedur. Wenn der Cursor bereits am Anfang bzw. am Ende des Unterprogramms steht, wird die vorangegangene bzw. nächste Prozedur

angezeigt. (Die Reihenfolge der Prozeduren orientiert sich an der Reihenfolge, in der die Prozeduren definiert wurden.)

Strg+↑ und Strg+Bild ↑ bzw. Strg+↓ und Strg+Bild ↓ zeigen unabhängig von der aktuellen Position des Textcursors in jedem Fall das vorige bzw. das nächste Unterprogramm an. F6 wechselt den aktiven Ausschnitt, wenn das Fenster geteilt ist.

Shift+F2 bewegt den Cursor zum Code der Prozedur, auf dessen Name der Cursor gerade steht (Kommando ANSICHT | PROZEDURDEFINITION). Wenn die betroffene Prozedur in einer anderen Datei des Projekts definiert ist, wechselt Visual Basic automatisch in das betreffende Codefenster. Strg+Shift+F2 springt zurück zur vorherigen Position. Visual Basic verwaltet dazu einen mehrstufigen Puffer für die Rücksprungpositionen.

Zum raschen Springen zu einem anderen Programmteil können Sie schließlich auch den Objektkatalog verwenden, in dem (unter anderem) sämtliche von Ihnen programmierte Prozeduren verzeichnet sind – siehe den folgenden Abschnitt.

Blöcke ein- und ausrücken

Damit der Programmcode leichter zu lesen ist, werden Blöcke innerhalb von Verzweigungen und Schleifen normalerweise eingerückt (wie in allen Programmlistings dieses Buchs). Die Einrückungen erfolgen nicht automatisch, sondern müssen durch die Eingabe von Leer- oder Tabulatorzeichen vorgenommen werden. Wenn Sie später die Struktur des Programms ändern (z.B. durch eine zusätzliche Sicherheitsabfrage), müssen Sie oft zahlreiche Zeilen ein- oder ausrücken. Anstatt das für jede Zeile manuell zu erledigen, können Sie sich von Visual Basic helfen lassen: Markieren Sie den gesamten Zeilenblock mit der Maus, und geben Sie dann Tab bzw. Shift+Tab ein. Visual Basic rückt den gesamten Block um eine Tabulatorposition ein oder aus.

Die Tabulatorweite kann im Optionenfenster (EXTRAS | OPTIONEN | EDITOR) beliebig eingestellt werden – sogar auf ein einziges Zeichen. Die Defaulteinstellung lautet vier Zeichen, in diesem Buch wurden aber nur zwei Zeichen verwendet, was zu einem weniger stark auseinandergezogenen Programmcode führt. (Visual Basic arbeitet übrigens nicht mit echten Tabulatoren. Die Tabulatorweite gibt nur an, wie viele Leerzeichen durch Tab eingefügt werden.)

Änderungen rückgängig machen

Wenn Sie versehentlich einen markierten Bereich löschen oder eine Änderung am Programmcode rückgängig machen möchten, können Sie den bisherigen Zustand des Programms mit dem Kommando BEARBEITEN | RÜCKGÄNGIG bzw. mit Alt+Backspace wiederherstellen. Mit BEARBEITEN | WIEDERHERSTELLEN bzw. mit Strg+Backspace können Sie auch das Rückgängig-Kommando wieder rückgängig machen. Diese Undo- und Redo-Funktion arbeitet mehrstufig, d.h., Sie können mehrere Änderungen zurücknehmen.

Variablendeklaration

Wenn Sie am Beginn des Programmcodes eines Moduls die Anweisung *Option Explicit* angeben, dann müssen Sie alle Variablen mit *Dim* deklarieren, bevor Sie sie zum ersten Mal verwenden. Dieser Schutzmechanismus bewahrt Sie vor Tippfehlern: Wenn Sie bei der Eingabe einer Variablen zwei Buchstaben vertauschen, einen Buchstaben vergessen etc., meldet sich Visual Basic mit der Fehlermeldung, daß die falsch geschriebene Variable nicht deklariert ist. Wenn im Dialogblatt EXTRAS | OPTIONEN | EDITOR die Option VARIABLENDEKLARATION ERFORDERLICH aktiviert ist, dann fügt Visual Basic die Anweisung *Option Explicit* bei allen neuen Modulen automatisch ein.

Die Option hat keinen Einfluß auf vorhandene Programme, d.h., sie wirkt nur auf neue Formulare oder Module. Sie können aber selbstverständlich die Anweisung *Option Explizit* nachträglich im Deklarationsabschnitt eines Moduls eintragen bzw. löschen.

Syntaxkontrolle

Visual Basic überprüft jede Programmzeile sofort auf syntaktische Fehler. Fehlerhafte Zeilen werden dabei rot markiert. Gleichzeitig erscheint ein Meldungsdialog, der auf die Ursache des Fehlers hinweist. Die ständige Benachrichtigung über Fehlerquellen ist allerdings ausgesprochen lästig, wenn Sie während der Eingabe einer Programmzeile das Fenster wechseln (beispielsweise um in einem anderen Programmcodefenster eine Definition nachzusehen).

Über die Option AUTOMATISCHE SYNTAXKONTROLLE im Dialogblatt EXTRAS | OPTIONEN | EDITOR können Sie die automatische Benachrichtigung abstellen. (Fehlerhafte Zeilen werden weiterhin rot markiert, was nach einer kurzen Gewöhnung an Visual Basic vollkommen ausreicht. Zudem meldet Visual Basic eventuell noch vorhandene Fehler beim Programmstart.)

3.4 Objektkatalog

Der Objektkatalog ist eine zentrale Referenz aller zur Zeit verfügbaren Funktionen, Methoden, Eigenschaften, Prozeduren etc. Die Referenz umfaßt Visual-Basic-Grundfunktionen ebenso wie Erweiterungen durch Zusatzsteuerelemente, durch Objektbibliotheken und selbst programmierte Funktionen. Soweit die Information verfügbar ist, wird im unteren Bereich des Objektfensters eine kurze Beschreibung des Schlüsselworts angegeben. Das Anklicken des ?-Buttons zeigt den Hilfetext zu diesem Schlüsselwort an. (Das funktioniert auch dann, wenn sich der Hilfetext in einer nicht zu Visual Basic gehörenden Hilfedatei befindet! Objektbibliotheken speichern ihre eigene Referenz auf Hilfedateien.)

Der Objektkatalog kann wahlweise über das Menükommando ANSICHT | OBJEKTKATALOG, mit F2 oder durch Anklicken des Objektkatalogsymbols aufgerufen werden.

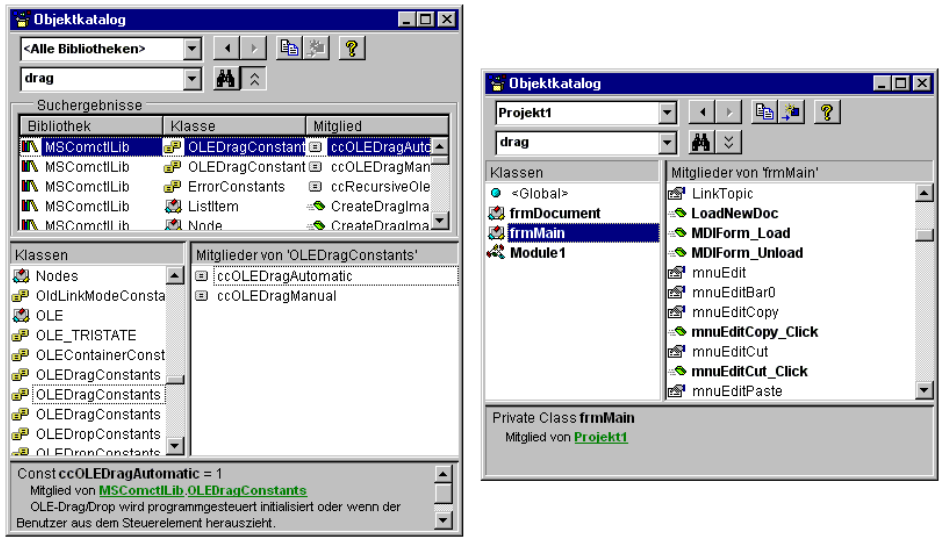

Bild 3.3: Zweimal der Objektkatalog: links mit den Ergebnissen einer Suche nach Drag, rechts mit Prozeduren und eigenen Datentypen des aktuellen Projekts

