

Überblick

Auf gehts: Sie erfahren, wie Sie mit Visual Basic eigene Computerprogramme schreiben! In dieser ersten Woche lernen Sie die Grundlagen kennen, die Sie für den Umgang mit Visual Basic brauchen. Sie erfahren, wie Sie sich in der Visual-Basic-Umgebung bewegen und wie die visuellen Elemente eines Windows-Programms angelegt werden. Darüber hinaus lernen Sie die Grundlagen der Programmiersprache Visual Basic kennen.

Was Sie erwartet

Obwohl diese erste Woche eher eine Einführung darstellt, werden Sie schon am ersten Tag, »Willkommen bei Visual Basic«, mit Visual Basic arbeiten und dabei Erfahrungen sammeln. In der ersten Lektion erstellen Sie eine funktionierende Visual-Basic- Applikation, die sich wie alle anderen Windows-Programme verhält, die Sie bereits kennen. Die folgenden Lektionen bauen auf den hier vermittelten Kenntnissen auf und zeigen Ihnen neue Elemente für Ihre Programme und wie Sie diese noch leistungsfähiger machen. Jeder Tag wird mit einem Übungsteil abgeschlossen, wo Sie auch Fragen zur Wiederholung finden. Hier werden die wichtigsten Dinge aus dem Kapitel wiederholt, so daß Sie erkennen, wo Sie noch Lücken haben. Außerdem sammeln Sie hier praktische Erfahrungen für die Programmentwicklung.

Für die Programmierung genügt es nicht, eine Programmiersprache zu beherrschen. Während Sie die Kapitel in dieser und den beiden folgenden Wochen lesen, werden Sie verstehen, wie wichtig es ist, klare, gut dokumentierte Programme zu schreiben. Die Umgebungen, in denen die Computerprogramme eingesetzt werden, ändern sich, und das sollte auch in den Programmen berücksichtigt werden. Wenn Sie sich von Anfang an einen guten Programmierstil angewöhnen, werden Ihre Programme leichter verständlich und können einfacher aktualisiert werden.

Mit Visual Basic werden Windows-Programme geschrieben. In dieser ersten Woche lernen Sie fast jedes Element der Visual-Basic-Programmierung ein kleines bißchen kennen. Sie erfahren, wie man Objekte auf einem Windows-Bildschirm anlegt, wie man Pulldown-Menüs erstellt und verwaltet und wie man die Interaktion zwischen Programm und Benutzer realisiert. Und Sie lernen die Seele von Visual Basic kennen: die Programmiersprache, die allem einen Sinn gibt.

Die Visual-Basic-Programmierung ist eine der angenehmsten Methoden, Programme zu entwickeln. Man legt einfach ein grafisches Objekt auf dem Bildschirm an und setzt seine Attribute, die bestimmen, wie das Objekt aussieht und wie es sich verhält. Visual Basic ist heute die einzige Programmiersprache, die auch Anfänger ganz leicht erlernen können. Darüber hinaus ermöglicht es fortgeschrittenen Programmierern, leistungsfähige Windows-Applikationen zu schreiben.

Stecken Sie sich Ihre Ziele ruhig höher! Wenn Sie noch nie programmiert haben, oder wenn Sie noch nie in Visual Basic programmiert haben, werden Sie die Arbeit genießen. Und Sie werden überrascht sein, was mit Visual Basic alles möglich ist.

Woche 1

Tag 1

Willkommen bei Visual Basic

Visual Basic 6 ist die neueste und beste Version der Programmiersprache Visual Basic von Microsoft. Die Programmentwicklung kann manchmal recht mühsam sein, aber Visual Basic reduziert den von Ihnen geforderten Aufwand und macht die Programmierung zu einem Vergnügen. Viele Aufgaben können gelöst werden, indem man einfach nur ein grafisches Objekt mit der Maus auf den Bildschirm zieht.

Heute beginnt Ihr 21-Tage-Kurs in Visual Basic. Noch bevor Sie dieses Kapitel ganz gelesen haben, werden Sie Ihr erstes Visual-Basic-Programm geschrieben haben. In den nächsten drei Wochen lernen Sie, Visual Basic 6 zu beherrschen. Sie werden in der Lage sein, Applikationen zu entwickeln, die wirklich sinnvolle Arbeiten für Sie erledigen.

Heute lernen Sie die folgenden Dinge:

- Die Geschichte von Visual Basic
- Programmentwurf und Realisierung
- Die visuelle Benutzeroberfläche von Visual Basic - so macht Programmieren Spaß
- Der Anwendungs-Assistent
- Die Bedeutung der ereignisorientierten Programmierung in einer Windows-Umgebung

Die Entstehungsgeschichte von Visual Basic

Wenn Sie die Entstehungsgeschichte von Visual Basic kennen, werden Sie Visual Basic 6 besser verstehen und für den Einsatz von Visual Basic gerüstet sein. Microsoft baute Visual Basic auf einer *Programmiersprache* auf, die für Anfänger entwickelt worden war, *BASIC*. BASIC gab es 15 Jahre lang in den unterschiedlichsten Variationen. Die Entwickler, die diese Sprache ins Leben gerufen hatten, wollten eine Programmiersprache für Anfänger schaffen. Mit BASIC waren neue Programmierer schnell vertraut. Andere Programmiersprachen, die zur selben Zeit verwendet wurden, beispielsweise COBOL, FORTRAN oder Assembler, machten viel mehr Lernaufwand erforderlich, bevor man sie effektiv anwenden konnte.



BASIC steht für *Beginner's All-purpose Symbolic Instruction Code*. Gut, daß es Abkürzungen gibt!

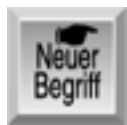


Eine *Programmiersprache* besteht aus Befehlen und Befehlsoptionen, sogenannten *Argumenten*, mit denen Sie dem Computer Anweisungen geben. Computer verstehen die menschliche Sprache (noch) nicht, weil Menschen mehrdeutige Aussagen treffen können, die ein Computer nicht nachvollziehen kann. Eine Programmiersprache muß präziser sein als eine gesprochene Sprache.



Programmiersprachen sind einfacher zu lernen als Fremdsprachen. Computersprachen umfassen oft weniger als 300 Befehle, und viele dieser Befehle sind ganz normale Wörter, die man leicht versteht, beispielsweise *Open* oder *Next*.

Die Sprache BASIC war zwar für Anfänger gedacht, aber ein *BASIC-Programm* war eher kompliziert zu lesen. Listing 1.1 zeigt ein Programm in BASIC. Dieses Programm soll die Quadrate aller Zahlen von 1 bis 10 ausdrucken. Sie können zwar einen Teil der Programmlogik erraten und die Befehle zuordnen, aber das Programm ist sicher nicht das deutlichste der Welt, und Sie müssen BASIC schon verstehen, damit Sie begreifen, wozu die einzelnen Komponenten gut sind.



Programme bestehen häufig aus mehreren Dateien, die zusammenarbeiten. Manchmal werden als Synonyme für die Dateien eines Programms auch die Begriffe *Applikation* oder *Anwendung* gebraucht. Das Programm oder die Anwendung, die in einer Programmiersprache geschrieben sind, enthalten Anweisungen, die den Computer steuern.

Listing 1.1: Frühe BASIC-Programme verwendeten Zeilennummern und waren schwer zu verstehen.

```
10 REM Dieses Programm berechnet die ersten zehn Quadratzahlen
20 REM und druckt sie aus
30 CLS
40 PRINT "Quadratzahlen von 1 bis 10"
50 PRINT "wert", "Quadrat"
60 FOR N = 1 TO 10
70 PRINT N, (N*N)
80 NEXT N
90 PRINT
100 END
```

Was Sie lassen sollten

Lassen Sie sich nicht durch all dieses Gerede über die Quadratzahlen von 1 bis 10 beunruhigen. Sie mögen keine Mathematik? Kein Problem! Visual Basic übernimmt die ganze Mathematik für Sie.

Die Ausführung dieses BASIC-Programms erzeugt die folgende Ausgabe:

	Quadratzahlen von 1 bis 10	
	Wert	Quadrat
	1	1
	2	4
	3	9
	4	16
	5	25
	6	36
	7	49
	8	64
	9	81
	10	100

Beachten Sie, daß BASIC eine rein textbasierte Sprache ist. Sowohl das Programm als auch die Ausgabe bestehen ausschließlich aus Text. Es ist nicht möglich, eine Grafikausgabe in Fenstern zu erzeugen, wie wir sie von den heutigen Programmen gewöhnt sind.

Microsoft hat Visual Basic nicht direkt von der Originalsprache BASIC abgeleitet. BASIC entwickelte sich in seiner 35jährigen Geschichte zwar in unterschiedlichen Phasen weiter, aber es behielt dabei seine Originalstruktur größtenteils bei. Als Microsoft beschloß, BASIC als seine Haupt-Programmiersprache zu verwenden, die zusammen mit dem MS-DOS-Betriebssystem ausgeliefert wurde, verbesserte es die Sprache und ergänzte ihre Funktionalität. Dazu wurden mehrere Varianten von BASIC geschaffen, mit Namen wie MBASIC (für Microsoft BASIC), GWBASIC, BASICA (für BASIC Advanced), QuickBASIC und QBasic (das auf den CDs von Windows-Betriebssystemen immer noch bereitgestellt wird).

Während dieser ganzen Entwicklung blieb BASIC einfach, obwohl es jetzt neue, leistungsfähige Befehle erhielt. Textbasierte Sprachen wie beispielsweise QBasic sind von neuen Programmierern sehr viel schneller zu erlernen als viele nicht-textbasierte Sprachen, wie beispielsweise Visual C++. Um die einfache Handhabung weiterhin zu gewährleisten, wollte Microsoft alle BASIC-Versionen als *interpretierte* im Gegensatz zu einer *kompilierten* Sprache beibehalten. Ein Programmierer kann ein Programm, das auf einer interpretierten Sprache basiert, unmittelbar ausführen und sieht sofort die Ergebnisse und die Fehler. Ein solches Feedback ist vor allem für Anfänger wichtig, die schnelle Antworten brauchen, wenn sie lernen zu programmieren. Kompilierte Sprachen erzeugen zwar schnellere Programme und sind besser für professionelle Entwicklungsumgebungen geeignet, bedingen jedoch auch viel mehr Arbeitsaufwand.

	Eine <i>interpretierte</i> Sprache, wie beispielsweise BASIC, erlaubt Ihnen, die Programme während der Entwicklung auszuführen. Interpretierte Sprachen sind einfach zu erlernen, weil sie ein schnelles Feedback bieten. Bei einer kompilierten Sprache sind zusätzliche Schritte erforderlich, das <i>Kompilieren</i> und das <i>Linken</i> , bevor der Programmierer das Programm ausführen kann. Der Computer legt das <i>kompilierte</i> Programm in seiner eigenen Sprache ab, und nicht in der Programmiersprache, die der Programmierer für die Entwicklung verwendet hat.	

Als Windows sich immer mehr durchsetzte, erkannte Microsoft, daß das textbasierte QBasic nicht als fensterorientierte Programmiersprache eingesetzt werden konnte. Microsoft entwickelte Visual Basic, eine Sprache, die auf BASIC aufbaute, die aber für die heute üblichen fensterorientierten Umgebungen

besser geeignet ist. Während QBasic und alle anderen BASIC-Varianten textbasiert waren, ist Visual Basic grafisch. Ein Visual-Basic-Programm könnte zwar *Code* enthalten, der ein bißchen aussieht wie der in Listing 1.1 gezeigte, aber hauptsächlich besteht es aus grafischen Elementen, die nur noch wenig mit dem textbasierten Code aus Listing 1.1 gemeinsam haben. Abbildung 1.1 zeigt einen Visual-Basic-Bildschirm, der mehrere Bestandteile eines Visual-Basic-Programms demonstriert.



Code ist ein anderer Name für die Gesamtheit der Anweisungen in einem Programm.



In diesem Buch werden Sie alle Elemente kennenlernen, die Sie in Abbildung 1.1 sehen. Der Bildschirm erscheint zwar überfüllt und kompliziert, aber Visual Basic ist ganz leicht zu verstehen.

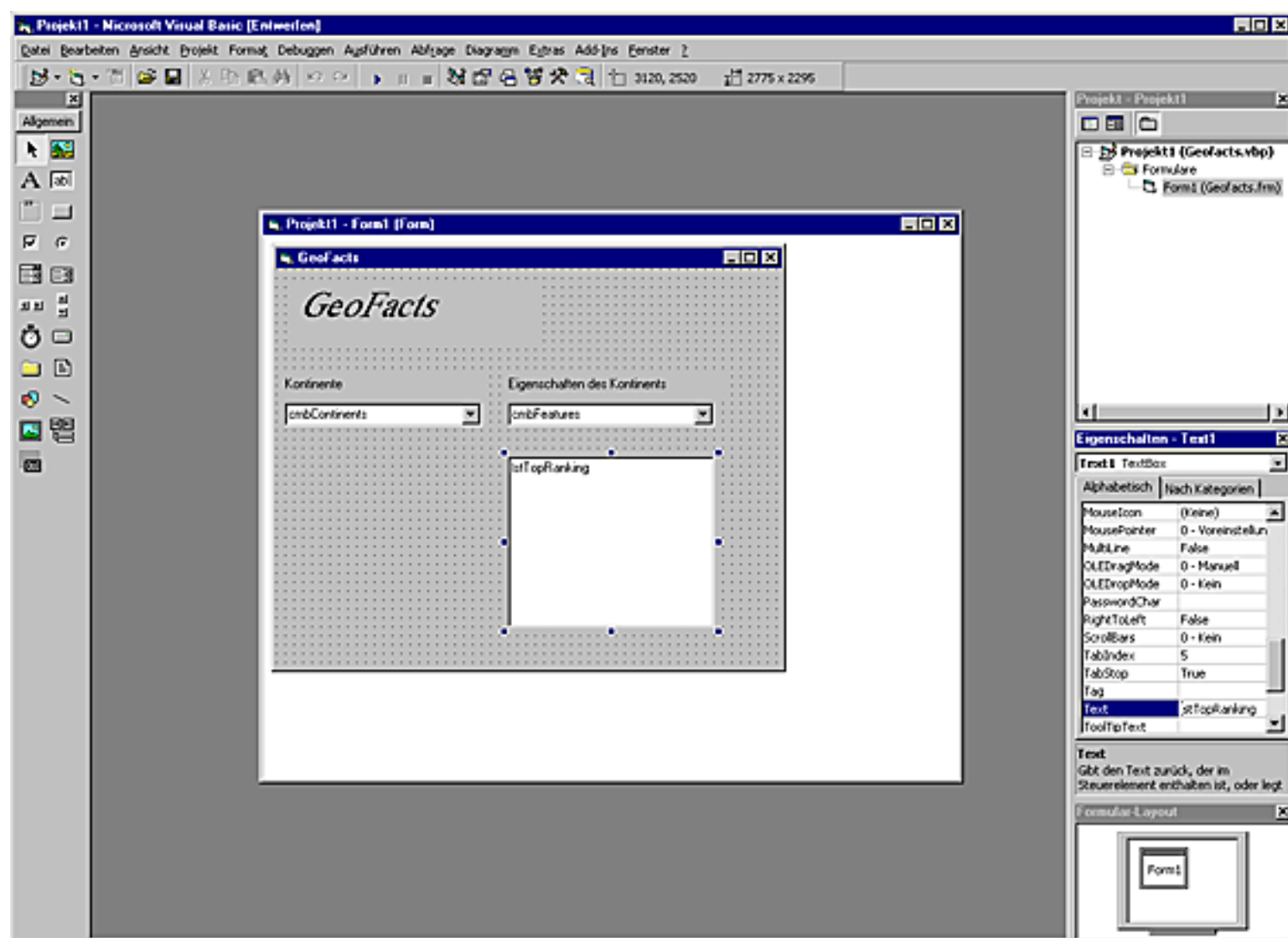
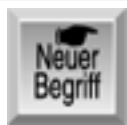


Abbildung 1.1:
Die Programmieroberfläche von Visual Basic scheint auf den ersten Blick kompliziert, ist aber einfach anzuwenden.

Visual Basic ist nicht nur grafisch und einfach zu bedienen, sondern auch eine der gebräuchlichsten Programmiersprachen, weil es sowohl interpretiert als auch kompiliert ist! Sie können ein Visual-Basic-Programm während der Entwicklung testen, indem Sie es interpretiert ausführen, bis alle *Bugs* entfernt sind. Nachdem Sie Ihr Programm sorgfältig getestet haben, kompilieren Sie es zu einer

schnellen und sicheren (niemand kann mehr Änderungen am Programm vornehmen) ausführbaren Programmdatei, die Sie an andere weitergeben können.

Dieser Kompilierungsprozeß erfolgt unter Visual Basic über eine einfache Menüauswahl. Visual Basic übernimmt die schwierigeren Schritte der Kompilierung für Sie (auch das sogenannte *Link-Editing*), für die Sie in anderen Programmiersprachen selbst verantwortlich sind.



Ein *Bug* ist ein Programmfehler. Wenn das Programm, das Sie entwickeln, nicht richtig funktioniert, müssen Sie ein *Debugging* ausführen, um alle Bugs zu entfernen.

Zu dem Zeitpunkt, als Microsoft die erste Version von Visual Basic veröffentlichte, sagten viele Leute den Untergang der Sprache BASIC (und ihrer Nachkommen, wie beispielsweise QBasic) voraus. Diese Schwarzmaler dachten, daß keine Sprache, die auf BASIC basiert sei, für ernsthafte Programme verwendet werden könnte, weil sie BASIC nie als ernsthafte Sprache betrachtet hatten. Sprachen wie C, C++ oder Pascal waren angesagt, weil sie kompilierbar waren und ihre Programmstruktur besser für eine Windows-Umgebung geeignet war. Mit Visual Basic lehrte Microsoft der Programmiergemeinschaft die folgenden Lektionen:

- Eine BASIC-ähnliche Sprache kann sowohl leicht verständlich als auch leistungsfähig sein.
- Mit der richtigen Oberfläche ist auch eine BASIC-ähnliche Sprache für eine Windows-Umgebung geeignet.
- Visual Basic kann sowohl interpretiert als auch kompiliert arbeiten, abhängig davon, was der Programmierer gerade braucht.
- Statt zu veralten, kann eine auf BASIC basierende Sprache zu einer der weit verbreitetsten Sprachen der ganzen Welt werden.

Die visuelle Erscheinung von Visual Basic

Wie Sie in Abbildung 1.1 gesehen haben, ist Visual Basic 6 mehr als nur eine Programmiersprache. Das Geheimnis von Visual Basic ist in seinem Namen ausgedrückt: *visual*. Bei den heutigen Windows-Betriebssystemen muß ein Programm in der Lage sein, mit dem Bildschirm, der Tastatur, der Maus und dem Drucker grafisch umzugehen. Ältere Programmiersprachen, wie beispielsweise BASIC, funktionierten in einer textbasierten Umgebung ausgezeichnet, aber sie unterstützen die grafische Benutzeroberfläche nicht, die für die heutigen Computer unabdingbar ist.

Im ersten Teil dieses Buchs werden Sie nicht viel über die Programmiersprache Visual Basic erfahren, weil bei der Programmierung unter Visual Basic hauptsächlich die Arbeit mit der visuellen Umgebung eine Rolle spielt. Sie brauchen nur wenig Kenntnisse über die Programmiersprache, um funktionierende Programme zu entwickeln. Erst wenn Sie komplexere Programme schreiben, brauchen Sie mehr Erfahrung mit der Sprache Visual Basic, als nur die paar Befehle, die Sie in den ersten Tagen hier kennenlernen werden.



Visual Basic ist nicht nur deshalb so einfach zu erlernen und zu benutzen, weil ihm die Programmiersprache BASIC zugrunde liegt. Ein Großteil der Programmentwicklung besteht darin, Elemente mit der Maus auf den Visual-Basic-Bildschirm zu ziehen. Statt komplizierte Eingabe- und Ausgabeanweisungen zu schreiben, um mit den Benutzern zu kommunizieren, verschieben Sie einfach ein paar Steuerelemente auf dem Bildschirm, beispielsweise Textfelder und Befehlsschaltflächen. Visual Basic sorgt dafür, daß die Steuerelemente bei der Ausführung des Programms richtig funktionieren.



Ein *Anwender* ist eine Person, die ein Programm benutzt. Sie, der Programmierer, der ein Programm entwickelt, sind ebenfalls ein Anwender, weil Sie Programme verwenden, die Sie und andere schreiben. Das Programmiersystem Visual Basic ist nichts anderes als ein Programm, das Sie verwenden, um andere Programme zu schreiben.

Visual Basic gibt es in mehreren Varianten, unter anderem:

- Visual Basic Standard Edition: Die klassische Einstiegsversion, mit der Sie schon die meisten Übungen in diesem Buch vollziehen können.
- Visual Basic Enterprise Edition: Diese Variante wurde für Team-Programmierungsumgebungen und Client/Server-Programmierung entwickelt, wobei die Anwendungen die Verarbeitung und die Daten auf mehrere Computer verteilen.
- Visual Basic Professional Edition: Diese Variante ist für professionelle Programmierer vorgesehen, die die Visual-Basic-Programmierungsumgebung so gut wie möglich nutzen wollen. Sie enthält alle Werkzeuge und Assistenten, die dem Programmierer helfen, Anwendungen zu verpacken und an die Benutzer weiterzugeben und ist damit die optimale Version, diesen Ihnen vorliegenden Kurs Schritt für Schritt durchzuarbeiten.



Es gibt noch eine spezielle Version von Visual Basic, *Visual Studio*. Visual Studio ist eine Programmierungsumgebung, die mehrere Microsoft-Sprachen unterstützt, unter anderem Visual Basic, Visual C++ und Visual J++. Wenn Sie Visual Basic einsetzen, können Sie dieselbe Umgebung benutzen, die die Benutzer dieser anderen Programmiersprachen einsetzen. Wenn Sie irgendwann auf eine andere Programmiersprache umsteigen wollen, brauchen Sie sich nicht an neue Menüs und Dialogfelder zu gewöhnen.

Warum schreibt man Programme?

Die meisten Computeranwender werden nie in ihrem Leben Programme schreiben. Einige Leute kaufen alle ihre Programme im Laden oder bestellen sie über Versandgeschäfte, und sie brauchen nie irgendwelche Anpassungen für ihre speziellen Bedürfnisse. Aber für ganz bestimmte Aufgaben findet man manchmal kein Programm von der Stange, insbesondere, wenn Sie Ihren Computer in Ihrem Geschäft oder für wissenschaftliche Forschungen einsetzen. Oder vielleicht haben Sie das Konzept für ein neues Spiel im Kopf, das Sie zum Bestseller der Computerspiele machen möchten, damit Sie sich irgendwann in der Südsee zur Ruhe setzen können. Wenn Sie eine spezielle Anwendung brauchen, aber im Laden nichts geeignetes finden, oder wenn Sie einfach nur neue Programme entwickeln möchten, dann müssen Sie diese Programme mit einer Programmiersprache wie Visual Basic entwerfen und schreiben.



Sie wissen, daß Sie dem Computer nicht einfach mitteilen können, was er zu tun hat, und dann erwarten, daß er es auch macht. Ein Computer braucht eine detaillierte Anweisungsliste, weil er letztlich eine dumme Maschine ist, die nichts selbständig erledigen kann. Sie übergeben Ihrem Computer diese Anweisungen in Form eines Programms. Ein Visual-Basic-Programm besteht aus Programmcode (ähnlich dem in Listing 1.1) und grafischen Elementen, die den Bildschirm und die Windows-Steuerelemente definieren, mit denen der Benutzer bei der Programmausführung arbeitet.



Wenn Sie Visual Basic lernen, lernen Sie gleichzeitig, wie man gebräuchliche Anwendungsprogramme automatisiert, beispielsweise Programme aus Microsoft Office. Microsoft Office besteht aus mehreren Programmen, die zusammenarbeiten können, beispielsweise eine Textverarbeitung, eine Tabellenkalkulation und ein Datenbankprogramm. Microsoft Office enthält außerdem die vollständige Programmiersprache Visual Basic 6, mit der Sie Microsoft-Office-Anwendungen automatisieren können. (Microsoft Office 95, die Version vor Microsoft Office 97, enthält *Visual Basic für Applikationen* (VBA), das ähnlich aber nicht voll kompatibel zur Version Visual Basic 6 ist.) Sie könnten beispielsweise die Buchungsprozeduren am Monatsende automatisieren, indem Sie ein Programm schreiben, das die Excel-Tabellenkalkulationen für das Monatsende automatisch ausführt. Das Visual Basic, das in diesen Anwendungen enthalten ist, ist nicht das vollständige Entwicklungssystem Visual Basic, das Sie mit Visual Basic 6 erhalten, aber es enthält die vollständige Sprachdefinition, so daß Sie die Anwendungen steuern können.

Die Programmierung

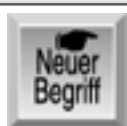
Mit der Zeit werden Sie eine eigene Methode entwickeln, Programme zu schreiben, die für Ihre Bedürfnisse am besten geeignet sind. Nichtsdestotrotz sollten Sie bei der Entwicklung Ihrer Visual-Basic-Programme den folgenden, standardisierten Schritten folgen:

1. Legen Sie fest, was Ihre Anwendung tun soll, indem Sie einen vollständigen Entwurf anfertigen.
2. Legen Sie die Oberfläche Ihrer Anwendung fest (die Bildschirme und Menüs, die Sie Ihren Benutzern anbieten).
3. Fügen Sie den Programmcode ein, um die visuellen Elemente zu verknüpfen und das Programm zu automatisieren.
4. Testen Sie Ihr Programm, um alle Bugs zu erkennen und zu entfernen.
5. Kompilieren Sie Ihr getestetes Programm und geben Sie es an Ihre Benutzer weiter.

Was Sie tun sollten	Was Sie lassen sollten
Testen Sie Ihr Programm, um alle Bugs zu entfernen, und geben Sie es dann erst an andere weiter. Es ist fast unmöglich, alle Bugs zu erkennen, aber Sie sollten so lange wie möglich testen und alle Programmfunktionen ausprobieren, um sicherzustellen, daß Sie vor dem Kompilieren und der Weitergabe des Programms so viele Fehler wie möglich eliminieren.	Seien Sie nicht erstaunt, wenn ein Anwender noch einen Fehler (oder sogar mehrere) findet. Auch der sorgfältigste Test garantiert nicht, daß alle Fehler erkannt wurden. Je mehr ein Programm leistet, desto wahrscheinlicher ist es, daß irgendwann ein Fehler auftaucht, und zwar dann, wenn Sie und Ihre Benutzer es am wenigsten erwarten.

Sie können den Testprozeß beschleunigen, indem Sie mit der Kompilierung des Programms warten, bis es sorgfältig getestet wurde. Wenn Sie Ihr Programm interaktiv testen, finden Sie die Bugs schneller und einfacher und können entsprechende Maßnahmen ergreifen. Visual Basic beinhaltet ein spezielles

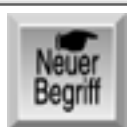
Hilfssystem, einen sogenannten *Debugger*, der Ihnen hilft, Bugs zu finden, die während des Testens auftreten. In Kapitel 21 erfahren Sie mehr über die Verwendung des Debuggers.



Ein *Debugger* ist ein interaktives Überwachungssystem, das Sie innerhalb von Visual Basic aktivieren oder deaktivieren, um fehlerhafte Anweisungen zu erkennen. Wenn Sie beispielsweise ein Programm ausführen, das Sie gerade entwickeln, und es berechnet einen fehlerhaften Betrag, kann Ihnen der Debugger helfen, die Anweisung im Programm zu finden, wo der Fehler entsteht.

Vor der Einführung von Visual Basic war es mühevoll, ein Programm zu schreiben. Das hatte mehrere Gründe. In einer textbasierten Umgebung mußte man alle Bildschirme, die dem Benutzer präsentiert werden sollten, auf Papier entwerfen. Anschließend trugen Sie diese Skizzen zu den Anwendern, um zu erfahren, ob das genau das war, was sie brauchten. Wenn Sie ein Programm entwickelten, das sehr viele Benutzer ansprechen sollte, wie beispielsweise ein Spiel oder eine allgemeine Geschäftsanwendung, mußten Sie trotzdem alle Bildschirme skizzieren, komplizierte Datenflußdiagramme zeichnen, die Dateien anlegen, die das Programm brauchte, und überhaupt jedes Detail berücksichtigen, bevor Sie nur irgendeine Taste drücken konnten.

Durch die visuelle Natur von Visual Basic werden Sie ermutigt, die Tastatur schon sehr viel früher einzusetzen. Statt auf Papier entwerfen Sie die Bildschirme mit den visuellen Werkzeugen von Visual Basic. Abbildung 1.2 zeigt einen solchen Bildschirm. Sie brauchen keinerlei Code, um diesen Bildschirm zu erstellen. Sie ziehen einfach die verschiedenen Steuerelemente in das *Formularfenster*.



Das *Formularfenster*, auch als *Formular* bezeichnet, bildet den Bildschirmhintergrund für ein Visual-Basic-Programm. Es enthält Elemente wie beispielsweise Befehlsschaltflächen und Bildlaufleisten. Programme setzen sich aus einem oder mehreren Formularfenstern zusammen, abhängig davon, was sie leisten und wie komplex sie sind.

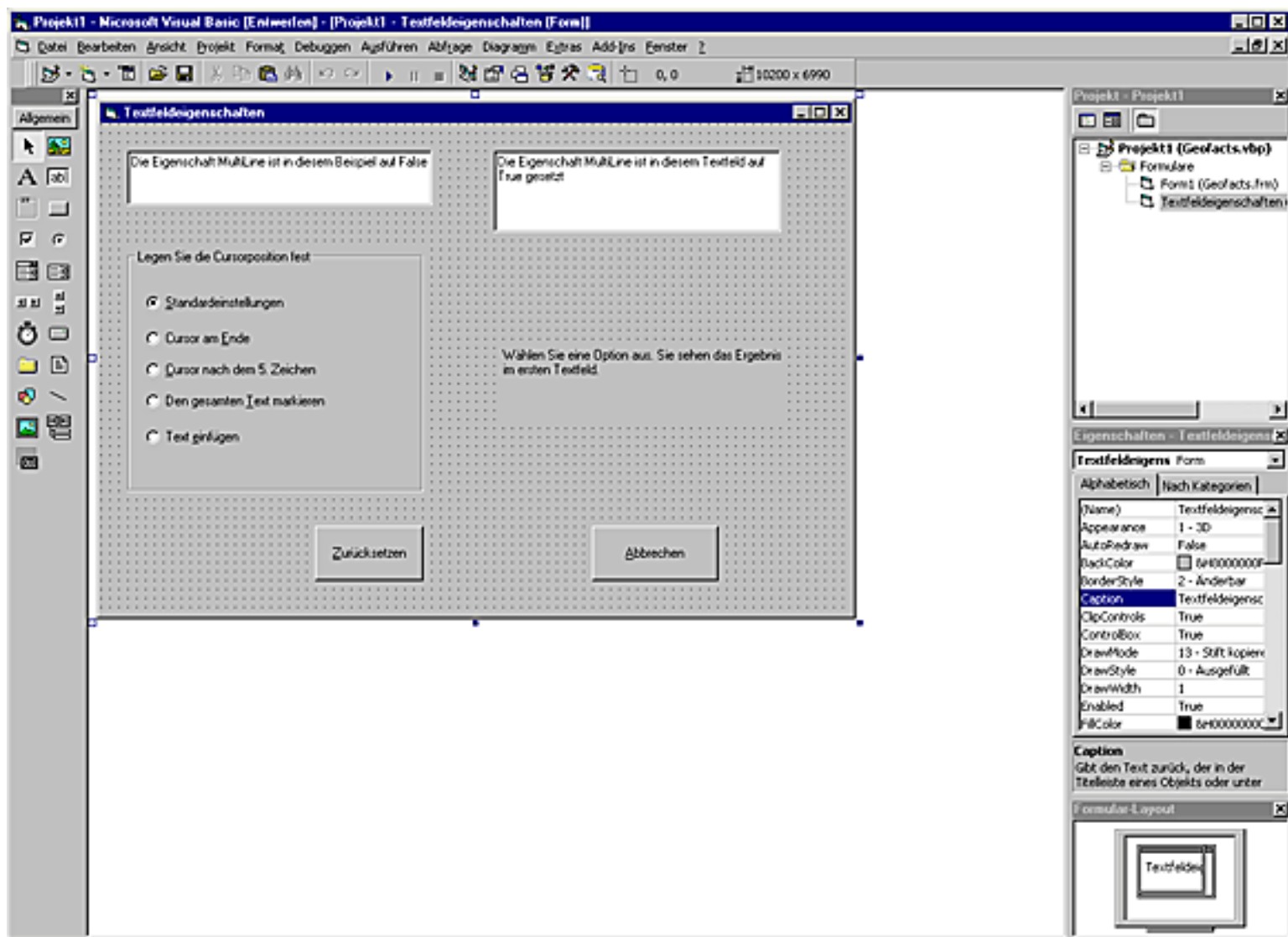


Abbildung 1.2:
Visual Basic ermöglicht Ihnen, Oberflächen während der Programmierung zu entwerfen und anzulegen.

Noch bevor Sie irgendwelchen Code schreiben, können Sie Ihre Programmbildschirme testen (jedes Formular ist die Grundlage für einen Bildschirm), weil Ihnen Visual Basic erlaubt, Ihr Programm interaktiv auszuführen, nachdem Sie mindestens ein Formular angelegt haben. Sie überzeugen sich, daß Ihre Bildschirme gut aussehen, und zeigen Ihren *Prototyp* dann den Benutzern, die das Programm bestellt haben, um sicherzustellen, daß sie mit Ihrer Arbeit zufrieden sind. Änderungen an diesem Prototyp sind in der Phase vor der Programmierung viel einfacher als nach der Codierung. Diese Prototyp-Funktion von Visual Basic ist eine der Methoden, wie Ihnen Visual Basic hilft, Ihre Programme schnell und präzise zu entwickeln.

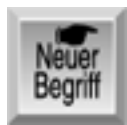
	Ein <i>Prototyp</i> ist ein Testprogramm, das fast keine Funktionalität beinhaltet, aber einige oder alle Bildschirme demonstriert, die das endgültige Programm anbieten wird. Sie und die Benutzer können den Prototyp testen, um herauszufinden, ob alle benötigten Elemente vorhanden sind.



Nachdem Sie Ihr Programm entwickelt, getestet, kompiliert und an die Benutzer weitergegeben haben, ist es immer noch möglich, Änderungen vorzunehmen. Das ist allerdings sehr mühsam. Außerdem müssen Sie alle Anwendungsdateien noch einmal an die Benutzer weitergeben. Je früher Sie Probleme erkennen, desto einfacher sind sie zu beheben.

Die Programmwartung

Fehler sind nicht die einzige Ursache, warum Sie nach der Fertigstellung eines Programms weiter daran arbeiten sollten. Die *Programmwartung* ist wichtig, weil sich Anforderungen, Firmensituationen und Gesetze ändern. Sie müssen die Programme so modifizieren, daß sie überlebensfähig bleiben; dazu ist es erforderlich, sie in regelmäßigen Zeitabständen zu aktualisieren, um die erforderlichen Änderungen vorzunehmen. Und auch die Benutzer stellen immer wieder neue Anforderungen an das Programm.



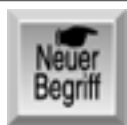
Der Begriff *Programmwartung* bezeichnet die Aktualisierung eines Programms, nachdem es an die Benutzer ausgeliefert ist. Diese Aktualisierung kann aufgrund einer Anforderung von Seiten der Benutzer erfolgen, aber auch, wenn sich Dinge ändern, die dem Programm zugrundeliegen.

Man sagt, ein Programm wird einmal geschrieben und oft geändert. Je mehr Programmwartung Sie betreiben, desto wahrscheinlicher bleibt Ihr Programm aktuell und wird eingesetzt. Möglicherweise ist es sinnvoll, den Benutzern die neuen Versionsnummern auf dem Eröffnungsbildschirm anzuzeigen, so daß sie immer sofort erkennen, welche Version auf ihrem System installiert ist.



Dokumentieren Sie Ihre Programme so, daß auch andere Programmierer Ihren Code verstehen, wenn sie später Änderungen daran vornehmen müssen.

Wenn Sie mehr über Visual Basic erfahren haben, werden Sie lernen, wie man sauberen Code schreibt, und wie man eine *Dokumentation* für ein Programm anlegt. Je mehr *Kommentare* Sie in Ihrem Programm unterbringen, und je klarer Sie den Programmcode schreiben und dabei auf mühsame, komplizierte und trickreiche Programmanweisungen verzichten, desto einfacher wird es Ihnen und anderen fallen, Fehler zu erkennen und das Programm zu warten.



Die *Dokumentation* umfaßt alle Beschreibungen des Programms. Sie können die Dokumentation innerhalb des Programms ablegen, so daß Sie (oder ein anderer) bei einer späteren Änderung lesen können, welche Abschnitte für welche Aufgaben vorgesehen sind, ohne erst mühselig herausfinden zu müssen, was der Code eigentlich macht. Interne Programmbeschreibungen werden auch als *Anmerkungen* oder *Kommentare* bezeichnet.

Schreiben Sie die Kommentare unmittelbar bei der Programmierung, weil Sie das Programm zu diesem Zeitpunkt noch am besten verstehen. Wenn Sie warten, bis eine Anwendung komplett fertig ist, wie es viele Programmierer tun, dann wird Ihr Programm voraussichtlich nie richtig dokumentiert, weil andere Projekte schon wieder Vorrang haben und die Dokumentation häufig beiseite geschoben wird, bis ein anderes Projekt abgeschlossen ist.

Darüber hinaus sollten Sie auch eine externe Dokumentation anlegen, mit Screenshots der verschiedenen Programmbildschirme und Beschreibungen, was der Benutzer tun muß, um das Programm aufzurufen, zu bedienen und zu beenden. Je besser die Dokumentation für den Benutzer ist, desto eher beherrscht er Ihr Programm und will dann auch weitere Programme von Ihnen haben.

Ihr erstes Programm

Falls Sie bereits andere Windows-Produkte kennen, beispielsweise den Microsoft Publisher, dann kennen Sie die *Assistenten*, die Ihnen helfen, Ihre Dokumente anzulegen. Auch Visual Basic unterstützt die Assistenten-Technologie, die Ihnen bei der Programmierung helfen soll. Bei der Entwicklung eines Visual-Basic-Programms haben Sie die Wahl, eine Anwendung von Grund auf neu zu schreiben, oder einen Assistenten aufzurufen, der das Gerüst oder die allgemeine Struktur der Anwendung für Sie aufbaut. Nachdem der Assistent das Gerüst für die Anwendung angelegt hat, ergänzen Sie die Details.