Im Objektkatalog werden folgende Daten angezeigt:

- Objekte mit deren Eigenschaften und Methoden
- globale Funktionen und Kommandos von Visual Basic
- vordefinierte Konstanten
- Formulare, Module, Prozeduren und Datentypen des aktuellen Projekts (in fetter Schrift)

Generell werden alle Einträge alphabetisch geordnet. Mit dem Kontextmenükommando ELEMENTE GRUPPIEREN erreichen Sie, daß Objekte, Eigenschaften, Ereignisse etc. jeweils in Gruppen zusammengefaßt werden. Das ist insbesondere dann hilfreich, wenn Sie etwa einen raschen Überblick über alle Ereignisse eines Objekts gewinnen möchten.

Im Objektkatalog werden zwar viele, aber nicht alle Objekte, Eigenschaften, Methoden etc. angezeigt. Bewußt ausgeblendet sind diverse Elemente, die bei der Programmierung nicht verwendet werden sollen. Dazu zählen diverse interne Schlüsselwörter sowie Elemente, die nur noch aus Kompatibilitätsgründen zur Verfügung stehen. Sie können diese verborgenen Schlüsselwörter in grauer Schrift anzeigen lassen, wenn Sie das Kontextmenükommando VERBORGENE ELEMENTE ANZEIGEN ausführen.

Das im Objektkatalog gerade ausgewählte Objekt kann mit Strg+C bzw. mit Strg+Einfg in die Zwischenablage kopiert und anschließend bequem in den Programmcode eingefügt werden.

Bei Objekten, die im Programmcode des aktuellen Projekts definiert sind, gelangen Sie mit einem Doppelklick im Objektkatalog an die entsprechende Stelle im Programmcode.

3.5 Programmausführung, Kompilierung

3.5.1 Programmausführung in der Entwicklungsumgebung

Die Programmausführung wird am bequemsten mit F5 gestartet. Wenn ein Programm aus mehreren Formularen besteht, fragt Visual Basic vor dem ersten Start, welches Formular nach dem Programmstart automatisch angezeigt werden kann (PROJEKT | EIGENSCHAFTEN). Alternativ dazu besteht die Möglichkeit, die Programmausführung mit der Prozedur *Main* zu beginnen. Der Programmcode für *Main* muß dazu in einem Modul angegeben werden.

VERWEIS

Die Fehlersuche in Visual-Basic-Programmen, also das Setzen von Haltepunkten, die Einzelschrittausführung, Überwachungsausdrücke etc., werden in Kapitel 8 ab Seite 365 beschrieben.

Bevor Visual Basic ein Programm ausführt, wird dessen Code in P-Code umgewandelt. (P-Code ist eine Zwischenform zwischen wirklich kompiliertem Code und purem ASCII-Text, der interpretiert wird. `*.exe`-Dateien früherer Visual-Basic-Versionen basierten auf P-Code.)

Standardgemäß erfolgt diese Kompilierung nur nach Bedarf für die gerade benötigten Prozeduren. Der Vorteil besteht darin, daß die Programmausführung praktisch verzögerungslos beginnt. Das ist natürlich auch mit einem Nachteil verbunden: Visual Basic führt vor dem Programmstart keine ordentliche Syntaxkontrolle durch. Eine Menge Fehler, die beim Kompilieren entdeckt würden, treten so erst irgendwann während der Programmausführung auf (oder gar nicht, wenn die betroffenen Programmteile bei einem Testlauf nicht ausgeführt werden).

Aus diesem Grund ist es zur Fehlersuche oft sinnvoller, eine vollständige Kompilierung vor dem Programmstart zu erzwingen. Dazu deaktivieren Sie in EXTRAS | OPTIONEN | ALLGEMEIN | KOMPILIEREN das Kontrollkästchen BEI BEDARF.

Befehlszeile

An Visual-Basic-Programme kann eine Kommandozeile (Befehlszeile) übergeben werden. Um die Auswertung einer Befehlszeile auch in der Entwicklungsumgebung zu testen, können Sie in PROJEKT | EIGENSCHAFTEN | ERSTELLEN eine entsprechende Zeichenkette angeben. Im Visual-Basic-Programm erfolgt die Auswertung der Kommandozeile dann mit der Funktion *Command*.

Automatisch speichern

Visual Basic sieht zwar keine Möglichkeit vor, Ihr Projekt regelmäßig (etwa alle 10 Minuten) zu speichern, aber immerhin können Sie via EXTRAS | OPTIONEN | UMGEBUNG angeben, ob Sie (mit oder ohne Rückfrage) das Projekt vor jedem Ausführen speichern möchten. Vor allem wenn Sie potentiell absturzgefährdete Programmteile entwickeln (Einbindung von DLL-Funktionen etc.) ist es empfehlenswert, die Optionen ÄNDERUNGEN SPEICHERN zu setzen.

3.5.2 Kompilierung zu *.exe- oder *.dll-Dateien

Wenn die Entwicklung eines Programms abgeschlossen ist, kompilieren Sie Ihr Programm oder Steuerelement zu einer eigenständigen Datei, die je nach Programmtyp die Kennungen `*.exe`, `*.dll` oder `*.ocx` aufweist. Sofern Sie mit der Professional- oder Enterprise-Version arbeiten, stehen zwei Varianten zur Auswahl: Sie können P-Code oder echten Binärcode erstellen. Da Binärcode meist deutlich schneller ist und die resultierenden Dateien dabei nicht wesentlich größer werden, besteht kein vernünftiger Grund, diese neue Möglichkeit nicht zu nutzen. Dazu müssen Sie lediglich im Kompilierungsdialog (oder schon vorher in PROJEKT | EIGENSCHAFTEN | KOMPILIEREN) die Option NATIVE CODE auswählen.

Schon interessanter sind die zahlreichen Compiler-Optionen: Damit können Sie Einfluß auf den Code nehmen. Allerdings müssen Sie dabei zumeist einen Kompromiß eingehen: Kompakter Code oder schneller Code? Sicherer Code oder schneller Code? Dazu einige Informationen:

Rücksicht auf die Größe des Kompilats brauchen Sie eigentlich nur dann zu nehmen, wenn Sie Internet-Komponenten entwickeln. Bei allen anderen Anwendungen ist die (ohnedies nicht sehr große) Platzersparnis kein wirkliches Kriterium.

Die Optimierung für Pentium-Pro-Prozessoren bedeutet, daß einige Besonderheiten dieses Prozessors bezüglich der Parallelverarbeitung von CPU-Kommandos berücksichtigt werden. Der Code kann zwar weiterhin auf allen Intel-CPUs (bis hinab zum 386er) ausgeführt werden, allerdings dann etwas langsamer als ohne Pentium-Pro-Optimierung.

Wenn Sie die Option TESTINFORMATIONEN FÜR SYMBOLISCHE DEBUGGER wählen, wird der Code signifikant größer. Dafür können Sie selbst im Kompilat noch nach Fehlern su-

chen. Voraussetzung ist allerdings, daß Sie einen symbolischen Debugger besitzen, wie er etwa mit Visual C++ mitgeliefert wird.