Ein *Assistent* stellt Ihnen eine Reihe von Fragen, auf die Sie antworten. Anhand dieser Antworten erzeugt der Assistent eine Anwendung, die den vorgegebenen Kriterien entspricht. Visual Basic beinhaltet mehrere Assistenten, aber am häufigsten werden Sie voraussichtlich den sogenannten *Anwendungs-Assistenten* benutzen.

Manchmal ist es schwierig zu entscheiden, ob man mit Hilfe des Anwendungs-Assistenten ein Anwendungsgerüst anlegen und die Details für die spezielle Situation später eintragen soll, oder ob man eine Anwendung von Grund auf neu entwickeln soll. Einige Programmierer legen sich, nachdem sie bereits eine Anwendung entwickelt haben, die mit der neuen vergleichbar ist, eine Kopie dieser ersten Anwendung an und ändern sie entsprechend ab. Mit der Zeit werden Sie lernen, was für die jeweilige Situation am besten geeignet ist.

Für die ersten Erfahrungen mit Visual Basic zeigt Ihnen dieser Abschnitt, wie Sie Ihre allererste Anwendung schreiben. Sie werden sehen, wie einfach es ist, mit dem Anwendungs-Assistenten ein Programmgerüst zu entwickeln. Die resultierende Anwendung wird zwar noch nicht viel leisten (schließlich handelt es sich nur um ein Gerüst), aber Sie werden sehen, wie viel Visual Basic automatisch für Sie tut, wenn Sie den Anwendungs-Assistenten nutzen. In der morgigen Lektion werden Sie lernen, wie Sie eine Anwendung ohne den Anwendungs-Assistenten von Grund auf neu entwickeln.



Erstaunlicherweise werden Sie sehr wahrscheinlich mehr Anwendungen von Grund auf neu schreiben, als den Anwendungs-Assistenten zu verwenden, oder eine Kopie eines ähnlichen Programms anzulegen und die entsprechenden Änderungen vorzunehmen. Der Anwendungs-Assistent erzeugt zwar ein vollfunktionales Programmgerüst, aber Sie entwickeln mit der Zeit einen eigenen Programmierstil, so daß es einfacher wird, eine Kopie eines existierenden Programms abzuändern, als zuerst ein Gerüst anzulegen und dieses dann zu ergänzen. Welchen Stil Sie sich angewöhnen, wird sich später entscheiden. Lehnen Sie sich einfach zurück und genießen Sie es, Visual Basic kennenzulernen. Probieren Sie alles aus, haben Sie keine Angst, etwas kaputt zu machen, und seien Sie auf Fehler gefaßt, wenn Sie ein Programm schreiben. Programmieren ist eine kreative Arbeit, und Sie werden feststellen, daß sie mit Visual Basic sehr viel Spaß macht.

Beim Starten von Visual Basic steht sofort der Anwendungs-Assistent zur Verfügung, um Ihnen zu

helfen. Wenn Sie Visual Basic vom Startmenü von Windows aufrufen, erscheint das in Abbildung 1.3 gezeigte Dialogfeld **Neues Projekt**. Die Registerkarten im Dialogfeld **Neues Projekt** bieten die folgenden Auswahlen:

- Mit **Neu** legen Sie eine neue Anwendung an, entweder mit Hilfe verschiedener Assistenten oder von Grund auf neu.
- Mit **Vorhanden** wählen Sie ein existierende Visual-Basic-Projekt aus und öffnen es.
- Mit **Aktuell** zeigen Sie eine Liste der Visual-Basic-Projekte an, die Sie vor kurzem geöffnet oder angelegt haben.

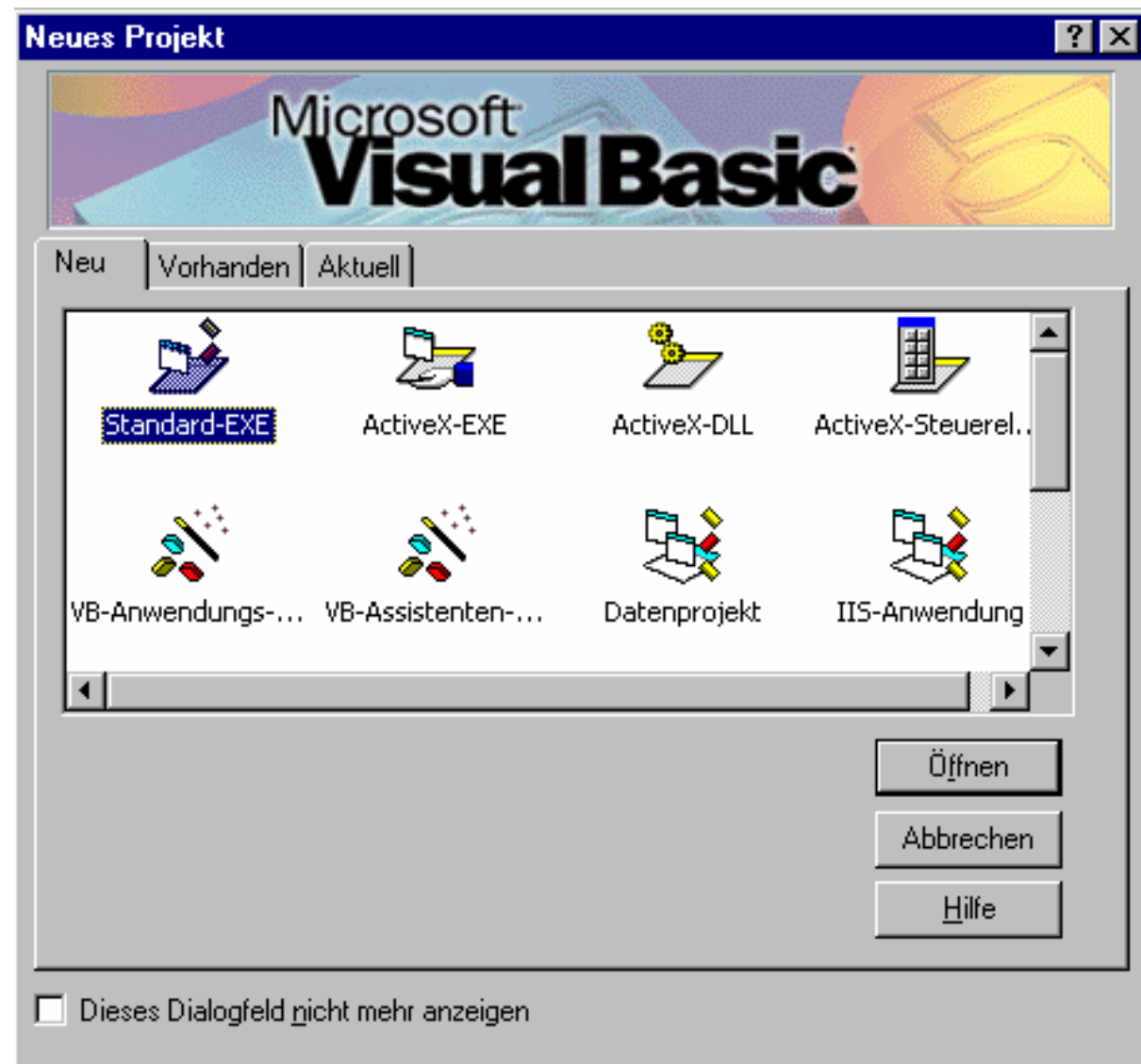
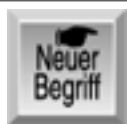


Abbildung 1.3:
Im Dialogfeld Neues Projekt können Sie den Anwendungs-Assistenten aufrufen.

Falls Sie das Dialogfeld **Neues Projekt** schon geschlossen haben und den Anwendungs-Assistenten später ausführen möchten, wählen Sie **Datei, Neues Projekt**, um das Dialogfeld **Neues Projekt** wieder anzuzeigen. Dieses Dialogfeld **Neues Projekt** zeigt jedoch die Registerkarten **Vorhanden** und **Aktuell** nicht an, weil Sie durch Ihre Menüauswahl bereits festlegen, daß Sie ein neues Projekt anlegen möchten.



Ein *Projekt* ist der Container für die Dateien, aus denen sich Ihre Anwendung zusammensetzt. Eine einzelne Anwendung kann mehrere Dateien umfassen, und das Projekt nimmt diese Dateien auf. Eine oder mehrere dieser Dateien können Code enthalten, oder Bildschirmdefinitionen mit den zugehörigen Formularfenstern. Eine oder mehrere der Dateien können komplexe Programminformationen enthalten, mit deren Hilfe Ihr Programm mit anderen Programmen und Modulen im Betriebssystem kommuniziert.

Was Sie tun sollten

Wenn das Dialogfeld **Neues Projekt** nicht bei jedem Starten von Visual Basic angezeigt werden soll, markieren Sie das Kontrollkästchen **Dieses Dialogfeld nicht mehr anzeigen**. Beim nächsten Aufruf von Visual Basic wird das Dialogfeld **Neues Projekt** nicht mehr angezeigt.

Wenn Sie auf der Registerkarte **Neu** das Icon **VB Anwendungs-Assistent** anklicken, beginnt der Assistent seine Arbeit. Als erstes teilt Ihnen eine Meldung auf dem Bildschirm mit, daß der Assistent jetzt die Arbeit aufnimmt. (Darüber hinaus können Sie auf diesem Bildschirm ein Profil laden, das Optionen definiert, aber Sie brauchen keine zusätzlichen Profile, um die Aufgaben in diesem Buch nachzuvollziehen.) Wie bei den meisten Assistenten klickt man auch hier auf die Schaltfläche **Weiter**, um den nächsten Bildschirm anzuzeigen. Abbildung 1.4 zeigt den nächsten Bildschirm des Assistenten, wo Sie den Schnittstellentyp auswählen.



Abbildung 1.4:
Der Schnittstellentyp bestimmt, wie Ihre Anwendung mehrere Fenster verwaltet.

Die folgenden Auswahlen stehen zur Verfügung:

- **MDI (Multiple Document Interface)** ermöglicht, in einer Anwendung mehrere

Dokumentfenster zu verwenden. Diese Oberfläche erlaubt, unterschiedliche Datenmengen in verschiedenen Fenstern Ihres Programms zu verarbeiten. Die einzelnen Dokumentfenster werden auch als *Kindfenster* bezeichnet.

- **SDI (Single Document Interface)** begrenzt Ihre Anwendung auf jeweils ein geöffnetes Dokumentfenster. Die meisten Ihrer Anwendungen sind voraussichtlich SDI-Anwendungen.
- **In der Art des Explorers** ermöglicht Ihnen, Ihrer Anwendung eine ähnliche Oberfläche zu geben, wie der Windows-Explorer sie benutzt. Dabei werden im linken Feld Themenüberblicke angezeigt, im rechten Feld Details zu einem ausgewählten Thema.

Klicken Sie eine dieser drei Optionen an. Sie sehen eine kurze Beschreibung und eine kleine Skizze eines Beispiel-Programmfensters. Viele Ihrer Anwendungen verwenden die SDI-Oberfläche, weil sie jeweils nur ein einziges Fenster mit Daten brauchen. Für unser erstes Beispiel verwenden Sie die Option SDI.

Auf dem Assistenten-Bildschirm geben Sie jetzt den Namen Ihres Projekts ein. Der Standardname, *Project1*, läßt sehr zu wünschen übrig. Geben Sie dem Projekt also den Namen *ErstesProgramm* (es sind keine Leerzeichen erlaubt). Klicken Sie auf **Weiter**, um das nächste Fenster des Assistenten anzuzeigen, wie in Abbildung 1.5 gezeigt.

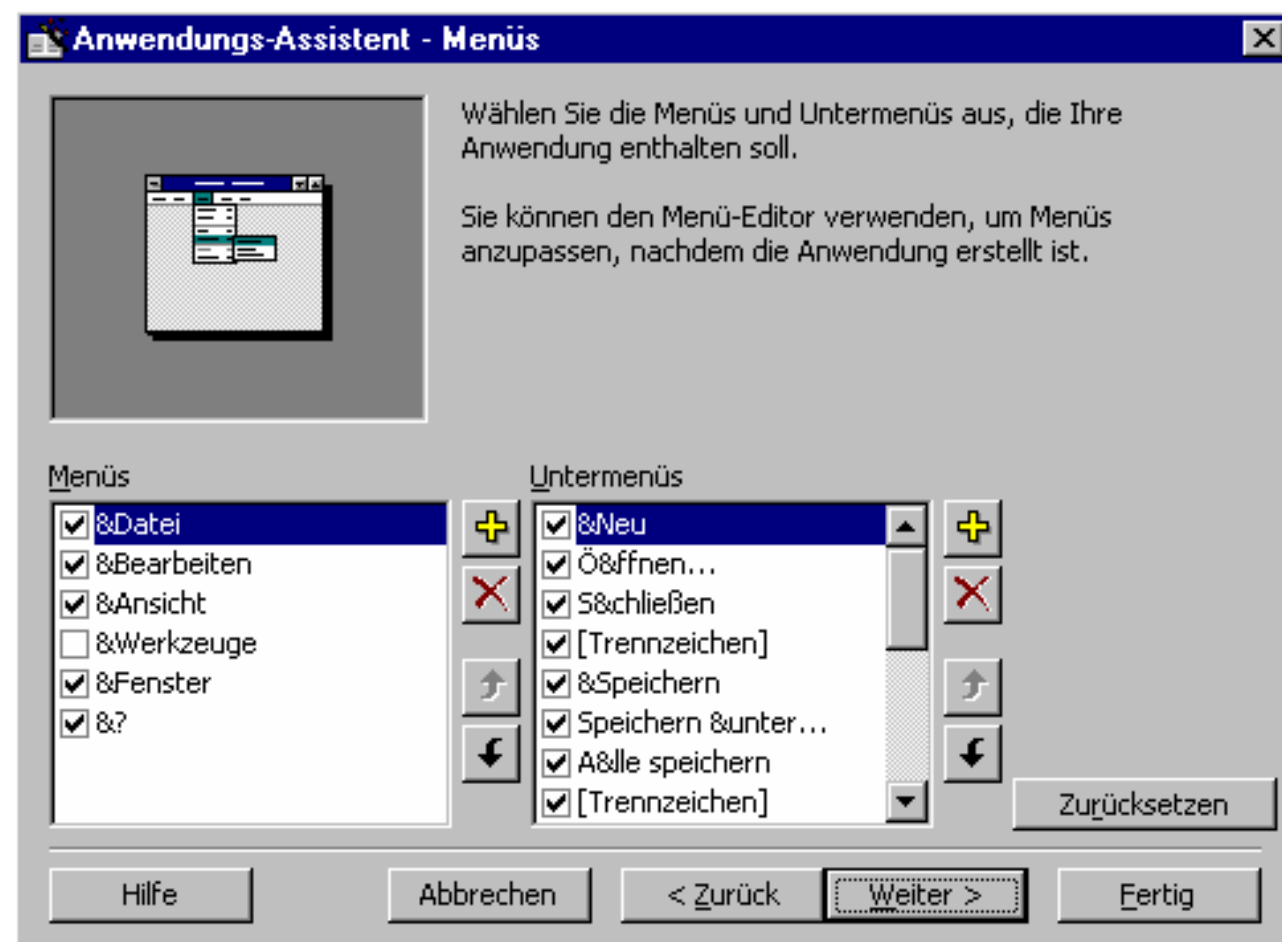


Abbildung 1.5:
Wählen Sie die Optionen aus, die das Menü Ihres Programms enthalten soll.

Der Anwendungs-Assistent fügt die hier ausgewählten Menüeinträge in Ihr Programmenü ein. Es handelt sich dabei um gebräuchliche Windows-Optionen, die man in den meisten Windows-Programmen findet. Die Menüs werden in der typischen Dropdown-Darstellung von Windows implementiert. Hier legen Sie fest, welche Einträge die Menüleiste haben soll (beispielsweise **Datei**, **Bearbeiten** usw.), ebenso wie die Untermenüeinträge, wie etwa **Neu**, **Öffnen** und **Beenden**. Wenn neben einem Buchstaben im Namen eines Menüeintrags das Ampersand-Zeichen

(&) steht, wird der betreffende Buchstabe unterstrichen dargestellt und stellt das Shortcut-Zeichen dar. Mit anderen Worten, &Neu bedeutet, daß im Menü der Eintrag **Neu** angezeigt wird (achten Sie auf den Unterstrich). Der Benutzer drückt (Alt)+(N) Wenn Sie ein Ampersand im Namen anzeigen wollen, geben Sie es doppelt ein. A&&B beispielsweise erzeugt **A&B**. In diesem Programm behalten Sie für alle Optionen die Vorgabewerte bei (die Optionen **Datei**, **Bearbeiten**, **Ansicht**, **Fenster** und **Hilfe** sind markiert) und klicken auf **Weiter**.



Nachdem der Anwendungs-Assistent Ihr Programm angelegt hat, funktionieren die Menüoptionen genau so, wie man es von ihnen erwartet. Das **Datei**-Menü beispielsweise erscheint, wenn Sie (Alt)+(D) drücken oder auf den Menüeintrag **Datei** klicken.

Auf dem nächsten Bildschirm des Assistenten, den Sie in Abbildung 1.6 sehen, wählen Sie aus, welche Schaltflächen in der Symbolleiste angezeigt werden. Wie Sie sehen, nimmt Ihnen der Assistent sehr viel Arbeit ab. Durch die Bereitstellung einer zunächst einfachen Symbolleiste erspart er Ihnen viel Zeit und Mühe. Im linken Feld des Fensters sehen Sie, welche Schaltflächen für die Symbolleiste bereitstehen. Im rechten Feld werden die Schaltflächen angezeigt, die Sie für Ihre Symbolleiste ausgesucht haben (ebenso wie die Trennleisten zwischen den Schaltflächen). Klicken Sie auf **Weiter**, um die Vorgabewerte zu übernehmen.



Abbildung 1.6:
Der Anwendungs-Assistent spart Ihnen eine Menge Zeit - zum Beispiel beim Anlegen einer Symbolleiste

Auf dem nächsten Bildschirm werden die Ressourcen angezeigt, die Sie in Ihrem Programm nutzen können, beispielsweise mehrsprachige Textdateien. Für einfache Programme sind solche Ressourcen häufig gar nicht nötig, In diesem Beispiel behalten wir die Auswahl **Nein** bei und klicken auf die Schaltfläche **Weiter**.

Auf dem nächsten Bildschirm geht es um die Verbindung zum Internet. Sie können Ihrem Programm eine Schnittstelle zum Internet mitgeben. Wenn Sie in diesem Fenster die Option **Ja** auswählen (was Sie hier nicht machen sollten!), fügt der Anwendungs- Assistent einen vollständigen Internet-Browser in Ihr Programm ein, der sich ähnlich verhält wie der Internet Explorer. Der Benutzer kann auf diese Weise auf das Internet zugreifen, ohne daß Sie etwas dazutun müssen. Wenn er eine Internet-Adresse eingibt (auch als *URL*, *Uniform Resource Locator*, bezeichnet), wie beispielsweise `http://www.mut.com`, zeigt der Browser diese Web-Seite im Browser-Fenster der Anwendung an.

Dazu wird der Benutzer gegebenenfalls mit dem Standard-Internet-Dienst des PC am Internet angemeldet. Sie können eine Standardadresse für die Seite angeben, die beim Starten des Browsers angezeigt werden soll.



Wenn Sie Ihrer Anwendung den Browser hinzufügen, setzen Sie damit voraus, daß der Benutzer Internet-Zugriff hat. Falls nicht, wird ein Fehler angezeigt, wenn der Benutzer versucht, den Browser zu benutzen.

In dieser ersten Anwendung brauchen wir keinen Internet-Zugriff, klicken Sie also auf **Weiter**, ohne die Standardoptionen zu ändern. Auf dem nächsten Bildschirm erhalten Sie die Möglichkeit, Ihrer Anwendung die folgenden Standardformulare hinzuzufügen:

- Der **Eröffnungsbildschirm** zeigt beim Programmstart den Namen der Anwendung an.
- Der **Anmeldedialog** ist ein Dialogfeld, das die ID und das Kennwort des Benutzers abfragt, falls Ihre Anwendung diese Sicherheitsmaßnahmen verwendet.
- Der **Optionsdialog** ist ein leeres Dialogfeld mit Registerkarten, wo Ihre Benutzer Attribute auswählen können, die Sie für die Anwendung zur Verfügung stellen.
- Das **Info-Feld** ist ein Dialogfeld, das erscheint, wenn Ihr Benutzer im Anwendungsmenü die Einträge **Hilfe**, **Info** auswählt.

Für dieses erste Programm klicken Sie auf die Option **Info-Feld**.

Nachdem Sie das Standardformular **Info-Feld** ausgewählt haben, klicken Sie auf **Weiter**, um den Bildschirm des Assistenten zu überspringen, auf dem externe Datenbankdateien für die Anwendung ausgewählt werden. Klicken Sie auf die Schaltfläche **Fertig**. Visual Basic legt das Gerüst für Ihr erstes Programm an.



Die Schaltfläche **Formularvorlagen** ermöglicht eine Auswahl aus verschiedenen *Formularschablonen*, die sich im Template-Ordner von Visual Basic befinden. Unter anderem gibt es eine Add-In-Schablone, mit der Sie ein Formular aus Ihrer eigenen Bibliothek, einfügen können, ein ODBC- Anmelde-Formular, das den Benutzern einen erweiterten Datenbankzugriff ermöglicht, oder Tips&Tricks, womit ein zufällig ausgewählter Tip angezeigt wird, wenn Ihr Benutzer das Programm aufruft.



Eine *Formularvorlage* ist ein Modell für ein Formular, das Sie beliebig anpassen können. Formularvorlagen sind Formulare mit ähnlichen Eigenschaften, die in unterschiedlichen Anwendungen verwendet werden können.



Die Schaltfläche **Bericht anzeigen** zeigt einen Überblick über das angelegte Projekt und beschreibt, welche Änderungen Sie vornehmen können und welche weiteren Assistenten zur Verfügung stehen, mit denen Sie Ihrem Projekt Funktionalität geben können.

Herzlichen Glückwunsch! Damit haben Sie Ihr erstes Programm erstellt, ohne besonders viel über Visual Basic und die Programmiersprache Visual Basic zu wissen! Ein paar Klicks auf dem Bildschirm - und Visual Basic zeigt ein Dialogfeld an, in dem es Ihnen mitteilt, daß Ihr Programm fertig ist. Wenn Sie auf **OK** klicken, verschwindet dieses Dialogfeld und Sie können Ihr Programm ausführen.



Nachdem Sie eine Anwendung geladen oder erstellt haben, führen Sie sie aus, um zu prüfen, ob sie funktioniert - so wie Ihre Benutzer es irgendwann tun werden, sobald Sie die Anwendung getestet und kompiliert haben. Visual Basic ist mit einer Küche vergleichbar. Sie sind der Koch, die Anwendung ist das Rezept. Ändern Sie die Anwendung (das Rezept), und es entsteht ein völlig neues Programm (Gericht). Die Programmierphase kann relativ lange dauern, wenn Sie eine sehr komplizierte Anwendung entwickeln, selbst wenn Sie mit dem Anwendungs-Assistenten ein Grundgerüst dafür anlegen. Bei der Entwicklung des Programms tut dieses überhaupt nichts, bevor Sie es nicht ausführen.

Führen Sie das Programm aus, indem Sie auf **Ausführen** , **Starten** klicken (das Programm wird standardmäßig interaktiv ausgeführt). Im Menü sehen Sie, daß für die Ausführung der Anwendung (F5) als Shortcut bereitgestellt wird. Sie sehen das in Abbildung 1.7 gezeigte Fenster.



Abbildung 1.7:
Ihre erste Anwendung ist fertig!

Sie haben mit dem Anwendungs-Assistenten ein vollständiges Programm angelegt (allerdings nur mit einer einfachen Oberfläche, die nur wenige Funktionen bietet), indem Sie nur die Fragen des Assistenten beantwortet haben. Ihre Anwendung kann das folgende:

- Ein Standard-Programmfenster erscheint, das Sie verschieben und dessen Größe Sie ändern können. Der Name des Projekts, *ErstesProgramm*, wird in der Titelleiste des Fensters angezeigt.
- In einer Statusleiste werden Datum und Zeit angezeigt. Im **Ansicht**-Menü kann diese Statusleiste deaktiviert werden.
- Ein Menü mit fünf Einträgen wird angezeigt. Nur die Option **Hilfe**, **Info** funktioniert (probieren Sie das aus). Die normalen Menüauswahlen, wie beispielsweise **Datei**, **Öffnen** (womit ein Dialogfeld angezeigt wird, in dem Sie nach einer bestimmten Datei suchen) oder **Bearbeiten**, **Ausschneiden**, werden zwar schon angezeigt, müssen aber noch durch aktiven Code ergänzt werden. Das Dialogfeld **Info** entspricht der Standardkonvention von Windows zur Anzeige von Systeminformationen beim Anklicken der Schaltfläche **System-Info**.
- Eine Standard-Symbolleiste wird angezeigt, der Sie zusätzliche Funktionalität hinzufügen und die sie im **Ansicht**-Menü aktivieren oder deaktivieren können.



Der Bildschirm **Microsoft Systeminfo** zeigt einen vollständigen Überblick über die Hardware und das Betriebssystem des Benutzer-PCs. Dieser Überblick erscheint, nachdem Visual Basic ein Systeminfo-Programm ausgeführt hat, das den Computer des Benutzers nach spezieller Hardware und Systeminformationen durchsucht. (Sie können das Systeminfo-Programm auch von anderen Positionen als dem Info-Feld aufrufen.) Ein solcher Überblick kann sehr praktisch sein, wenn Ihnen Ihre Benutzer mitteilen, daß sie ein Problem mit Ihren Anwendungen haben. Sie bitten die Benutzer, die Systeminformation anzuzeigen, um zu überprüfen, ob ein geeignetes Betriebssystem und die passende Hardware für Ihr Programm verwendet wird. Darüber hinaus können Sie im System-Info-Fenster feststellen, wie viele Ressourcen zur Verfügung stehen, beispielsweise Plattenspeicher und Hauptspeicher.

Die Anwendung leistet nur wenig, ist aber fertig - und Sie können die Lücken ausfüllen. Es ist ganz einfach, Anwendungen, ihre Menüs und ihre Fenster zu ändern oder zu ergänzen. Die Anwendung stellt nur eine Oberfläche für die Dinge dar, die noch kommen, und doch hat der Anwendungs-Assistent ein vollständiges Projekt angelegt und Ihnen auf diese Weise viel der mühseligen Arbeit abgenommen, die Sie erledigen müssen, wenn Sie die Anwendung von Grund auf neu erstellen.

Im nächsten Kapitel erfahren Sie, daß es ganz einfach ist, funktionierende Projekte selbst anzulegen, aber der Anwendungs-Assistent stellt die grundlegende Funktionalität bereit, die von Anwendungen häufig benötigt wird.

Um die laufende Anwendung zu verlassen, wählen Sie **Datei, Beenden**. Antworten Sie mit **Nein**, wenn Windows Sie fragt, ob Sie das Projekt speichern möchten. Sie brauchen diese Oberfläche nicht zu speichern, weil Sie sie ganz einfach neu erstellen können, indem Sie den Anwendungs-Assistenten ausführen.

Ereignisorientierte Programmierung

Abbildung 1.8 zeigt ein Fenster aus einem Windows-Programm. Das Fenster enthält mehrere Windows-Steuerelemente, beispielsweise Befehlsschaltflächen, Kontrollkästchen und eine Bildlaufleiste. Diese Steuerelemente sind nur Beispiele für die vielen Windows-Steuerelemente, die Ihnen in der Programmierumgebung von Visual Basic zur Verfügung stehen.

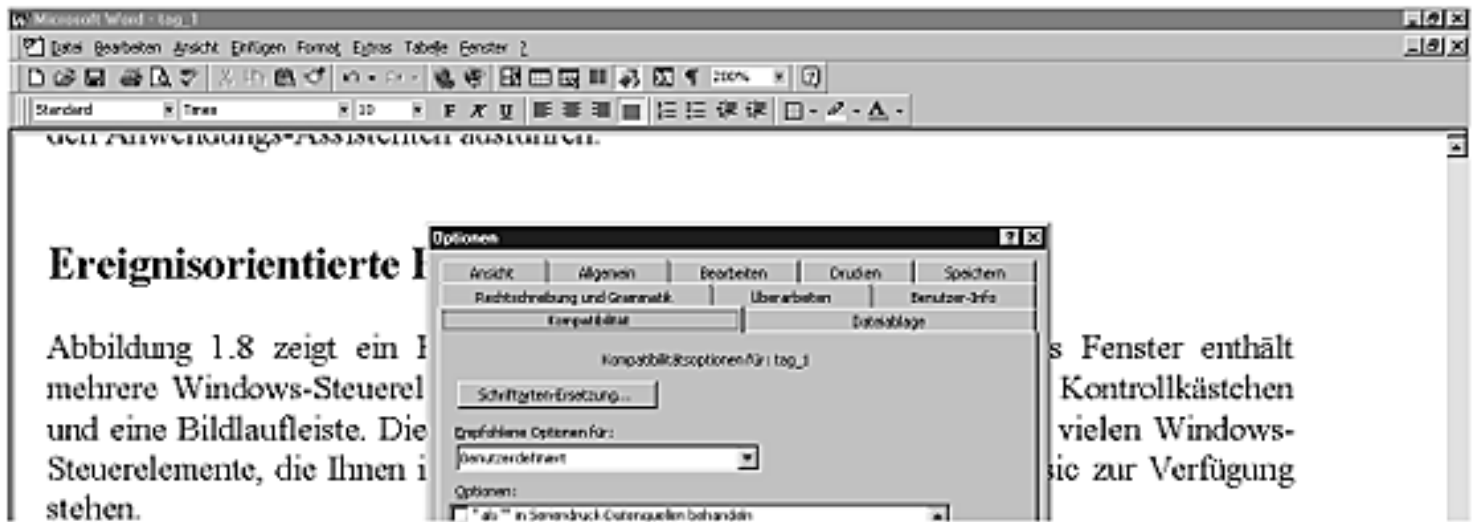


Abbildung 1.8:
Windows-Programme reagieren auf Ereignisse.

Diese Steuerelemente sind wichtig, weil Visual Basic visuell bedient wird, im Gegensatz zu Programmen, die in älteren, textbasierten Sprachen geschrieben sind. Windows- Programme reagieren auf *Ereignisse*. Diese Ereignisse werden von den Steuerelementen erzeugt, ebenso wie von internen Aktivitäten, etwa vom Zeitgeber des PC. Ereignisse treten in zufälliger Reihenfolge auf. Der Benutzer des Fensters in Abbildung 1.8 beispielsweise könnte eine Befehlsschaltfläche anklicken, ein oder mehrere Kontrollkästchen markieren oder das Dropdown-Listefeld öffnen. Er kann diese Ereignisse bei jeder Ausführung des Programms in einer unterschiedlichen Reihenfolge erzeugen. Sie brauchen *ereignisorientierte Programmierstechniken*, um auf die Aktionen des Benutzers und andere Aktivitäten, die Ereignisse auslösen können, richtig zu reagieren.



Ein *Ereignis* ist eine Aktivität, die während der Programmausführung auftritt, beispielsweise ein Mausklick oder ein Tastendruck. Die *ereignisorientierte Programmierung* reagiert auf diese Windows-Ereignisse.



Ihre Programme müssen nicht nur Ereignisse in zufälliger Reihenfolge verarbeiten, sondern auch erkennen, auf welche Ereignisse sie reagieren sollen, falls mehrere Windows-Programme ausgeführt werden.

Wie Abbildung 1.9 zeigt, verarbeitet Windows ein paar Ereignisse, gibt aber die meisten an die aktuell ausgeführten Programme weiter. Windows ist ein Multitasking-Betriebssystem, deshalb können mehrere Programme gleichzeitig ausgeführt werden. Ihr Programm muß alle Ereignisse verarbeiten, für die es verantwortlich ist, und die anderen ignorieren. Wenn Ihr Programm beispielsweise nach einem zuvor festgelegten Zeitintervall eine Warnmeldung anzeigen soll, muß es das Timer-Ereignis überprüfen, um zu sehen, ob seit der letzten Warnung die richtige Zeitspanne vergangen ist. Ein anderes Programm, das gleichzeitig ausgeführt wird, und das den Zeitgeber nicht angefordert hat, ignoriert alle Timer-Ereignisse, die ihm Windows übergibt.

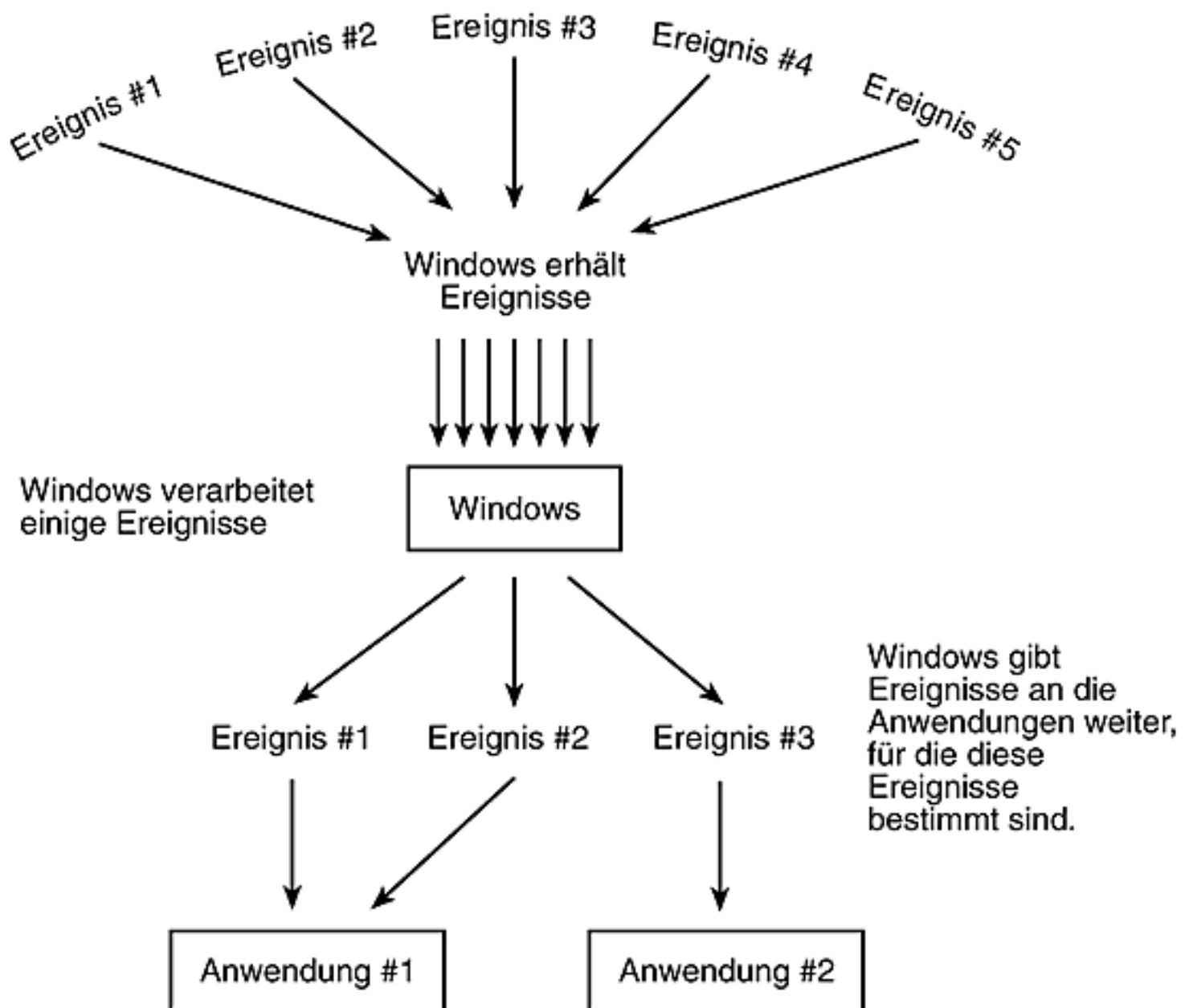


Abbildung 1.9:
Einige Ihrer Programme müssen auf Ereignisse reagieren, andere dagegen sie ignorieren.

Ein Visual-Basic-Programm besteht aus der visuellen Oberfläche, die alle Fenster und Steuerelemente

anzeigt, die der Benutzer sieht und benutzt. Daneben gibt es Programmcode, der alles verbindet. Steuerelemente reagieren automatisch auf den Programmcode, können aber auch von Ihnen festgelegte Funktionen übernehmen. Eine Befehlsschaltfläche beispielsweise wird wirklich als »gedrückt« angezeigt, wenn der Benutzer mit der Maus darauf klickt. Sie brauchen nichts weiter zu tun, als die Schaltfläche auf dem Formular anzulegen (im Programmfenster), damit sie auf diese Weise funktioniert. (Sie können eine Befehlsschaltfläche sowohl mit der Maus als auch mit der Eingabetaste drücken.) Andere Aspekte der Befehlsschaltfläche unterliegen Ihrer Kontrolle, beispielsweise der Titel oder das Bild, das darauf angezeigt wird, ihre Größe, ihre Farbe usw. Dabei handelt es sich um die *Eigenschaften*, die von Visual Basic mit einem Standardwert vorgegeben werden, den Sie jedoch ändern können. Befehlsschaltflächen unterscheiden sich durch ihre Eigenschaften voneinander.

	Eine <i>Eigenschaft</i> hilft, ein Steuerelement von anderen Steuerelementen zu unterscheiden, weil sie das Erscheinungsbild oder das Verhalten eines Steuerelements genau bestimmt. Eigenschaften haben Werte, beispielsweise Farben, Beschriftungen, Größen und Positionen auf dem Formular. Wenn Sie ein Steuerelement auf einem Formular ablegen, weisen Sie ihm Eigenschaften zu, die es eindeutig von den anderen Steuerelementen unterscheiden.

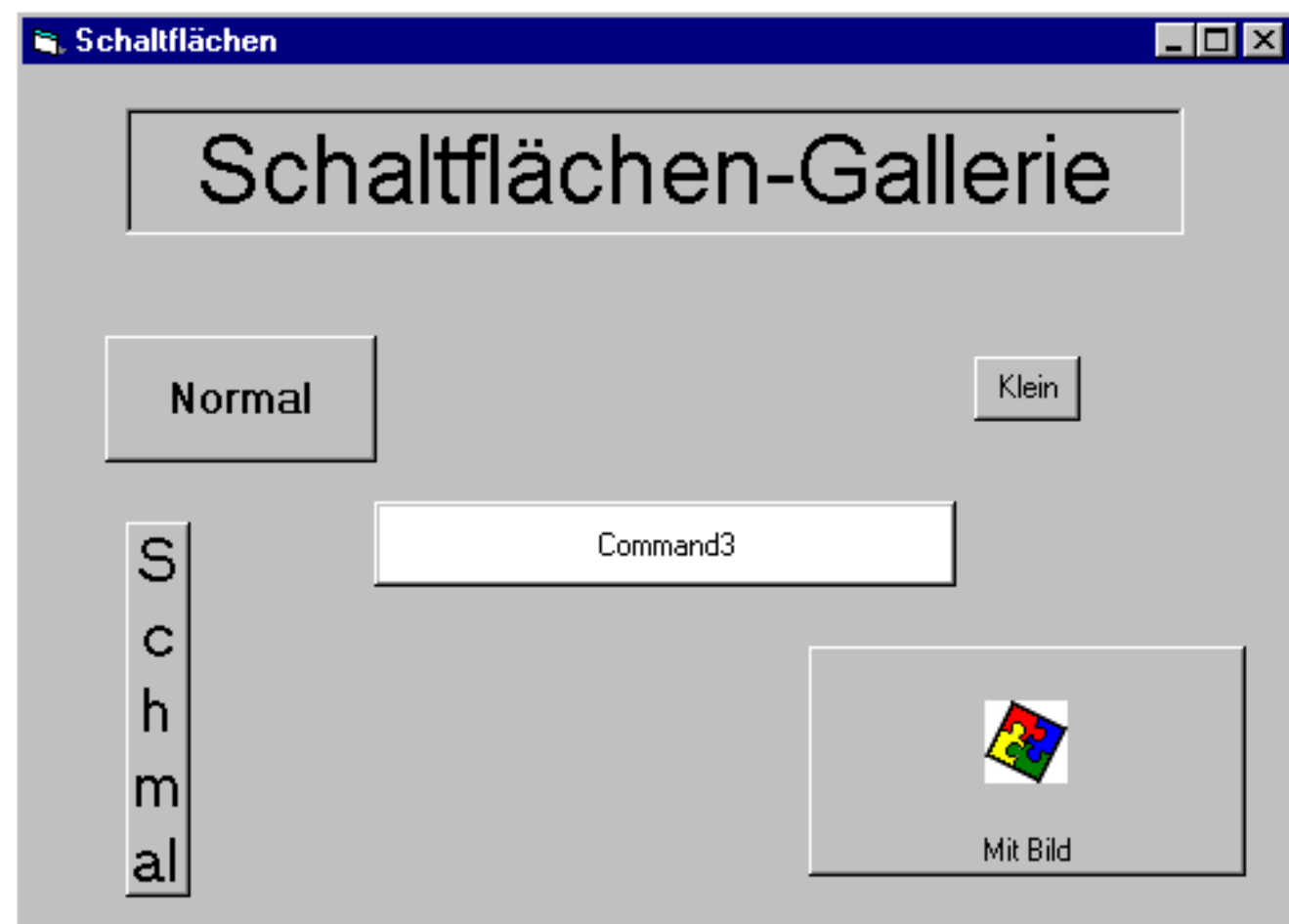


Abbildung 1.10:
Steuerelemente sehen unterschiedlich aus, wenn sie unterschiedliche Eigenschaftswerte haben.

Nachdem Sie auf einem Formular Steuerelemente angelegt und ihnen verschiedene Werte zugeordnet haben, können Sie Programmcode schreiben, der auf Ereignisse reagiert. Ein Steuerelement kann unterschiedliche Ereignisse auslösen.

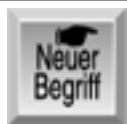
Beispielsweise könnte eine Befehlsschaltfläche einen Klick oder einen Doppelklick erzeugen, abhängig davon, was der Benutzer tut. Der Code, den Sie für das Programm schreiben, legt fest, welche dieser Ereignisse Ihr Programm ignoriert, und welche es auf die eine oder andere Weise

verarbeitet.



Wenn Sie Code für ein bestimmtes Ereignis schreiben, reagiert Ihr Programm auf dieses Ereignis, falls es während der Programmausführung auftritt. Wenn Sie dagegen für ein bestimmtes Ereignis keinen Code schreiben, und dieses Ereignis tritt auf, ignoriert das Programm es, falls Windows es ihm übergibt.

Ihr Programmiercode hinter dem Formular verhält sich nicht wie eine lange Textauflistung - und sieht auch nicht so aus -, sondern wie mehrere kleine Codeabschnitte, die jeweils auf ein bestimmtes Ereignis der Steuerelemente auf dem Formular reagieren. Diese Abschnitte tun überhaupt nichts, solange das Ereignis nicht aufgetreten ist; wenn es auftritt, führt das Programm unmittelbar den zugehörigen Code aus. Angenommen, durch einen Klick mit der rechten Maustaste auf ein *Objekt*, wie beispielsweise eine bestimmte Befehlsschaltfläche, werden ein Warnton und eine Meldung ausgegeben. Damit das funktioniert, müssen Sie Code schreiben, der den Warnton und die Meldung erzeugt. Das ausführende Programm führt diesen Code nur dann aus, wenn der Benutzer mit der rechten Maustaste auf die Schaltfläche klickt.



Ein *Objekt* ist ein Element in einem Visual-Basic-Programm, beispielsweise ein Steuerelement, ein Formular oder ein Codemodul, das Programmanweisungen enthält.

Wie arbeiten alle diese Dinge zusammen? Die Antwort kennen Sie in etwa 20 Tagen. Im nächsten Kapitel erfahren Sie, wie man die Eigenschaften von Steuerelementen festlegt, und wie man auf die Ereignisse dieser Steuerelemente reagiert. Sie werden dabei Ihr erstes Visual-Basic-Programm ohne die Hilfe des Anwendungs-Assistenten erzeugen. Grau ist alle Theorie - Sie müssen sich wirklich an die Tastatur setzen und Steuerelemente erzeugen, ihre Eigenschaften festlegen und Code schreiben, der auf Ereignisse reagiert.

Zusammenfassung

Sie sind auf dem besten Weg, Visual Basic zu beherrschen. Heute haben Sie den Hintergrund kennengelernt, den Sie für die Programmierung brauchen. Mit Hilfe dieser Informationen über den Programmierprozeß sind Sie besser für den Einsatz von Visual Basic gerüstet - eine der heute komplexesten Programmierungsumgebungen.

Im heutigen Kapitel haben Sie erfahren, wie man Programme entwirft und entwickelt. Visual Basic hat die Methode, wie Programmierer Programme entwerfen, völlig geändert. Es erlaubt, auf einfache Weise Prototypen Ihres Programms zu entwerfen und diese schließlich bis zur fertigen Anwendung weiterzuentwickeln. Während der Programmierung erfolgen viele Korrekturen und Bearbeitungen. Programme funktionieren nur selten gleich auf Anhieb perfekt. Wie Sie jedoch heute erfahren haben, nimmt Ihnen die interaktive Umgebung von Visual Basic viel Arbeit ab, so daß sich die Fehler auf ein Minimum reduzieren.

Der Anwendungs-Assistent erzeugt eine Programmoberfläche, der Sie Details hinzufügen, so daß aus dem Programm eine eigene, funktionierende Anwendung wird, die die geforderte Aufgabe erfüllt. Dabei werden Steuerelemente eingefügt, Eigenschaftswerte definiert und Code geschrieben, mit dessen Hilfe ein Programm mit den Steuerelementen zusammenarbeitet und darauf reagiert. Die restlichen Kapitel dieses Buchs werden Ihnen zeigen, wie Sie diese Details anlegen, um funktionale Programme zu schaffen.

Fragen & Antworten

Frage:

Muß ich für alle Programme, die ich schreibe, diesen Programmierprozeß befolgen (Entwurf, visuelle Elemente anlegen usw.), oder nur für die kleinen?

Antwort:

Je größer ein Programm ist, desto mehr müssen Sie diese Grundlagen der Programmentwicklung befolgen. Programme werden sehr schnell sehr komplex, wenn sie immer mehr Funktionen erfüllen sollen. Eine Funktion könnte andere Funktionen beeinflussen. Wenn Sie also von Anfang an alle Funktionen planen, müssen Sie später weniger korrigieren. Glücklicherweise ist es in der Visual-Basic-Umgebung meistens ganz einfach, Programmänderungen vorzunehmen, auch wenn sie sich auf den strukturellen Entwurf beziehen. Wenn Sie von der Oberfläche ausgehen, die der Anwendungs-Assistent für Sie erzeugt hat, ist der Entwurf Ihres Programms der zweite Schritt. Wenn Sie im Laufe dieses Buchs lernen, Programme zu schreiben, erfahren Sie auch mehr über den sauberen Programmentwurf.

Frage:

Erzeugt der Anwendungs-Assistent Programmcode?

Antwort:

Der Anwendungs-Assistent erzeugt Code, aber nicht viel. Die Aufgabe von Programmanweisungen ist es, eine bestimmte Aufgabe mit einem Programm auszuführen, beispielsweise die Berechnung von Zahlen oder die Verwaltung von Eingangsrechnungen. Ihre Aufgabe als Programmierer ist es, diesen Code zu spezifizieren.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welche Sprache verwendete Microsoft als Grundlage für Visual Basic?
2. Warum ist Visual Basic sowohl für Anfänger als auch für fortgeschrittene Programmierer geeignet?
3. Was ist wichtiger für Visual-Basic-Anfänger: die Programmiersprache oder die visuelle Oberfläche?
4. Was ist der Unterschied zwischen einem Formularfenster und dem Programmfenster?
5. Was bedeuten die Begriffe *Bug* und *Debug*?
6. Was läuft schneller: ein Programm in einer interpretierten Sprache oder ein Programm in einer kompilierten Sprache?
7. Was ist einfacher zu debuggen: ein Programm in einer interpretierten Sprache oder ein Programm in einer kompilierten Sprache?

8. Was ist der Unterschied zwischen einem Eröffnungsbildschirm und einem Tip-des-Tages-Bildschirm?
9. Was ist der Unterschied zwischen einem Steuerelement und dem Eigenschaftswert eines Steuerelements?
10. Richtig oder falsch? Steuerelemente enthalten Code, der auf Eingaben des Benutzers reagiert.

Übung

Erstellen Sie mit dem Anwendungs-Assistenten eine Anwendung, die neben den anderen Optionen, die Sie für Ihr erstes Projekt ausgewählt haben, ein Internet-Browser- Fenster und einen Splash-Bildschirm anzeigt. Führen Sie die Anwendung aus, um zu sehen, wie der Internet-Zugriff funktioniert. Wenn Sie keinen Internet-Zugriff haben, wird beim Öffnen des Browser-Fensters ein Fehler angezeigt. Legen Sie das Projekt jedoch trotzdem an, um sich Übung zu verschaffen.

Woche 1

Tag 2

Erstes Arbeiten mit Visual Basic

Nachdem Sie in Kapitel 1 erfahren haben, wie einfach die Arbeit mit dem Anwendungs-Assistenten ist, sollten Sie den Sprung wagen und ein Programm von Grund auf neu anlegen. Es ist zwar nicht schwer, ein Programm ohne den Anwendungs-Assistenten zu entwickeln, aber Sie sollten etwas mehr über die Visual-Basic-Umgebung wissen, bevor Sie Ihr erstes Programm schreiben. In diesem Kapitel erfahren Sie, wie Sie sich innerhalb der Visual-Basic-Umgebung bewegen, so daß Sie ohne den Anwendungs-Assistenten ein neues Programm entwickeln können.

Heute lernen Sie die folgenden Dinge kennen:

- Komponenten der Visual-Basic-Umgebung
- Platzierung von Steuerelementen auf einem Formular
- Speichern von Projekten und ihren Dateien
- Funktionen des Eigenschaftenfensters
- Zugriff auf das Codefenster

Die Visual-Basic-Umgebung

In allen weiteren Kapiteln dieses Buchs werden Sie in der Visual-Basic-Umgebung Programme entwickeln. Je eher Sie sich mit dieser Umgebung vertraut machen, insbesondere mit den Aufgaben der einzelnen Fenster, desto schneller sind Sie in der Lage, Visual-Basic-Programme zu schreiben. Abbildung 2.1 zeigt den Bildschirm von Visual Basic mit mehreren gebräuchlichen Elementen und ihrer Bezeichnung.

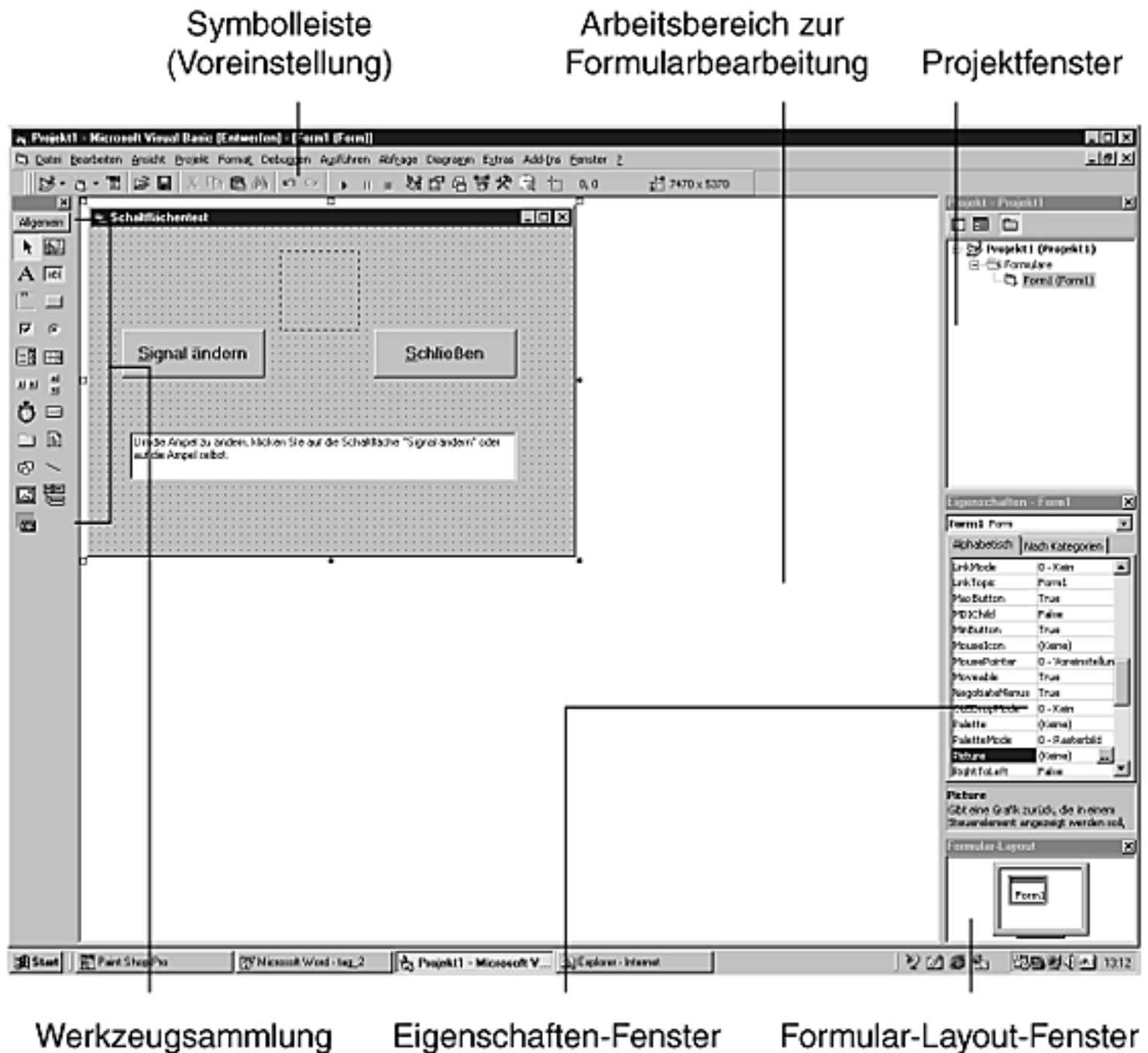


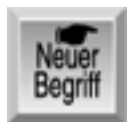
Abbildung 2.1:
 Sie sollten wissen, wofür Sie die verschiedenen Komponenten von Visual Basic einsetzen können.

Das Fenster Neues Projekt

Wie Sie in Kapitel 1 bereits gesehen haben, erscheint beim Starten von Visual Basic das Fenster **Neues Projekt**, und ebenso, wenn Sie **Datei, Neues Projekt** auswählen. In diesem Buch werden Sie die meisten neuen Anwendungen in diesem Fenster beginnen.

Wenn Sie Ihre Anwendungs-Shell nicht mit dem VB Anwendungs-Assistenten erzeugen wollen, wie es in Kapitel 1 gezeigt wurde, dann entwickeln Sie wahrscheinlich ein Standalone-Programm. Dazu wählen Sie das Icon **Standard-EXE** aus. Der Name dieses Icons repräsentiert die Dateinamenerweiterung (.exe für eine ausführbare Datei), die Ihre Anwendung erhält, nachdem Sie sie kompiliert haben. Auch wenn Sie Ihre Anwendung hier noch nicht kompilieren werden, verwenden Sie

während Ihrer Lernphase hauptsächlich das Icon **Standard-EXE**.



Eine *Standard-EXE-Anwendung* ist eine Anwendung, die Sie kompilieren oder interpretiert ausführen können.



Mehrere Icons im Fenster **Neues Projekt** tragen die Bezeichnung *ActiveX ...* *ActiveX* ist der Name, den die von Ihnen selbst entwickelten Steuerelemente erhalten. Diese Steuerelemente haben die Dateinamenerweiterung *.OCX*. Sie können sie der Visual-Basic-Umgebung hinzufügen, so daß sie in der Werkzeugsammlung angezeigt werden. Damit ist es möglich, Anwendungen zu schreiben, die zu neuen Steuerelementen werden und der Visual-Basic-Umgebung für die zukünftige Programmentwicklung hinzugefügt werden. *ActiveX* ist ein relativ komplexer Begriff, der sich auch auf andere Bereiche der Programmierung erstreckt.



Sie wissen, daß auch Visual Basic nur ein Windows-Programm ist (wenn auch ein sehr umfassendes), mit dem Sie wiederum neue Windows-Programme entwickeln können. Die Symbolleisten, Dialogfelder und Fenster in der Visual-Basic-Umgebung verhalten sich hier so wie in allen anderen Windows-Programmen - die grundsätzliche Bedienung von Visual Basic sollte also kein Problem für Sie darstellen.

Die Symbolleiste

Die Symbolleiste, die unterhalb der Titelleiste angezeigt wird, ändert ihr Aussehen abhängig davon, wofür Sie Visual Basic gerade einsetzen. Visual Basic beinhaltet insgesamt vier Symbolleisten:

- **Debuggen.** Diese Symbolleiste erscheint, wenn Sie die interaktiven Debugging- Werkzeuge benutzen, um Fehler zu erkennen und zu korrigieren.
- **Bearbeiten.** Diese Symbolleiste unterstützt Sie bei der Bearbeitung von Visual- Basic-Code.
- **Formular-Editor.** Diese Symbolleiste hilft Ihnen, Objekte auf Formularen auszurichten.
- **Voreinstellung.** Dies ist die Standardsymbolleiste, die unterhalb der Menüleiste angezeigt wird.

Diese Symbolleisten werden im Menü **Ansicht, Symbolleisten** aktiviert und deaktiviert. Jede Symbolleiste enthält mehrere Schaltflächen, die Abkürzungen für Befehle darstellen, für deren Aufruf Sie andernfalls mehrere Menüs durchqueren müßten. Wenn Sie mit Visual Basic arbeiten, werden Ihnen mehrere Schaltflächen begegnen, die Sie sehr praktisch finden, während Sie andere vielleicht nie verwenden. In diesem Buch lernen Sie viele Schaltflächen der Symbolleiste kennen, mit denen Sie Ihre Arbeit beschleunigen können, aber nicht alle Schaltflächen haben unbedingt diesen Effekt.

Was Sie tun sollten	Was Sie lassen sollten
Schieben Sie den Mauszeiger über eine Schaltfläche, die Sie nicht zuordnen können. In einem kleinen Popup-Fenster wird eine kurze Beschreibung der Schaltfläche angezeigt.	Versuchen Sie nicht, sich die Bedeutung aller Schaltflächen in der Symbolleiste zu merken.



Alle Symbolleisten können von ihrer Position unterhalb der Menüleiste an eine andere Stelle verschoben werden. Damit ist es beispielsweise möglich, eine Symbolleiste neben den Komponenten zu platzieren, wo Sie sie gerade brauchen, so daß die Schaltflächen unmittelbar bereitstehen. Anschließend können Sie die Symbolleiste wieder unterhalb der Menüleiste ablegen.

Die Werkzeugsammlung

Die Werkzeugsammlung unterscheidet sich von der Symbolleiste. In diesem Fenster finden Sie mehrere Werkzeuge, die die Steuerelemente repräsentieren, welche Sie auf einem Formular anlegen. Dieses Buch wird Ihnen zeigen, wie Sie Werkzeuge in die Werkzeugsammlung einfügen und daraus entfernen. Abbildung 2.2 zeigt die gebräuchlichsten Komponenten der Werkzeugsammlung.

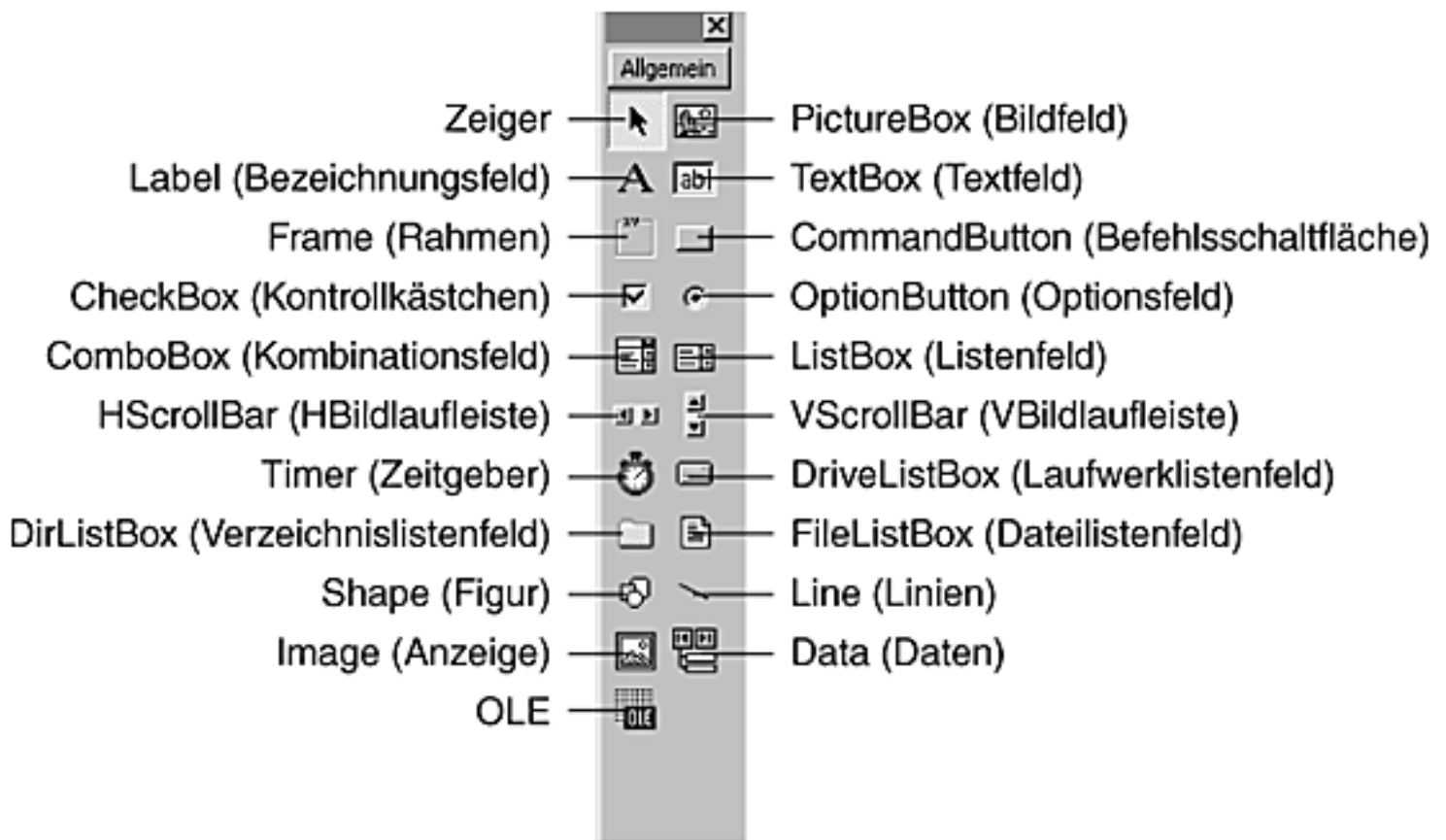


Abbildung 2.2:
Die Werkzeugsammlung enthält Werkzeuge (Steuerelemente), die Sie im Formularfenster Ihres Programms anlegen können.

Der Werkzeugsammlung gehen die Werkzeuge nicht aus! Wenn Sie in Ihrem Programm mehrere Schaltflächen brauchen, können Sie alle diese Schaltflächen mit dem CommandButton-Werkzeug der Werkzeugsammlung anlegen. Mit den Schaltflächen der Werkzeugsammlung legen Sie die Steuerelemente dort auf Ihrem Formular an, wo Sie sie brauchen, wie Sie im letzten Abschnitt dieses Kapitels bei der Entwicklung eines neuen Programms noch sehen werden.

Das Formularfenster

Ein Großteil Ihrer Arbeit spielt sich im Formularfenster ab. Im zentralen Arbeitsbereich, wo das Formularfenster angezeigt wird, entwickeln Sie die Formulare für Ihre Anwendungen, die der Benutzer als Hintergrundfenster sieht. Die Größe des Formularfensters kann beliebig angepaßt werden - so klein oder so groß, wie Sie es für Ihre Anwendung brauchen. (Mit Hilfe von Bildlaufleisten blättern Sie im Formularfenster, falls Sie Komponenten anzeigen wollen, die nicht mehr auf den Bildschirm passen.)

Merken Sie sich, daß eine Anwendung aus mehreren Formularen bestehen kann. Wie in Abbildung 2.3 gezeigt, können diese Formulare in eigenen Formular-Arbeitsbereichen angezeigt werden. Die Titelleiste des aktiven Formulars ist hervorgehoben dargestellt. Ein Formular wird durch einen Klick in das Fenster oder in die Titelleiste aktiviert.

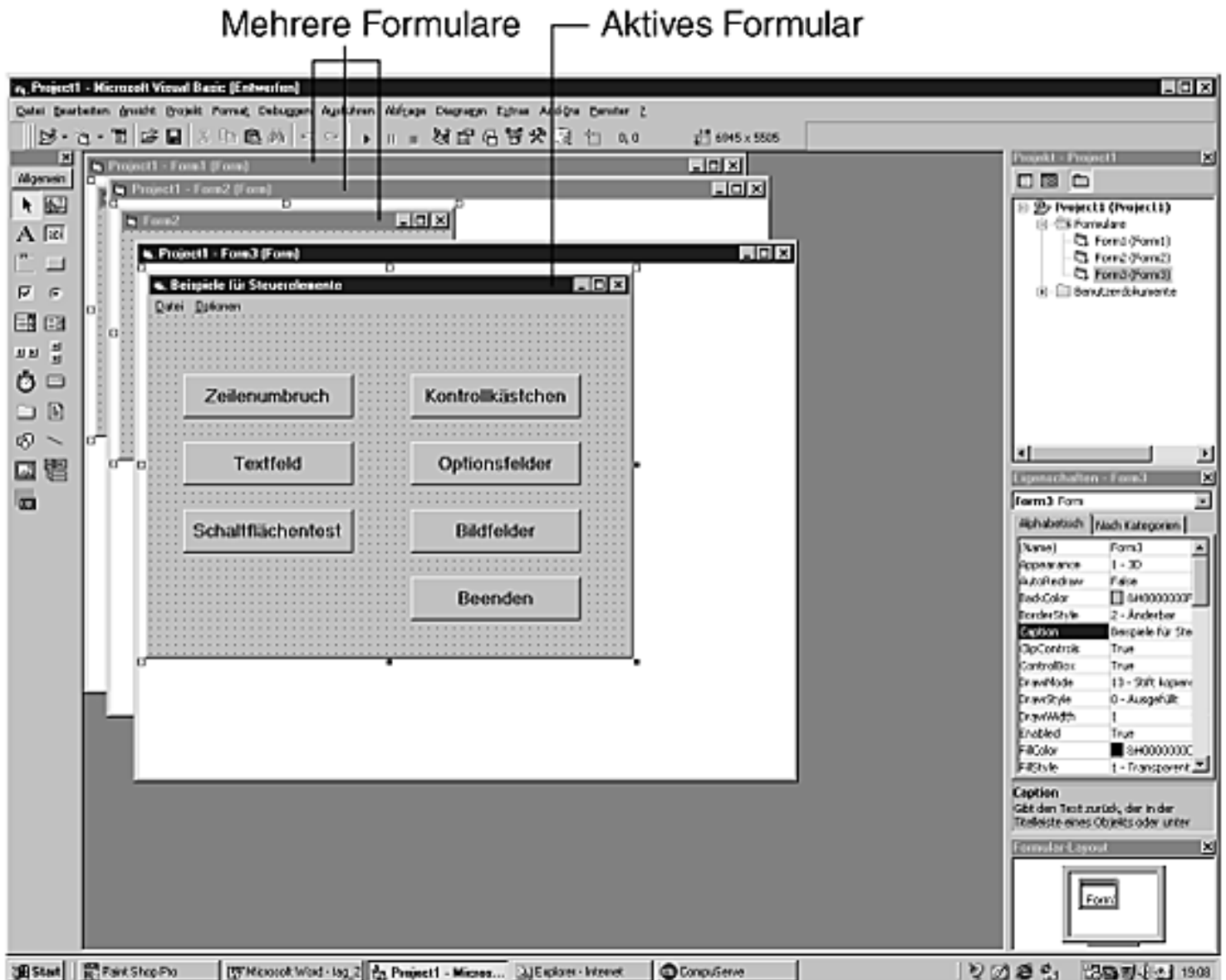


Abbildung 2.3:
Im Arbeitsbereich des Formularfensters bearbeiten Sie die Formulare für Ihre Anwendung.

Das Formular-Layout-Fenster

Das Formular-Layout-Fenster ist ein interessantes kleines Fenster, das Sie direkt neben dem Formularfenster finden. Dort sehen Sie eine Vorschau auf die Position des Formularfensters. Wenn Sie in Ihrem Formularfenster mehrere Formulare bearbeiten, sehen Sie Miniaturskizzen aller dieser Formulare im Formular-Layout-Fenster. So erkennen Sie, wo die einzelnen Formulare bei der Ausführung der Anwendung auf dem Bildschirm erscheinen.

Das Formular-Layout-Fenster zeigt nicht nur, wo ein Formular bei der Ausführung im Verhältnis zu den Bildschirmrändern angezeigt wird, sondern Sie können hier auch bestimmen, wo das Formular angezeigt werden soll, indem Sie es einfach an eine andere Position verschieben. Wenn Sie also möchten, daß Ihr Formular in der Bildschirmmitte angezeigt wird, verschieben Sie es im Formular-Layout-Fenster in die Mitte. Visual Basic platziert das Formular bei der Ausführung dann an der festgelegten Stelle.



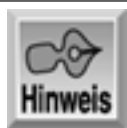
Nachdem Sie mehr über die Programmiersprache Visual Basic erfahren haben, können Sie auch Code schreiben, der ein Formular an einer beliebigen Bildschirmposition platziert. Sie können Visual Basic sogar anweisen, ein Formular beim Aufruf auf dem Bildschirm zu zentrieren, unabhängig davon, was im Formular-Layout-Fenster bei der Entwicklung festgelegt wurde.



Viele Visual-Basic-Programmierer schließen das Formular-Layout-Fenster, um Platz für andere Fenster in der Entwicklungsumgebung zu schaffen.

Das Projektfenster

Im Projektfenster werden die Komponenten Ihrer Anwendung verwaltet. Wie in Abbildung 2.4 gezeigt, kann das Projektfenster sehr stark belegt sein. Ein Windows-Programm, genauer gesagt, eine Anwendung, wie in Kapitel 1 beschrieben, kann aus mehreren Dateien bestehen. Bevor Sie ein Windows-Programm kompilieren, kann es noch sehr viel mehr Visual-Basic-bezogene Dateien geben. Das Projektfenster ermöglicht Ihnen, alle diese Komponenten zu verwalten und jeweils genau die Komponente, die Sie gerade bearbeiten möchten, im Arbeitsbereich anzuzeigen.



Das Projektfenster wird auch als Projekt-Explorer bezeichnet, weil es eine dem Explorer ähnliche Oberfläche aufweist, wo Sie Objektgruppen auf- und zuklappen können.

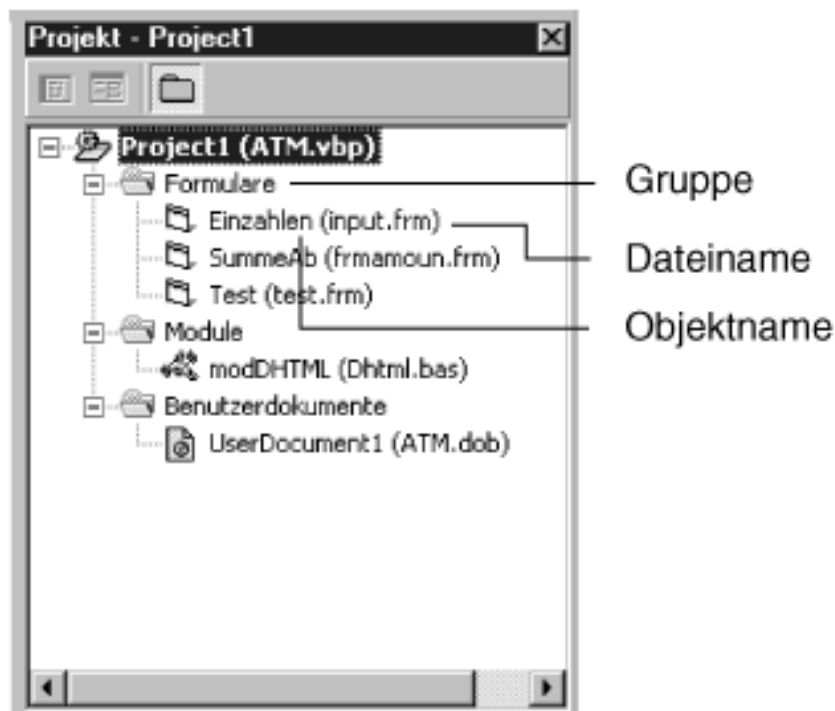


Abbildung 2.4:
Das Projektfenster enthält die
Komponenten Ihres Projekts.

Das Projektfenster zeigt die Komponenten in einer baumähnlichen Auflistung an. Gleiche Objekte werden gruppiert. Sie können die Details anzeigen oder ausblenden, indem Sie auf das Plus- oder Minuszeichen neben der betreffenden Objektgruppe anklicken. Wenn Sie beispielsweise auf das Pluszeichen neben dem Objekt **Formulare** klicken, wird eine Liste der Formulare des aktuellen Projekts angezeigt. Wenn Sie hier auf den Namen eines Formulars doppelklicken, erscheint das betreffende Formularfenster im Arbeitsbereich.

Jedes Element im Projektfenster hat einen Projektnamen und einen Dateinamen. In Visual Basic können Sie Objekten Namen zuweisen, beispielsweise Formularen und Modulen. Jedes der Elemente im Projektfenster wird auf Ihrer Festplatte in einer separaten Datei abgelegt. Der Dateiname, der sich von dem Projektnamen unterscheidet (Projektnamen haben beispielsweise anders als Dateinamen keine Dateinamenerweiterung), erscheint in Klammern neben dem zugehörigen Element. Sie können also im Projektfenster alle Dateinamen und alle Projektnamen für alle Dateien in Ihrem Projekt erkennen und jedes beliebige Fenster aktivieren, indem Sie das betreffende Objekt im Projektfenster anklicken.



Beachten Sie, daß das Projektfenster eine Symbolleiste mit drei Schaltflächen enthält. Die Schaltfläche **Code anzeigen** zeigt das Codefenster für das ausgewählte Objekt an, so daß Sie den Code bearbeiten können. (Das Codefenster wurde in Abbildung 2.1 nicht angezeigt. Sie sehen es am Ende des Kapitels, wenn Sie einer Anwendung Code hinzufügen.) Die Schaltfläche **Objekt anzeigen** zeigt das Objektfenster für das ausgewählte Element an. Viele Objekte haben sowohl ein Codefenster als auch ein Objektfenster. Jedes Formular beispielsweise hat ein Codemodul und ein entsprechendes Formularfenster. Die Schaltflächen **Code anzeigen** und **Objekt anzeigen** erlauben Ihnen, schnell zwischen dem Code und den visuellen Komponenten eines Elements zu wechseln. Die Schaltfläche **Ordner wechseln** gruppiert Elemente im Projektfenster auf einer dem Explorer ähnlichen Oberfläche bzw. hebt die Gruppierung auf.

Im Projektfenster gibt es die folgenden Objekttypen:

- **Projekte.** Eine Anwendung kann mehrere Projekte umfassen, beispielsweise wenn Sie ActiveX-Steuerelemente erzeugen. Projekte haben immer die Dateinamenerweiterung .VBP.
- **Formulare.** Das Projektfenster zeigt eine Liste der Formulare in Ihrem Projekt an. Formulardateien haben immer die Dateinamenerweiterung .FRM.
- **Module.** Ihre Projektmodule enthalten allgemeine und wiederverwendbare Routinen, die sich aus Visual-Basic-Anweisungen zusammensetzen. Module haben immer die Dateinamenerweiterung .BAS.
- **Klassenmodule.** Klassenmodule sind spezielle Codemodule, die die Objekte definieren, die Sie für ein Projekt entwickelt haben. Klassenmoduldateien haben immer die Dateinamenerweiterung .CLS.
- **Benutzersteuerelemente.** Benutzersteuerelemente sind ActiveX-Steuerelemente, die Sie dem Projekt hinzugefügt haben. ActiveX-Steuerelementdateien haben immer die Dateinamenerweiterung .OCX.
- **Benutzerdokumente.** Benutzerdokumente sind Dokumentobjekte, die Teile Ihrer Projekte beschreiben. Benutzerdokumentdateien haben immer die Dateinamenerweiterung .DOB.
- **Eigenschaftenseiten.** Eigenschaftenseiten (wie beispielsweise die, die man in den Dialogfeldern mit Registerkarten findet), die in der Projektdatei erscheinen, beschreiben ein bestimmtes Steuerelement. Eigenschaftendateien haben immer die Dateinamenerweiterung .PAG.