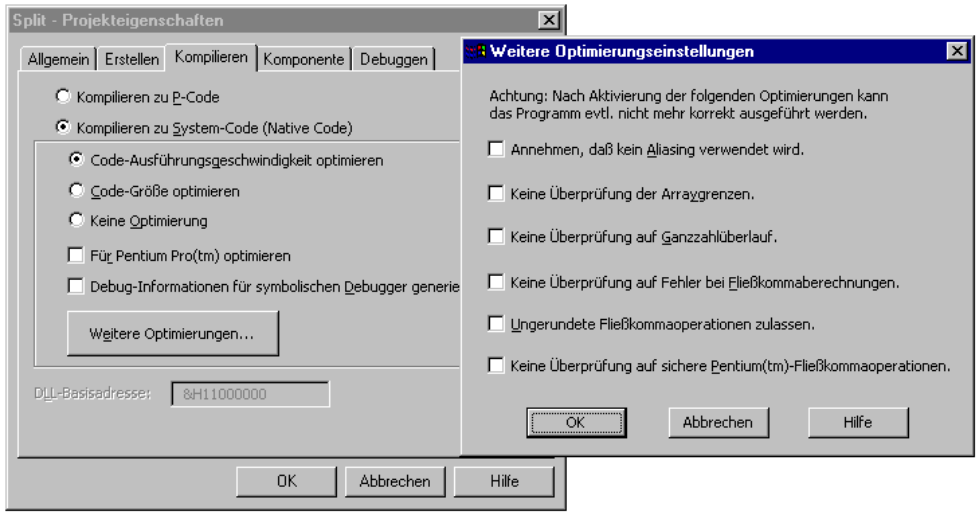

Bild 3.4: Optionen des Compilers

Die weiteren Optionen betreffen diverse automatische Tests, die Visual Basic üblicherweise durchführt: Tests etwa auf Überlauffehler, auf den Zugriff nicht-existenter Feldelemente etc. Wenn Sie wirklich sicher sind, daß Ihr Programm fehlerfrei ist, können Sie diese Kontrollen abschalten. Wenn es dann doch zu einem Fehler kommt, ist die Reaktion des Programms allerdings unklar. Im ungünstigsten Fall stürzt das Programm einfach ab. Wenn Sie die Fehlerüberprüfung dagegen aktiviert lassen, tritt ein definierter Fehler auf, den Sie mit *On Error* abfangen können. Auf diese Weise können Sie dem Anwender beispielsweise noch die Möglichkeit geben, seine Daten zu speichern. (Wie oft hätte ich mir das bei WinWord 97 auch gewünscht!)

Wieviel Geschwindigkeitsgewinn bzw. Platzersparnis die jeweiligen Optimierungen tatsächlich bringen, können Sie nur durch Ausprobieren feststellen. Bei den meisten typischen Visual-Basic-Anwendungen sind die Unterschiede so gering, daß sich ein Experimentieren mit den Optionen kaum lohnt. Wenn Sie wirklich sehr rechenintensive Algorithmen verwenden, kann der Unterschied spürbar werden – aber dann stellt sich ohnedies die Frage, ob es nicht eine bessere Programmiersprache als Visual Basic gäbe.

3.5.3 Bedingte Kompilierung

Manchmal kommt es vor, daß Sie parallel zu einem Programm eine zweite Version verwalten möchten (etwa eine Demoversion mit eingeschränkten Merkmalen oder eine Debugversion mit zusätzlichen Sicherheitsabfragen). Dazu können Sie in PRO-

JEKT | EIGENSCHAFTEN | ERSTELLEN im Textfeld ARGUMENTE FÜR BEDINGTE KOMPILIERUNG eine Konstante definieren, beispielsweise *demo=1*. Im Programmcode können Sie den Inhalt der Konstanten dann mit *#If*-Anweisungen auswerten.

Je nach Ergebnis der *#If*-Abfrage wird entweder der eine oder andere Zweig ausgeführt. Im Unterschied zu normalen *If*-Abfragen erfolgt die Unterscheidung zwischen den beiden Varianten allerdings schon bei der Kompilierung. Das Kompilat enthält nur eine Variante und keine *#If*-Abfragen, es ergibt sich also kein Geschwindigkeitsnachteil. Die folgenden Zeilen zeigen, wie Programmcode mit *#If*-Anweisungen aussehen kann:

```
Sub Command1_Click()
  #If demo Then
    MsgBox "In der Demoversion kann nichts gespeichert werden"
  #Else
    ' ... Programmcode zum Speichern
  #End If
End Sub
```

3.6 Designer

Einfache VB-Projekte bestehen aus Dateien, die direkt durch die VB-Entwicklungsumgebung erstellt werden können. Seit Version 5 besteht darüberhinaus die Möglichkeit, sogenannte Designer-Komponenten zu verwenden (Dateityp `*.dsr`). Dabei handelt es sich um Dateien, die zwar nahtlos in das Projekt integriert werden, zu deren Manipulation aber externe Zusatzprogramme – eben ActiveX-Designer – eingesetzt werden. Die wichtigsten Designer-Komponenten in Visual Basic 6 sind:

- *UserForm*: stellen eine Alternative zu den Standardformularen dar und ermöglichen die Verwendung von MS-Forms-Steuerelementen (Seite 234)
- *DataEnvironment*: stellen die Verbindung zwischen einem VB-Programm und einer ADO-Datenquelle her (Seite 806)
- *DataReport*: ermöglichen die Anzeige und den Ausdruck von Datenbankberichten (Seite 862)
- *DHTMLPage*: helfen dabei, ein HTML-Dokument mit Client-seitigem Code zu verbinden (Seite 1059)
- *WebClass*: helfen dabei, ein HTML-Dokument mit Server-seitigem Code zu verbinden (Seite 1127)

Designer-Komponenten werden mit PROJEKT | XY HINZUFÜGEN bzw. mit PROJEKT | WEITERE ACTIVEX-DESIGNER in ein Projekt eingefügt. Unter Umständen muß der Designer vorher durch PROJEKT | KOMPONENTEN | DESIGNER aktiviert werden.

Das gemeinsame Merkmal aller Designer-Komponenten besteht darin, daß die in der `*.dsr`-Datei gespeicherten Informationen visuell durch den Designer bearbeitet werden (und nicht durch die VB-Entwicklungsumgebung). Aus diesem Grund sind auch die Standardmenüs nicht aktiv, solange eine Designer-Datei bearbeitet wird.

Jede Designer-Komponente kann auch mit Programmcode verbunden werden. Der Code wird ebenfalls in der `*.dsr`-Datei gespeichert. Die Code-Eingabe erfolgt allerdings nicht durch den Designer, sondern durch die VB-Entwicklungsumgebung.

Keine Gemeinsamkeiten gibt es hingegen darin, ob und wie Designer-Komponenten im fertigen Programm sichtbar werden. Beispielsweise stellt die *DataEnvironment*-Komponente lediglich einige unsichtbare Objekte zur Verfügung. Hier besteht der Sinn des Designers nur darin, bei der Einstellung der Eigenschaften dieser Objekte zu helfen. Im Gegensatz dazu wird eine *DataReport*-Komponente wie ein gewöhnliches Formular in einem eigenen Fenster angezeigt.

HINWEIS

Nicht alle Designer stammen unmittelbar von Visual Basic oder können nur darin verwendet werden. So ist der MS-Forms-Designer ein Bestandteil des Office-Pakets und dient dort zur Gestaltung von Dialogen. Es besteht momentan keine Möglichkeit, selbst mit Visual Basic neue Designer zu programmieren.

3.7 Assistenten

Seit Visual Basic 4 besteht die Möglichkeit, Add-Ins (also Erweiterungen zur Entwicklungsumgebung) unter Visual Basic selbst zu programmieren. Seither stehen jede Menge Assistenten zur Verfügung, mit denen sich wiederholende Schritte bei der Entwicklung neuer Programme, beim Entwurf von Formularen zur Anzeige von Daten aus einer Datenbank, bei der Umwandlung vorhandener Programme in ActiveX-Dokumente etc. vermieden werden können bzw. sich effizienter gestalten lassen.