Manchmal tauchen auch andere Elemente im Projektfenster auf, beispielsweise Ressourcen und andere Dokumente, die Sie Ihrem Projekt hinzugefügt haben.

Größtenteils werden Sie bei der Entwicklung Ihrer Visual-Basic-Anwendungen nur Formulare und Codemodule verwenden, insbesondere in den ersten 21 Kapiteln dieses Buchs.

Das Eigenschaftfenster

Ein Formular kann mehrere Steuerelemente enthalten. Um einem Formular Steuerelemente hinzuzufügen, wählen Sie es aus und klicken darauf. Wenn Sie ein Steuerelement auswählen, werden im Eigenschaftfenster seine Eigenschaften aufgelistet. Wie Sie am Ende dieses Kapitels sehen werden, setzt Visual Basic Standardwerte für ein neu angelegtes Steuerelement. Im Eigenschaftfenster für ein Steuerelement können Sie dessen Eigenschaften ändern.

Abbildung 2.5 zeigt ein Eigenschaftfenster für ein Bezeichnungsfeld. Beachten Sie, daß der Name, der Typ und die Beschreibung im Eigenschaftfenster das ausgewählte Steuerelement reflektieren. Um einer Eigenschaft einen Wert zuzuweisen, selektieren Sie diese und geben einen neuen Wert ein. Falls für die Werte einer Eigenschaft ein Wert von mehreren vorgegebenen Werten gewählt werden kann, erscheint ein Dropdown-Listefeld.



Abbildung 2.5:
Das Eigenschaftenfenster enthält alle Eigenschaften des ausgewählten Steuerelements.

Jede Eigenschaft hat einen Namen, über den Sie sie ansprechen, und jede Eigenschaft hat einen Wert, den Sie oder Visual Basic ihr zuweisen. Beispielsweise bezeichnet Visual Basic die erste Befehlsschaltfläche, die Sie einer Anwendung hinzufügen, immer als *Command1*. Die Name-Eigenschaft für die erste Schaltfläche enthält also als Vorgabe den Wert *Command1*. Sie sollten der Schaltfläche einen etwas aussagekräftigeren Namen zuweisen. Eine Schaltfläche, die einen Bericht ausdruckt, könnte beispielsweise als *cmdReportPrint* bezeichnet werden.

Sie sollten jedem Objektnamen, den Sie zuweisen, ein dreistelliges Präfix voranstellen, das den Objekttyp bezeichnet. Wenn Sie die Liste der Objekte betrachten, erkennen Sie daran sofort deren Typen (Befehlsschaltfläche, Textfeld, Formular usw.). Tabelle 2.1 listet die gebräuchlichsten Präfixe für Objektnamen unter Visual Basic auf. Greifen Sie bei der Programmentwicklung in diesem Buch immer wieder auf diese Angaben zurück. Wenn Ihr Projekt sehr viele Steuerelemente enthält, können diese Namen helfen, die Aufgabe und den Typ der Steuerelemente zu identifizieren.

Tabelle 2.1: Präfixe für Objektnamen

Präfix	Objekttyp
cbo	Kombinationsfeld (Combo Box)
chk	Kontrollkästchen (Check Box)
cmd	Befehlsschaltfläche (Command Button)
dir	Verzeichnislistenfeld (Directory List Box)
fil	Dateilistenfeld (File List Box)
fra	Rahmen (Frame)
frm	Formular (Form)

grd	Raster (Grid)
hsb	Horizontale Bildlaufleiste (Horizontal Scrollbar)
img	Anzeige (Image)
lbl	Bezeichnungsfeld (Label)
lin	Linien (Line)
lst	Listenfeld (List Box)
mnu	Menü (Menu)
mod	Modul (Module)
ole	OLE
opt	Optionsfeld (Option Button)
pic	Bildfeld (Picture Box)
res	Ressource (Resource)
shp	Figur (Shape)
tmr	Zeitgeber (Timer)
txt	Textfeld (Text Box)
typ	Benutzerdefinierter Datentyp
vsb	Vertikale Bildlaufleiste (Vertical Scrollbar)

Was Sie tun sollten	Was Sie lassen sollten
Verwenden Sie für das Präfix vor den Visual-Basic-Objektnamen immer Kleinbuchstaben.	Verwenden Sie das Präfix nicht für die Dateinamen von Objekten.

	Beachten Sie, daß Sie jedes Visual-Basic-Fenster verschieben, vergrößern oder verkleinern und schließen können. Manchmal will man mehr oder weniger von einem Fenster sehen, um Platz für andere Fenster zu schaffen.

Hilfe

Visual Basic enthält ein großes Sortiment an Online-Werkzeugen, die Ihnen gegebenenfalls Hilfe bereitstellen. Bevor Sie heute im letzten Abschnitt des Kapitels eine Anwendung erstellen, sollen Sie erfahren, wie man auf die verschiedenen Hilfoptionen zugreift, um gegebenenfalls mehr über eine Prozedur zu erfahren.

Lokale Hilfe

Größtenteils erhalten Sie in der Visual-Basic-Umgebung alle Hilfe, die Sie brauchen, ohne irgendwelche anderen Quellen zu Rate zu ziehen (außer diesem Buch natürlich!). Die erste Option im Hilfemenü, **Inhalt**, erzeugt einen HTML-basierten Windows-Hilfebildschirm, wie in Abbildung 2.6 gezeigt. Im linken Feld sehen Sie mehrere Symbole für Online-Bücher, die Sie öffnen und lesen können, und das rechte Feld enthält eine geführte Tour durch mehrere Hilfethemen mit Dr. Gui (*GUI* ist die Abkürzung für *Graphical User Interface*, also Grafische Benutzeroberfläche).



Das Hilfesystem von Visual Basic basiert auf Books Online, einer Hilfe-Datenbank, die man bereits in älteren Microsoft-Produkten findet. Sie brauchen die MSDN CD-ROMs, um auf die gesamte Online-Hilfe zugreifen zu können.



MSDN ist die Abkürzung für *Microsoft Developer's Network* und steht für mehrere Online-Artikel, CD-ROMs und Newsletters, die Microsoft in den vergangenen paar Jahren für Programmierer herausgegeben hat. Die Hilfe von Visual Basic ist jetzt Teil des MSDN-Materials. Die Online-Bildschirme für MSDN stehen nur über ein Abonnement-System zur Verfügung. Um Online-MSDN zu abonnieren, klicken Sie auf dem Bildschirm **Hilfe, Inhalt** auf den Hyperlink **MSDN Online**.



Ihre Hilfe unterscheidet sich möglicherweise von der hier angezeigten, abhängig von dem Datum Ihrer Visual-Basic-6-Kopie. Microsoft ändert die Hilfebildschirme manchmal, nachdem die ersten Pakete ausgeliefert sind.

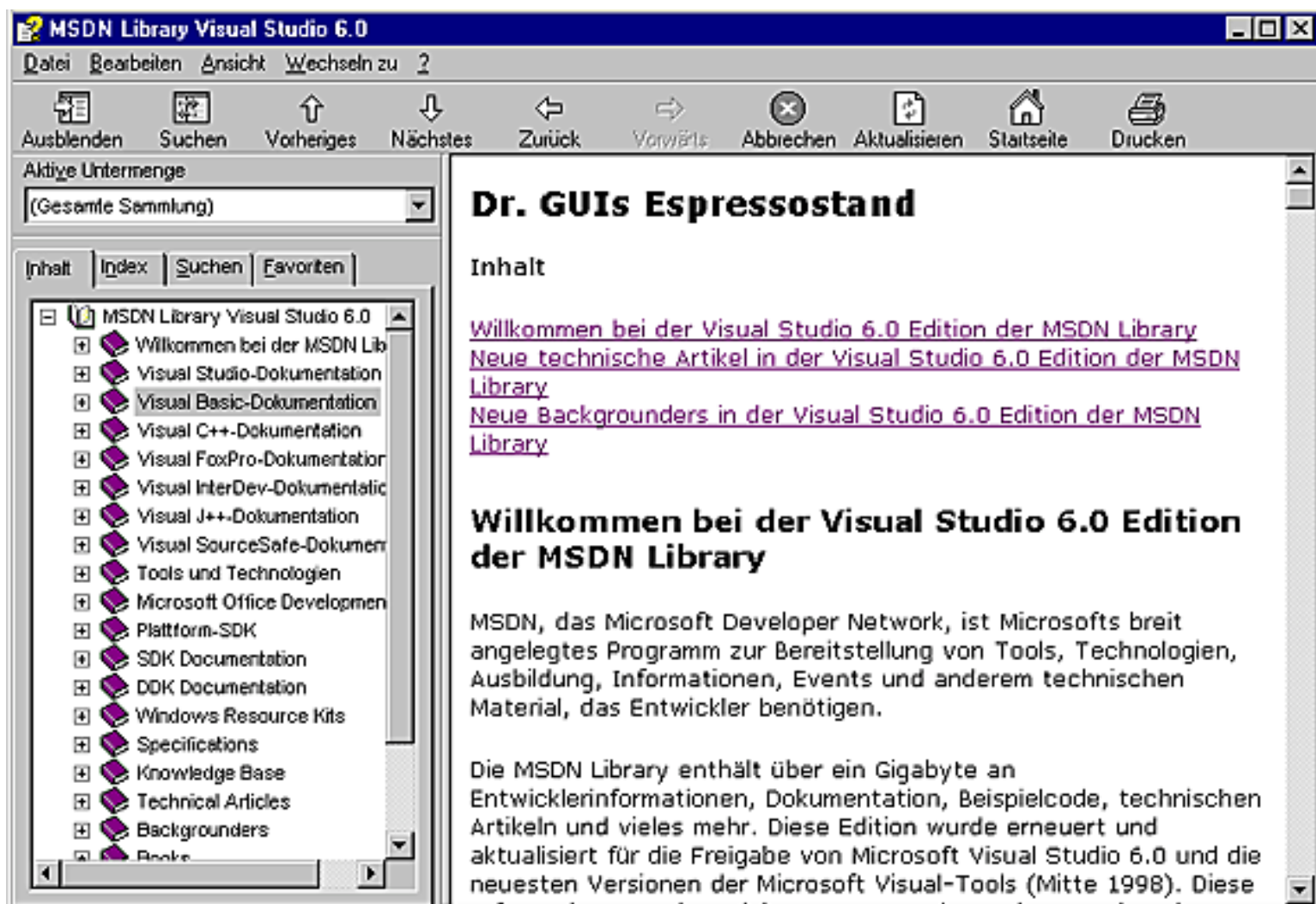


Abbildung 2.6:
Die Online-Hilfe von Visual Basic unterstützt Sie, wenn Sie Probleme haben.

Das Dialogfeld **Hilfe** bietet Online- und Direkthilfe, und zwar in der folgenden Form:

- **Inhalt.** Diese Option bietet in Büchern angeordnete Hilfe, beispielsweise »Visual Basic-Dokumentation« oder »Tools und Technologien«.
- **Index.** Diese Option ermöglicht Ihnen, durch die Eingabe von Stichwörtern Hilfethemen aufzurufen.
- **Suchen.** Diese Option ermöglicht Ihnen, innerhalb von Artikeln nach einem bestimmten Text zu suchen.
- **Favoriten.** Mit Hilfe dieser Option können Sie Hilfethemen speichern, die für Sie besonders interessant sind.

Haben Sie sich schon einmal gefragt, wie so ein umfangreiches Entwicklungssystem wie Visual Basic ohne dicke, sperrige Handbücher auskommt? Es *gibt* viele Handbücher für Visual Basic, aber sie stehen über das MSDN-Hilfesystem online zur Verfügung. Klicken Sie auf eines der Bücher im linken Feld, um es zu »öffnen«, und gehen Sie in ein Kapitel, um eine bestimmte Seite zu lesen. Die Seite wird im rechten Feld des Hilfefensters angezeigt.



Die Hilfebildschirme erscheinen in einem eigenen Fenster, das unabhängig vom Visual-Basic-Fenster ist. Sie können beide Fenster gleichzeitig öffnen und mit Hilfe von (Alt)+(Y__) oder durch Anklicken der entsprechenden Schaltfläche in der Taskleiste zwischen beiden Fenstern wechseln.

Das Hilfesystem bietet Informationen über Visual Basic, Datenbankverbindungen, ActiveX-Programmierung und andere Aspekte der Programmierung, die Sie häufig verwenden werden, um schnelle Antworten auf Probleme bei der Programmentwicklung zu erhalten. Sie können sich das Hilfesystem als Sammlung teurer Nachschlagewerke vorstellen, die, wären sie gebunden und dem Programmpaket Visual Basic beigelegt, die Kosten für Ihre Software wesentlich erhöhen würden, und die Suche darin wäre bei weitem nicht so einfach, wenn Sie Hilfe zu bestimmten Themen suchen.

Ein weiterer Eintrag im Hilfemenü, **Info zu Microsoft Visual Basic**, erzeugt ein Info-Dialogfeld, in dem Sie die Versionsnummer, die Seriennummer und den registrierten Namen für Ihre Visual-Basic-Version sehen. Wenn Sie auf die Schaltfläche **Systeminfo** klicken, erscheint nach einer Überprüfung Ihres Systems das in Abbildung 2.7 gezeigte Dialogfeld. Die Systeminformation enthält Informationen über Software und Hardware.

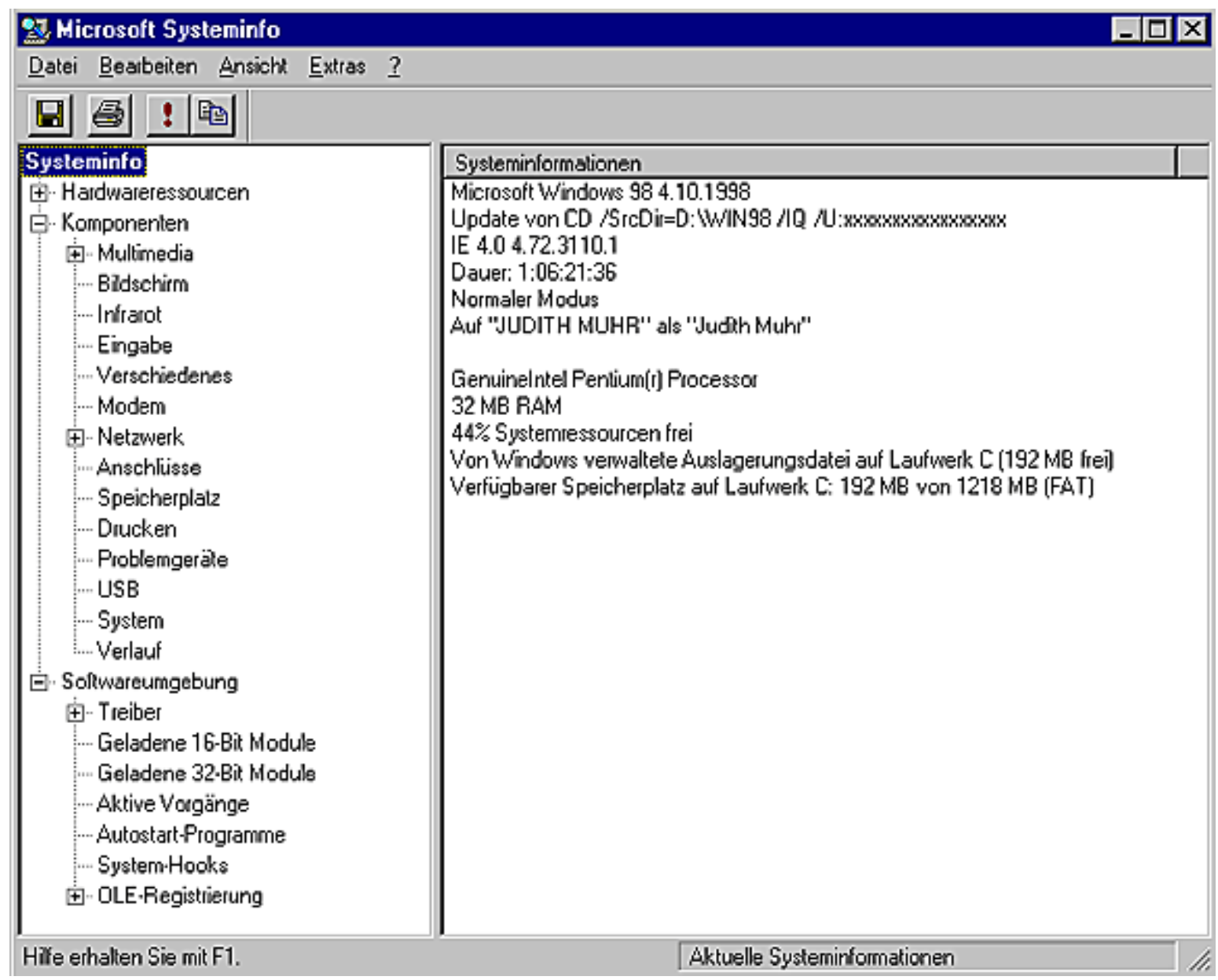


Abbildung 2.7:
Die Schaltfläche Systeminfo ermittelt Informationen über Ihr System.

In Kapitel 20 erfahren Sie, wie Sie Ihren Anwendungen eine Online-Hilfe hinzufügen.

Software-Service

Durch Auswahl von **Hilfe, Software-Service** erscheint ein Dialogfeld, das Ihnen mitteilt, wie Sie den technischen Support von Microsoft erreichen. Für spezielle Fragen sind die Online-Hilfe und Books Online möglicherweise nicht ausreichend. Wenn Sie beispielsweise vermuten, daß sich Visual Basic so seltsam verhält, weil sich ein Bug im System befindet, sollten Sie Microsoft um Hilfe bitten. (Bei solchen Problemen sollten Sie immer versuchen, Visual Basic neu zu installieren, um zu prüfen, ob das Problem damit gelöst werden kann, bevor Sie sich an den technischen Support wenden. Diesen Rat würde man Ihnen dort zunächst vielleicht auch geben, Sie können also durch die Neuinstallation vor dem Anruf Zeit sparen.)



Sie fragen sich vielleicht, warum Sie Unterstützung brauchen sollten, um den technischen Support zu erreichen. Brauchen Sie nicht die gebührenfreie Telefonnummer und die Zeitangaben, wann die Hotline besetzt ist? Microsoft bietet mehrere Support-Ebenen, von kostenlosem bis hin zu gebührenpflichtigem Kundendienst und jährlichen Abonnements. In der Hilfe finden Sie einen Überblick darüber, welche Möglichkeiten Ihnen zur Verfügung stehen. Wenn Sie außerhalb der Vereinigten Staaten leben, sollten Sie das Büro in Ihrer Nähe anrufen.

Online-Hilfe

Mit **Hilfe, Microsoft im Web** zeigt Visual Basic eine große Auswahl von Online-Support-Optionen an. (Für alle diese Optionen ist ein Internet-Zugriff erforderlich.) Wenn Sie **Online Software Service** wählen, erscheint die Webseite von Microsoft für Visual Basic. Diese Seite sollten Sie regelmäßig lesen, auch wenn Sie keine Online-Hilfe brauchen. Dort finden Sie wichtige Update-Informationen, Fehlerkorrekturen, Tips, Patches, Beispielcode sowie Links zu weiteren interessanten Seiten. Andere Optionen für die Web-Menü-Option sind Links zur Microsoft-Homepage, eine Web-Suchmaschine und sogar ein Feedback-Link zu Microsoft, so daß Sie Microsoft Ihre Kommentare und Verbesserungsvorschläge zu Visual Basic mitteilen können.



Besuchen Sie regelmäßig die FAQ-Seite. Dort finden Sie Antworten auf viele allgemeine Fragen von Visual-Basic-Entwicklern.

Gewöhnen Sie sich an den Bildschirm

Schon an Ihrem ersten Tag mit diesem Buch haben Sie eine vollständige, funktionierende Visual-Basic-Anwendung entwickelt. Genauer gesagt, Sie haben die Anwendung mit Hilfe des Anwendungs-Assistenten von Visual Basic erstellt, der die gesamte Arbeit für Sie übernommen hat. Sie werden diese Leistung jetzt noch verbessern und eine ganze Anwendung von Grund auf neu erstellen.

Sie wissen jetzt schon viel mehr über die Visual-Basic-Umgebung und wie Sie gegebenenfalls Hilfe erhalten. Bevor Sie den Schritten im nächsten Abschnitt folgen, um eine neue Anwendung zu erstellen, sollten Sie sich einen Moment Zeit nehmen und eine existierende Anwendung aus einem der Beispiele von Visual Basic laden. Anschließend machen Sie sich mit den Fenstern vertraut, die daraufhin erscheinen. Die folgenden Anweisungen sollen Ihnen dabei helfen:

1. Starten Sie Visual Basic.
2. Legen Sie die MSDN-CD 1 in Ihr CD-ROM-Laufwerk ein.
3. Klicken Sie im Dialogfeld **Projekt öffnen** auf die Registerkarte **Vorhanden** und gehen Sie in den Ordner **Samples**.
4. Doppelklicken Sie auf das Icon **Controls**, um das Projekt **Controls** zu öffnen.

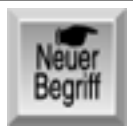


Möglicherweise sehen Sie im Dialogfeld **Projekt öffnen** die Dateinamenerweiterung **.VBP** für das Controls-Projekt nicht, abhängig davon, wie Sie Ihren Windows-Explorer eingestellt haben. Unabhängig davon, ob Sie die Dateinamenerweiterungen in Windows-Dialogfeldern anzeigen, können Sie die Dateitypen anhand des Icons links von den Dateinamen erkennen. Das Fenster **Projekt öffnen** zeigt nur Projektdateien an (es sei denn, Sie haben das im Dropdown-Listefeld **Dateityp** geändert), und das Icon neben dem Controls-Projekt ist das Icon, das vor allen Projektdateien in Visual Basic steht.

5. Nachdem Sie das Controls-Projekt geöffnet haben, erhalten Sie möglicherweise ein Dialogfeld, in dem Sie gefragt werden, ob Sie dem Projekt ein sogenanntes *SourceSafe* hinzufügen möchten. Antworten Sie auf diese Abfrage für alle Projekte in diesem Buch immer mit Nein. Visual Basic zeigt zwei weitere Dialogfelder für SourceSafe an. Klicken Sie beide Male auf OK, um sie zu schließen.



SourceSafe ist ein Werkzeug von Visual Studio (das für alle Visual-Studio- Sprachen zur Verfügung steht), mit dem Sie die Versionen Ihres *Quellprogramms* verwalten können.

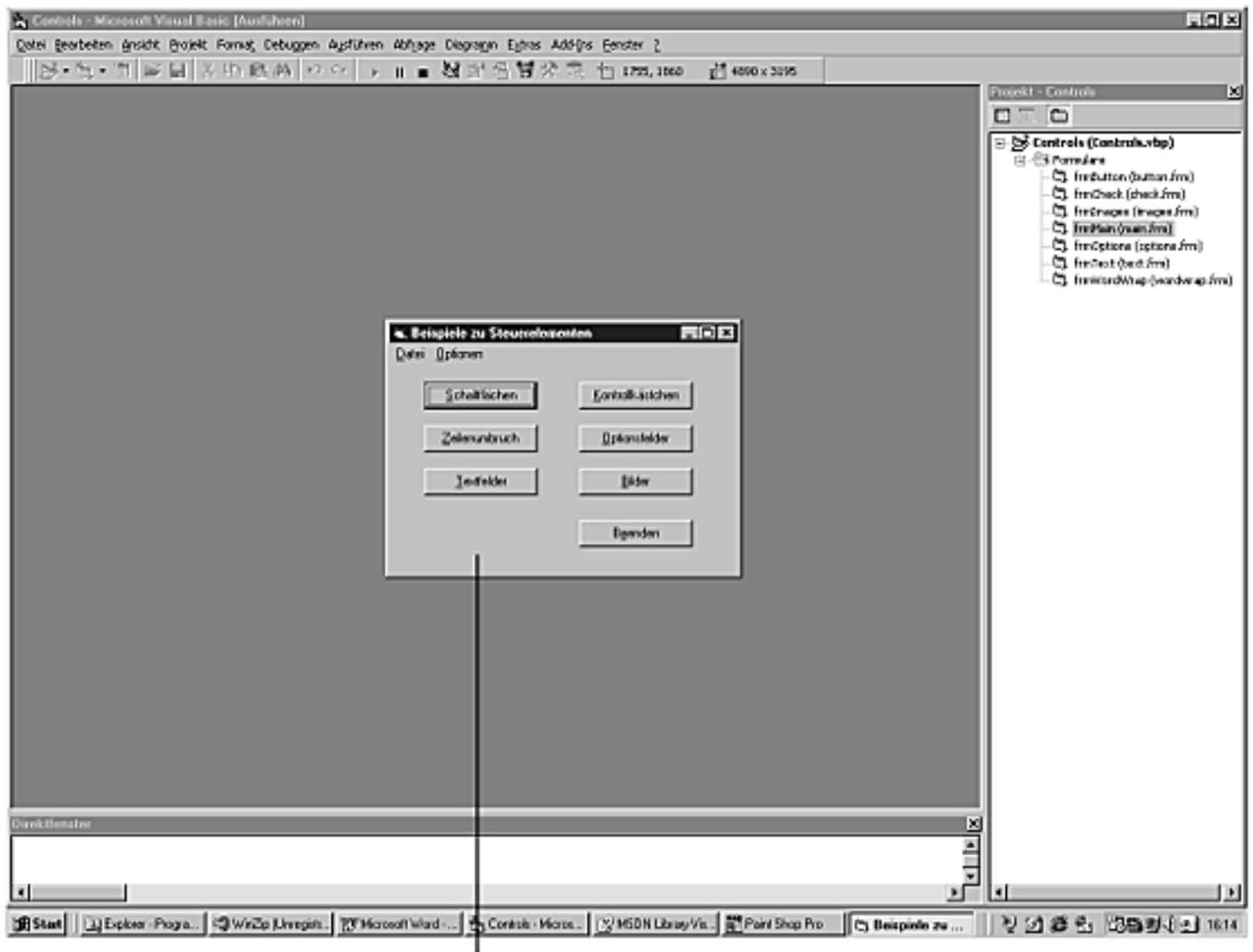


Das *Quellprogramm* setzt sich aus dem Code und den visuellen Bildschirmen Ihrer Anwendung zusammen. Dieses Quellprogramm geben Sie nicht an Ihre Benutzer weiter, weil man dort Änderungen vornehmen kann. Statt dessen geben Sie nur kompilierte Anwendungen weiter.

6. Nachdem Sie das Controls-Projekt geöffnet haben, sieht Ihr Visual-Basic-Bildschirm völlig unverändert aus. Die Controls-Anwendung wird nicht gestartet, weil Sie das Projekt erst von der CD geladen haben. Um die Anwendung (interpretiert) auszuführen, wählen Sie **Ausführen, Starten**. Sie sehen das in Abbildung 2.8 gezeigte Programm.



Wenn Sie eine Anwendung interpretiert ausführen, bleibt das Programmfenster innerhalb der aktiven Visual-Basic-Umgebung, so daß Sie das Programm anhalten und es ändern oder problematische Bereiche auswerten können. Wenn Sie ein Visual-Basic-Programm kompilieren, führen Sie es innerhalb von Windows, aber außerhalb der Visual-Basic-Umgebung aus.



Das Controls-Programmfenster

Abbildung 2.8:
Das Controls-Programm wird innerhalb der Visual-Basic-Umgebung ausgeführt.

Was Sie tun sollten	Was Sie lassen sollten
Um ein Programm zu starten, drücken Sie (F5) oder klicken Sie in der Symbolleiste auf die Schaltfläche Starten .	Wählen Sie nicht Ausführen , Starten im Menü aus.

7. Das Controls-Programm demonstriert mehrere Windows-Steuerelemente aus der Werkzeugsammlung. Klicken Sie auf eine Schaltfläche und probieren Sie die daraufhin erscheinenden Steuerelemente aus.

8. Nachdem Sie mehrere Programmoptionen getestet haben, klicken Sie auf die Schaltfläche **Schliessen**, um die Programmausführung zu beenden und das Programmfenster zu schließen. Sie gelangen zurück in die Visual-Basic-Umgebung. Die Controls-Anwendung ist weiterhin in die Umgebung geladen, läuft jedoch nicht. Jetzt können Sie einige der Fenster von Visual Basic betrachten.

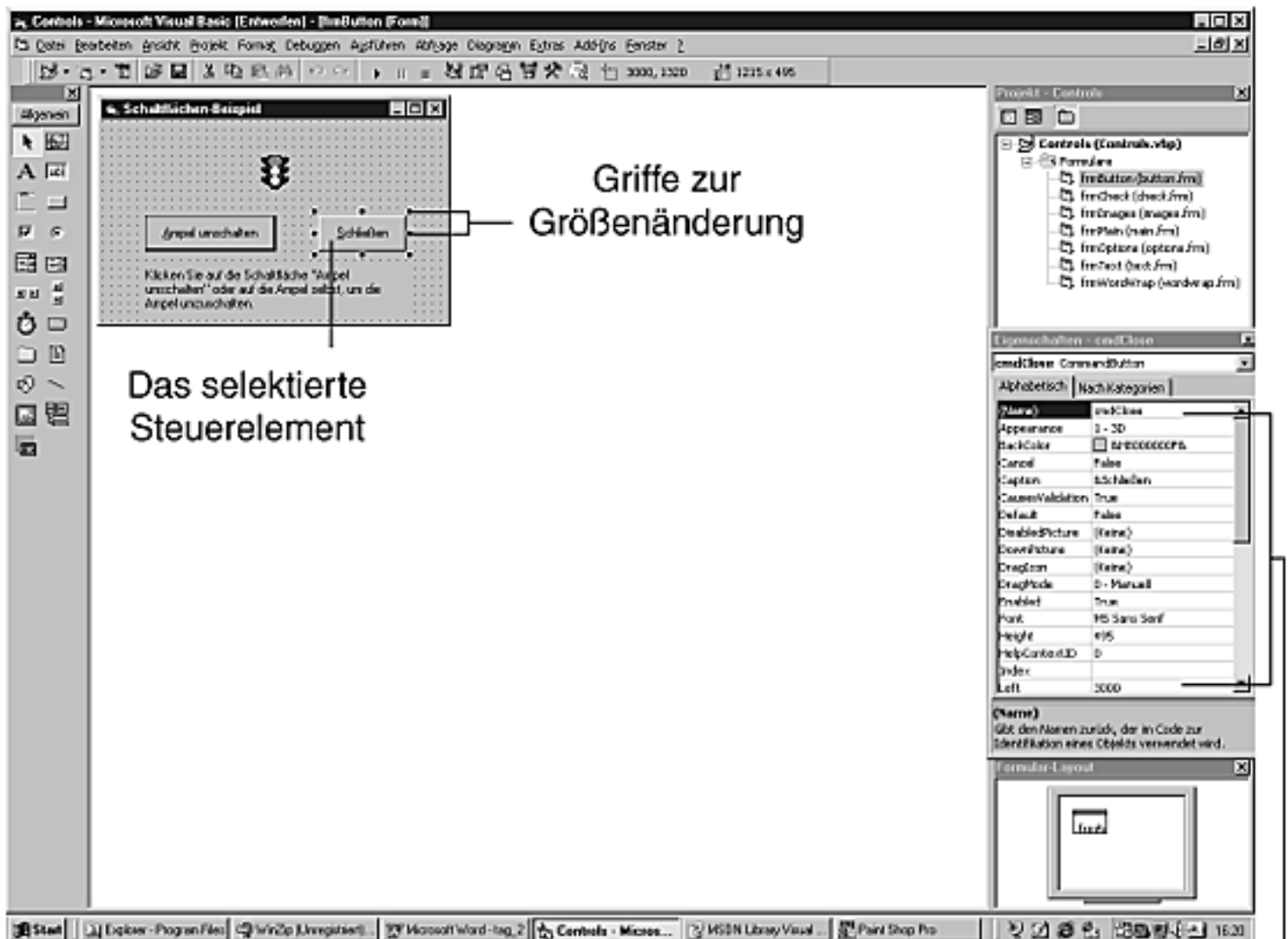
9. Sehen Sie das Projektfenster an. Das Controls-Programm besteht nur aus Formularen. Es gibt zwar

Code (klicken Sie im Projektfenster auf die Schaltfläche **Code anzeigen**, um das Codefenster zu sehen, und klicken Sie dann wieder auf die Schaltfläche **Objekt anzeigen**, um zur Formularliste zurückzugelangen), der sich aber in den sieben Formulardateien für das Projekt befindet.

10. Doppelklicken Sie im Projektfenster auf einen Formelnamen. Das Formular erscheint im Arbeitsbereich. Es sieht fast so aus wie während der Programmausführung. Sehen Sie in das Formular-Layout-Fenster (wählen Sie **Ansicht, Formular-Layout-Fenster** , falls es nicht angezeigt wird). Dort erkennen Sie, wo das Formular bei der Programmausführung auf dem Bildschirm angezeigt wird.

11. Ziehen Sie die Miniaturskizze des Formulars im Formular-Layout-Fenster an eine andere Position. Wenn Sie das Programm jetzt ausführen, wird das erste Formularfenster an der Stelle angezeigt, wohin Sie es im Formular-Layout-Fenster verschoben haben.

12. Betrachten Sie das Eigenschaftenfenster. Dort sehen Sie die Eigenschaftswerte für das ausgewählte Steuerelement. Beachten Sie, daß das Eigenschaftenfenster jeweils nur die Eigenschaftswerte für ein einziges Steuerelement des Formularfensters anzeigt. In Abbildung 2.9 sehen Sie das Eigenschaftenfenster für das selektierte Optionsfeld (gekennzeichnet durch die acht »Griffe«, die der Größenanpassung dienen).



Eigenschaftswerte für das selektierte Steuerelement

Abbildung 2.9:
Das Eigenschaftensfenster zeigt die Eigenschaftswerte für ein selektiertes Steuerelement an.

13. Blättern Sie in den Eigenschaftswerten nach unten, um alle Eigenschaften für das ausgewählte Steuerelement anzusehen.

14. Klicken Sie auf ein anderes Steuerelement im Formular und sehen Sie sich das aktualisierte Eigenschaftenfenster an. Wenn Sie ein Steuerelement durch Anklicken auswählen, werden um das Steuerelement herum Griffe für die Größenänderung angezeigt, und das Eigenschaftenfenster zeigt die Eigenschaften des neu ausgewählten Steuerelements an.

Lassen Sie das Projekt für den nächsten Abschnitt geöffnet. Schließen Sie die Symbolleisten **Bearbeiten** und **Formular-Editor**, weil Sie sie im nächsten Abschnitt nicht brauchen. Nachdem Sie sich nun mit der Visual-Basic-Umgebung besser vertraut gemacht haben, können Sie sie mit einer eigenen Kreation füllen.

Eine Anwendung von Grund auf neu anlegen

In diesem letzten Abschnitt des Kapitels werden Sie selbst eine Anwendung entwickeln. Sie verstehen sicher noch nicht alles, was in den nächsten paar Minuten geschehen wird, aber Sie sollten das Beispiel unbedingt nachvollziehen, weil es Sie auf alle weiteren Dinge vorbereiten wird. Diese erste Anwendung ist einfach, zeigt aber schon, wie einfach die Programmierung unter Visual Basic ist. In den weiteren Kapiteln werden die hier angesprochenen Aspekte detailliert beschrieben.

Einrichtung des Formulars

Diese erste Anwendung zeigt ein Bild und eine Schaltfläche an. Das Programm ändert das Bild, wenn Sie auf die Schaltfläche klicken. Gehen Sie dazu wie folgt vor:

1. Wählen Sie **Datei, Neues Projekt**, um das Dialogfeld **Neues Projekt** anzuzeigen. Falls Sie noch eine Anwendung aus einem früheren Abschnitt geöffnet haben, fragt Visual Basic Sie, ob Sie die Änderungen an dieser Anwendung speichern wollen. Antworten Sie mit **Nein**, um keine Änderungen zu speichern.
2. Wählen Sie das Icon **Standard-EXE** aus. Ihre Visual-Basic-Umgebung enthält nur ein einziges Formular, **Form1** (wie in der Titelleiste angezeigt). Das Formular erscheint auf dem weißen Arbeitsbereich zur Formularbearbeitung.
3. Klicken Sie auf die **Maximieren**-Schaltfläche, um den Arbeitsbereich für die Formularbearbeitung auf die maximale Größe zu bringen (den weißen Hintergrund, *nicht* das graue Formular). Damit haben Sie Platz, das Formular zu vergrößern.



Um das Formular herum erscheinen die Griffe für die Größenänderung, weil es momentan das einzige Objekt im Arbeitsbereich ist. Das Eigenschaftenfenster zeigt die Eigenschaften des Formulars an. Formulare haben wie alle anderen Objekte Eigenschaften, die Sie ändern können.

4. Ziehen Sie den Griff für die Größenänderung, der sich an der unteren rechten Ecke befindet, nach unten und nach rechts. Beachten Sie, wie sich die Breiten- und Höhenangaben rechts in der Symbolleiste ändern. Vergrößern Sie das Formular so, daß es jetzt 7400 x 5200 *Twips* groß ist. Damit erhalten Sie einen kleinen Hintergrund für Ihr Programm. Abbildung 2.10 zeigt Ihren Bildschirm. (Ihr

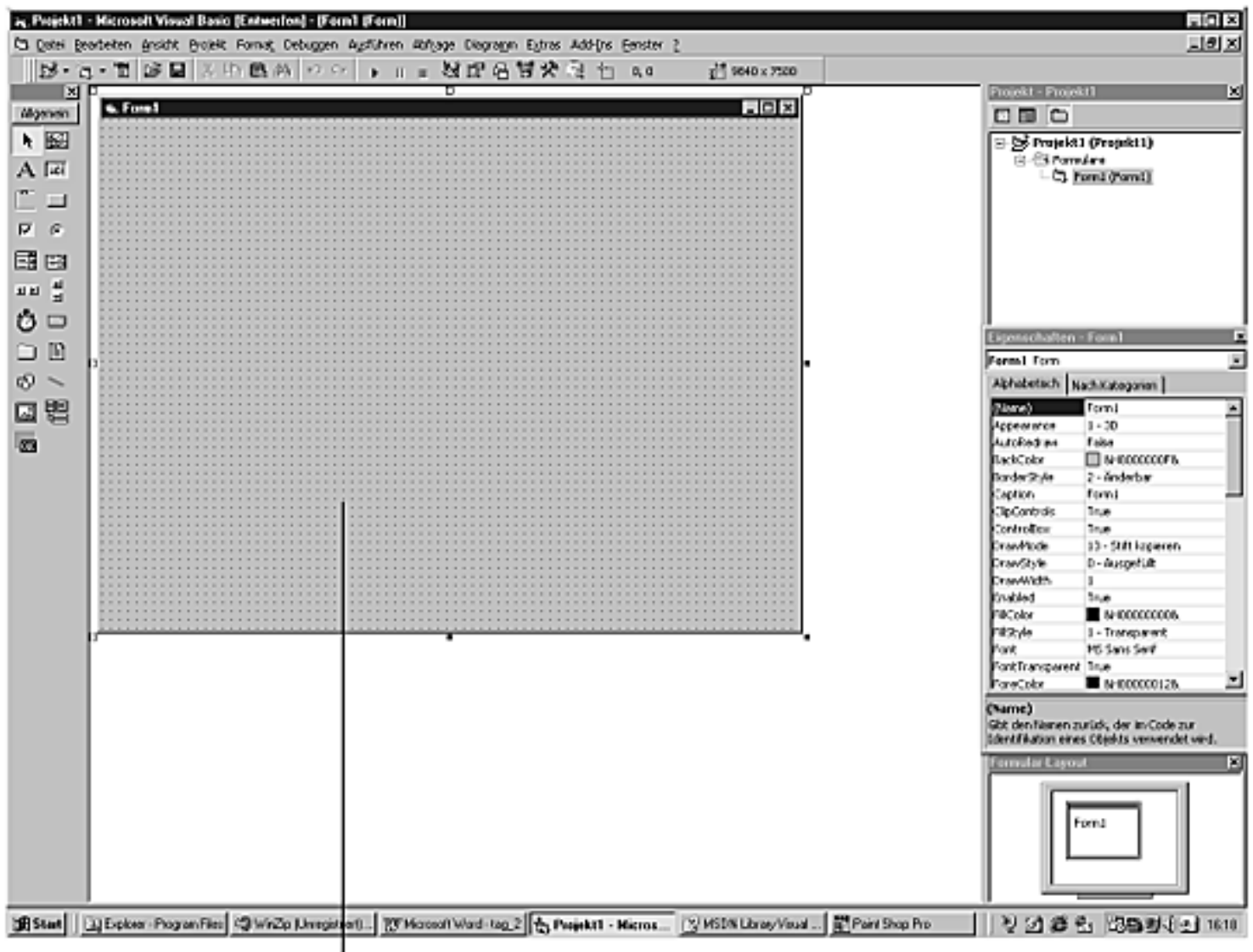
Formular-Layout-Fenster wird möglicherweise an einer anderen Stelle angezeigt.)



Ein *Twip* ist ein Bildschirmmaß. Ein Twip kann man sich wie einen Punkt auf dem Bildschirm vorstellen. Verschiedene Bildschirme und Grafikkarten erzeugen jedoch unterschiedliche Auflösungen und damit eine unterschiedliche Anzahl Punkte. Ein Twip ist ein von der Auflösung unabhängiges Maß für einen imaginären kleinen Bildschirmpunkt (kleiner als die höchste Auflösung erlaubt). Wenn Sie ein Formular also mit einer Größe von 7400 Twips anlegen, haben Sie nicht wirklich 7400 Punkte (sogenannte *Pixel*) auf dem Bildschirm.



Wenn Sie Formularfenster positionieren und ihnen eine Größe zuteilen, achten Sie auf die Positionskoordinaten und die Größenkoordinaten rechts in der Symbolleiste. Diese Werte werden immer paarweise angezeigt. Der erste Wert im Positionskoordinaten-Paar gibt die Anzahl der Twips an, wo das Fenster rechts neben dem linken Bildschirmrand angezeigt wird. Der zweite Wert gibt die Anzahl der Twips an, wo das Fenster unterhalb des oberen Bildschirmrands angezeigt wird. Das zweite Wertepaar, die Größenkoordinaten, gibt an, wie breit und wie hoch das Fenster sein soll. Die Formulareigenschaften für die Position heißen **Left** und **Top**. Sie stellen die Anzahl der Twips für den Abstand vom linken und oberen Bildschirmrand dar. Die Formulareigenschaften für die Größenkoordinaten sind **Width** und **Height**. Sie stellen die Breite und die Höhe des Formularfensters dar. Visual Basic aktualisiert diese Werte im Eigenschaftenfenster automatisch, wenn Sie das Formular im Arbeitsbereich verschieben oder seine Größe ändern.

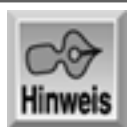


Formularfenster

Abbildung 2.10:
Mit der Größe des Formularfensters legen Sie gleichzeitig die Größe Ihres Programmfensters fest.

5. Wählen Sie **Ansicht, Formular-Layout-Fenster**, um das Fenster anzuzeigen, und zentrieren Sie das Symbol für das Formular dort, so daß es beim Programmstart ebenfalls zentriert angezeigt wird. Das Formularfenster selbst ändert sich nicht, aber die Positionskoordinaten reflektieren die neue Position.

6. Schließen Sie das Formular-Layout-Fenster, so daß Sie mehr Platz für die anderen Fenster haben.



Die Punkte, die innerhalb des Formularfensters erscheinen, bilden das *Raster*. Diese Punkte werden durch Auswahl von **Extras, Optionen** auf der Registerkarte **Allgemein** mit der Option **Raster anzeigen** aktiviert oder deaktiviert.



Ein *Raster* ist das Punktmuster, das Sie im Hintergrund des Formularfensters sehen. Die Dichte dieser Rasterpunkte wird im Dialogfeld **Extras, Optionen** eingestellt.

7. Weisen Sie dem Formular einen aussagekräftigeren Namen als `Form1` zu. Dabei lernen Sie gleichzeitig, wie man mit dem Eigenschaftenfenster umgeht. Die Eigenschaft (**Name**) (in Klammern angezeigt, so daß sie in der alphabetisch sortierten Liste immer oben erscheint) enthält den Namen des ausgewählten Formulars. (Im weiteren Text werden wir die Klammern um die Eigenschaft **Name** weglassen.) Blättern Sie im Eigenschaftenfenster nach oben, bis Sie die Eigenschaft **Name** sehen. Ihr Wert ist momentan `Form1`.

8. Klicken Sie auf die **Name**-Eigenschaft des Formulars und geben Sie als Namen `frmMyFirst` ein. Bei der Eingabe des Namens erscheint der Name sowohl rechts neben dem Eigenschaftsnamen **Name**, als auch in der Titelleiste von Visual Basic.



Alle Eigenschaften im Eigenschaftenfenster werden auf dieselbe Weise geändert, wie Sie sie eben für den Formularenamen kennengelernt haben. Blättern Sie zu der betreffenden Eigenschaft, klicken Sie darauf und geben Sie den neuen Wert ein (bzw. wählen Sie einen Wert aus einer bereitgestellten Dropdown-Liste aus).

9. Ändern Sie die Titelleiste des Formulars von dem ursprüngliche Wert auf »Neues Programm«. Dazu wählen Sie die Eigenschaft **Caption** aus und geben `Neues Programm` ein. Die **Caption**-Eigenschaft legt fest, was bei der Programmausführung in der Titelleiste des Formulars erscheint. Der neue Name erscheint sowohl im Eigenschaftenfenster als auch in der Titelleiste des Formulars.

10. Bevor Sie weitermachen, speichern Sie das Formular auf der Festplatte. Wählen Sie **Datei, Projekt speichern**. Die Option **Projekt speichern** speichert alle Dateien Ihres Projekts (es enthält momentan nur ein einziges Formular), und außerdem eine Projektbeschreibungsdatei mit der Dateinamenerweiterung `.VBP`. Visual Basic fordert Sie auf, den Dateinamen für Ihr Formular einzugeben. Es bietet den Wert der **Name**-Eigenschaft als Standarddateinamen an, mit der Dateinamenerweiterung `.FRM`. (Wenn Ihr Projekt mehrere Formulare oder Module oder andere Objekte enthält, fragt Visual Basic auch nach den Namen für diese anderen Elemente.) Anschließend fordert Visual Basic Sie auf, einen Projektnamen für die Projektbeschreibungsdatei einzugeben. Geben Sie dem Projekt den Namen `NeuProg`, bevor Sie es speichern. Antworten Sie mit **Nein**, wenn Visual Basic fragt, ob es dem Projekt die SourceSafe-Bibliothek hinzufügen soll.



Die Projektbeschreibungsdatei wird geladen, wenn Sie später einmal mit Ihrem Projekt arbeiten möchten. Wenn Sie eine solche Projektbeschreibungsdatei laden, lädt Visual Basic alle dem Projekt zugeordneten Dateien; ihre Namen erscheinen im Projektfenster.

Die Details

Nachdem Sie den Hintergrund der Anwendung angelegt haben, können Sie mit den Steuerelementen Details einrichten. Zum Einfügen von Steuerelementen sind in der Regel einer oder mehrere der folgenden Schritte erforderlich:

1. Wählen Sie das betreffende Steuerelement aus der Werkzeugsammlung aus.
2. Legen Sie das Steuerelement an der richtigen Position ab.
3. Passen Sie die Größe des Steuerelements an.

4. Legen Sie die Eigenschaften des Steuerelements fest.

5. Aktivieren Sie das Steuerelement gegebenenfalls mit Visual-Basic-Code.

In den folgenden Schritten lernen Sie schnell, wie man Steuerelemente aus der Werkzeugsammlung auswählt und sie auf dem Formular ablegt. Im allgemeinen gibt es dafür zwei Methoden:

- Sie doppelklicken auf das Icon, das in der Werkzeugsammlung für das Steuerelement angezeigt wird. Visual Basic platziert das Steuerelement in der Mitte des Formulars. Anschließend verschieben Sie das Steuerelement an die richtige Position und passen die Größe mit Hilfe der Größenänderungs-Griffe an.
- Sie klicken auf das Icon, das in der Werkzeugsammlung für das Steuerelement angezeigt wird. Klicken Sie mit der Maustaste und halten Sie diese gedrückt. Ziehen Sie die Maus an die Position, an der das Steuerelement angelegt werden soll. Während Sie die Maus verschieben, zeichnet Visual Basic den Umriß des Steuerelements auf Ihr Formular. Nachdem Sie sich an der Position befinden, wo das Steuerelement angelegt werden soll, lassen Sie die Maustaste los.

In den folgenden Schritten wird die bisher angelegte Anwendung ergänzt:

1. Doppelklicken Sie auf das Bezeichnungsfeld-Steuerelement und schieben Sie den daraufhin angezeigten Fadenkreuz-Mauszeiger an die Position im Formular, wo das Steuerelement angelegt werden soll. Das Icon für das Bezeichnungsfeld-Steuerelement zeigt den Buchstaben A, wie Sie im vorigen Abschnitt bereits gesehen haben. (Wenn Sie nicht wissen, wofür ein Icon in der Werkzeugsammlung steht, schieben Sie den Mauszeiger darüber, um den Tooltip zu sehen). Jetzt ist das Bezeichnungsfeld auf Ihrem Arbeitsbereich selektiert, deshalb werden die Griffe für die Größenänderung dafür angezeigt. Darüber hinaus zeigt das Eigenschaftsfeld die Eigenschaften des Bezeichnungsfelds an, und in der Symbolleiste sehen Sie die Positions- und Größenkoordinaten, die jetzt die Maße des Bezeichnungsfeldes darstellen.

Ein Bezeichnungsfeld zeigt in einem Formular Text an. Dieses neue Bezeichnungsfeld nimmt die Überschrift für Ihre Anwendung auf.

2. Verschieben Sie das Bezeichnungsfeld nach oben, bis es sich etwa 1320 Twips vom linken Rand und 120 Twips vom oberen Rand des Formularfensters befindet. In der Symbolleiste sehen Sie die Positionskoordinaten.



Wenn Sie die Option **Am Raster ausrichten** auf der Registerkarte **Allgemein** im Dialogfeld **Extras, Optionen** nicht deaktiviert haben, richtet Visual Basic alle Steuerelemente automatisch am nächsten Rasterpunkt in Ihrem Formular aus, so daß Ihre Steuerelemente gleichmäßig angeordnet sind.

3. Doppelklicken Sie in der Werkzeugsammlung auf das Schaltflächen-Steuerelement, um in der Mitte Ihres Formulars eine Schaltfläche anzulegen.

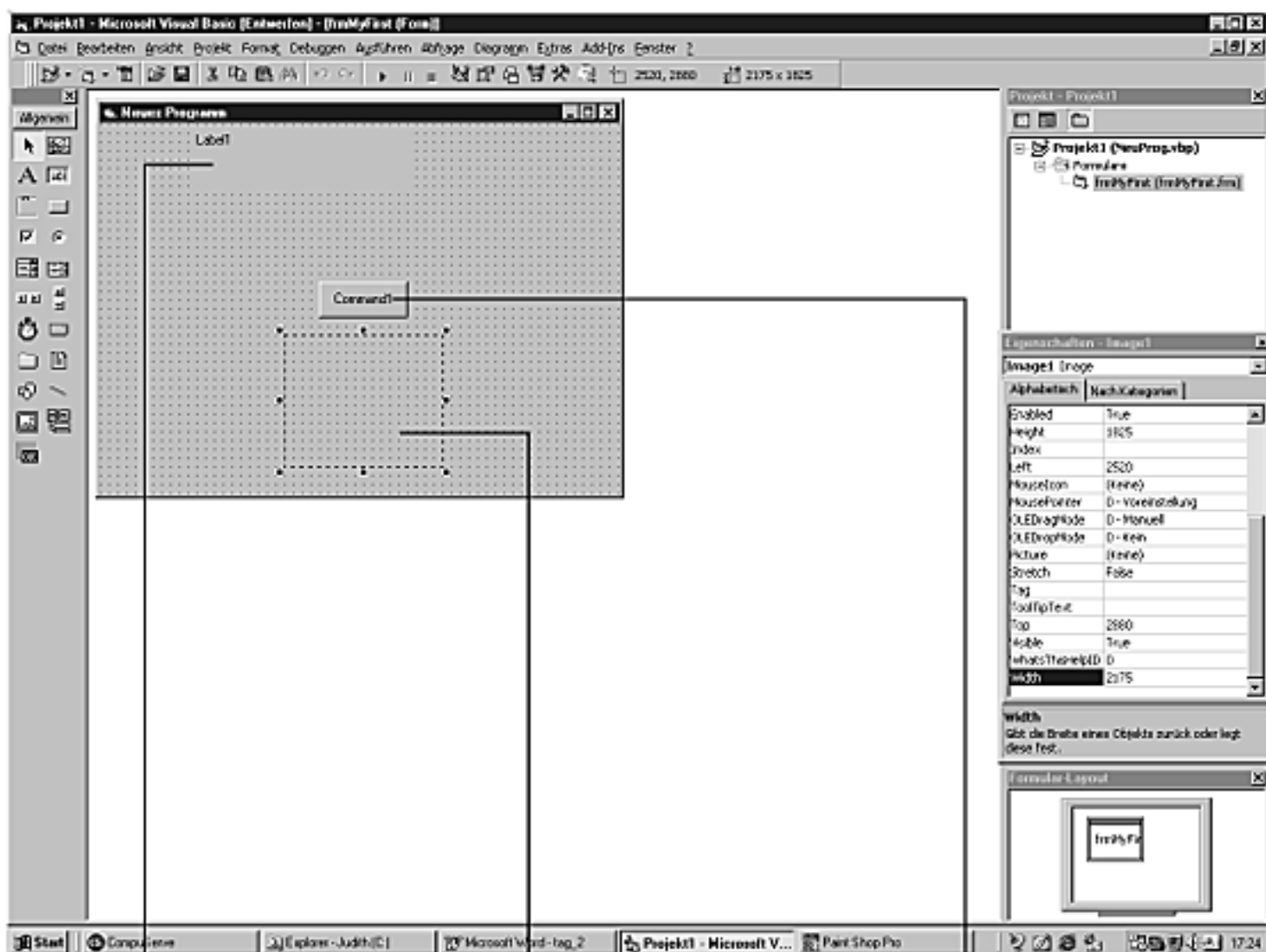
4. Klicken Sie einmal auf das Anzeigefeld-Steuerelement, und nicht zweimal, wie Sie es beim Bezeichnungsfeld gemacht haben. Schieben Sie Ihren Mauszeiger in das Formularfenster und spannen Sie das Anzeigefeld auf. Versuchen Sie, das Anzeigefeld 2520 Twips von der linken Kante des Formulars und 2880 Twips von der oberen Kante anzuordnen. Machen Sie das Anzeigefeld 2175 breit und 1825 hoch. Ziehen Sie langsam an den Griffen des Anzeigefelds, so daß der Tooltip die jeweiligen Koordinaten anzeigen kann. Nachdem Sie die richtigen Koordinaten eingestellt haben, lassen Sie die Maustaste los, um das Anzeigefeld mit dieser Größe anzulegen. Abbildung 2.11 zeigt Ihren Bildschirm an dieser Stelle. Das Anzeigefeld zeigt bei der Programmausführung ein grafisches Bild an.



Sie können das Anzeigefeld-Steurelement zwar in etwa an derselben Position und in derselben Größe anlegen, wie hier beschrieben, aber Sie können die Größe und die Position auch exakt bestimmen, indem Sie die entsprechenden Werte für die zuvor beschriebenen Eigenschaften eingeben: **Left: 2520, Top: 2880, Width: 2175 und Height: 1825**. Diese Notation wird für alle weiteren Beispiele in diesem Buch verwendet. Sie wissen jetzt, daß Sie dafür den Eigenschaftennamen anklicken und dann den neuen Eigenschaftswert eingeben müssen, um den Eigenschaften neue Werte zuzuweisen.



Positionskoordinaten und Größenkoordinaten werden immer paarweise angegeben. Häufig sehen Sie sie in Klammerschreibweise, beispielsweise einen Positionswert von (2520, 2880).



Bezeichnungsfeld

Anzeigefeld

Befehlsschaltfläche

Abbildung 2.11:
Ihre Anwendung nimmt Form an.

5. Nachdem Sie wissen, wie man Eigenschaftswerte für Steuerelemente setzt, sind Sie gerüstet, um weitere Eigenschaften für das Formular und die Steuerelemente zu setzen - auch wenn Sie viele dieser

Eigenschaften noch nicht verstehen -, um Ihre Anwendung zu verschönern. Sie legen geeignete Eigenschaftswerte fest und fügen dann Code hinzu, der die Steuerelemente verknüpft und eine Zusammenarbeit zwischen ihnen realisiert.

Tabelle 2.2 enthält eine Liste aller Eigenschaftswerte, die Sie für das Formular und die drei Steuerelemente setzen sollten. Beachten Sie, daß Sie dazu jeweils das Formular oder das betreffende Steuerelement selektieren müssen, um die Eigenschaftswerte zu ändern. Um das Formular zu selektieren, klicken Sie irgendwo in das Formular oder in die Titelleiste, aber nicht an einer Position, wo ein Steuerelement angelegt wurde. Das Eigenschaftenfenster zeigt die Eigenschaften des Formulars an. Klicken Sie auf das Bezeichnungsfeld, die Schaltfläche oder das Anzeigefeld, um diese zu selektieren und ihre Eigenschaften zu ändern. Dazu klicken Sie jeweils einen Eigenschaftennamen an und geben den neuen Wert ein.

	Auf den ersten Blick erscheinen Ihnen die Schrift-Informationen für ein Steuerelement möglicherweise verwirrend. Wenn Sie die Font-Eigenschaft für ein Steuerelement auswählen, erscheint hinter dem Eigenschaftswert eine Schaltfläche mit drei Punkten. Diese drei Punkte weisen darauf hin, daß Sie für die Font-Eigenschaft mehrere Werte festlegen können. Klicken Sie auf die Schaltfläche mit den drei Punkten. Das Dialogfeld Schriftart erscheint, wie in Abbildung 2.12 gezeigt. Nachdem Sie hier die gewünschten Werte ausgesucht und auf OK geklickt haben, werden alle Eigenschaftswerte, die sich auf die Schrift beziehen, auf das Bezeichnungsfeld angewendet. Das Steuerelement wird unmittelbar mit den neuen Werten angezeigt.

Tabelle 2.2: Weisen Sie dem Formular und den Steuerelementen der Anwendung die hier aufgelisteten Eigenschaftswerte zu

Steuerelement	Eigenschaft	Eigenschaftswert
Formular	MaxButton	False (die möglichen Werte sehen Sie im Dropdown-Listenfeld)
Bezeichnungsfeld	Alignment	2 - Zentriert (die möglichen Werte sehen Sie im Dropdown-Listenfeld)
Bezeichnungsfeld	Name	lblHappy
Bezeichnungsfeld	Caption	Schönen Tag!
Bezeichnungsfeld	Font	Courier New
Bezeichnungsfeld	Schriftschnitt	Fett
Bezeichnungsfeld	Size	36
Bezeichnungsfeld	Left	1320
Bezeichnungsfeld	Height	1695
Bezeichnungsfeld	Top	120
Bezeichnungsfeld	Width	4695
Anzeigefeld	Name	imgHappy
Anzeigefeld	Stretch	True

Befehlsschaltfläche	Name	cmdHappy
Befehlsschaltfläche	Caption	Klicken Sie hier!



Während Sie Ihre Anwendung entwickeln, können Sie sie immer wieder ausführen, um zu prüfen, was sie bisher schon kann. Wenn Sie beispielsweise jetzt (F5) drücken, überprüft Visual Basic Ihr Programm und zeigt ein aktives Programmfenster mit einer Schaltfläche an, die Sie bereits klicken können. Es passiert jedoch nichts, wenn Sie die Schaltfläche anklicken - außer daß Sie sehen, wie sie gedrückt wird. Es erscheint auch nichts, was Sie im Anzeigefeld platziert haben. Im nächsten Abschnitt werden Sie lernen, diese beiden nachrangigen Probleme zu lösen. Um das Programm zu verlassen, klicken Sie auf die **Schliessen**-Schaltfläche im Programmfenster. Wie Sie eine Möglichkeit schaffen, das Programm ordentlich zu verlassen, erfahren Sie in Kapitel 3.

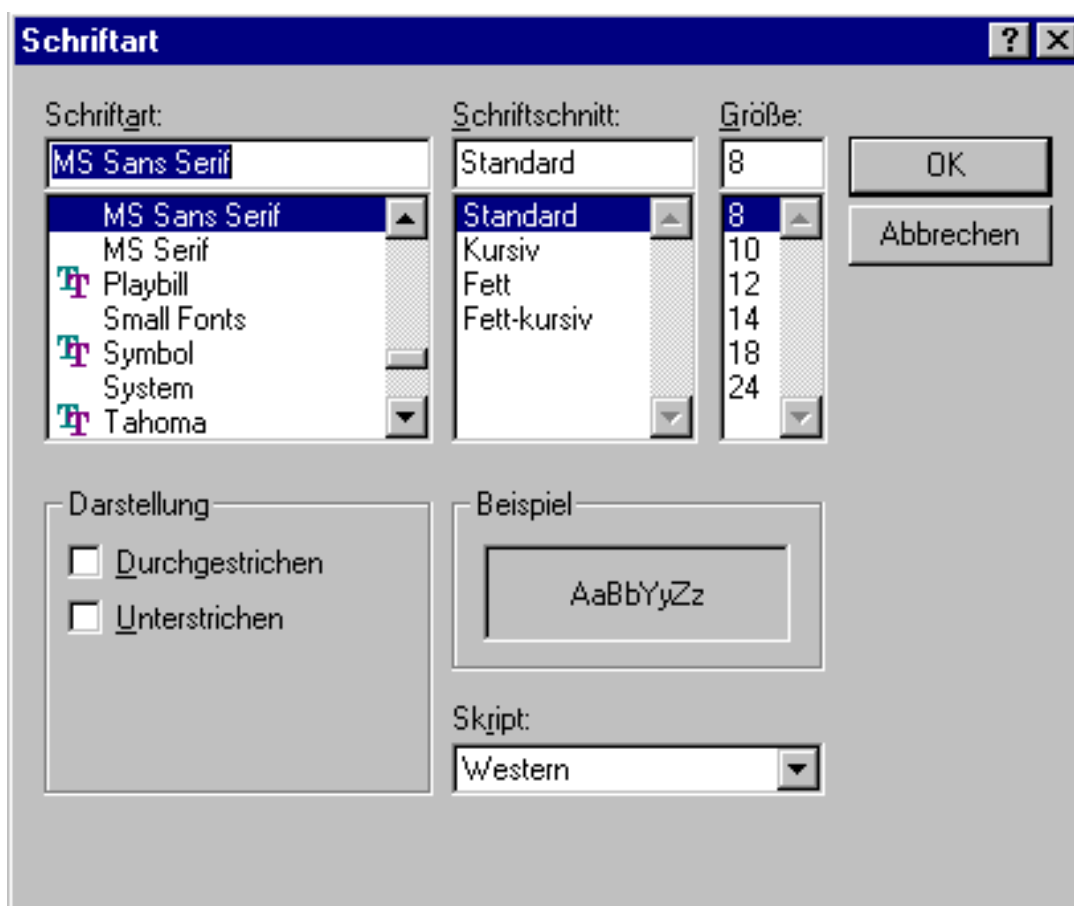


Abbildung 2.12:

Im Dialogfeld Schriftart setzen Sie mehrere Eigenschaftswerte für die Font-Eigenschaft.

Code - das Finish

Visual-Basic-Programmieranweisungen bewirken, daß Ihre Anwendung funktioniert - auch wenn sie bisher noch einfach ist. Der folgende Prozeß erscheint Ihnen vielleicht wie Zauberei, weil Sie Code, der sehr kompliziert aussieht, in ein Codefenster schreiben, das völlig unerwartet erschienen ist. Befolgen Sie einfach die nächsten Schritte, um den Code einzufügen - die nächsten Kapitel werden Ihnen erklären, was Sie getan haben.

1. Doppelklicken Sie irgendwo in das Raster des Formularfensters. Das Formular verschwindet und das Codefenster wird angezeigt. Ganz oben im Codefenster sehen Sie die beiden folgenden Zeilen:

```
Private Sub Form_Load()
```

```
End Sub
```

Diese Zeilen sind zwei der vier Zeilen, die Sie für den Code im Formular brauchen. Das Codefenster arbeitet wie eine kleine Textverarbeitung, wo Sie Anweisungen, die in Ihrem Programmcode erscheinen, hinzufügen, löschen und ändern können.



Der gesamte Code wird in *Prozeduren* angeordnet. Alle Prozeduren brauchen eine Anfangs- und Endzeile, die die Anfangs- und Endposition definieren. Visual Basic fügt die erste und die letzte Zeile vieler Prozeduren automatisch ein, wie hier gezeigt.



Eine *Prozedur* ist ein Abschnitt im Visual-Basic-Programmcode, der Visual-Basic-Programmieranweisungen enthält und eine bestimmte Aufgabe ausführt, beispielsweise das Zentrieren eines Formulars auf dem Bildschirm.

2. Jetzt können Sie die folgenden Zeilen eingeben. Drücken Sie dreimal die Leertaste, bevor Sie eine Zeile eingeben, um sie gegenüber den Anfangs- und Endzeilen einzurücken.

Visual-Basic-Programmierer rücken den Rumpf von Prozeduren häufig ein, um den Anfang und das Ende in einer Liste mehrerer Prozeduren einfacher zu erkennen. Der folgende Code stellt sicher, daß Ihr Anwendungsfenster immer in der Bildschirmmitte erscheint, egal mit welcher Bildschirmauflösung das Programm ausgeführt wird.

```
frmMyFirst.Left = (Screen.Width - frmMyFirst.Width) / 2  
frmMyFirst.Top = (Screen.Width - frmMyFirst.Height) / 2
```

Nachdem Sie in der ersten Zeile ein paar Zeichen geschrieben haben, zeigt Visual Basic das Dropdown-Listenfeld **Elemente automatisch auflisten** an, wie in Abbildung 2.13 gezeigt. Sobald Visual Basic erkennt, daß Sie den Eigenschaftswert eines Steuerelements schreiben möchten, bietet es Ihnen dieses Dropdown-Listenfeld an, in dem alle Eigenschaften für dieses Steuerelement aufgelistet werden, so daß Sie sie einfach auswählen, statt den vollständigen Namen eintippen zu müssen. Nachdem Sie eine Eigenschaft ausgewählt und die Leertaste gedrückt haben, fügt Visual Basic die Eigenschaft ein und Sie schreiben in der Zeile weiter.

3. Klicken Sie im Projektfenster auf die Schaltfläche **Objekt anzeigen**, um in das Formularfenster zurückzukehren.

4. Doppelklicken Sie auf die Befehlsschaltfläche, um wieder das Codefenster anzuzeigen. Dort wird der zuvor eingegebene Code angezeigt, ebenso wie zwei neue Anfangs- und Endanweisungen für eine neue Prozedur, die der Befehlsschaltfläche zugeordnet ist. Drücken Sie die Leertaste dreimal und geben Sie zwischen den bereits vorhandenen Zeilen die folgende Zeile ein:

```
imgHappy.Picture = LoadPicture("\Program Files\Microsoft Visual  
Studio\Common\Graphics\Bitmaps\Assorted\Happy.bmp")
```

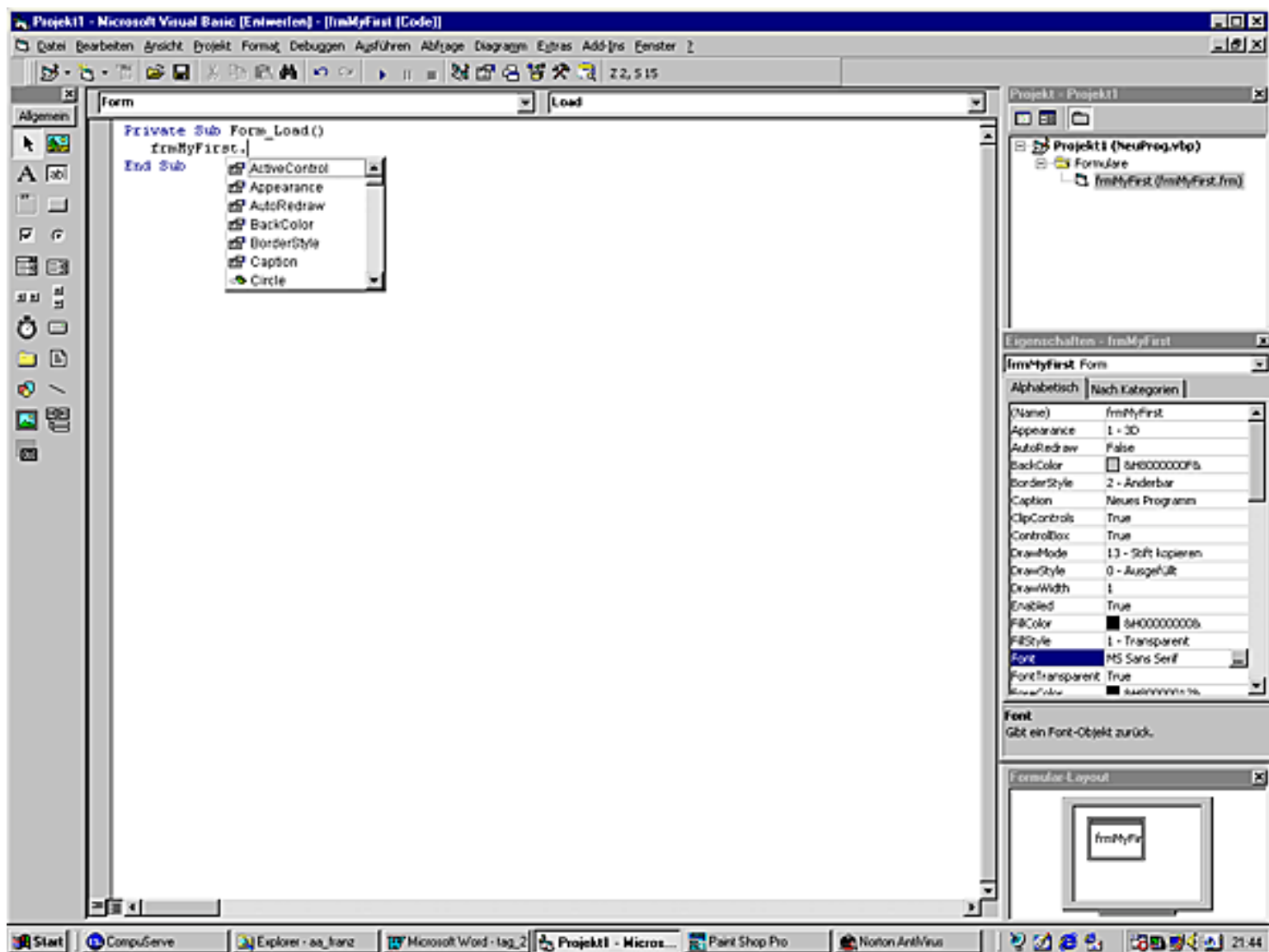


Abbildung 2.13:
Visual Basic hilft Ihnen, Ihre Codeeingabe zu beschleunigen.

Sobald Sie die öffnende Klammer für LoadPicture eingeben, zeigt Visual Basic eine Popup-Hilfe an, ähnlich dem Dropdown-Listefeld, das die Elemente automatisch auflistet, und das Sie bereits kennengelernt haben. Einige Visual-Basic- Anweisungen, insbesondere solche, wie Sie hier in dieser Anwendung sehen, machen es erforderlich, einen oder mehrere Werte einzugeben. Mehr über diese Werte und warum sie notwendig sind, erfahren Sie später, wenn Sie mehr über die Sprache gelernt haben. Visual Basic ist eine sehr umfangreiche Sprache, deshalb ist diese Unterstützung sehr praktisch.

5. Führen Sie Ihr Programm aus und klicken Sie auf die Befehlsschaltfläche. Ein Bild wie das in Abbildung 2.14 gezeigte erscheint. Sie haben Ihre neue Anwendung angelegt, ohne dazu den Anwendungs-Assistenten zu brauchen. Diese Anwendung zeigt ein Bild an, sobald Sie auf die Befehlsschaltfläche klicken. Die Anwendung enthält Code, und alle Steuerelemente haben Eigenschaftswerte, die Sie selbst gesetzt haben.
6. Klicken Sie auf die Schliessen-Schaltfläche des Fensters, um das Programm zu beenden. Speichern Sie Ihr Projekt, bevor Sie Visual Basic verlassen.



Abbildung 2.14:

Ihre Anwendung erzeugt ein grafisches Bild, wenn Sie mit der Maus auf die Befehlsschaltfläche klicken.

Zusammenfassung

Dieses Kapitel hat Ihnen die Visual-Basic-Umgebung vorgestellt. Sie werden nur dann ein guter Visual-Basic-Programmierer, wenn Sie die Arbeitsweise der verschiedenen Fenster und der Oberfläche verstanden haben. Visual Basic bietet mehrere Hilfeebenen, unter anderem Online-Hilfe, Web-Unterstützung und einen persönlichen technischen Support, der auf unterschiedliche Weise bereitgestellt wird. Wenn die Visual-Basic-Oberfläche oder Sprache ein Problem aufwirft, das Sie alleine nicht lösen können, haben Sie die Hilfe unmittelbar zur Hand.

Um eine Anwendung zu erstellen, legen Sie ein neues Projekt an, fügen im Formularfenster die gewünschten Steuerelemente ein, setzen die Eigenschaften für das Formular und die Steuerelemente und aktivieren diese Steuerelemente mit Code. Das Projekt, das Sie in diesem Kapitel angelegt haben, war sicher überraschend einfach - insbesondere, wenn man bedenkt, wie wenig Code dafür erforderlich war (drei Zeilen).

Kapitel 3 wird einige Fragen zu Steuerelementen und ihren Eigenschaften beantworten.

Fragen & Antworten

Frage:

Wie groß sollte das Formularfenster für meine Anwendung sein?

Antwort:

Ihre Anwendung bestimmt, wie groß Ihr Formularfenster sein sollte, und ebenso, wie viele Formulare Sie brauchen. Für einfache Programme ist ein einziges Formular ausreichend, aber ihre Größe ist davon abhängig, wie viele Steuerelemente Sie darauf ablegen möchten, und welche Aufgabe ein Programm hat.

Das Projekt aus diesem Kapitel hätte auch mit einem maximierten Formular angelegt werden können, aber bei nur drei Steuerelementen gäbe es dann zu viel leeren Platz auf dem Formular.

Frage:

Was hat der Code aus dem heutigen Kapitel bewirkt?

Antwort:

Der Code war erforderlich, damit die heute entwickelte Anwendung korrekt funktioniert. Die Zeile mit dem Schlüsselwort LoadPicture ist wichtig, weil sie das Bild lädt, wenn der Benutzer auf die Befehlsschaltfläche klickt. Die beiden anderen Zeilen, die Sie eingetragen haben, nachdem Sie in das Formularfenster doppelgeklickt haben, um das Codefenster zu öffnen, zentrieren das Formular auf dem Bildschirm, egal welche Bildschirmgröße verwendet wird.

Frage:

Wenn der Code das Formularfenster zentriert, muß ich es dann auch im Formular-Layout-Fenster zentrieren?

Antwort:

Egal, wo im Formular-Layout-Fenster Sie das Formular plziert haben, die beiden Zeilen in der Prozedur für das Formular zentrieren es, sobald das Programm ausgeführt wird.

Das Formular-Layout-Fenster soll Ihnen nur in etwa zeigen, wo das Formularfenster beim Laden angezeigt wird. Eine genaue Kontrolle, insbesondere, wenn Sie nicht wissen, wie groß der Bildschirm ist, auf dem Ihr Programm ausgeführt wird, ist nur durch die Positionierung des Formulars vom Codefenster aus möglich.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist der Unterschied zwischen der Symbolleiste und der Werkzeugsammlung?
2. Welche Namen hat der Abonnement-Online-Service, den Microsoft Programmierern bietet?
3. Richtig/Falsch. Das Formularfenster enthält jeweils nur ein Formular.

4. Was passiert, wenn Sie in der Werkzeugsammlung auf ein Steuerelement klicken?
5. Was passiert, wenn Sie in der Werkzeugsammlung auf ein Steuerelement doppelklicken?
6. Richtig/Falsch. Die Steuerelementeigenschaften werden im Fenster der Werkzeugsammlung gesetzt.
7. Wie erkennt Visual Basic, welche Steuerelementeigenschaften im Eigenschaftenfenster erscheinen sollen?
8. Worauf weist die Schaltfläche mit den drei Punkten in einem Eigenschaftenfenster hin?
9. Welchen Namen hat die Eigenschaft, mit der der Titel einer Befehlsschaltfläche festgelegt wird?
10. Warum sollten Sie für die Namen Ihrer Steuerelemente nicht die Vorgabewerte verwenden?