Es bestehen zwei Möglichkeiten, Assistenten aufzurufen: Zum einen können Sie beim Beginn eines neuen Projekts oder beim Einfügen von Formularen / Modulen / Klassen etc. einen Assistenten starten, anstatt einfach nur eine leere Schablone des jeweiligen Objekts einzufügen. Zum anderen können Sie die Menüeinträge des ADD-IN-Menüs verwenden. Vorher müssen Sie den jeweiligen Assistenten allerdings im ADD-IN-Manager aktivieren. Wenn Sie einen Assistenten häufig benötigen, sollten Sie dort die Option BEIM START LADEN anklicken – der Assistent wird dann bei jedem Start der Entwicklungsumgebung aktiviert.

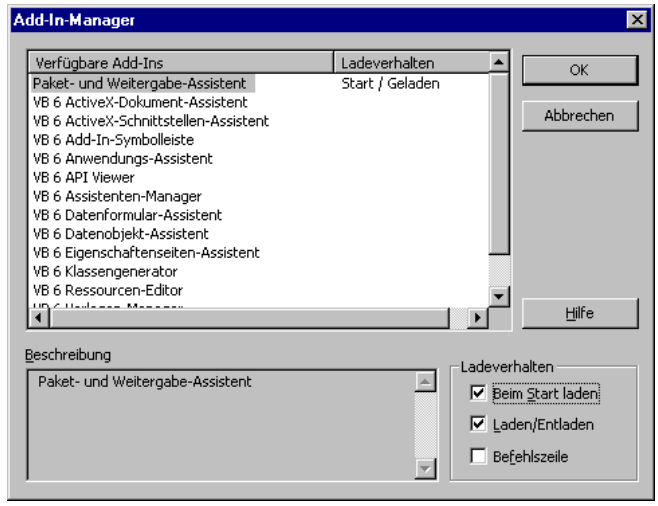

Bild 3.5: Der Add-In-Manager

Assistenten sind ein zweischneidiges Schwert. Auf der einen Seite gelingt es damit sehr rasch, funktionierende Programme zu erzeugen. Auf der anderen Seite besteht die Gefahr, daß Sie den Code nicht verstehen und beim Versuch, den Code zu ändern, mehr Zeit benötigen als bei einer Neuentwicklung. Erschwerend kommt hinzu, daß die von Assistenten produzierten Lösungen oft alles andere als optimal sind. Allzuviel Vertrauen in die Fähigkeiten der Assistenten ist also nicht angebracht. Die folgende Liste nennt einige Assistenten, die in diesem Buch zumindest kurz behandelt werden:

- PAKET- UND WEITERGABE-ASSISTENT: Hilft bei der Zusammenstellung von Installationsdateien (Seite 88). In diesem Buch wird statt des offiziellen Wortungetüms der Begriff Installationsassistent verwendet. Im Englischen finden Sie wiederum oft die Abkürzung PD-Wizard (für *Package and Deployment Wizard*).
- KLASSENGENERATOR: Hilft bei der Neudefinition einer Klassenhierarchie (Seite 193).
- API-VIEWER: Zeigt *Declare*-Definitionen zum Aufruf von DLL-Betriebssystemfunktionen an (Seite 616).
- DATENFORMULAR-ASSISTENT: Hilft beim Entwurf von Formularen, in denen Daten aus einer Datenbank angezeigt werden (Seite 734).
- ACTIVEX-SCHNITTSTELLENASSISTENT: Hilft bei der Definition der Schnittstelle zwischen einem neu entwickelten ActiveX-Steuerelement und der Außenwelt (Seite 1025).
- EIGENSCHAFTSSEITENASSISTENT: Hilft bei der Entwicklung eines Eigenschaftsdialogs zu einem neuen ActiveX-Steuerelement (Seite 1025).

- ACTIVEX-DOKUMENTASSISTENT: Hilft bei der Konvertierung eines herkömmlichen Visual-Basic-Programms in ein ActiveX-Dokument, das im Internet Explorer angezeigt und ausgeführt werden kann (Seite 1040).

TIP

Weitere Informationen zu den Assistenten finden Sie in der Online-Dokumentation:

VB-DOKUMENTATION | REFERENZ | ASSISTENTEN

3.8 Installationsassistent

3.8.1 Grundlagen

Wenn Sie ein Visual-Basic-Programm weitergeben möchten, ist es mit dem Kopieren der `*.exe`-Datei nicht getan; vielmehr müssen Sie zahllose `*.dll`- und `*.ocx`-Dateien mitliefern. Soll die Weitergabe via Internet geschehen, müssen diese Daten in `*.cab`-Dateien verpackt werden. In jedem Fall wollen diese Dateien ordnungsgemäß am Rechner des Anwenders installiert und registriert sein. Um all diese Aufgaben kümmert sich der sogenannte PAKET- UND WEITERGABE-ASSISTENT, der in diesem Buch allerdings kurz als Installationsassistent bezeichnet wird.

Die Funktionsvielfalt des Programms ist beeindruckend:

- Das Programm analysiert Ihr Programm und ermittelt eine Liste mit allen erforderlichen Dateien.
- Diese Dateien werden komprimiert und in ein Verzeichnis der Festplatte kopiert.
- Der Assistent erstellt ferner ein Setup-Programm zu den Installationsdateien, mit dem der Anwender Ihres Programms dieses vollautomatisch installieren kann.
- Dieses Setup-Programm kümmert sich darum, daß vorhandene Systemdateien nur dann überschrieben werden, wenn die Version auf den Installationsdisketten höher als die auf dem Rechner ist. Dadurch wird vermieden, daß womöglich veraltete Versionen von Systemdateien installiert werden. Das Setup-Programm erledigt auch die Registrierung von ActiveX-Komponenten. Schließlich protokolliert es den Installationsvorgang und ermöglicht später eine mühelose Deinstallation.
- Bei Internet-Komponenten ist das Setup-Programm nicht erforderlich – diese Aufgabe übernimmt dann der Internet Explorer. Dafür müssen die Installationsdateien aber in ein spezielles Format konvertiert werden.
- Die Installationsdateien von Internet-Komponenten können mit dem Assistenten direkt in ein Verzeichnis des Internet-Servers exportiert werden. Damit ist nun auch dieser letzte Schritt der Auslieferung automatisiert (neu in Version 6).

VERWEIS

Der Assistent ist hervorragend dokumentiert, die Dokumentation ist allerdings etwas verstreut:

VB-DOKUMENTATION | ARBEITEN MIT VB | EINSATZMÖGLICHKEITEN | VERTRIEB

VB-DOKUMENTATION | ARBEITEN MIT VB | KOMPONENTENHANDBUCH |
- ERSTELLEN VON INTERNET-ANWENDUNGEN |
- DOWNLOADEN VON ACTIVEX-KOMPONENTEN

VB-DOKUMENTATION | REFERENZ | ASSISTENTEN | PAKET- UND WEITERGABE-ASSISTENT

Lesen Sie außerdem die Datei `Readmevb.htm` (siehe unten)!

Probleme

Leider war der Installationsassistent seit jeher einer der Schwachpunkte von Visual Basic, d.h., nicht alle obigen Punkte werden tatsächlich so klaglos erledigt, wie es den Anschein hat. Der weitgehend neu entwickelte Assistent von Version 6 ist in dieser Beziehung leider keine Ausnahme; im Gegenteil, die Anzahl der Probleme ist im Vergleich zu Version 5 deutlich größer geworden.

- Das gravierendste Problem besteht darin, daß das vom Assistenten erzeugte Setup-Programm auf Windows-95-Rechnern unter Umständen mit einer Fehlermeldung abbricht. (Das Problem tritt nur bei alten Windows-95-Versionen auf, nicht bei der OEM-Version 2.*n*. Das Problem tritt nur auf, wenn am Rechner weder der Internet Explorer 3 / 4 noch Office 97 noch ein VB5-Programm installiert ist.)