Übung

1. Laden Sie die Anwendung, die Sie heute entwickelt haben, so daß Sie damit arbeiten können. Fügen Sie ihr ein bißchen Farbe hinzu, indem Sie den Hintergrund des Formulars blau machen. Darüber hinaus fügen Sie eine weitere Schaltfläche ein, Beenden, und geben ihr einen entsprechenden Titel. Fügen Sie die folgende Zeile in die neue Prozedur ein, die im Codefenster für die Befehlsschaltfläche erscheint:

End

Führen Sie die Anwendung aus und klicken Sie auf die **Beenden**-Schaltfläche, um die Anwendung auf diese Weise ordentlich zu schließen als zuvor.

Woche 1

Tag 3

Steuerelemente - Verwaltung

Nachdem Sie nun zwei Anwendungen entwickelt haben - eine mit dem Anwendungs- Assistenten, die andere ohne -, sollen Sie langsam erfahren, wie Visual Basic funktioniert. In den beiden ersten Kapiteln haben Sie gesehen, wie einfach es ist, Programme zu entwickeln. Und Sie haben sich ein bißchen an die Visual-Basic-Umgebung gewöhnt. Heute werden Sie lernen, wie die Komponenten eines Visual-Basic-Programms zusammenarbeiten, insbesondere die Steuerelemente und die Eigenschaften.

Sie lernen die folgenden Dinge kennen:

- Allgemeine Steuerelementeigenschaften
- Warum haben Steuerelemente so viele Eigenschaften?
- Die allgemeinen Werkzeuge in der Werkzeugsammlung
- Wie die Beschleunigungstasten die Dateneingabe vereinfachen
- Wie der Fokus den Benutzern hilft, Steuerelemente auszuwählen
- Ereignisprozeduren

Steuerelemente

In der Werkzeugsammlung finden Sie alle Steuerelemente, die Sie in Ihren Anwendungen einsetzen können. Diese Werkzeugsammlung enthält beliebig viele dieser Steuerelemente, d.h. Sie können so oft Sie möchten beispielsweise auf das Bezeichnungsfeld-Steuerelement klicken, um damit beliebig viele Bezeichnungsfelder auf Ihrem Formular anzulegen.

Welche Steuerelemente (auch als Werkzeuge bezeichnet) in der Werkzeugsammlung angezeigt werden, hängt davon ab, was Sie für Ihre Anwendung brauchen. Größtenteils verwenden Sie für die Übungen in diesem Buch die normale Werkzeugsammlung, die Sie in Kapitel 2 kennengelernt haben. Diese Werkzeuge sind Standardsteuerelemente, die beim Anlegen einer neuen Anwendung geladen werden.

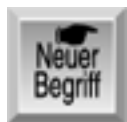


Ein Steuerelement steht Ihrem Programm nur dann zur Verfügung, wenn es in der Werkzeugsammlung angezeigt wird. Sie können beispielsweise Ihrem Formular kein Steuerelement für das Internet-Browsing hinzufügen, wenn Sie dieses Ihrer Werkzeugsammlung nicht hinzugefügt haben. Wenn Sie den Anwendungs-Assistenten verwenden, fügt der Assistent die Steuerelemente in Ihre Werkzeugsammlung ein, die für Ihr Projekt gebraucht werden.



Wenn Sie für eine Aufgabe in diesem Buch ein Steuerelement brauchen, das nicht in der Standard-Werkzeugsammlung enthalten ist, wird erklärt, wie Sie es hinzufügen.

Wie in Abbildung 3.1 gezeigt, kann Ihre Werkzeugsammlung relativ kompliziert werden, wenn zu viele Steuerelemente dort angelegt werden. Wenn die Werkzeugsammlung so groß ist, kann sie Platz einnehmen, den Sie für andere Fenster dringend brauchen. Sie können die Werkzeugsammlung an der Titelleiste an eine andere Position verschieben, wenn Sie beispielsweise links den Platz für die Anzeige eines Formulars brauchen. Außerdem können Sie das Fenster für die Werkzeugsammlung beliebig vergrößern oder verkleinern.



Visual-Basic-Puristen verwenden strengere Namen für die bereitgestellten Steuerelemente. Die Steuerelemente, die standardmäßig in der Werkzeugsammlung erscheinen, sind die *internen Steuerelemente*. ActiveX-Steuerelemente mit der Dateinamenerweiterung .OXC sind *externe Steuerelemente*, die Sie der Werkzeugsammlung hinzufügen können. *Einfügbare Steuerelemente* sind Steuerelemente aus externen Anwendungen, beispielsweise Microsoft Excel.



Abbildung 3.1:
Das Fenster für die Werkzeugsammlung sollte eine sinnvolle Größe haben.

Was Sie lassen sollten
Machen Sie die Werkzeugsammlung nicht so klein, daß Sie die Werkzeuge nicht mehr sehen. Für die Werkzeugsammlung werden keine Bildlaufleisten bereitgestellt. Um Steuerelemente zu benutzen, die außerhalb der Fenstergrenzen liegen, müssen Sie das Fenster für die Werkzeugsammlung wieder vergrößern, bis Sie das betreffende Steuerelement sehen.

Viele Steuerelemente haben ähnliche Eigenschaften. Tabelle 3.1 listet einige der Eigenschaften auf, die von den meisten Steuerelementen unterstützt werden. Sie erkennen vielleicht, warum die in Tabelle 3.1 beschriebenen Eigenschaften von so vielen Steuerelementen unterstützt werden. Alle Steuerelemente haben eine Position auf dem Bildschirm (angegeben durch die Eigenschaften `Left` und `Top`), eine Größe (angegeben durch die Eigenschaften `Width` und `Height`), und die meisten haben auch Vorder- und Hintergrundfarben, falls sie Text anzeigen.

Tabelle 3.1: Allgemeine Eigenschaften für die Visual-Basic-Steuerelemente

Eigenschaft	Beschreibung

Alignment	Legt fest, ob der Text im Steuerelement, wie beispielsweise in einem Bezeichnungsfeld oder einer Befehlsschaltfläche, links oder rechts ausgerichtet oder zentriert dargestellt werden soll.
BackColor	Bestimmt die Hintergrundfarbe des Steuerelements. Diese Farbe wählen Sie aus einer Farbpalette aus, die im Dropdown-Listenfeld der Eigenschaft angezeigt wird.
BorderStyle	Gibt an, ob für das Steuerelement ein Rahmen angezeigt werden soll.
Caption	Gibt an, welcher Text auf dem Steuerelement angezeigt wird.
Enabled	Diese Eigenschaft wird mit Hilfe eines Dropdown-Listenfelds gesetzt. Sie ist True , wenn Sie möchten, daß das Steuerelement auf Eingaben des Benutzers reagiert, andernfalls False . Diese Eigenschaft ist praktisch, um Steuerelemente zu aktivieren oder zu deaktivieren, wenn sie während der Programmausführung nicht zur Verfügung stehen.
Font	Zeigt das Dialogfeld Schriftart an, wo Sie verschiedene Schrifteigenschaften für die Schrift eines Steuerelements festlegen können, beispielsweise Größe und Stil.
ForeColor	Bestimmt die Vordergrundfarbe des Steuerelements. Diese Farbe wählen Sie aus einer Farbpalette aus, die im Dropdown-Listenfeld der Eigenschaft angezeigt wird.
Height	Gibt an, wie viele Twips das Steuerelement hoch ist.
Left	Gibt das Anfangs-Twip für den Abstand vom linken Formularrand an, wo das Steuerelement erscheinen soll. Für ein Formular gibt die Eigenschaft Left an, wie viele Twips vom linken Bildschirmrand entfernt dieses angezeigt werden soll.
MousePointer	Bestimmt, wie der Mauszeiger angezeigt wird, wenn der Benutzer die Maus zur Laufzeit über das Steuerelement schiebt.
Name	Legt den Namen für das Steuerelement fest. Wie Sie in Kapitel 2 gesehen haben, zeigt das Eigenschaftsfenster die Eigenschaft Name in Klammern an, so daß sie immer ganz oben in der Auflistung erscheint.
ToolTipText	Enthält den Text, der erscheint, wenn der Benutzer den Mauszeiger zur Laufzeit über das Steuerelement schiebt (ähnlich ScreenTips).
Top	Gibt das Anfangs-Twip für den Abstand vom oberen Formularrand an, wo das Steuerelement erscheinen soll. Für ein Formular gibt die Eigenschaft Top an, wie viele Twips vom oberen Bildschirmrand entfernt dieses angezeigt werden soll.
Visible	Diese Eigenschaft wird in einem Dropdown-Listenfeld gesetzt. Sie ist True , wenn das Steuerelement auf dem Formular sichtbar sein soll, andernfalls False .
Width	Gibt an, wie viele Twips das Steuerelement breit ist.

Beachten Sie, daß alle Steuerelemente Vorgaben für ihre Eigenschaften besitzen, noch bevor Sie einen Wert ändern. Sobald Sie ein Steuerelement anlegen, richtet Visual Basic bestimmte vordefinierte Eigenschaftswerte dafür ein (die gebräuchlichsten Werte). Darüber hinaus weist es den Steuerelementen Standardnamen und Titel zu, die Sie jedoch ändern sollten. Die meisten Standardwerte können unverändert übernommen werden. Es wird sicher nie vorkommen, daß Sie alle Eigenschaftswerte eines Steuerelements ändern.

Sie wissen bereits, daß Sie die Eigenschaften eines Steuerelements ändern können, während Sie die Anwendung entwickeln, aber Sie können sie auch während der Programmausführung ändern. Beispielsweise wird die Eigenschaft **Enabled** häufig während der Programmausführung geändert, wenn dem Benutzer ein bestimmtes Steuerelement vorübergehend nicht zur Verfügung stehen soll. Sie könnten beispielsweise eine Befehlsschaltfläche deaktivieren, die einen Bericht ausdruckt, bis der Benutzer den Inhalt dieses Berichts spezifiziert hat.



Im Eigenschaftenfenster werden nicht alle Eigenschaften für alle Steuerelemente angezeigt. Eigenschaften, die Sie nur über den Code setzen können, erscheinen nicht im Eigenschaftenfenster.



Einige Steuerelementeigenschaften, wie beispielsweise die Werte für **Alignment** , sehen seltsam aus, weil im Dropdown-Listenfeld dafür Nummern angegeben sind. Beispielsweise kann die **Alignment**-Eigenschaft die Werte **0 - Links**, **1 - Rechts** oder **2 - Zentriert** annehmen. Sie können diese Werte mit Hilfe der Maus aus der Liste auswählen, ohne sich um die Nummern kümmern zu müssen, Sie können aber auch das Dropdown-Listenfeld öffnen, die entsprechende Nummer eingeben, und so die Eingabe beschleunigen. Die Nummern sind auch praktisch, wenn die Werte im Code gesetzt werden sollen. Sie weisen einfach den numerischen Wert zu, statt den gesamten Wert zu tippen, wie beispielsweise **0 - Links**. Die Nummern stellen Abkürzungen dar, die Sie im Code verwenden können, wie Sie in den nächsten Kapiteln noch erfahren werden.

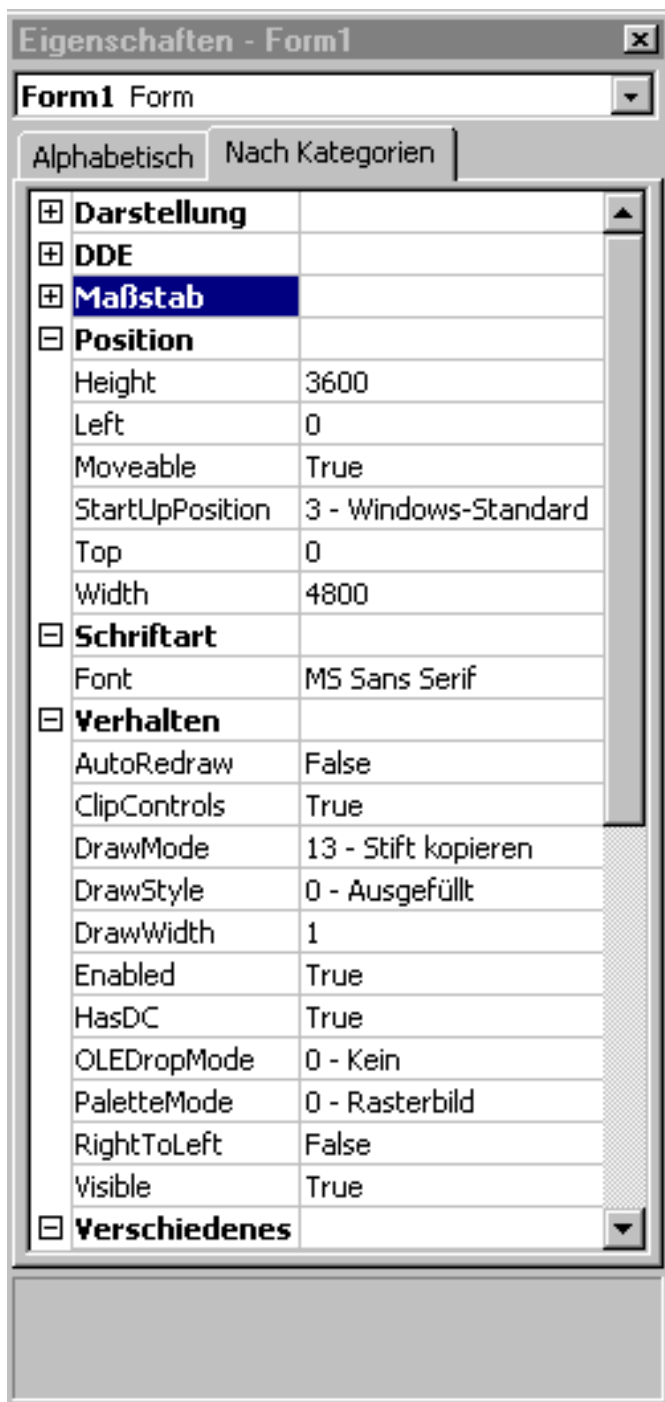


Abbildung 3.2:

Sie können die Eigenschaften nach Kategorien anzeigen, um das Gesuchte schneller zu finden.

Die folgenden Abschnitte beschreiben die wichtigsten Steuerelemente aus der Standard-Werkzeugsammlung. Dort werden Sie auch mehr über ihre Eigenschaften erfahren. Sie werden heute jedoch nicht alle Steuerelemente der Werkzeugsammlung kennenlernen, sondern nur diejenigen, die Sie für die ersten Tage der Visual-Basic- Programmierung benötigen.

Sie brauchen nicht alle Eigenschaften dieser Steuerelemente zu kennen, weil einige davon wirklich selten verwendet werden. Die meisten Visual-Basic-Programmierer kennen nur die wichtigsten Eigenschaften. Wenn Sie möchten, daß sich ein Steuerelement auf eine bestimmte Weise verhält, dann gibt es sehr wahrscheinlich auch eine Eigenschaft, die dies für Sie bewerkstelligt. Durch eine kurze Beschreibung der Steuerelemente und ihrer Eigenschaften erhalten Sie ein besseres Verständnis für die Arbeit mit Steuerelementen, und Sie können sich besser vorstellen, welche Eigenschaften dafür verwendet werden können.



Einige Programmierer lassen sich die Eigenschaften im Eigenschaftenfenster nach der Kategorie sortiert anzeigen. Um die Eigenschaften so zu gruppieren, klicken Sie im Eigenschaftenfenster auf die Registerkarte **Nach Kategorien**.

Formulareigenschaften

Viele Formulareigenschaften entsprechen den Eigenschaften anderer Steuerelemente, die Sie in Tabelle 3.1 bereits kennengelernt haben. Das Formular ist etwas Besonderes, weil es sich nicht auf einem Formular befindet, sondern direkt im Fenster des Benutzers erscheint. Deshalb entsprechen die Eigenschaften **Left**, **Top**, **Width** und **Height** den Rändern des Bildschirms und nicht einem Formularfenster.

Die folgenden Formulareigenschaften sind wichtig:

- **BorderStyle.** Diese Eigenschaft legt fest, wie das Fenster darauf reagiert, wenn ein Benutzer versucht, seine Größe zu ändern. Einige Werte, die Sie vielleicht brauchen, sind **0 - None**, so daß ein Formular ohne Ränder und Titelleiste angezeigt wird, und **1 - Fest Einfach** für ein Fenster, dessen Größe nicht verändert werden kann (der Benutzer kann das Fenster schließen, es aber nicht maximieren oder minimieren und es auch nicht in der Größe ändern). **2 - Sizable** (der Standardwert) zeigt ein reguläres Fenster an, dessen Größe verändert werden kann, ebenso wie Maximier- und Minimier-Schaltflächen.
- **ControlBox.** Die Werte **True** oder **False** dieser Eigenschaft legen fest, ob das *Steuermenü* für das Formular angezeigt werden soll.



Ein *Steuermenü* erscheint, wenn Sie auf das Icon in der oberen linken Ecke des Fensters klicken. Das Steuermenü ermöglicht Ihnen, ein Fenster zu verschieben, zu minimieren oder maximieren, es in der Größe zu ändern und es zu schließen.

- **Icon.** Diese Eigenschaft spezifiziert einen Icon-Dateinamen für ein Icon in der Windows-Symbolleiste, das erscheint, wenn der Benutzer das Formular minimiert.
- **MaxButton.** Diese Eigenschaft bestimmt, ob das Formular eine aktive Maximier- Schaltfläche enthält.
- **MinButton.** Diese Eigenschaft bestimmt, ob das Formular eine aktive Minimier- Schaltfläche enthält. (Wenn Sie **MaxButton** und **MinButton** auf **False** setzen, wird keine dieser Schaltflächen angezeigt.)
- **Movable.** Diese Eigenschaft bestimmt, ob der Benutzer das Formular verschieben kann, oder ob es an einer bestimmten Position bleiben muß.
- **ShowInTaskbar.** Diese Eigenschaft bestimmt mit den Werten **True** und **False**, ob das geöffnete Formular in der Taskleiste des Benutzers angezeigt wird.
- **StartPosition.** Diese Eigenschaft bietet eine schnelle Methode, die Anfangsposition des Formulars auf dem Bildschirm anzuzeigen. Einer der praktischsten Werte ist **2 - Bildschirmmitte**, der das Formular beim Starten in der Bildschirmmitte anzeigt.
- **WindowState.** Diese Eigenschaft legt die Größe (normal, maximiert oder minimiert) des Formulars fest (das ist praktisch, um ein minimiertes Formular aufzurufen).

Der Zeiger

Der Zeiger ist das einzige Element der Werkzeugsammlung, bei dem es sich nicht um ein Steuerelement handelt. Die einzige Aufgabe dieses Werkzeugs ist, den Fadenkreuz-Cursor verschwinden zu lassen, der erscheint, wenn Sie eines der anderen Steuerelemente in der Werkzeugsammlung angeklickt haben.

Das Bezeichnungsfeld (Label)

Das Bezeichnungsfeld zeigt Text an. Dieser Text kann zwar vom Code aus geändert werden, nicht jedoch zur Laufzeit vom Benutzer. (Mehr Informationen finden Sie in Kapitel 5.) Programmierer verwenden Bezeichnungsfelder häufig für Beschriftungen, Eingabeaufforderungen und Beschreibungen. Wenn Sie beispielsweise möchten, daß der Benutzer etwas in ein anderes Steuerelement eingibt, etwa in ein Textfeld, schreiben Sie ein Bezeichnungsfeld neben das betreffende Steuerelement. Ohne diese Beschriftung erkennt der Benutzer vielleicht sonst nicht, was er eingeben soll.

Die beiden Eigenschaften `AutoSize` und `WordWrap` bestimmen, wie der Text in einem Bezeichnungsfeld angezeigt wird. Wenn Sie `AutoSize` auf `True` setzen, wird das Bezeichnungsfeld automatisch horizontal über das Formular vergrößert, um den gesamten Text anzuzeigen, den Sie der Eigenschaft `Caption` zugewiesen haben. Wenn Sie `WordWrap` auf `True` setzen, behält Visual Basic die Breite des Bezeichnungsfelds bei, vergrößert es aber vertikal, um so viele Zeilen anzuzeigen, wie für den gesamten Inhalt erforderlich sind. Abbildung 3.3 zeigt drei Versionen eines Bezeichnungsfeldes, die alle dieselbe `Caption`-Eigenschaft haben, aber mit unterschiedlichen Kombinationen der Eigenschaften `AutoSize` und `WordWrap`. Hier sehen Sie, wie sich die verschiedenen Werte auswirken.



Damit `WordWrap` funktioniert, müssen Sie `AutoSize` auf `True` setzen. Visual Basic muß in der Lage sein, das Bezeichnungsfeld wenigstens teilweise zu verbreitern, wenn ein einzelnes Wort im Titel größer als seine Breite ist.

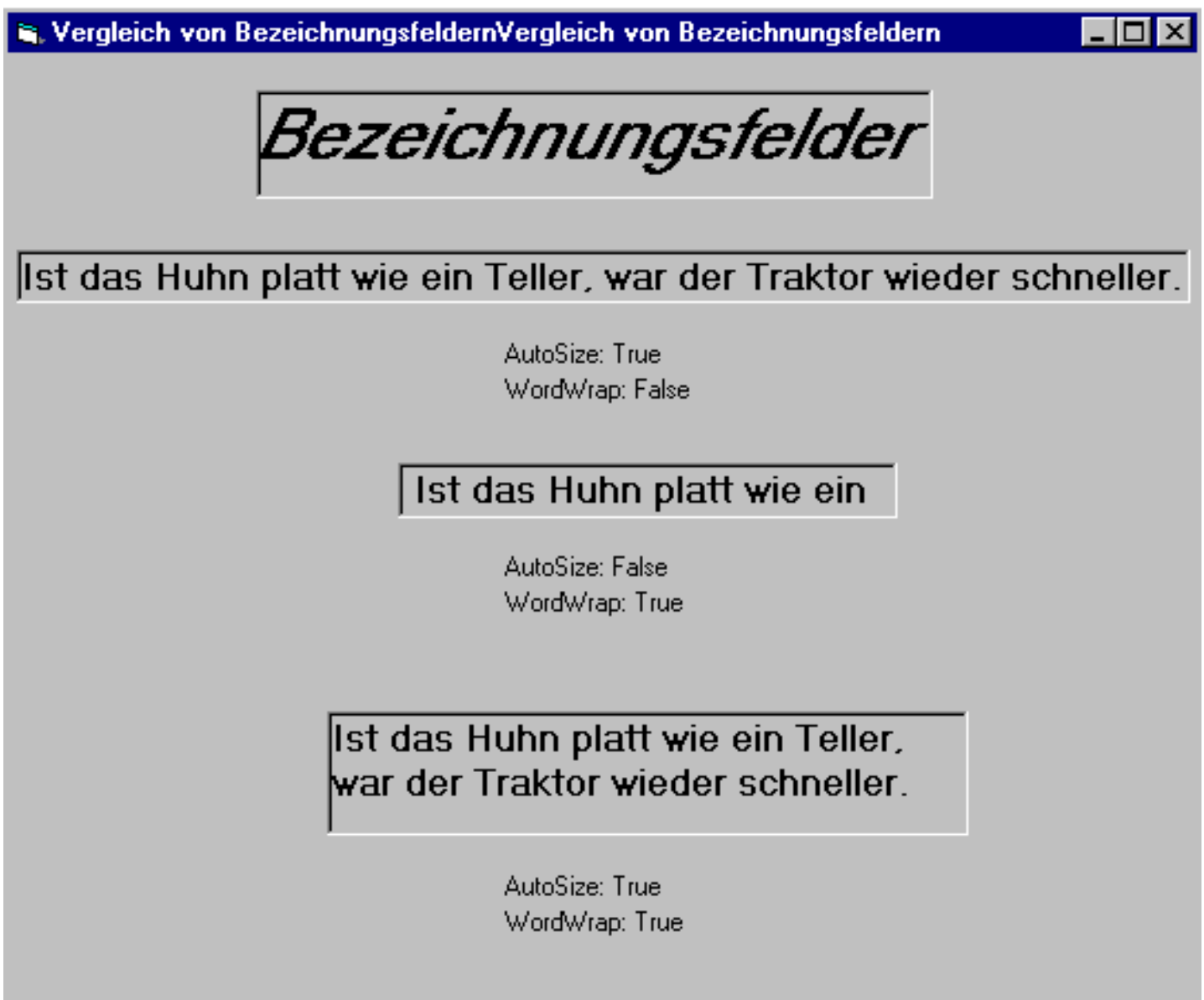


Abbildung 3.3:
Die Eigenschaften AutoSize und WordWrap bestimmen, wie ein Bezeichnungsfeld seinen Titel anzeigt.

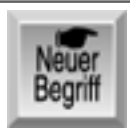
Das Textfeld (TextBox)

Wenn der Benutzer etwas eingeben soll, beispielsweise die Antwort auf eine Abfrage, in der Sie den Namen und die Adresse erfahren möchten, verwenden Sie am besten ein Textfeld. Häufig ist ein Vorgabewert für Ihre Benutzer sehr hilfreich, den Ihr Visual-Basic-Programm bereitstellen kann. Wenn Sie beispielsweise den Benutzer auffordern, ein Datum einzugeben, könnten Sie das aktuelle Datum im Textfeld vorgeben, so daß er das Datum einfach übernehmen kann.

Für Ja/Nein- oder Richtig/Falsch-Antworten sind Textfelder weniger geeignet. Mit anderen Worten, wenn Sie Ihren Benutzer auffordern möchten, zwischen zwei Werten zu wählen, oder auf eine Ja/Nein-Frage zu antworten, sind andere Steuerelemente besser geeignet, beispielsweise ein Optionsfeld oder ein Listenfeld. Diese Steuerelemente lernen Sie in Kapitel 5 kennen.

Was Sie tun sollten

Zeigen Sie vor dem Textfeld ein Bezeichnungsfeld an, das dem Benutzer mitteilt, was er eingeben soll. Beispielsweise könnten Sie vor einem Textfeld das Bezeichnungsfeld **Geben Sie Ihren Vornamen ein:** anzeigen. Damit wissen Ihre Benutzer, welche Informationen sie eingeben sollen.



Programmierer unterscheiden zwischen *Entwicklungszeit* und *Laufzeit*. Die Entwicklungszeit ist der Zeitraum, in der Sie eine Anwendung entwickeln und warten. Die Laufzeit bezieht sich auf den Zeitpunkt, zu dem der Benutzer eine Anwendung ausführt. Wenn Sie einen Wert in die `Text`-Eigenschaft eines Textfelds schreiben, wird dieser Wert zur Laufzeit als Vorgabe angezeigt. Zur Laufzeit kann der Benutzer den Wert im Textfeld ändern, indem er ihn überschreibt oder die Vorgabe abändert.

Für Textfelder sind die folgenden Eigenschaften wichtig:

- **Alignment.** Diese Eigenschaft legt fest, wie der Text in einem Textfeld erscheint, dessen `MultiLine`-Eigenschaft auf `True` gesetzt ist. Die `Alignment`-Eigenschaft hat keine Bedeutung für einzeilige Textfelder.
- **Locked.** Diese Eigenschaft legt fest, ob der Benutzer einen Wert eingeben oder den Standardwert des Textfelds ändern kann. Wenn diese Eigenschaft gleich `True` ist, kann der Benutzer den Wert des Textfelds nicht ändern, bis das Programm ihr zur Laufzeit den Wert `False` zuweist.
- **MaxLength.** Diese Eigenschaft legt fest, wie viele Zeichen das Textfeld maximal aufnimmt. Der Wert 0 zeigt an, daß der Benutzer beliebig viele Zeichen eingeben darf.
- **MultiLine.** Wenn diese Eigenschaft `True` ist, kann das Textfeld mehrere Textzeilen aufnehmen. Wenn der Benutzer mehr Text eingibt, als in eine einzige Zeile paßt, wird eine vertikale Bildlaufleiste angezeigt, es sei denn, Sie haben die Bildlaufleisten mit Hilfe der Eigenschaft `ScrollBars` deaktiviert. Die `Alignment`-Eigenschaft, deren Standardwert gleich 0 - `Links` ist, bestimmt, wie mehrfache Zeilen ausgerichtet werden.
- **PasswordChar.** Diese Eigenschaft legt fest, welches Zeichen (beispielsweise ein Stern) statt der Zeichen angezeigt wird, die der Benutzer in das Textfeld eingibt. Mit anderen Worten, wenn der Benutzer einen Geheimcode eingibt, werden während der Eingabe nur Sterne auf dem Bildschirm angezeigt, so daß ihm niemand über die Schulter sehen und den Code ablesen kann. Die Sterne werden nur auf dem Bildschirm angezeigt, das Textfeld enthält den Wert, den der Benutzer eingegeben hat.
- **ScrollBars.** Diese Eigenschaft bestimmt, ob und wie viele Bildlaufleisten im Textfeld angezeigt werden. Der Wert 0 - `Kein` gibt an, daß überhaupt keine Bildlaufleisten angezeigt werden sollen. 1 - `Horizontal` erlaubt nur horizontale Bildlaufleisten. 2 - `Vertikal` erlaubt nur vertikale Bildlaufleisten. 3 - `Beides` erlaubt sowohl horizontale als auch vertikale Bildlaufleisten.

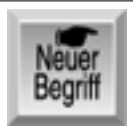


Damit vertikale Bildlaufleisten in einem Textfeld angezeigt werden können, müssen Sie die `MultiLine`-Eigenschaft auf `True` setzen.

- **Text.** Diese Eigenschaft gibt den Vorgabewert an, der im Textfeld erscheint.

Für Textfelder und Formulare gibt es einen speziellen Cursor (Einfügemarke). Ihr Benutzer gibt den Text mit Hilfe des *Textcursors* ein. Wenn Sie auf den Text in einem Textfeld klicken, wird der

Textcursor an der Stelle eingefügt, an der geklickt wurde. Damit kann der Benutzer innerhalb des Textfelds Text einfügen, ändern oder löschen.



Der *Textcursor*, auch als *Einfügemarke* bezeichnet, ist ein vertikaler Strich, mit dessen Hilfe Text in Steuerelemente wie Textfelder eingegeben und dort bearbeitet wird.



Laden und starten Sie die Beispielanwendung Controls aus Ihrem MSDN- Ordner Samples. Wählen Sie die Option **Textfeld**, um die Verwendung von Textfeldern zu üben. Abbildung 3.4 zeigt den Bildschirm, der erscheint. Sie können hier die Arbeit mit einzeiligen und mehrzeiligen Textfeldern sowie mit Textcursorn und Bildlaufleisten einüben. Dabei sehen Sie auch, daß Sie mit Hilfe der normalen Windows-Operationen Text innerhalb eines Textfelds kopieren und ausschneiden und in einem anderen Textfeld wieder einfügen können.

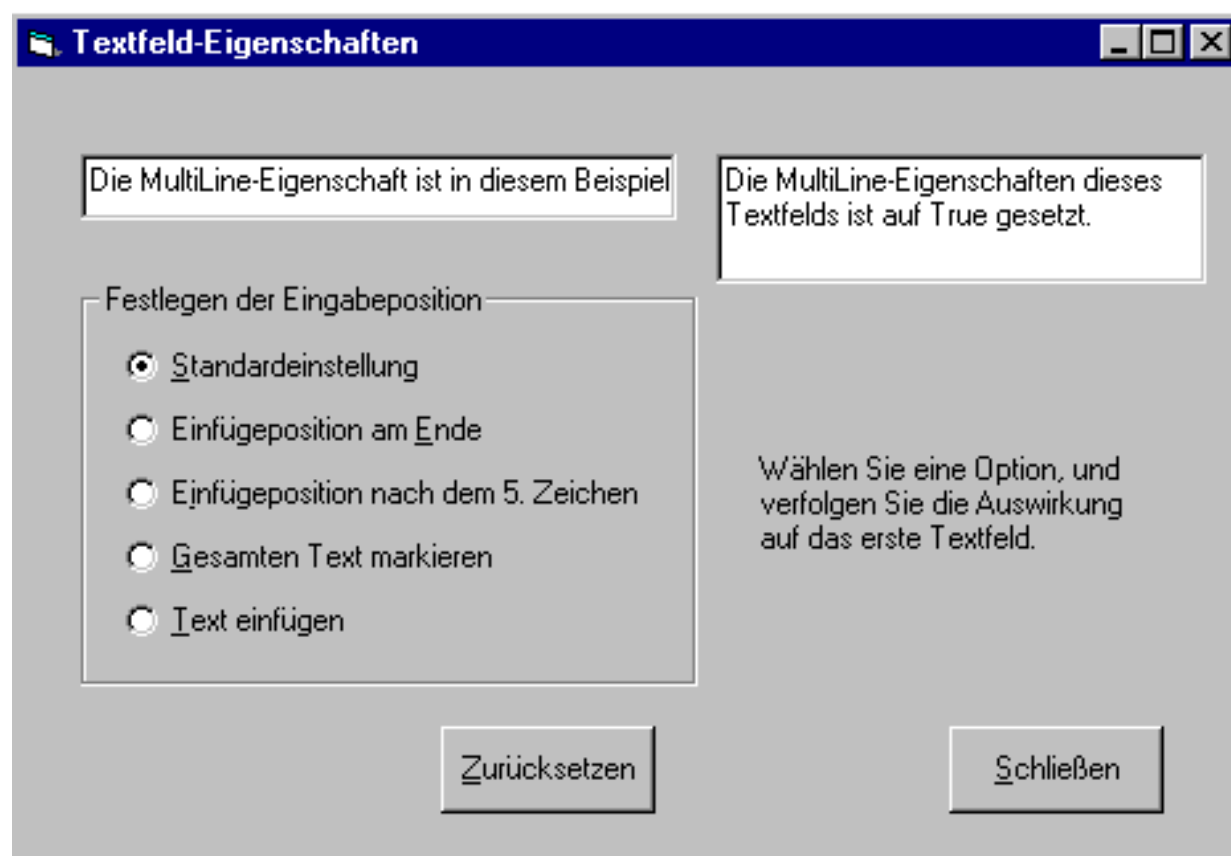


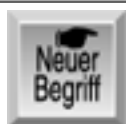
Abbildung 3.4:
In der Beispielanwendung Controls sehen Sie, daß Sie mit mehreren Textfeldstilen arbeiten können.

Die Befehlsschaltfläche (CommandButton)

Fast jedes Formular in einer Anwendung enthält eine Befehlsschaltfläche. Mit einem Klick auf die Befehlsschaltfläche kann der Benutzer anzeigen, daß er mit seiner Arbeit in dem Formular fertig ist, er kann bestätigen, daß er Papier in den Drucker eingelegt hat, oder daß er glaubt, es sei an der Zeit, das Programm zu beenden. Visual Basic stellt die Schaltfläche gedrückt dar, wenn der Benutzer sie anklickt.



Befehlsschaltflächen können nicht nur mit der Maus bedient werden! Sie können auch der Beschriftung einer Schaltfläche eine Beschleunigtaste hinzufügen. Der Benutzer aktiviert diese Schaltfläche, indem er die entsprechende Tastenkombination drückt, beispielsweise (Alt)+(R). Die `Caption`-Eigenschaft legt die Beschleunigtaste für die Schaltfläche fest. Außerdem kann der Benutzer eine Befehlsschaltfläche auch selektieren, indem er die Eingabetaste drückt, während die Schaltfläche den Fokus hat, oder wenn die `Default`-Eigenschaft gleich `True` ist. Manchmal möchten Sie, daß ein Benutzer in der Lage sein soll, eine Befehlsschaltfläche durch Drücken der (Esc)-Taste zu aktivieren, beispielsweise, wenn eine Befehlsschaltfläche das Beenden des Programms anbietet und die `Cancel`-Eigenschaft (Esc) verarbeitet.



Eine *Beschleunigtaste* (Shortcut) ist eine Taste, die in Kombination mit der (Alt)-Taste eine bestimmte Reaktion auslöst. Alle Einträge in der Menüleiste haben Beschleunigtasten, etwa (Alt)+(D) für den Menüeintrag **Datei**. Der unterstrichene Buchstabe in einer Menüauswahl oder auf der Beschriftung einer Schaltfläche kennzeichnet die Beschleunigtaste für dieses Objekt.

Die folgenden Eigenschaften sind für die Programmierung von Befehlsschaltflächen sehr praktisch:

- **Cancel.** Diese Eigenschaft legt fest, ob eine Befehlsschaltfläche auf die (Esc)-Taste reagiert. Ist sie `True`, kann der Benutzer die Befehlsschaltfläche auslösen (einen Klick simulieren), indem er (Esc) drückt. In jedem Formular kann jeweils nur eine Befehlsschaltfläche einen `Cancel`-Wert von `True` haben. Visual Basic setzt alle bis auf den letzten auf `False`.
- **Caption.** Diese Eigenschaft spezifiziert den Text, der auf der Befehlsschaltfläche erscheinen soll. Wenn Sie einem Buchstaben ein Ampersand-Zeichen (&) vorausstellen, macht Visual Basic diesen Buchstaben zur Beschleunigtaste (Shortcut). Der `Caption`-Wert `&Beenden` erzeugt eine Befehlsschaltfläche, die aussieht wie in Abbildung 3.5 gezeigt. Der Benutzer kann diese Befehlsschaltfläche auslösen, indem er darauf klickt oder (Alt)+(X) drückt.

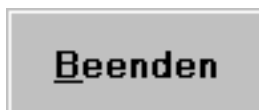


Abbildung 3.5:

Die `Caption`-Eigenschaft eines Steuerelements legt durch das Ampersand-Zeichen eine Beschleunigtaste fest.

- **Default.** Diese Eigenschaft legt fest, wie die Befehlsschaltfläche auf die Eingabetaste reagiert. Ist sie auf `True` gesetzt, kann der Benutzer die Befehlsschaltfläche auslösen (einen Klick simulieren), indem er die Eingabetaste drückt, es sei denn, er verschiebt zuvor den Fokus (siehe nächster Abschnitt) auf ein anderes Steuerelement. Nur jeweils eine Schaltfläche auf einem Formular kann den `Default`-Eigenschaftswert `True` haben. Visual Basic setzt alle anderen `Default`-Werte auf `False`. Die Befehlsschaltfläche mit dem `Default`-Wert `True` ist beim ersten Anzeigen des Formulars immer auf `True` gesetzt.
- **Picture.** Diese Eigenschaft bezeichnet ein grafisches Bild, das anstelle des Titels auf der Befehlsschaltfläche angezeigt wird. Die `Style`-Eigenschaft muß auf `1 - Grafisch` gesetzt sein, damit ein Bild angezeigt werden kann.
- **Style.** Diese Eigenschaft legt fest, ob eine Befehlsschaltfläche einen Texttitel (`0 - Standard`) oder ein Bild (`1 - Grafisch`) anzeigt.