  Abhilfe: Weisen Sie Ihre Kunden in einer `Readme`-Datei darauf hin, daß sie gegebenenfalls die Datei `Oleaut32.dll` selbst in das Windows-Systemverzeichnis kopieren. Eine aktuelle Version dieser Datei finden auf der Visual-Basic-CD-ROM im Verzeichnis `Os\System`.

- Wenn Ihr Programm ADO-Datenbankfunktionen nutzt, muß auf dem Windows-9x-Rechner DCOM installiert werden, bevor das Setup-Programm ausgeführt wird! DCOM-Installationsdateien befinden sich auf der Visual-Basic-CD-ROM im Verzeichnis `Dcom98`. Das Setup-Programm überprüft nicht, ob DCOM installiert ist – dafür ist abermals Ihr Kunde zuständig.

- Die Datei `Readmevb.htm`, die mit Visual Basic mitgeliefert wird, listet eine Menge weiterer Probleme auf, die hier nicht alle wiedergegeben werden. Der Abschnitt 'Themen zu Assistenten' in `Readmevb.htm` ist eine unablässige Pflichtlektüre, bevor Sie den Installationsassistenten anwenden!

HINWEIS

Die goldene Regel für den Umgang mit dem Installationsassistenten lautet: Glauben Sie nie, daß die Installation auf einem fremden Rechner funktioniert, bevor Sie es nicht ausprobiert haben! (Am besten erfolgt der Test auf mehreren Rechnern mit unterschiedlichen Betriebssystemen ... Ein Test am eigenen Entwicklungsrechner ist aussagelos.)

Voraussetzungen zur Ausführung von Visual-Basic-Programmen

Wenn Sie ein Visual-Basic-Programm in eine `*.exe`-Datei kompilieren (DATEI|*.EXE-DATEI ERSTELLEN), bekommen Sie ein eigenständiges Programm, das Sie unabhängig von der Entwicklungsumgebung von Visual Basic ausführen können – aber leider nur auf Ihrem Rechner! Wenn Sie die `*.exe`-Datei auf eine Diskette kopieren und auf dem Rechner eines Bekannten installieren, ist eine Programmausführung nicht möglich. Das `*.exe`-Programm baut nämlich auf einer Unzahl von `*.dll`-, `*.ocx`- und anderer Dateien auf, die vorher installiert werden müssen. Auf Ihrem Rechner ist das durch die Installation der Visual-Basic-Entwicklungsumgebung automatisch der Fall – aber auf anderen Rechnern eben nicht.

Seit Version 1 benötigen Visual-Basic-Programme eine sogenannte Runtime-Library. In Version 6 trägt diese Datei den Namen `Msvbvm60.dll` (Microsoft Visual Basic Virtual Machine – Java läßt grüßen) und hat die stattliche Größe von 1.4 MByte (was gegenüber Version 5 aber nur eine geringfügige Vergrößerung darstellt). Zu dieser Datei gesellen sich einige weitere Dateien, die bei den meisten Projekten ebenfalls erforderlich sind (etwa die OLE-Bibliotheken, die die Verwendung von Objekten in Visual Basic erst möglich machen).

Das ist der Grund, warum die Installation des 20 kByte kleinen Hello-World-Programms Dateien mit einer Gesamtgröße von mehr als 2 Mbyte erfordert! Selbst in komprimierter Form finden diese Dateien nicht auf einer Installationsdiskette Platz. Ohnehin ist eine CD-ROM mittlerweile das einzig adäquate Medium zur Weitergabe von Visual-Basic-Programmen geworden.

3.8.2 Bedienung des Installationsassistenten

Vorarbeiten

Bevor Sie den Installationsassistenten starten, sollten Sie an Ihrem Programm einige Vorarbeiten leisten:

- Entfernen Sie alle Zusatzsteuerelemente, die zwar in der Toolbox angezeigt, aber von Ihrem Programm nicht verwendet werden.
- Entfernen Sie alle Referenzen auf nicht benötigte Programmbibliotheken (Menükommando PROJEKT|VERWEISE). Besonders in Programmen, die mit älteren Versionen von Visual Basic entwickelt wurden, wimmelt es oft nur so von überflüssigen

Verweisen. Wenn Sie sich nicht sicher sind, ob Sie eine Bibliothek brauchen oder nicht, versuchen Sie einfach, sie zu deaktivieren. Visual Basic läßt nicht zu, daß Sie Bibliotheken entfernen, die benötigt werden (bzw. meldet sich beim probeweisen Ausführen oder Kompilieren mit einer Fehlermeldung).

- Speichern, kompilieren und testen Sie das Programm nochmals. Werfen Sie auch einen Blick in PROJEKT | EIGENSCHAFTEN | KOMPILIEREN, und optimieren Sie eventuell die Compiler-Einstellungen.

Installationstypen

Der Assistent wird über das ADD-IN-Menü gestartet und automatisch auf das gerade geladene Projekt angewandt. Im Begrüßungsdialog können Sie zwischen drei Operationen auswählen:

- VERPACKEN: Installationsdateien erstellen
- VERTEILEN: zuvor erstellte Dateien auf einen Internet-Server exportieren
- SKRIPTS VERWALTEN: Konfigurationsdateien des aktuellen Projekts verwalten

Der folgende Text bezieht sich auf die VERPACKEN-Funktion, die sicherlich am häufigsten benötigt wird. Die nächste Frage bezieht sich auf den Pakettyp. Zur Auswahl stehen:

- STANDARD SETUP-PAKET: Installationsdateien mit Setup-Programm, die auf CD-ROM weitergegeben werden und am Rechner des Kunden installiert werden
- INTERNET-PAKET: `*.cab`-Dateien, die am Internet-Server zum Herunterladen bereit gestellt werden; diese Option steht nur bei ActiveX-Komponenten, nicht aber bei normalen Programmen zur Auswahl; Informationen zur Internet-Installation von ActiveX-Komponenten finden Sie auf Seite 997
- ABHÄNGIGKEITSDATEI: `*.dep`-Dateien, die Informationen enthalten, welche Bibliotheken zur Ausführung eines Programms oder einer Komponente erforderlich sind; Abhängigkeitsdateien werden im Regelfall nur für ActiveX-Komponenten erstellt, die an andere Programmierer weitergegeben werden

Standard-Setup-Paket

Als nächstes müssen Sie ein Verzeichnis angeben, in das die Installationsdateien geschrieben werden sollen. Dazu können Sie beispielsweise ein Unterverzeichnis `setup` im aktuellen Verzeichnis anlegen.

Im nächsten Schritt zeigt der Assistent alle Installationsdateien an (Bild 3.6): Dabei handelt es sich um das eigentliche Programm, die dazugehörigen Bibliotheken sowie um zwei Setup-Programme (deren Bedeutung unten noch beschrieben wird). Falls Ihr

Programm noch weitere, spezifische Dateien benötigt (etwa eine Hilfedatei, eine Datenbankdatei, zusätzliche Systembibliotheken etc.), können Sie diese HINZUFÜGEN.