Was Sie tun sollten	Was Sie lassen sollten
Verwenden Sie unterschiedliche Beschleunigertasten für Ihre Steuerelemente.	Verwenden Sie nicht dieselben Beschleunigertasten für mehrere Steuerelemente. Wenn zwei oder mehr Steuerelemente dieselben Beschleunigertasten verwendet, wählt Visual Basic nur das erste in der Fokus-Reihenfolge (siehe nächster Abschnitt) dafür aus und Ihr Benutzer kann die anderen Steuerelemente nicht mit dieser Beschleunigertaste auslösen.

Das Anzeigefeld (Image)

Das Anzeigefeld soll hier kurz erwähnt werden, weil Sie im gestrigen Programm ein Bild in einem Anzeigefeld platziert haben. Das Anzeigefeld ist eines von zwei Steuerelementen (neben dem Bildfeld), das grafische Bilder anzeigt. Die Bildeigenschaften legen fest, in welcher Datei das anzuzeigende Bild abgelegt ist, und ob das Bild so vergrößert oder verkleinert werden soll, daß es in das Steuerelement paßt bzw. es ausfüllt. Mehr über Bildfelder und Anzeigefelder erfahren Sie in Kapitel 14.

Es gibt viele andere Steuerelemente, die Sie ebenfalls kennenlernen sollten. Sie werden das in den nächsten Kapiteln tun. Im nächsten Abschnitt erfahren Sie, wie der Benutzer mit Hilfe der Tastatur alle Steuerelemente auf einem Formular auslösen kann.

Der Fokus

Jeweils nur ein Steuerelement auf einem Formular kann den *Fokus* besitzen. Welches Steuerelement als erstes den Fokus erhält, wird durch die Reihenfolge bestimmt, in der die Steuerelemente auf dem Formular angelegt wurden, oder durch die Reihenfolge, die durch die `TabIndex`-Eigenschaft der einzelnen Steuerelemente festgelegt ist. Nicht jedes Steuerelement kann den Fokus erhalten. Beispielsweise kann ein Bezeichnungsfeld den Fokus nicht erhalten, weil der Benutzer keine Veränderungen daran vornehmen kann.

	Der <i>Fokus</i> , auch als <i>Steuerelementfokus</i> bezeichnet, ist das aktuelle selektierte Steuerelement. Visual Basic stellt dieses Steuerelement hervorgehoben dar.

Betrachten Sie Abbildung 3.6. Sie erkennen die Schaltfläche mit dem Fokus sofort, weil sie durch einen gestrichelten Rahmen markiert ist. Visual Basic zeigt das Steuerelement mit dem Fokus normalerweise immer mit so einem gestrichelten Rahmen an. Hätte der Benutzer hier die Eingabetaste gedrückt, würde die mittlere Schaltfläche ausgelöst, weil sie den Fokus besitzt. Der Benutzer hätte jedoch auch (ÿ__) oder (^)+(ÿ__) drücken können, um den Fokus an ein anderes Steuerelement weiterzugeben. Hätte er die (ÿ__)-Taste gedrückt, hätte die dritte Befehlsschaltfläche den Fokus, so daß sie ausgelöst würde, hätte der Benutzer jetzt die Eingabetaste gedrückt.

	Beachten Sie, daß sich die Beschleunigungstasten für alle Steuerelemente unterscheiden müssen.



Wenn ein Steuerelement den Default-Wert True hat, hat dieses Steuerelement den Fokus, wenn der Benutzer das Formular lädt. Natürlich kann der Benutzer den Fokus an ein anderes Steuerelement weitergeben.



Die Markierung kennzeichnet den Fokus

Abbildung 3.6:
Das markierte Steuerelement hat den Fokus, aber der Benutzer kann diesen Fokus an ein anderes Steuerelement weitergeben, bevor er die Eingabetaste drückt.

Alle Steuerelemente haben die `TabIndex`-Eigenschaft. Wenn Sie Steuerelemente auf einem Formular platzieren, setzt Visual Basic automatisch die `TabIndex`-Eigenschaft auf 0, 1 usw., so daß eine eindeutige Reihenfolge für alle Steuerelemente entsteht. Auch Steuerelemente, mit denen der Benutzer normalerweise nicht arbeitet, wie beispielsweise Bezeichnungsfelder, haben die `TabIndex`-Eigenschaft. Die `TabIndex`-Eigenschaft legt die Fokusreihenfolge fest.

Sie platzieren nicht immer alle Steuerelemente sofort bei der Entwicklung in der gewünschten `TabIndex`-Reihenfolge auf dem Formular. Manchmal fügen Sie ein Steuerelement zwischen zwei anderen ein. Sie müssen also möglicherweise die `TabIndex`-Reihenfolge ändern, damit der Fokus so weitergegeben wird, wie Sie das möchten. Vielleicht möchten Sie den Fokus spaltenweise oder zeilenweise weitergeben, wenn der Benutzer die (ÿ__)-Taste drückt. Die Reihenfolge, in der der Fokus weitergegeben wird, wird ausschließlich durch die Werte in den `TabIndex`-Eigenschaften der Steuerelemente bestimmt.



Wenn das nächste Steuerelement, das den Fokus erhalten soll - wie in der `TabIndex`-Eigenschaft festgelegt - ein Bezeichnungsfeld ist, übergibt Visual Basic den Fokus dem nächsten Steuerelement in der Folge. Wenn Sie das wissen, können Sie dem Benutzer eine Beschleunigtaste für ein Textfeld zuweisen, was sonst nicht möglich ist. Betrachten Sie das Formular in Abbildung 3.7. Das Bezeichnungsfeld **Vorname:** hat einen `TabIndex`, der um eins größer als der des darauffolgenden Textfelds ist. Egal, welches Steuerelement den Fokus besitzt - wenn der Benutzer (Alt)+(F) drückt, sendet Visual Basic den Fokus an das Bezeichnungsfeld, das den Fokus unmittelbar an das Textfeld weitergibt, weil es selbst den Fokus normalerweise nicht annehmen kann. Wenn Sie Textfelder mit Bezeichnungsfeldern auf einem Formular verwenden, sollten Sie den Bezeichnungsfeldern Beschleunigtasten zuweisen, so daß Ihre Benutzer schnellen Zugriff auf alle Textfelder haben, wo sie irgendwelche Eingaben vornehmen möchten. Sie müssen dabei natürlich sicherstellen, daß die Bezeichnungsfeld/Textfeld-Paare aufeinanderfolgende `TabIndex`-Werte besitzen.

Datenformular für Kundenaufträge

Kundeninformation

Vorname

Nachname

Adresse

Stadt

Land

Postleitzahl

Textfeld, das den
Vornamen aufnimmt

Abbildung 3.7:
Der Benutzer drückt (Alt)+(V), um den Vornamen in das Textfeld einzugeben.

Ereignisprozeduren

Ereignisprozeduren erscheinen Neulingen der Visual-Basic-Programmierung manchmal kompliziert, aber das Konzept ist ganz einfach. Wenn der Benutzer auf eine Befehlsschaltfläche klickt oder Text in ein Textfeld eingibt, muß etwas passieren, das der Anwendung mitteilt, daß der Benutzer gerade irgend etwas getan. In Kapitel 2 haben Sie erfahren, daß Windows aus allen möglichen Quellen Ereignisse erhält. Die meisten Ereignisse für Windows-Anwendungen kommen direkt vom Benutzer, die dieser mit Hilfe der Maus oder der Tastatur auslöst.

Wenn Windows erkennt, daß der Benutzer ein Ereignis ausgelöst hat, und daß es sich dabei nicht um ein Systemereignis handelt, wie beispielsweise das Anklicken der Start- Schaltfläche von Windows, sondern um ein Ereignis, das von einer Anwendung benötigt wird, gibt es dieses Ereignis direkt an die Anwendung weiter. Wenn Sie eine Ereignisprozedur geschrieben haben, die auf das betreffende Ereignis reagiert, verarbeitet Ihre Anwendung das Ereignis. Wenn Sie keine Ereignisprozedur geschrieben haben, bleibt das Ereignis unverarbeitet.

Es gibt alle möglichen Ereignisse, beispielsweise Klick, Doppelklick oder Tastendruck. Darüber hinaus können mehrere Steuerelemente Ihres Formulars dieselben Ereignisse verarbeiten. Beispielsweise können sowohl eine Befehlsschaltfläche als auch ein Textfeld ein `Click`-Ereignis verarbeiten, weil der Benutzer jedes dieser Steuerelemente anklicken kann. Sie müssen für ein bestimmtes Ereignis also nicht nur eine Ereignisprozedur schreiben, sondern dabei auch spezifizieren, welchem Steuerelement dieses Ereignis zuzuordnen ist.



Das Konzept, Ereignisprozeduren für alle Ereignisse und für alle Steuerelemente zu schreiben, ist unabdingbar, wenn Sie Visual Basic verstehen möchten. Wie viele klick-bezogene Ereignisprozeduren müssen Sie schreiben, um drei Befehlsschaltflächen auf Ihrem Formular zu bedienen? Sie brauchen drei Ereignisprozeduren, weil der Benutzer jede dieser drei Befehlsschaltflächen anklicken kann. Wenn Sie eine **Click**-Prozedur schreiben, die keinem bestimmten Steuerelement zugeordnet ist, kann Ihr Programm nicht definiert auf die drei Schaltflächen reagieren. Sie müssen also für jede Befehlsschaltfläche eine separate **Click**-Prozedur schreiben. Wenn Windows Ihrem Programm das **Click**-Ereignis übergibt, übergibt es gleichzeitig das Steuerelement, das das Ereignis erzeugt hat. Nur wenn Sie eine Ereignisprozedur sowohl für das Steuerelement als auch für das Ereignis geschrieben haben, kann Ihre Anwendung auf das Ereignis reagieren.

Angenommen, Ihre Anwendung zeigt vier Befehlsschaltflächen auf dem Formular an. Dabei könnte folgendes passieren:

1. Wenn der Benutzer auf eine der Befehlsschaltflächen klickt, erkennt Windows, daß ein Ereignis aufgetreten ist.
2. Windows wertet das Ereignis aus und erkennt, daß es Ihrer Anwendung zuzuordnen ist.
3. Windows übergibt Ihrer Anwendung das Ereignis und überträgt ihr die Steuerung.
4. Wenn Ihre Anwendung eine Ereignisprozedur für das Steuerelement bereitstellt, das das Ereignis empfangen hat, wird der entsprechende Code (den Sie in das Codefenster eingegeben haben) ausgeführt.



Code in der Ereignisprozedur eines Steuerelements wird nie ausgeführt, wenn dieses Ereignis für das Steuerelement nie stattfindet. Visual Basic ist also ein sehr reaktives System, weil Sie alle möglichen Ereignisprozeduren schreiben können, die einfach nur da sind, bis genau das Ereignis auftritt, für das sie entwickelt wurden. Sie übernehmen nie die Kontrolle, wenn etwas anderes passiert.

Allgemeine Steuerelement-Ereignisse

Sie sollten sich mit den gebräuchlichsten Ereignissen vertraut machen, die für solche Steuerelemente auftreten, die Sie bereits kennen. Das Formular und alle seine Steuerelemente können Ereignisse verarbeiten. Hier einige gebräuchliche Formularereignisse, die während der Ausführung einer Anwendung auftreten können:

- **Activate.** Dieses Ereignis tritt auf, wenn ein Formular den Fokus erhält. Wenn eine Anwendung mehrere Formulare umfaßt, tritt das **Activate**-Ereignis auf, wenn der Benutzer auf ein anderes Formular wechselt, indem er dieses anklickt oder es in einem Menü auswählt.
- **Click.** Dieses Ereignis tritt auf, wenn der Benutzer irgendwo auf das Formular klickt. Wenn der Benutzer auf ein Formular klickt, das zum Teil verborgen ist, weil ein anderes Formular den Fokus hat, treten sowohl **Click** als auch **Activate** auf.



Das **Click**-Ereignis wird auch ausgelöst, wenn der Benutzer mit der rechten Maustaste über einem Formular klickt. Wenn Sie mehr über die Programmierung mit der Maus erfahren haben, lernen Sie auch, wie man erkennt, welche Maustaste das **Click**-Ereignis ausgelöst hat.

- **DblClick.** Dieses Ereignis tritt auf, wenn der Benutzer auf das Formular doppelklickt.
- **Deactivate.** Dieses Ereignis tritt auf, wenn ein anderes Formular den Fokus erhält. Wenn der Benutzer ein anderes Formular auswählt, treten also sowohl **Activate** als auch **Deactivate** auf. Sie können für beide Ereignisse jedes Formulars eine Ereignisprozedur schreiben, oder nur für jeweils ein Ereignis für jedes der Formulare, oder eine Kombination aus beidem, abhängig von den Anforderungen Ihrer Anwendung.
- **Initialize.** Dieses Ereignis tritt auf, wenn das Formular zum ersten Mal angezeigt wird.
- **Load.** Dieses Ereignis tritt auf, wenn das Formular in den aktiven Speicher geladen und auf dem Bildschirm angezeigt wird.
- **Paint.** Dieses Ereignis tritt auf, wenn Windows das Formular neu zeichnen muß, weil der Benutzer einen Teil davon unter einem anderen Objekt freigelegt hat, beispielsweise unter einem Icon.
- **Resize.** Dieses Ereignis tritt auf, wenn der Benutzer die Größe des Formulars ändert.
- **Unload.** Dieses Ereignis tritt auf, wenn die Anwendung im Code ein Formular aus dem Fenster entfernt. Wenn eine Anwendung beendet wird, werden alle geladenen Formulare zuvor entladen. Sie müssen also für jedes Formular eine **Unload**- Ereignisprozedur schreiben, wenn Sie am Ende jeder Sitzung einen sauberen Abschluß gewährleisten oder irgendwelche Dateien speichern möchten.

Die folgenden Ereignisse sind für Textfelder gebräuchlich:

- **Change.** Dieses Ereignis tritt auf, wenn der Benutzer den Text bearbeitet.
- **Click.** Dieses Ereignis tritt auf, wenn der Benutzer das Textfeld anklickt.
- **DblClick.** Dieses Ereignis tritt auf, wenn der Benutzer in das Textfeld doppelklickt.



Einige Ereignisse stehen für fast alle Steuerelemente zur Verfügung, beispielsweise Tastatur- und Mausereignisse. Sie werden später in diesem Buch beschrieben.

Die meisten dieser Textfeld-Ereignisse werden auch von Bezeichnungsfeldern unterstützt. Bezeichnungsfelder lösen die Ereignisse jedoch auf etwas andere Weise aus. Beispielsweise unterstützt ein Bezeichnungsfeld zwar das **Change**-Ereignis, aber der Benutzer kann es nicht direkt ändern. Im Code von Visual Basic kann ein Bezeichnungsfeld jedoch geändert werden, und genau wenn das passiert, findet das **Change**-Ereignis statt.

Das Anzeigefeld unterstützt einige der Ereignisse, die auch von Bezeichnungsfeldern unterstützt werden. Das Anzeigefeld ist einem Bezeichnungsfeld sehr ähnlich, außer daß es statt Text ein grafisches Bild anzeigt.



Es gibt sehr viel mehr Ereignisse, als Sie heute kennenlernen werden. Die hier beschriebenen Ereignisse brauchen Sie für Ihre ersten Schritte in Visual Basic.

Auch die Befehlsschaltflächen unterstützen die zuvor für Textfelder beschriebenen Ereignisse. Beachten Sie die folgenden Dinge, wenn Sie Ereignisse für Befehlsschaltflächen programmieren:

- Wenn sich nur eine Befehlsschaltfläche auf einem Formular befindet, löst das Drücken der Leertaste die Ereignisprozedur für die Befehlsschaltfläche aus, wenn diese den Fokus hat.
- Wenn die `Cancel`-Eigenschaft einer Befehlsschaltfläche `True` ist, wird durch Drücken von (Esc) das `Click`-Ereignis ausgelöst.
- Durch Drücken einer Beschleunigertastenkombination kann das `Click`-Ereignis einer Befehlsschaltfläche ausgelöst werden.



Nicht alle Ereignisse Ihrer Anwendung entstehen durch Aktionen des Benutzers. Auch der Visual-Basic-Code kann Ereignisse auslösen. Sie könnten beispielsweise Ihren Benutzer auffordern, eine Schaltfläche zu drücken, wenn er eine Gesamtsumme anzeigen möchte. Die Ereignisprozedur für das `Click`-Ereignis der Schaltfläche könnte beispielsweise die Gesamtsumme berechnen und sie ausdrucken. Nach einer bestimmten Zeit könnte auch Ihr Code dasselbe `Click`-Ereignis für die Schaltfläche auslösen. Die Gesamtsumme erscheint also irgendwann automatisch, auch ohne Zutun des Benutzers.

Ereignisprozeduren entwickeln

Ereignisprozeduren enthalten Visual-Basic-Code. Es handelt sich dabei um Codeabschnitte, die das Ereignis für ein bestimmtes Steuerelement verarbeiten. Einem Steuerelement können mehrere Ereignisprozeduren zugeordnet werden, wenn Sie auf verschiedene Ereignisse für dieses Steuerelement reagieren möchten.

Visual Basic verwendet den Namen einer Ereignisprozedur, um die beiden folgenden Informationen über die Prozedur auszusagen:

- Welches Steuerelement die Prozedur auslöst
- Welches Ereignis die Prozedur auslöst

Und hier das Format aller Ereignisprozedurnamen:

`SteuerelementName_EreignisName ( )`

Der Unterstrich trennt den Namen des Steuerelements vom Namen des Ereignisses und ist zwingend erforderlich. Alle Ereignisprozeduren werden auf diese Weise angesprochen. Die Ereignisprozedur `cmdExit_DbClick( )` beispielsweise wird nur dann ausgeführt, wenn das Ereignis `DbClick` für die Schaltfläche `cmdExit` auftritt.

Irgendwann werden Sie auch in die Klammern der Ereignisprozedur Werte eintragen - wenn Sie mehr über die Visual-Basic-Programmierung wissen. Die Klammern sind zwar hier leer, so wie in der Anwendung, die Sie gestern entwickelt haben, aber sie müssen angegeben werden. Die Klammern bieten eine Möglichkeit, zwischen Ereignisprozedurnamen und Steuerelementnamen zu unterscheiden, auch wenn sie nicht Teil des eigentlichen Namens sind.

Der Code innerhalb der Ereignisprozedur `cmdExit_DbClick( )` wird nur dann ausgeführt, wenn der Benutzer auf die Schaltfläche `cmdExit` doppelklickt. Wäre dies die einzige Ereignisprozedur in der Anwendung, würde sie alle anderen Ereignisse, die auftreten, ignorieren. Würde der Benutzer beispielsweise auf die Schaltfläche **Beenden** klicken, würde nichts passieren, weil ein Klick etwas

anderes als ein Doppelklick ist.

Fast alle Ereignisprozeduren, die Sie als Übung für dieses Buch schreiben werden, beginnen mit den Worten `Private Sub`. Das Schlüsselwort `Private` ist optional. Wenn Sie es nicht explizit angeben, setzt Visual Basic voraus, daß die Ereignisprozedur privat ist.

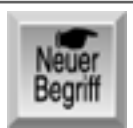
Visual Basic unterstützt zwei Prozedurtypen: *Funktionen* und *Sub-Prozeduren*. Alle Ereignisprozeduren sind Sub-Prozeduren. Der Rumpf einer Ereignisprozedur kann eine einzelne oder mehrere hundert Zeilen umfassen, aber man sollte sie so kurz wie möglich halten. Wenn Sie feststellen, daß Sie bei der Entwicklung eine extrem lange Prozedur schreiben, sollten Sie in Betracht ziehen, diese in mehrere kleinere Prozeduren zu zerlegen, die später leichter gewartet werden können. Listing 3.1 zeigt ein Beispiel dafür, wie `cmdExit_DblClick ( )` in einer Anwendung aussehen könnte.

Listing 3.1: Eine Ereignisprozedur, die ausgeführt wird, wenn der Benutzer auf die Befehlsschaltfläche klickt.

```
1: Private Sub cmdExit_DblClick ( )  
2: lblTitle.Caption = "New Page"  
3: intTotal = intCustNum + 1  
4: End Sub
```



Links im Code von Listing 3.1 sehen Sie Nummern, die in allen Listings dieses Buchs beibehalten werden. Diese Nummern sind nicht Teil des Programms. Sie dienen nur dazu, daß man die Zeilen leichter ansprechen kann, solange Sie hier Visual Basic lernen.



Eine *Funktion* sendet immer einen Wert, den sogenannten *Rückgabewert*, an irgendeine Stelle des Programms. Das Schlüsselwort `Function` weist darauf hin, daß eine Prozedur eine Funktion und keine Sub-Prozedur ist. Eine *Sub-Prozedur* wird durch das Schlüsselwort `Sub` gekennzeichnet. Sie sendet keine Werte, erledigt aber in ihrem Code bestimmte Arbeiten. Ereignisprozeduren sind immer Sub-Prozeduren. Funktionen werden für andere Aufgaben verwendet. Mehr über den Unterschied zwischen Funktionen und Sub-Prozeduren erfahren Sie in den weiteren Kapiteln dieses Buchs.

Die erste Zeile dieser Ereignisprozedur enthält viele Informationen. Hier erfahren Sie, daß die Prozedur privat ist (also nur im aktuellen Anwendungsmodul zur Verfügung steht). Außerdem wissen Sie, daß es sich bei der Ereignisprozedur um eine Sub-Prozedur handelt, also kein Wert zurückgegeben wird. Die Ereignisprozedur ist der Befehlsschaltfläche zugeordnet (zu erkennen an dem Präfix `cmd`), die der Entwickler als `cmdExit` angelegt hat. Sie wissen, daß die Ereignisprozedur nur auf Doppelklicks reagiert, die für diese Schaltfläche ausgeführt werden.

Der Rumpf der Ereignisprozedur umfaßt zwei Zeilen. Sie müssen den dort gezeigten Code jetzt noch nicht verstehen. Die letzte Zeile der Ereignisprozedur schließt die Prozedur ab, so daß Sie und Visual Basic ihr Ende erkennen. (Alle Funktionen enden, wie Sie vielleicht schon erraten haben, mit der Anweisung `End Function`.)

Der gesamte Code wird im Codefenster eingegeben. Das Codefenster verhält sich wie eine einfache Textverarbeitung. Wenn Sie eine Ereignisprozedur schreiben möchten, gelangen Sie auf unterschiedliche Weise in das Codefenster. Sie können das gewünschte Steuerelement, für das Sie eine Ereignisprozedur schreiben möchten, selektieren und in **Ansicht, Code** auswählen, oder Sie klicken im

Projektfenster auf die Schaltfläche **Code anzeigen**.

Auf einfachere Weise gelangen Sie in den Bearbeitungsbereich, indem Sie im Formularfenster einen Doppelklick auf das betreffende Steuerelement ausführen. Visual Basic öffnet automatisch das Codefenster für dieses Objekt, rät, welches Ereignis Sie bearbeiten möchten (das gebräuchlichste Ereignis für das Steuerelement) und schreibt die erste und letzte Zeile der Ereignisprozedur für Sie. Genau das haben Sie in der Anwendung in Kapitel 2 gemacht, wo Sie auf die Befehlsschaltfläche geklickt haben. Visual Basic wußte, daß das gebräuchlichste Ereignis für eine Befehlsschaltfläche **Click** ist, und hat das Codefenster mit den beiden folgenden Zeilen angezeigt:

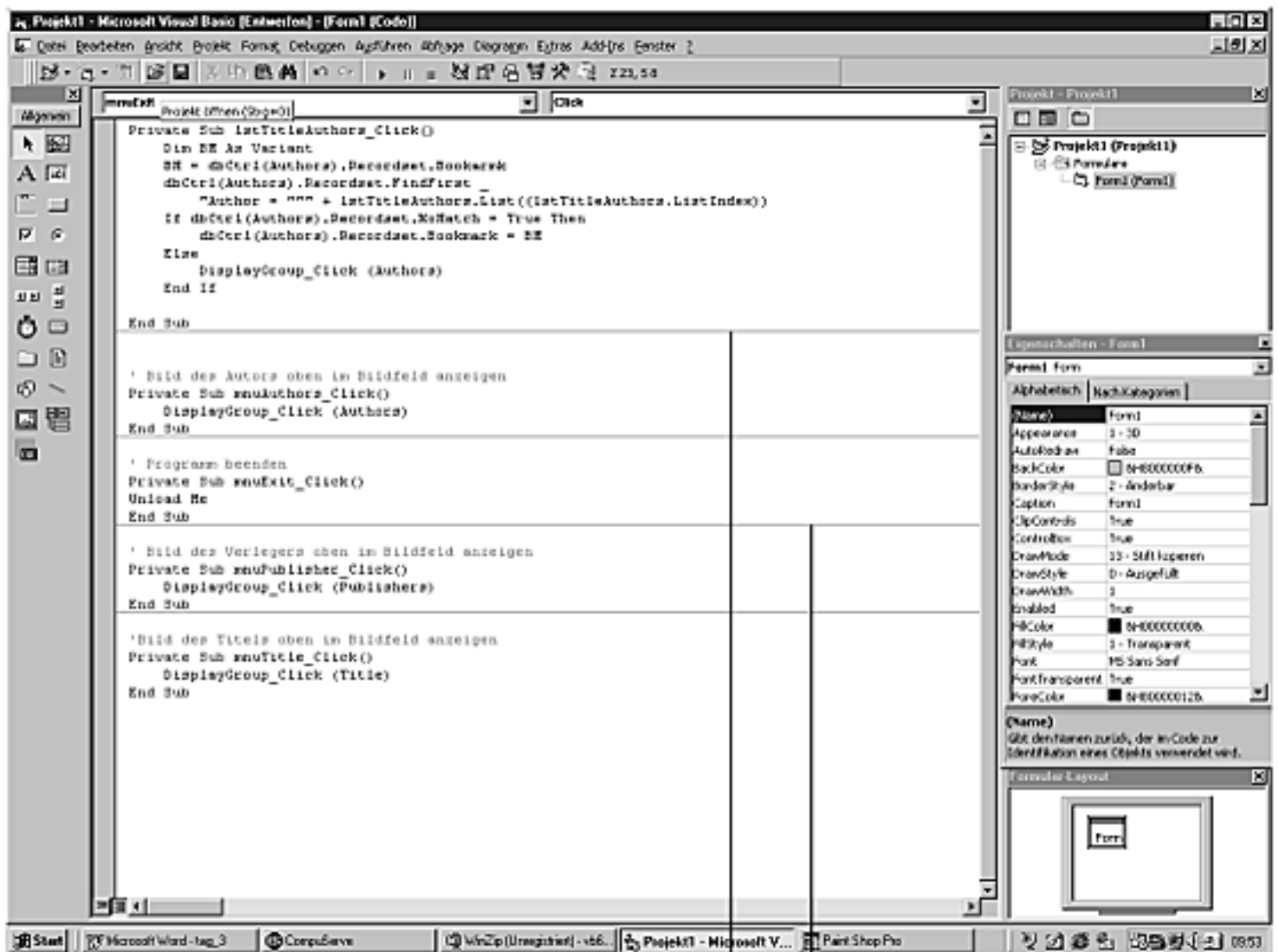
```
Private Sub cmdHappy_Click ()
```

```
End Sub
```

Visual Basic hat sogar den Textcursor zwischen den beiden Zeilen plazierte, so daß Sie sofort den Rumpf der Ereignisprozedur eingeben konnten. Nachdem Sie die Ereignisprozedur fertig haben, können Sie unmittelbar darunter die nächste schreiben (dann sind Sie aber selbst für die erste und die letzte Zeile verantwortlich). Durch Anklicken der Schaltfläche **Objekt anzeigen** im Projektfenster gelangen Sie in das Formularfenster zurück.



Wenn Visual Basic falsch geraten hat, und Sie eine Ereignisprozedur für ein anderes Ereignis schreiben möchten, ändern Sie den Ereignisnamen ab, beispielsweise in `cmdHappy_DblClick ()`.



Ereignisprozeduren werden durch Linien voneinander abgetrennt

Abbildung 3.8:
Das Codefenster verhält sich wie eine Textverarbeitung für Ihre Prozeduren.

Abbildung 3.8 zeigt ein Codefenster mit dem Code für mehrere Ereignisprozeduren. Das Codefenster teilt die einzelnen Prozeduren nicht nur für Sie ab, sondern unterstützt auch die gebräuchlichen Windows-Operationen wie Kopieren, Ausschneiden und Einfügen. Anders als eine Textverarbeitung umbricht das Codefenster die Zeilen nicht, weil sich jede Anweisung eines Visual-Basic-Programms in einer eigenen Zeile befinden muß. Sie können sehr lange Zeilen in der nächsten Zeile fortsetzen, indem Sie am Ende der ersten Zeile einen Unterstrich angeben, woran Visual Basic erkennt, daß die Anweisung in der nächsten Zeile fortgesetzt wird. Durch den Unterstrich behandelt Visual Basic die beiden Zeilen wie eine lange, fortgesetzte Zeile. Die zerlegten Zeilen sind einfacher zu lesen, weil man das Codefenster dann nicht nach links oder rechts blättern muß.

Bonus-Projekte

Heute haben Sie sehr viel Theorie kennengelernt. Damit sollten viele Fragen beantwortet sein, aber vielleicht wurden auch neue aufgeworfen. Um Ihnen zu helfen, alles einordnen zu können, erscheint zwischen diesem und dem nächsten Kapitel das Bonus-Projekt 1, »Steuerelemente, Eigenschaften und Ereignisse«. Dort werden Sie eine völlig neue Anwendung entwickeln, so daß Sie die Theorie aus diesem Kapitel in die Praxis umwandeln können. Sie erhalten nicht mehr so detaillierte Anweisungen wie für die Anwendung aus Kapitel 2, aber Sie brauchen das auch nicht mehr, weil Sie nun schon sehr viel mehr über Visual Basic wissen. In diesem Buch erscheinen immer wieder Bonus-Projekte zwischen den Kapiteln, um Ihr Wissen zu vertiefen. Betrachten Sie sie einfach als Hausaufgaben für das nächste Kapitel.

Zusammenfassung

Im heutigen Kapitel haben Sie sehr viel Theorie kennengelernt. Jetzt wissen Sie, was Steuerelemente, Eigenschaften und Ereignisse sind. Und auch über Prozeduren wissen Sie jetzt besser Bescheid, auch wenn Sie noch viel über Visual-Basic-Code lernen müssen. Für jedes Steuerelement und Ereignis, auf das Ihre Anwendung reagieren soll, muß es eine Ereignisprozedur geben. Ohne Ereignisprozedur wird ein Ereignis einfach ignoriert.

In Kapitel 4 erfahren Sie, wie Sie Ihrer Anwendung Menüs hinzufügen, so daß die Benutzer das Programm einfacher bedienen können.

Fragen & Antworten

Frage:

Warum soll man den Rumpf der Ereignisprozedur einrücken?

Antwort:

Sie sollten den Rumpf aller Prozeduren einrücken. Diese Einrückung ist nicht zwingend erforderlich, hilft aber, die Ereignisse in langen Codezeilen voneinander zu unterscheiden. Das Codefenster trennt die Prozeduren zwar schon durch Trennzeilen voneinander ab, aber wenn Sie ein Programmlisting ausdrucken, kann Ihnen die Einrückung helfen, sich auf einzelne Prozeduren zu konzentrieren.

Frage:

Kann ich neue Namen für Ereignisprozeduren einführen?

Antwort:

Die einzige Möglichkeit, den Namen einer Ereignisprozedur zu ändern, ist, den Namen des Steuerelements zu ändern, der die Prozedur auslöst. Durch das spezielle Namensformat der Ereignisprozeduren erkennt Visual Basic, welche Steuerelemente und welche Ereignisse die Ereignisprozedur auslösen sollen. Sie können die Namen der Sub-Prozeduren und Funktionen beliebig wählen, nicht jedoch die Ereignisprozedurnamen, die nach dem heute vorgestellten Modell aufgebaut sind.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist eine Beschleunigertaste?
2. Richtig/Falsch? Eigenschaften unterstützen mehrere Ereignisse.
3. Warum sollten Sie einer Befehlsschaltfläche das **Cancel**-Ereignis zuordnen?
4. Woher wissen Sie, welches Steuerelement den Fokus besitzt?
5. Wie kann der Benutzer den Fokus von einem Steuerelement zum nächsten weiterschalten?
6. Welche Eigenschaft bestimmt die Fokusreihenfolge?
7. Ist `LoadPicture ( )` eine Sub-Prozedur, eine Funktion oder eine Ereignisprozedur (raten Sie einfach)?
8. Richtig/Falsch? Visual Basic erzeugt automatisch die erste und die letzte Zeile für die **Click**-Ereignisprozedur, wenn Sie im Arbeitsbereich innerhalb des Formularfensters auf ein Steuerelement doppelklicken.
9. Richtig/Falsch? Sie können ein Benutzerereignis wie beispielsweise **DblClick** vom Visual-Basic-Code aus auslösen.
10. Welche Aufgabe hat die Eigenschaft `PasswordChar`?

Übungen

1. Schreiben Sie die erste Zeile für die **Load**-Ereignisprozedur eines Formulars. Der Name des Formulars ist `frmMyApp`.
 2. *Fehlersuche*: Warum kann der folgende Code keine Ereignisprozedur sein:

```
1: Private Function txtGetName_KeyDown ( )  
2: ' Bericht beginnen  
3: Call ReportPrint  
4: lblWarning.Caption = "Fertig...?"  
5: End Function
```
 3. Entwickeln Sie eine Anwendung mit drei mehrzeiligen Textfeldern. Machen Sie die Textfelder so groß, das sie drei oder vier Zeilen anzeigen können. Legen Sie für das erste Textfeld eine vertikale Bildlaufleiste, für das zweite eine horizontale Bildlaufleiste und für das dritte beide Bildlaufleisten an. Geben Sie in allen drei Textfeldern den Vorgabetext **Hier erfolgt die Eingabe** aus. Legen Sie eine Befehlsschaltfläche für das Beenden so an, daß der Benutzer das Programm mit (Alt)+(B) beenden kann.
-

Woche 1

Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse

Zwischen den Kapiteln dieses Buchs finden Sie immer wieder Bonus-Abschnitte, die Ihnen helfen, den bisher gelernten Stoff zu vertiefen. In diesen Bonus-Abschnitten entwickeln Sie vollständige Anwendungen. Sie sollten sich die Zeit nehmen, die Anwendungen wirklich nachzuvollziehen, weil Sie dadurch mit der Visual-Basic-Umgebung vertraut werden und schneller eigene Visual-Basic-Programme schreiben können.

Dieses erste Bonus-Projekt demonstriert die `PasswordChar`-Eigenschaft des Textfelds. Das Programm verwendet zur Anforderung eines Kennworts eine Textfeld-Eigenschaft. Nachdem der Benutzer das richtige Kennwort eingegeben hat, erscheint eine Grafik.

Experimentieren Sie ein bißchen mit den Bonus-Programmen herum. Wenn Sie mehr über Visual Basic gelernt haben, können Sie diese Programme abändern, um irgendwelche Dinge auszuprobieren und damit die Programmierung zu üben. Sie verstehen möglicherweise noch nicht den gesamten Code, den Sie in das Codefenster dieses Bonus-Projekts eingeben. Das ist aber nicht weiter schlimm - Sie werden ihn bald verstehen.

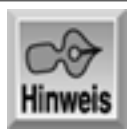


Das Kennwort für diese Anwendung lautet *Sams*. Pssst.... nicht weitersagen!

Die visuellen Elemente

Abbildung BP1.1 zeigt das Formularfenster für die neue Anwendung. Sie wissen bereits, wie man Steuerelemente auf einem Formular anlegt. Tabelle BP1.1 listet alle Steuerelemente und ihre Eigenschaftswerte auf, die Sie für die Anwendung brauchen. Gehen Sie zunächst vor wie gewohnt, um eine einfache Visual-Basic-Anwendung zu erstellen.

1. Wählen Sie **Datei, Neues Projekt**.
2. Wählen Sie das Icon **Standard-EXE**.
3. Setzen Sie die Eigenschaften des Formulars wie in Tabelle BP1.1 gezeigt.
4. Legen Sie alle in Tabelle BP1.1 beschriebenen Steuerelemente auf dem Formular an und setzen Sie ihre Eigenschaften. Behalten Sie für alle nicht in Tabelle BP1.1 aufgelisteten Eigenschaften die Standardwerte bei.



Sie müssen die Eigenschaften der Steuerelemente nicht sofort zuweisen, wenn Sie diese auf dem Formular angelegt haben. Sie können auch zunächst alle Steuerelemente plazieren und dann die Eigenschaften setzen.

Tabelle BP1.1: Legen Sie die Steuerelemente mit den hier beschriebenen Eigenschaften auf dem Formular an.