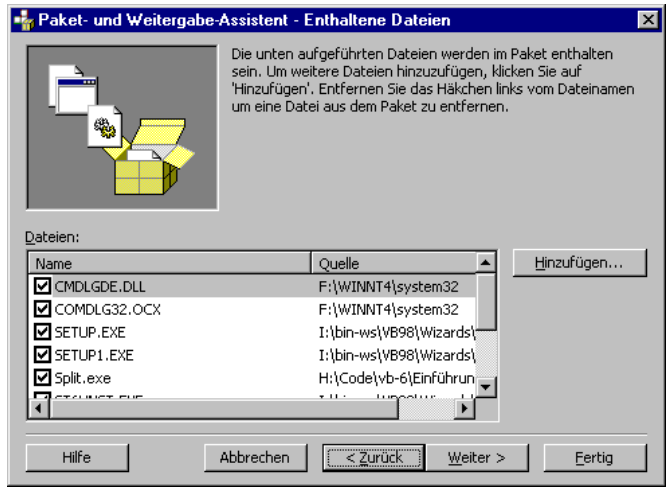

Bild 3.6: Der Installationsassistent

Anschließend geben Sie an, ob diese Dateien in eine riesige `*.cab`-Datei gepackt oder ob mehrere `*.cab`-Dateien mit einer Größe von jeweils 1.4 MByte gebildet werden sollen (zur Weitergabe auf Disketten). Eine weitere Frage betrifft den Ort, wo das Programm in das Windows-Startmenü eingetragen werden soll. Danach können Sie angeben, in welche Verzeichnisse die einzelnen Dateien am Rechner des Kunden installiert werden sollen (z.B. Windows-Systemverzeichnis). Im Regelfall können Sie hier die Vorgaben einfach bestätigen.

Als Ergebnis erhalten Sie letztlich im `Setup`-Verzeichnis je eine `*.cab`- und `*.lst`-Datei sowie das Programm `setup.exe`. Diese drei Dateien müssen Sie an Ihren Kunden weitergeben. Im `Setup`-Verzeichnis wird außerdem ein `Support`-Verzeichnis angelegt, das den gesamten Inhalt der `*.cab`-Datei in unkomprimierter Form enthält. Dieses Verzeichnis erleichtert eine spätere Aktualisierung der Installationsdateien: Sie brauchen nur einzelne Dateien austauschen und dann die `*.bat`-Datei ausführen, um eine aktualisierte `*.cab`-Datei zu erstellen. Aber auch das neuerliche Ausführen des Installationsassistenten ist kein Problem: Alle Konfigurationsinformationen werden in der Konfigurationsdatei `projektname.pdm` gespeichert, Sie müssen also nicht alle Eingaben wiederholen.

Abhängigkeitsdateien

Vielleicht fragen Sie sich, woher der Installationsassistent weiß, welche Dateien zur Ausführung des Projekts erforderlich sind. Eine wesentliche Informationsquelle sind dabei die Abhängigkeitsdateien mit der Kennung `*.dep`, die sich für alle Zusatz-

steuerelemente im Windows-Systemverzeichnis befinden. Diese Dateien enthalten Informationen darüber, welche Voraussetzungen erfüllt sein müssen, damit ein Zusatzsteuerelement verwendet werden kann. Ein Blick in diese Dateien ist recht aufschlußreich, wenn Sie wissen möchten, wozu die vielen Installationsdateien gebraucht werden.

Wenn der Installationsassistent Komponenten in Ihrem Projekt entdeckt, zu denen keine Abhängigkeitsdateien existieren, zeigt er eine Warnung an (Bild 3.7). An dieser Stelle können Sie sich über die Warnung nur hinwegsetzen (und durch Anklicken des Auswahlkästchens eine neuerliche Warnung beim nächsten Mal vermeiden). Wenn Sie die betreffende Komponente selbst entwickelt haben, ist es allerdings besser, dazu eine Abhängigkeitsdatei zu erstellen.

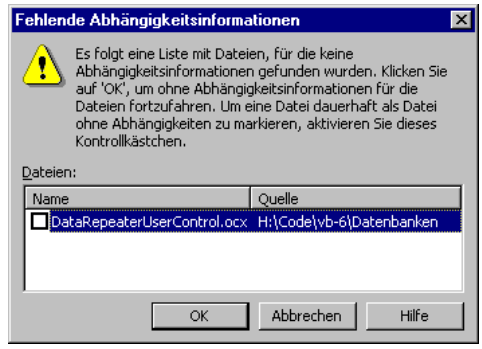

Bild 3.7: Der Installationsassistent beklagt sich über fehlende Abhängigkeitsinformationen

Probleme mit fehlenden Bibliotheken

Wenn ein ausgeliefertes Visual-Basic-Programm beim Kunden nicht funktioniert, liegt eine wahrscheinliche Fehlerursache bei Bibliotheken, die Sie via *CreateObject("bibliothek.obj")* nutzen, ohne einen entsprechenden Verweis darauf einzurichten. Das ist ohne weiteres möglich, und zahlreiche Beispiele in der Online-Dokumentation schlagen diese Vorgehensweise vor (insbesondere die Beispiele zur Scripting-Bibliothek, die das neue *Dictionary*-Objekt und die *File System Objects* (FSO) enthalten). Das Problem besteht darin, daß der Installationsassistent mangels Verweis nicht erkennen kann, daß Sie diese Bibliothek nutzen. Sie müssen die Bibliothek daher im Schritt HINZUFÜGEN manuell zu den Installationsdateien hinzufügen!

Um den Dateinamen der Bibliothek herauszufinden, starten Sie am besten eine zweite Instanz von Visual Basic und sehen im Dialog PROJEKT|VERWEISE nach. Die Scripting-Runtime-Bibliothek trägt beispielsweise den Namen `Scrrun.dll` und befindet sich im Windows-Systemverzeichnis.

Nach der Auswahl der Datei fragt der Installationsassistent möglicherweise nach Abhängigkeiten, d.h., ob andere Bibliotheken notwendig sind, um die gerade ausgeführte Bibliothek zu nutzen (siehe vorherigen Teilabschnitt).

Um festzustellen, welche Bibliotheken ein Visual-Basic-Programm benutzt, können Sie auch das Systeminformationsprogramm verwenden (Start unter Visual Basic mit HILFE | ÜBER VISUAL BASIC). Beenden Sie möglichst alle laufenden Programme und speichern Sie dann den aktuellen Status der Systeminformation (dazu wird automatisch der Dateiname `Msinfo32.txt` im Windows-Verzeichnis benutzt). Erstellen Sie eine Kopie dieser Datei, starten Sie Ihr Visual-Basic-Programm und wiederholen Sie die Status-Speicherung.

Jetzt brauchen Sie die beiden Textdateien nur noch zu vergleichen (insbesondere den Abschnitt *Active Modules*). Zum Textvergleich können Sie beispielsweise WinWord benutzen (Kommando EXTRAS | ÄNDERUNGEN VERFOLGEN | DOKUMENTE VERGLEICHEN). Die Aussagekraft dieses Vergleichs ist insofern eingeschränkt, als manche erforderlichen Bibliotheken unter Umständen auch auf Betriebssystemebene benutzt werden und daher nicht entdeckt werden können.

Adaption des Setup-Programms

Bei komplexen Programmen kann es sein, daß Sie dem Anwender bei der Installation mehrere Varianten anbieten möchten. Oder Sie möchten das Installationsprogramm mit Informationen über Ihre Firma ergänzen. Oder Sie möchten die Installation mit der Eingabe einer Lizenznummer absichern. Oder ...

All das ist möglich, erfordert aber einige Arbeit und vor allem ein Verständnis über den Ablauf der Installation: Die Installation beginnt mit dem Programm `Setup.exe`. Dieses Programm führt den ersten Teil der Installation durch und kopiert unter anderem die elementaren Visual-Basic-Libraries in das Windows-Systemverzeichnis (sofern sich dort nicht schon eine aktuelle Version befindet). Nachdem das erledigt ist, wird das Visual-Basic-Programm `Setup1.exe` gestartet, das für den zweiten Teil der Installation zuständig ist. Und eben dieses Programm `Setup1.exe` können Sie nach Ihren eigenen Vorstellungen anpassen.

Der Programmcode für dieses Programm befindet sich im Visual-Basic-Verzeichnis `Wizards\PDWizard\Setup1`. Bevor Sie irgendetwas ändern, erstellen Sie eine Sicherheitskopie vom gesamten Verzeichnis! Der Startpunkt des umfangreichen Programms ist die Prozedur *Form_Load* des Formulars *frmSetup1*. Der Code ist zwar gut dokumentiert, Veränderungen bedürfen aber dennoch einer intensiven Einarbeitung.

Internet-Installation

Wenn Sie mit Visual Basic nicht herkömmliche Programme, sondern ActiveX-Komponenten entwickelt haben, gibt es eine zweite Installationsvariante: Cabinet-Dateien. Das sind spezielle komprimierte Dateien, die vom Internet Explorer über ein lokales

Netz oder über das Internet geladen werden können – siehe Seite 997. (An dieser Stelle wird auch das Thema Sicherheit und ActiveX eingehend diskutiert.)

VERWEIS

Das Buch beschreibt in den betreffenden Kapitel einige weitere Sonderfälle beim Umgang mit dem Installationsassistenten (etwa die Integration von DLLs und eigener Type-Libraries). Werfen Sie einen Blick in das Stichwortverzeichnis, Eintrag 'Installationsassistent'!

3.8.3 Installation mit dem Setup-Programm ausführen

Ihr Kunde startet die Installation einfach mit dem Programm `Setup.exe` auf der ersten Installationsdiskette oder CD-ROM. Default erfolgt die Installation in ein Unterverzeichnis des `Programme`-Verzeichnisses von Windows – dieses Verzeichnis kann aber verändert werden. Vor Beginn der Installation sollten alle laufenden Visual-Basic-Programme und -Komponenten beendet werden, damit gegebenenfalls Visual-Basic-Libraries aktualisiert werden können.

Wenn es sich um die erste Installation eines VB6-Programms am Rechner handelt, beginnt die Installation mit der für den Kunden unerfreulichen Nachricht, daß einige Systemdateien aktualisiert werden müssen. Dazu muß der Rechner neu gestartet werden! (Diese Unsitte greift immer mehr um sich. Betriebssysteme wie Unix / Linux laufen oft jahrelang ohne Reboot! Davon ist Windows (auch NT) noch meilenweit entfernt – mittlerweile erfordert jede noch so minimale Installation schon einen Neustart. Davon abgesehen haben viele Windows-Anwender mittlerweile ein berechtigtes Mißtrauen gegen jedes Programm, das Systemdateien verändert. Alles in allem wird der erste Eindruck, den der Kunde von Ihrem Programm hat, leider ein negativer sein.)

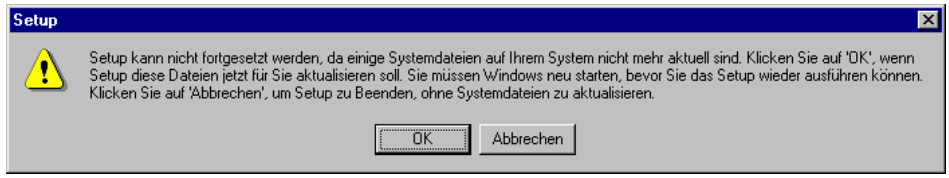

Bild 3.8: Das Setup-Programm fordert zum Rechnerneustart auf

Nach dem Neustart muß der Kunde `Setup.exe` neuerlich starten. Jetzt sollte alles glatt laufen. Das Programm meldet sich mit dem folgenden Dialog, in dem das Installationsverzeichnis verändert werden kann:

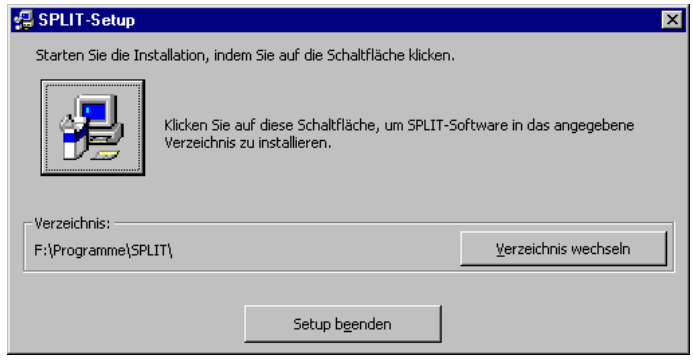

Bild 3.9: Das Setup-Programm

Installationsprobleme

Je nach Windows-Version können beim Ausführen des Installationsprogramms erhebliche Probleme auftreten. Um es nochmals zu wiederholen: Lesen Sie unbedingt die mit Visual Basic mitgelieferte Datei `Readmevb.htm`! Beim Autor sind mit Windows 95 B noch zwei weitere Probleme aufgetreten:

- Einzelne Dateien im Windows-Systemverzeichnis sind unter Umständen schreibgeschützt. Das Setup-Programm beklagt sich dann darüber, daß es die Dateien nicht verändern kann. Abhilfe: Markieren Sie alle Dateien im Systemverzeichnis, wählen Sie das Kontextmenü EIGENSCHAFTEN aus und deaktivieren Sie das Attribut SCHREIBGESCHÜTZT.
- Das Setup-Programm beklagt sich darüber, daß auf diverse Dateien momentan zugegriffen wird. Abhilfe: Beenden Sie alle anderen Programme (auch den Explorer)!

Beachten Sie, daß diese Probleme nicht bei Ihnen, sondern am Rechner Ihres Kunden auftreten (können). Bereiten Sie eine entsprechende Readme- oder Hilfe-Datei vor!

Generell sollten Sie jede Installation vor der Auslieferung an den Kunden auf einem 'jungfräulichen' Rechner mit einem möglichst alten Betriebssystem testen, d.h. idealerweise auf einem frisch installierten Windows-95-System *ohne* irgendwelche Systemerweiterungen, ohne vorherige Installation einer aktuellen Internet-Explorer-Version etc. Nur wenn Ihr Programm auf diesem System läuft, können Sie einigermaßen sicher sein, daß Sie nichts vergessen haben.

Auf Ihrem Entwicklungssystem ist dieser Test nicht möglich, weil dort Visual Basic, der Internet Explorer 4 usw. mit allen erforderlichen Bibliotheken ohnedhin installiert sind. (Der Autor ist sich durchaus bewußt, daß dieser Ratschlag schneller formuliert als tatsächlich ausgeführt ist. Woher einen Rechner bzw. eine Partition nehmen, auf die schnell ein Windows 95 installiert werden kann? Aber vermutlich ist dieser Aufwand immer noch besser als nachträgliche Reklamationen und Beschwerden.)

Deinstallation

Während der Installation werden alle dabei durchgeführten Aktionen – also die Installation von Bibliotheken in das Systemverzeichnis, die Veränderung von Einträgen in der Registrierdatenbank etc. – in der Datei `St6unst.log` protokolliert. Diese Datei wird im selben Verzeichnis wie das Visual-Basic-Programm gespeichert und darf nicht gelöscht werden! Sie enthält die für die Deinstallation erforderlichen Informationen. Die Deinstallation erfolgt über das Icon SOFTWARE INSTALLIEREN / ENTFERNEN der Windows-Systemsteuerung.

3.9 MSDN-Dokumentation

Seit Visual Basic 6 steht die gesamte Dokumentation zu Visual Basic in Form der Microsoft Developer Network Library zur Verfügung (nur Professional- und Enterprise-Version). Das eine feine Sache: erstmals können Sie zentral in einem Programm nach Stichwörtern suchen.

Generell stellt die MSDN-Library eine großartige Informationsquelle dar – es lohnt sich wirklich, den Umgang damit zu erlernen. Auch wenn diverse Details verbesserungswürdig sind (z.T. funktioniert etwa der kontextabhängige Aufruf mit F1 nicht), muß man immer vor Augen haben, welche unglaublichen Textmengen hier ansprechend formatiert und einigermaßen strukturiert zur Verfügung stehen. Der für das Internet oft fragwürdige Werbeslogan *information at your finger tips* wird der MSDN-Library fast uneingeschränkt gerecht.

Der Umgang mit der Library ist an sich einfach: Das Programm wird entweder durch F1 oder über das Hilfemenü gestartet. Die Navigation kann auf drei Arten erfolgen: über das hierarchische Inhaltsverzeichnis (das leider bisweilen unübersichtlich tief strukturiert ist), über den alphabetischen Index (im dem aber vereinzelt Einträge fehlen) oder über die Volltextsuche. Vor allem letzteres führt fast immer zum Ziel – liefert aber oft weit mehr Ergebnisse, als Ihnen lieb ist.

Anbei einige Tips, um den Umgang mit der Library zu optimieren:

- Sie können die MSDN-Library nicht nur von Visual Basic aus starten, sondern auch als eigenständiges Programm (Startmenü PROGRAMME | DEVELOPER NETWORK | MICROSOFT MSDN). Das hat zwei Vorteile: Erstens läuft die Library dann auch nach einem Absturz von Visual Basic weiter; und zweitens können Sie das Programm mehrfach öffnen, was beim Wechseln zwischen zwei oder drei Themen sehr hilfreich ist.
- Nutzen Sie die Möglichkeit, Lesezeichen zu setzen (Dialogblatt FAVORITEN). Es wird Ihnen sonst immer wieder passieren, daß Sie einmal gefundene Informationen kein zweites Mal wiederfinden (wie im Internet).
- Schränken Sie den Inhalt, den Index und die Stichwortsuche gezielt auf Teile der Dokumentation ein (Listenfeld AKTIVE TEILMENGE)! Wenn Ihre Englischkenntnisse

bescheiden sind, wählen Sie den Eintrag ÜBERSETZTER INHALT! (Nur ein Bruchteil der MSDN-Library ist ins Deutsche übersetzt. Die Qualität der Übersetzung ist allerdings nicht immer optimal, daß man sich bisweilen das besser verständliche englische Original zurückwünscht.)

- Sortieren Sie die Suchergebnisse nach der Herkunft (Spaltenüberschrift POSITION). Sehr oft ist die gewünschte Information dann rasch lokalisiert.
- Mit dem Kommando ANSICHT | THEMA IM INHALT SUCHEN können Sie meistens (wenn auch nicht immer) in das Inhaltsverzeichnis wechseln, wo der gerade angezeigte Text markiert wird. Das ist vor allem nach Suchoperationen praktisch, weil Sie so leichter ein Überblick über verwandte Themen finden.
- Leider passiert es recht häufig, daß Code-Beispiele nicht von der MSDN-Library in ein Code-Fenster von Visual Basic kopiert werden können, weil die Zeilenumbrüche verloren gehen. Statt mühsam zu versuchen, die Zeilenenden zu rekonstruieren, können Sie mit dem Kontextmenüeintrag QUELLE ANZEIGEN den HTML-Quellcode in einem Notepad-Fenster öffnen. Dieser Text läßt sich problemlos in das Code-Fenster kopieren. Sie müssen jetzt nur noch mit Suchen und Ersetzen einige HTML-Codes durch die entsprechenden Sonderzeichen ersetzten (etwa *&* durch das Zeichen &).

Knowledge Base

Ein wichtiger Teil der MSDN-Library ist die sogenannte *Knowledge Base* (kurz KB), eine schier endlose Liste von Artikeln, die kurze Informationen zu oft gestellten Fragen, Problemen, Bugs etc. enthalten. Das Problem besteht darin, daß die für ein Problem gerade relevanten Informationen unter einem Berg anderer Daten versteckt sind. Nützen Sie die Suchfunktionen, schränken Sie die Suche gezielt auf die Knowledge Base ein (bzw. auf den VB-Teil der Knowledge Base)!

In diesem Buch werden manchmal KB-Artikelnummern angegeben, etwa Q171583. Diese Nummern sind zumeist innerhalb der gesamten MSDN-Library eindeutig, d.h. Sie finden den entsprechenden Artikel ganz rasch durch eine Volltextsuche.

Die MSDN-Library steht unter der Adresse `support.microsoft.com/support` auch Online zur Verfügung. Die Online-Version hat natürlich den Vorteil, daß sie aktueller ist und bereits diverse VB6-Artikel enthält (vornehmlich zu den zahllosen Fehlern der neuen Version). Leider entspricht das Online-Suchformular nicht dem der MSDN-Library.

3.10 Tastenkürzel

Der Abschnitt gibt einen Überblick über die wichtigsten Tastenkürzel, die während der Programmentwicklung benötigt werden. Nicht mit aufgenommen wurden Ta-

stenkürzel, die generell unter Windows gelten (etwa Strg+C zum Kopieren in die Zwischenablage). Die Tastenkürzel wurden nach Bereichen geordnet, in denen sie am häufigsten benötigt werden. Viele Tastenkürzel stehen aber generell unter Visual Basic zur Verfügung (unabhängig davon, welches Fenster gerade aktiv ist).

Generell

Strg+N	Neues Projekt
Strg+O	Projekt öffnen
Strg+D	Datei hinzufügen
Strg+S	Datei speichern
F1	Online-Hilfe

Wechsel des aktuellen Fensters

Strg+Tab	wechselt zwischen allen Visual-Basic-Fenstern
Alt+F6	wechselt zwischen den beiden zuletzt aktiven Fenstern
Strg+E	Menüeditor aufrufen
Strg+G	ins Direktfenster (Debugfenster) wechseln
Strg+L	Liste der aufgerufenen Prozeduren anzeigen (Debugging)
Strg+R	ins Projektfenster wechseln
Strg+T	Dialog für Zusatzsteuerelemente anzeigen
F2	in den Objektkatalog wechseln
F4	ins Eigenschaftsfenster wechseln
Shift+F4	Eigenschaftsdialog zum Steuerelement anzeigen
F7	ins Codefenster wechseln
Shift+F7	ins Formularfenster wechseln

Programmausführung

F5	Programm starten
Strg+F5	Programm zuerst vollständig kompilieren, dann starten
Strg+Untbr	Programm unterbrechen
F8	ein einzelnes Kommando ausführen (single step)
Shift+F8	Kommando / Prozeduraufruf ausführen (procedure step)
Strg+F8	Prozedur bis zur Cursorposition ausführen
Strg+Shift+F8	aktuelle Prozedur bis zum Ende ausführen
F9	Haltepunkt setzen
Strg+F9	Ort des nächsten Kommandos bestimmen

Im Formularfenster

Tab	zum nächsten Steuerelement wechseln
Strg+J	Steuerelement in den Hintergrund bringen
Strg+K	Steuerelement in den Vordergrund bringen
Strg+Cursortasten	Steuerelement verschieben

Im Eigenschaftsfenster

Shift+Tab	springt ins Steuerelement-Listenfeld
Strg+Shift+X	springt zur Eigenschaft mit dem Anfangsbuchstaben X

Im Codefenster

Tab	markierten Zeilenblock einrücken
Shift+Tab	markierten Zeilenblock ausrücken
Strg+Y	Zeile löschen
Alt+Backspace	Änderung widerrufen (Undo)
Strg+Z	Änderung widerrufen (Undo)
Strg+Backspace	Widerruf rückgängig (Redo)
Strg+↑ / ↓	Cursor zur vorigen / nächsten Prozedur
Shift+F2	zur Prozedurdefinition bzw. zur Variablendeklaration
Strg+Shift+F2	zurück zur letzten Cursorposition (Undo zu Shift+F2)
F6	Codeausschnitt wechseln (bei zweigeteiltem Fenster)
Strg+F	Suchen
F3	Weitersuchen
Strg+H	Suchen und Ersetzen
Strg+Leertaste	Schlüsselwort / Variablennamen vervollständigen
Tab	Auswahl im IntelliSense-Listenfeld durchführen
Esc	IntelliSense-Listenfeld verlassen