Name des Steuerelements	Eigenschaftswert
Formular Name	frmPassword
Formular Caption	Geben Sie ein Kennwort ein
Formular Height	5610
Formular Width	8475
Anzeigefeld Name	imgPassword
Anzeigefeld BorderStyle	1- Fest Einfach
Anzeigefeld Height	1890
Anzeigefeld Left	3000
Anzeigefeld Stretch	True
Anzeigefeld Top	2640
Anzeigefeld Width	2295
Bezeichnungsfeld Name	lblPrompt
Bezeichnungsfeld Alignment	2 - Zentriert
Bezeichnungsfeld BorderStyle	1- Fest Einfach
Bezeichnungsfeld Caption	Geben Sie im folgenden Textfeld das Kennwort ein.
Bezeichnungsfeld Font	MS Sans Serif
Bezeichnungsfeld Schriftgröße	14
Bezeichnungsfeld Schriftschnitt	Fett
Bezeichnungsfeld Height	855
Bezeichnungsfeld Left	2520
Bezeichnungsfeld Top	600
Bezeichnungsfeld Width	3375
Textfeld Name	txtPassword
Textfeld Height	375
Textfeld Left	3360
Textfeld PasswordChar	*

Textfeld Text	(Leer - der Vorgabewert wird gelöscht)
Textfeld Top	1800
Textfeld Width	1695
Befehlsschaltfläche Name	cmdTest
Befehlsschaltfläche Caption	Kennwort &testen
Befehlsschaltfläche Left	6360
Befehlsschaltfläche Top	3000
Befehlsschaltfläche #2 Name	cmdExit
Befehlsschaltfläche #2 Caption	&Beenden
Befehlsschaltfläche #2 Left	6360
Befehlsschaltfläche #2 Top	3720

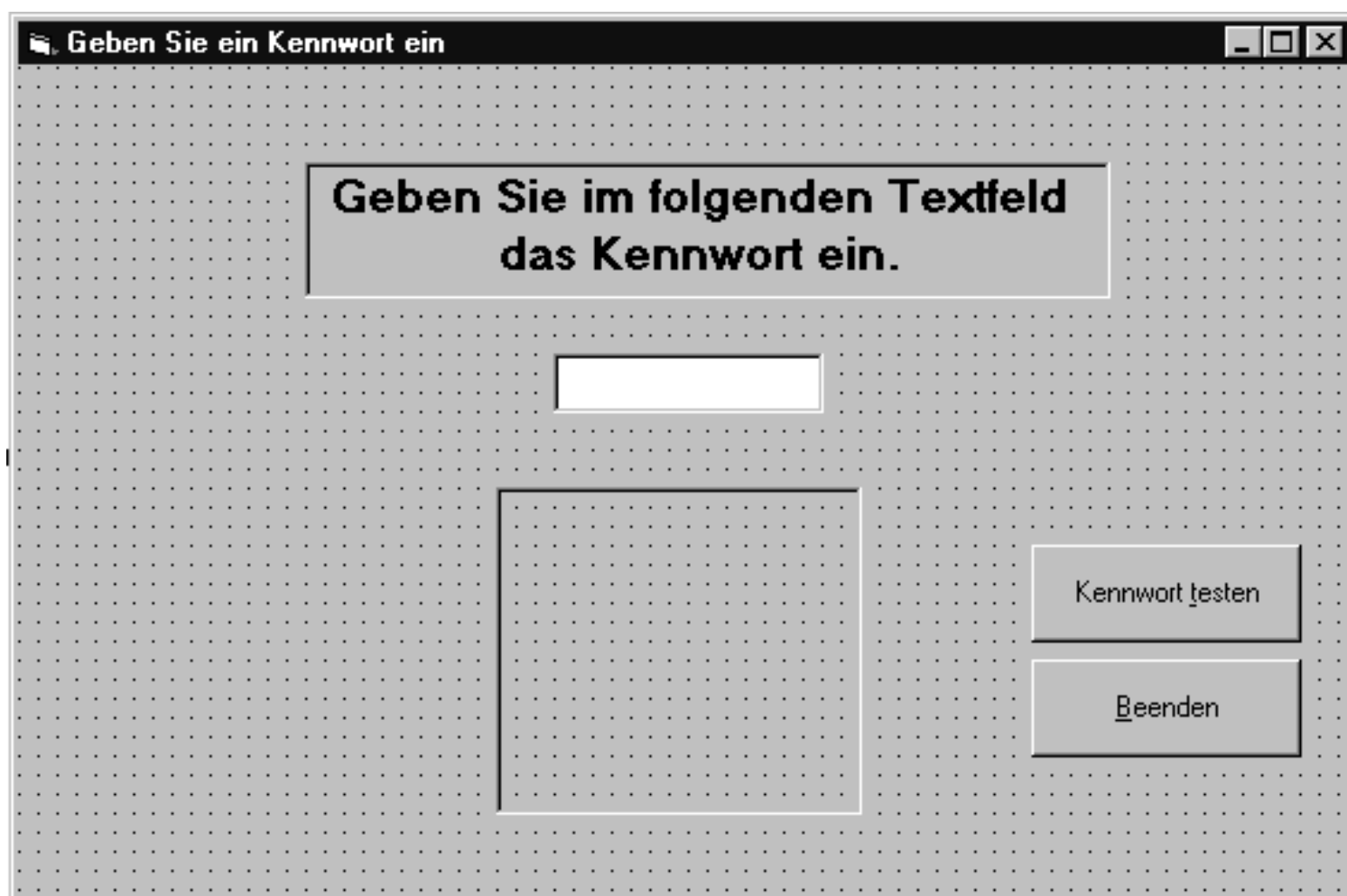


Abbildung BP1.1:
Diese Anwendung benutzt die PasswordChar-Eigenschaft des Textfelds.

Der Code

Nachdem Sie das Formularfenster angelegt haben, können Sie den Code eingeben. Listing BP1.1 zeigt diesen Code. Es gibt zwei Ereignisprozeduren: `cmdExit_Click()` und `cmdTest_Click()`. Jede dieser Ereignisprozeduren reagiert auf das `Click`-Ereignis der betreffenden Befehlsschaltfläche. Wenn der Benutzer auf die Befehlsschaltfläche `cmdExit` klickt, wird `cmdExit_Click()` ausgeführt. Wenn er auf die Befehlsschaltfläche `cmdTest` klickt, wird `cmdTest_Click()` ausgeführt.



Das `Click`-Ereignis ist das gebräuchlichste Ereignis für Befehlsschaltflächen. Die Ereignisprozedur für die beiden Befehlsschaltflächen legen Sie ganz einfach an, indem Sie es Visual Basic überlassen, die erste und letzte Zeile der Prozeduren zu erstellen. Um den einfachen, einzeiligen Rumpf für die `Click`-Ereignisprozedur der Steuerelement `cmdExit` einzugeben, doppelklicken Sie im Formularfenster auf die Schaltfläche `cmdExit` und füllen den Prozedurrumpf aus. Um den Rumpf für die `Click`-Ereignisprozedur der Schaltfläche `cmdTest` einzugeben, doppelklicken Sie auf die Schaltfläche `cmdTest` und füllen den Prozedurrumpf aus.

Listing BP1.1: Dieser Code aktiviert das kennwortorientierte Formular

```
1: Private Sub cmdExit_Click()  
2: End  
3: End Sub  
4:  
5: Private Sub cmdTest_Click()  
6: ' Diese Ereignisprozedur wird ausgeführt, wenn der  
7: ' Benutzer das von ihm eingegebene Kennwort überprüfen lassen will  
8: If txtPassword.Text = "Sams" Then  
9: ' Erfolg! Das Kennwort stimmt!  
10: Beep  
11: Beep ' Jetzt wird das Bild angezeigt  
12: imgPassword.Picture = LoadPicture("C:\Program Files\" _  
13: & "Microsoft Visual Studio\Common\Graphics\MetaFile\" _  
14: & "Business\coins.wmf")  
15: lblPrompt.Caption = "Zeig mir das Geld!"  
16: Else  
17: lblPrompt.Caption = "Falsches Kennwort - Noch ein Versuch!"  
18: txtPassword.Text = "" ' Altes Kennwort löschen  
19: txtPassword.SetFocus ' Fokus in das Textfeld  
20: End If  
21: End Sub
```

Analyse

Sie werden noch lernen, wie die Programmieranweisungen von Visual Basic auszuwerten sind - hier sollen nur einige Erklärungen zu diesem Code folgen, die als Vorbereitung für die Grundlagen zur Codierung betrachtet werden können, die Sie in Kapitel 5 kennenlernen werden.

Die Zeilen 1 bis 3 bilden die **Click**-Ereignisprozedur für die Schaltfläche `cmdExit`. Wenn der Benutzer auf die Schaltfläche `cmdExit` klickt, veranlaßt die **End**-Anweisung, daß die Anwendung beendet wird. Die Anwendung wird so lange auf dem Bildschirm angezeigt (auch wenn der Benutzer ein übereinstimmendes Kennwort eingegeben hat), bis der Benutzer auf die Schaltfläche `cmdExit` klickt oder das Programmfenster schließt.

Während der Codeeingabe sehen Sie, daß Visual Basic für bestimmten Text eine andere Farbe verwendet. Diese *Syntaxfarbgebung* ist ein praktisches Werkzeug, mit dem Sie Fehler bereits in der Programmierphase erkennen. Während Ihrer Arbeit werden Sie feststellen, daß Visual Basic bestimmten Texttypen bestimmte Farben zuweist. Ein Visual-Basic-Schlüsselwort, wie beispielsweise ein Befehl, wird immer blau dargestellt. Ein Objekt, wie beispielsweise ein Steuerelementname und seine Eigenschaft, wird immer schwarz dargestellt. Andere Codewörter sind grün. Wenn Sie sehen, daß ein Schlüsselwort grün angezeigt wird, erkennen Sie sofort, daß Sie etwas Falsches eingegeben haben und daß Sie den Schreibfehler beseitigen sollten, damit Visual Basic den Code richtig erkennt. Welche Farben dazu verwendet werden, können Sie über **Extras, Optionen, Editorformat** festlegen.



Die *Syntax* bestimmt die Grammatik- und Rechtschreibregeln einer Programmiersprache. Wenn Sie einen Befehl falsch schreiben oder ein Interpunktionszeichen vergessen, tritt ein Syntaxfehler auf.

Visual Basic erkennt die meisten Syntaxfehler, sobald Sie am Ende einer Zeile die Eingabetaste drücken. In Abbildung BP1.2 sehen Sie das Dialogfeld, das erscheint, wenn ein Befehl fehlerhaft eingegeben wurde. (Das Dialogfeld bezeichnet den Fehler nicht immer als Syntaxfehler; manchmal ist die Fehlermeldung auch viel spezifischer und weist auf einen falschen Befehl hin.) Wenn Sie das Problem erkennen (wie hier beispielsweise das doppelte Gleichheitszeichen), drücken Sie auf OK und korrigieren das Problem. Wenn Sie weitere Hilfe brauchen, klicken Sie auf die Hilfe-Schaltfläche. Visual Basic gibt Ihnen dann weitere Informationen über die mögliche Ursache des Problems.



Visual Basic kann die Problemursache nicht immer erkennen, wenn Sie etwas Falsches eingeben. Manchmal moniert Visual Basic nicht das eigentliche Problem (obwohl es ihm in Abbildung BP1.2 gelungen ist), weil es vielleicht nicht erkennt, daß ein Problem vorliegt, bis ein paar weitere Schlüsselwörter hinter dem wirklichen Problem ausgewertet wurden. Sie sollten also die gesamte Zeile lesen, um das Problem zu erkennen, falls Visual Basic es nicht richtig gekennzeichnet hat.

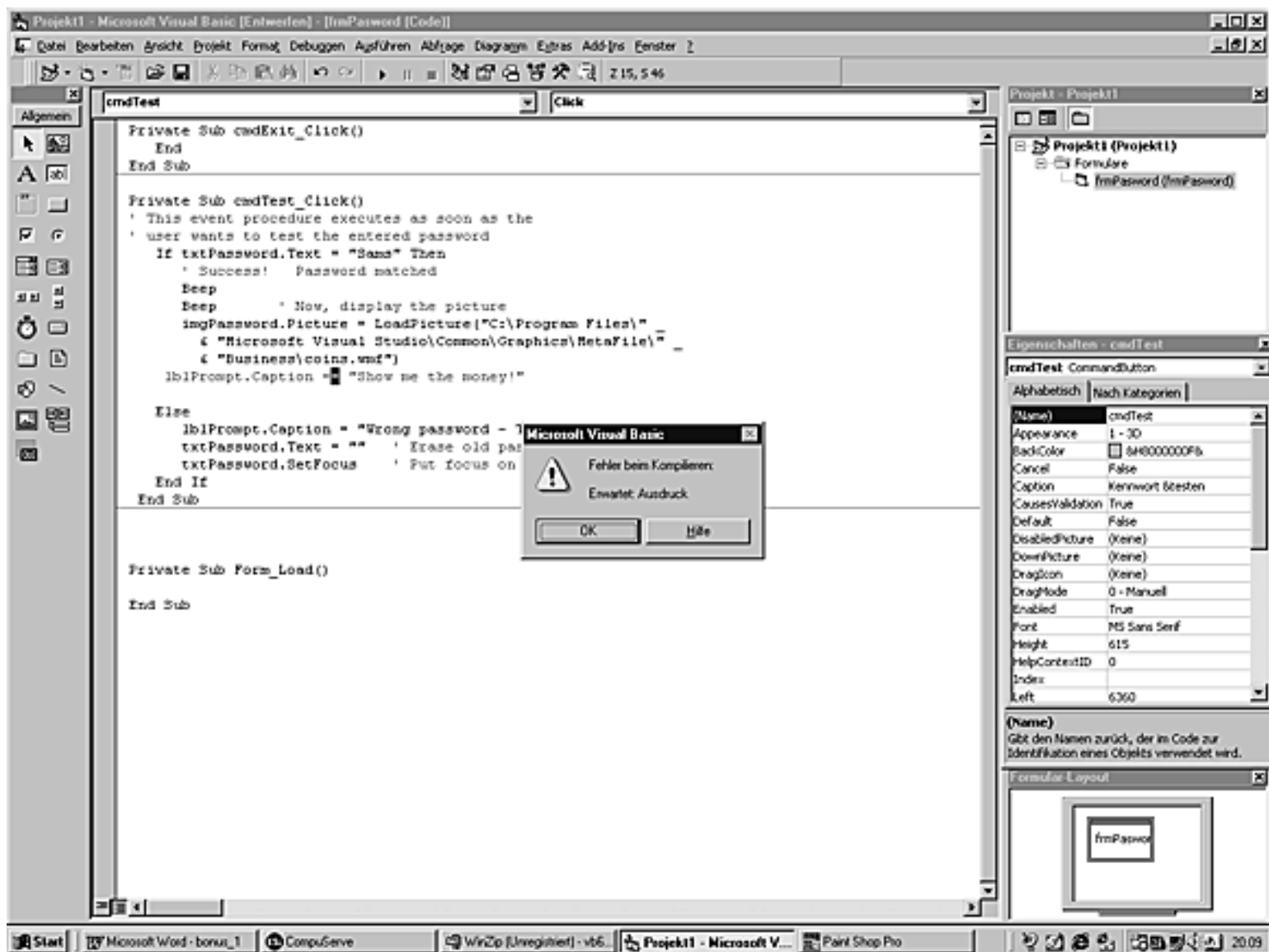


Abbildung BP1.2:
Visual Basic findet Syntaxfehler, sobald Sie die Eingabetaste gedrückt haben.

Die Zeilen 6 und 7 demonstrieren, wie Sie Ihren Visual-Basic-Code dokumentieren sollten. In Kapitel 1 haben Sie erfahren, daß die Dokumentation wichtig ist, weil Sie Ihren Code voraussichtlich längere Zeit warten müssen. Je mehr Beschreibungen Sie innerhalb des Codes unterbringen, desto schneller verstehen Sie später, was Sie sich bei der Entwicklung gedacht haben. Die Zeilen 6 und 7 sind Beispiele für *Kommentare*. Visual Basic ignoriert alle Zeilen, die mit einem einfachen Apostroph beginnen, der einen Kommentar kennzeichnet. Kommentare dienen nur den Menschen, nicht den Computern.

	Ein <i>Kommentar</i> ist eine Anmerkung in einem Visual-Basic-Programm, die den Code beschreibt. Manchmal fügt ein Programmierer oben in einem Programm einen Kommentar mit seinem Namen und seiner Telefonnummer ein. Auf diese Weise kann jeder, der den Code später bearbeitet und eine Frage hat, mit dem Autor Kontakt aufnehmen.

Was Sie tun sollten

Fügen Sie für alle Änderungen, die Sie vornehmen, oben im Code eine Zeile mit dem Datum und einer Beschreibung ein. Dieses Wartungsprotokoll hilft Ihnen (oder demjenigen, der das Programm irgendwann wartet) später, unmittelbar zu erkennen, welche Änderungen seit der ursprünglichen Programmentwicklung vorgenommen wurden.

Jetzt kennen Sie die Programmiersprache Visual Basic schon ein bißchen besser! Kommentare sind zwar für die Menschen vorgesehen, aber sie stellen gültige und wichtige Visual-Basic-Anweisungen dar. Sie sollten Ihr ganzes Programm mit Kommentaren versehen und dabei in einfachen Worten erklären, was das Programm macht. Wie Sie in Zeile 11 sehen, kann ein Kommentar auch rechts von einer Programmanweisung stehen, wenn es sich um einen kurzen Kommentar handelt. Versuchen Sie, Ihren gesamten Visual-Basic-Code und auch den Kommentar so in das Codefenster zu schreiben, daß keine horizontale Bildlaufleiste erforderlich ist. Wenn Sie den gesamten Code innerhalb eines Fensters unterbringen, werden die Bearbeitung und das Debugging einfacher.



Nutzen Sie die Werkzeuge, die Ihnen Visual Basic bietet, um den Code zu vervollständigen. Wenn Sie beispielsweise in Zeile 8 `txtPassword.Text` eingeben, zeigt Visual Basic alle Eigenschaften an, die für Textfelder zur Verfügung stehen, sobald Sie den Punkt gedrückt haben. Drücken Sie auf (T) und (E). Visual Basic geht zur Eigenschaft `Text`. Sie drücken einfach nur die Leertaste und schreiben weiter, ohne die beiden letzten Buchstaben schreiben zu müssen, weil Visual Basic das für Sie übernimmt. Damit ersparen Sie sich natürlich nur die Eingabe von zwei Zeichen, aber für andere Eigenschaften, wie beispielsweise `lblPrompt.Caption`, ist dieses Werkzeug wirklich praktisch.

Die Zeilen 12, 13 und 14 zeigen eine einzige Visual-Basic-Anweisung, die sich über drei Zeilen erstreckt. Die Anweisung ist so lang, weil für die Grafik so ein langer Pfadname erforderlich ist. Sie können die ganze Anweisung in einer Zeile eingeben, aber das würde die Breite des Codefensters bei weitem überschreiten. Visual Basic bietet eine Möglichkeit, eine logische Zeile in zwei oder mehr physische Zeilen zu unterteilen. Wenn Sie eine Zeile in eine weitere Zeile umbrechen möchten, drücken Sie die Leertaste und geben einen Unterstrich ein. Der Unterstrich als letztes Zeichen in einer Zeile zeigt Visual Basic, daß die nächste physische Zeile eine Fortsetzung der aktuellen Zeile ist.



Der Pfad, der in Zeile 12 beginnt, setzt voraus, daß Sie bei der Installation von Visual Basic die Beispielbilder installiert haben. Andernfalls funktioniert dieser Pfadname nicht. Suchen Sie mit der Windows-Menüoption **Suchen** auf Ihren Visual-Basic-CDs nach der Datei `Coins.wmf`. Um diese Grafik hinzuzufügen, legen Sie Ihre Visual-Basic-CD in das Laufwerk ein und wählen **Hinzufügen/Ändern**, um die Grafik auf Ihre Festplatte zu kopieren.

Ein spezielles Problem liegt vor, wenn Sie langen Text umbrechen möchten, der in Anführungszeichen eingeschlossen ist (wie hier gezeigt). Sie müssen die Anführungszeichen schließen, bevor Sie das Leerzeichen und den Unterstrich eingeben, und die nächste Zeile mit einem Ampersand-Zeichen (&) gefolgt von einem weiteren Anführungszeichen fortsetzen. In Kapitel 5 erfahren Sie mehr über Textstrings, so daß Sie besser verstehen, wozu das Ampersand-Zeichen gut ist.



Wenn Sie irgendwann mehr über die Programmiersprache Visual Basic wissen, verstehen Sie, warum einige der Anweisungen in Listing BP1.1 weiter eingerückt sind als andere.

Und hier noch ein letzter Gedanke für den Tag: Sie sind vielleicht überrascht, daß diese Anwendung die Befehlsschaltfläche **Kennwort testen** verwendet. Warum lassen Sie den Benutzer nicht einfach das Kennwort eingeben und dann die Eingabetaste drücken? In Kapitel 7 erfahren Sie, daß das Erkennen bestimmter Tastencodes, wie beispielsweise der Eingabetaste, zusätzlichen Code in diesem Programm erforderlich machen würde. Diese Anwendung sollte so einfach wie möglich gehalten werden, weil Sie noch nicht viel über Visual Basic wissen.

❖Kapitel Inhalt Index ❖ Top Kapitel❖

Woche 1

Tag 4

Menüs

Eine Menüleiste beinhaltet spezielle Steuerelemente, mit denen der Anwender Optionen auswählen und Befehle ausführen kann. Einige Menüoptionen erfüllen möglicherweise dieselben Aufgaben wie die Steuerelemente auf Ihrem Formular. Sie könnten dem Benutzer beispielsweise sowohl die Schaltfläche **Beenden** als auch die Menüoption **Datei, Beenden** bereitstellen, um die Anwendung zu beenden. Darüber hinaus können Menüoptionen dieselbe Funktion wie bestimmte Schaltflächen in der Symbolleiste haben. Andere Menüoptionen stellen dagegen vielleicht die einzige Methode dar, auf bestimmte Bereiche Ihres Programms zuzugreifen. In Visual Basic ist es ganz einfach, Menüs anzulegen.

Heute lernen Sie die folgenden Dinge kennen:

- Die vom Anwendungs-Assistenten erzeugten Menüs
- Menüoptionen
- Menüleisten, Dropdown-Menüs und Untermenüs
- Menü-Ereignisprozeduren
- Die Dropdown-Listenfelder **Objekt** und **Prozedur**

Menüs mit dem Anwendungs-Assistenten erstellen

Den Anwendungs-Assistenten haben Sie bereits in Kapitel 1 kennengelernt. Abbildung 4.1 zeigt den Bildschirm des Assistenten, wo Sie eine Menüleiste für Ihre Anwendung anlegen. Nachdem Sie alle Menüoptionen und Untermenüs angeklickt haben, die Sie in Ihrer Anwendung brauchen, erzeugt der Anwendungs-Assistent die entsprechenden Menüsteuerelemente und platziert sie in der erzeugten Anwendung.



Menüs sind sehr praktische Steuerelemente, weil Ihre Benutzer bereits wissen, wie man sie bedient. Benutzer verwenden am liebsten solche Menübefehle, die sie verstehen.

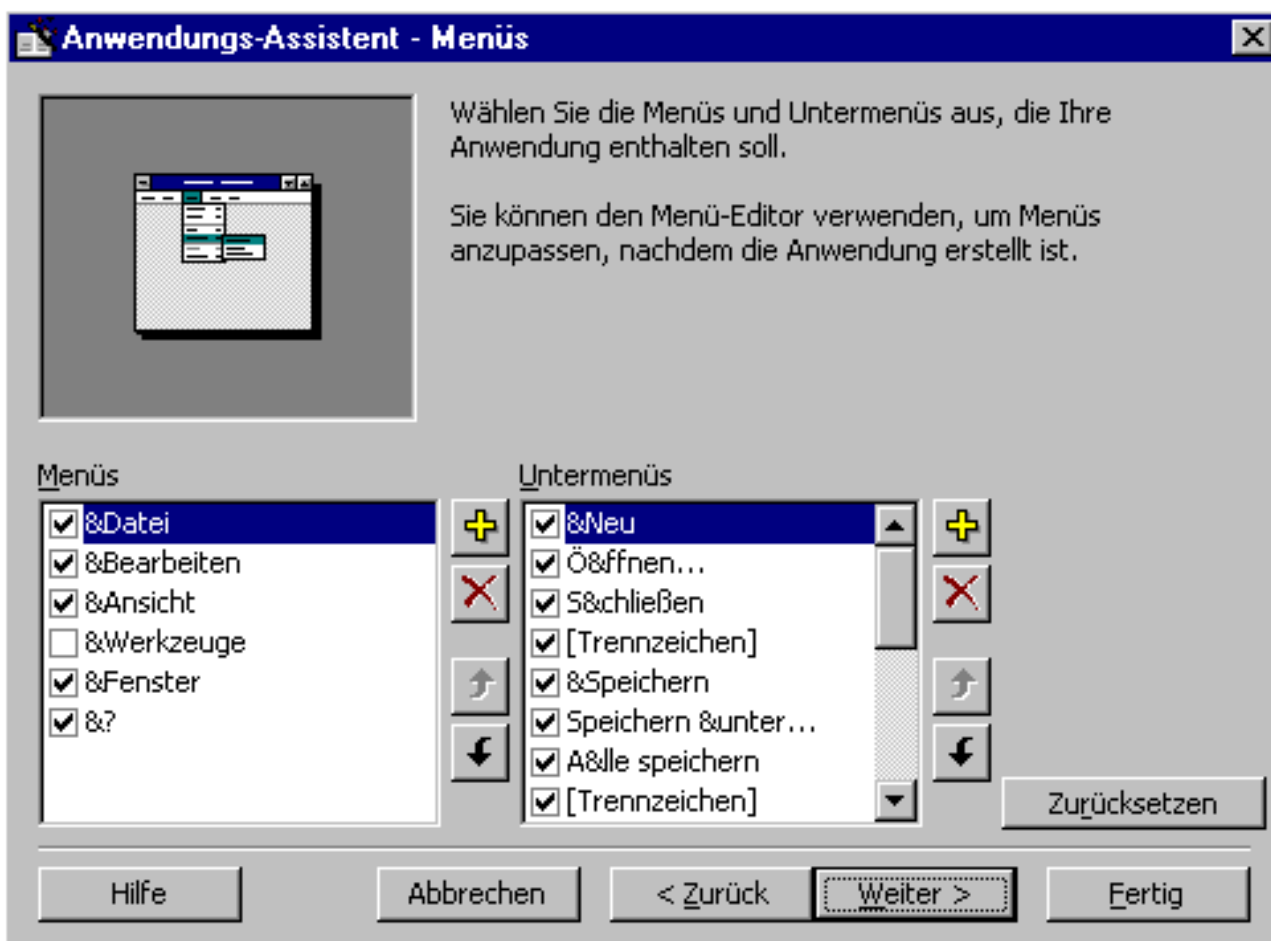


Abbildung 4.1:
Mit Hilfe des Anwendungs-Assistenten legen Sie Windows-Menüleisten und Optionen an.

Wenn Sie bereits viel mit Windows-Programmen gearbeitet haben, kennen Sie die meisten Menü-Optionen, die im Anwendungs-Assistenten angeboten werden. Menü-Optionen wie **Datei**, **Bearbeiten** oder **Fenster** erscheinen in den meisten Windows-Programmen, und auch die entsprechenden Untermenü-Optionen, die im Anwendungs-Assistenten angeboten werden, sind Ihnen vertraut, beispielsweise die Option **Beenden** im Menü **Datei**.

Es ist unwahrscheinlich, daß Sie je so viele Menüleistenoptionen und Untermenüs verwenden wie ein riesiges, kommerzielles Programm, wie beispielsweise Microsoft Word; nichtsdestotrotz stellen Word und die anderen Bestseller-Windows-Programme praktische Richtlinien für den Menüentwurf dar. Fast jedes Windows-Programm enthält die Option **Datei**, **Beenden**. Fast jede Windows-Anwendung enthält die Befehle **Bearbeiten**, **Ausschneiden** und **Bearbeiten**, **Kopieren**. Ihre Menüs werden nicht genau mit denen anderer Windows-Anwendungen übereinstimmen, weil Ihre Anwendungen andere Ziele haben. Wo immer es jedoch möglich ist, die allgemeinen Richtlinien zu Windows-Menüs zu befolgen, sollten Sie das tun.

Ihre Benutzer haben dann weniger Probleme, sich an Ihr Programm zu gewöhnen. Und je schneller Ihre Benutzer Ihr Programm kennenlernen, desto eher nutzen sie es und werden auch alle weiteren Versionen einsetzen.

Der Anwendungs-Assistent macht es so einfach, ein erstes Menü zu entwickeln, daß Sie wirklich in Betracht ziehen sollten, ihn zu verwenden, wenn Ihre Anwendung ein kompliziertes Menü erhalten soll. Im nächsten Abschnitt erfahren Sie, wie Sie mit den Werkzeugen von Visual Basic eigene Menüs anlegen, aber der Anwendungs-Assistent ist in der Bedienung noch einfacher. Sie brauchen eine Option nur anzuklicken, um sie in Ihre Anwendung zu übernehmen. Auch wenn Sie keine Symbolleiste anlegen möchten, oder eines der anderen Dinge, die der Assistent anbietet, können Sie

das Anwendungsgerüst dort anlegen, um ein vollständiges Menü zu erzeugen, bevor Sie die Details und die speziellen Menüoptionen einfügen.

Was Sie tun sollten

Lesen Sie in der MSND-Online-Hilfe nach, welche Richtlinien es für den Softwareentwurf gibt. Folgen Sie unbedingt den Windows-Standards für Menüs und andere Steuerelemente. Dadurch stellen Sie sicher, daß Ihre Benutzer von Anfang an mit Ihrem Programm vertraut sind.

Menüs - Grundlagen

Sie haben in den ersten Kapiteln dieses Buchs schon oft Windows-Menüs benutzt, weil Visual Basic eine Standard-Windows-Menüleiste einsetzt. Abbildung 4.2 zeigt das Visual-Basic-Menü mit aufgeklapptem **Datei**-Menü. Einige Komponenten sind beschriftet. Auch wenn Sie mit Windows-Programmen aufs beste vertraut sind, sollten Sie die Beschriftungen lesen, um die Visual-Basic-Namen für bestimmte Menüelemente kennenzulernen. Alle weiteren Abschnitte dieses Kapitels beschreiben, wie man diese Elemente in eigene Anwendungsmenüs aufnimmt.



Wenn eine Menüoption ein Dialogfeld aufruft, sollten Sie hinter dem Optionsnamen drei Punkte anzeigen, wie in Abbildung 4.2 gezeigt. Wenn Sie ein Menü mit Untermenü anlegen, fügt Visual Basic automatisch einen Pfeil nach rechts als Kennzeichner ein.

Einige Menüoptionen rufen weitere Menüs auf. Abbildung 4.3 zeigt das **Ansicht**-Menü von Visual Basic, wobei für den Menüpunkt **Symbolleisten** ein Untermenü aufgeklappt wurde. Beachten Sie die Markierung im Untermenü **Symbolleisten**. An dem Häkchen erkennen Sie, daß der Benutzer eine Option ausgewählt hat (»Chekked «). Durch Entfernen des Häkchens wird die Option deaktiviert. Mit dem Menü-Editor können Sie markierte (»checked«) und normale Menüoptionen anlegen.

Beschleunigtaste

Menüleiste

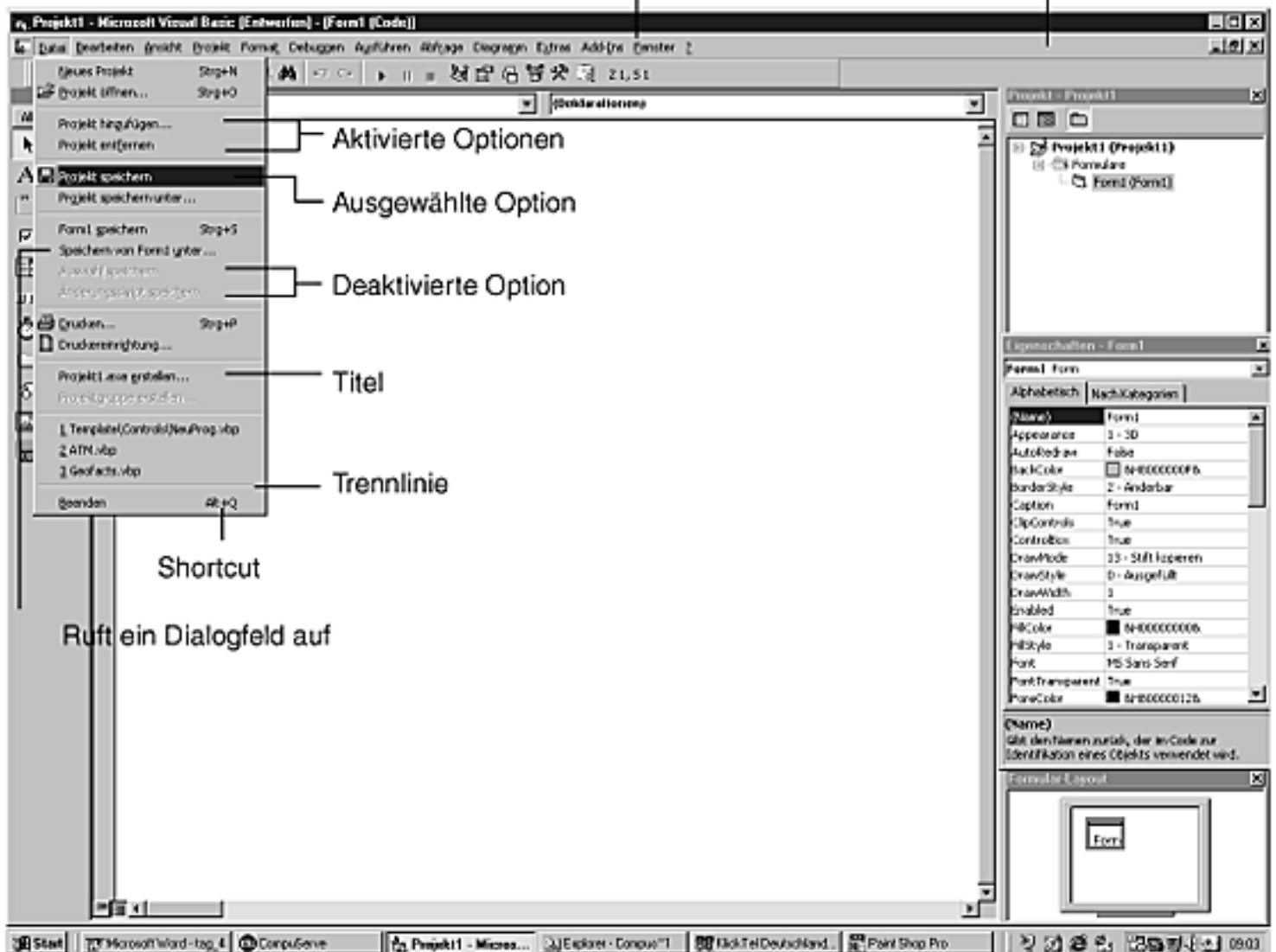


Abbildung 4.2:
Visual Basic legt für Sie ein Standard-Windows-Menü und seine Optionen an.

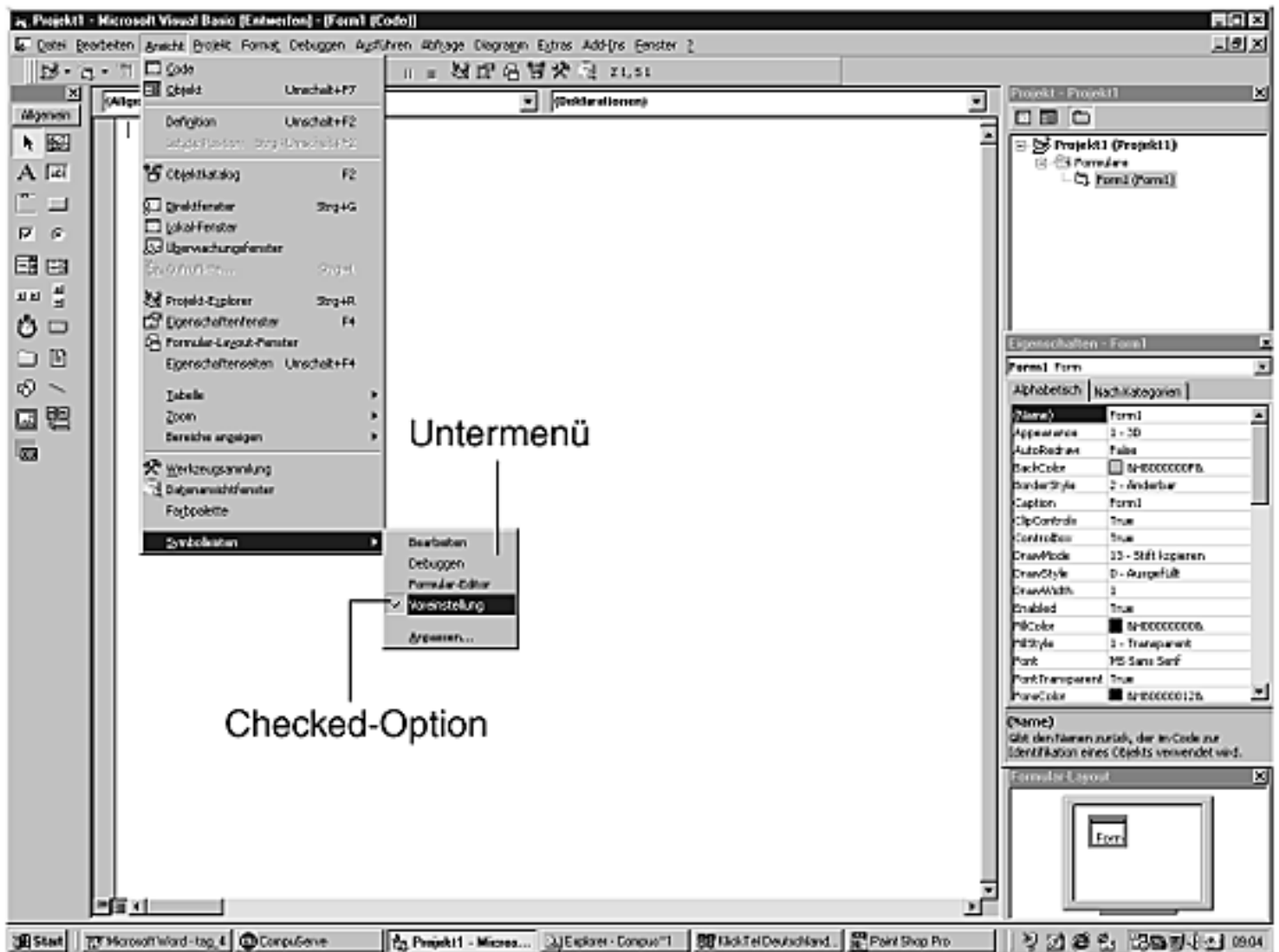
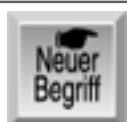


Abbildung 4.3:
Visual Basic legt für Sie Untermenüs an.

Der Menü-Editor

Die Symbolleiste enthält keine Werkzeuge zum Anlegen von Menüs. Statt dessen gibt es eine spezielle Menüauswahl, *Menü-Editor*, die Sie in Abbildung 4.4 sehen. Mit diesem Werkzeug legen Sie Ihre Menüs an. Im Formularfenster drücken Sie (Strg)+(E) (die Beschleunigtaste für **Extras**, **Menü-Editor**), um den Menü-Editor anzuzeigen.



Mit Hilfe des *Menü-Editors* legen Sie Menüs für Ihre Anwendungen an. Der Menü-Editor verhält sich in gewisser Weise wie ein Eigenschaftsfenster für die Menüleiste, weil Sie hier die Namen der Menüsteuerelemente sowie die Titel, die dem Benutzer angezeigt werden, festlegen. Der Menü-Editor wird schon seit mehreren Jahren in den Programmier-Produkten von Microsoft eingesetzt und wurde dabei nur unwesentlich verändert. Seine Langlebigkeit ist darauf zurückzuführen, daß er so leistungsfähig und einfach ist.

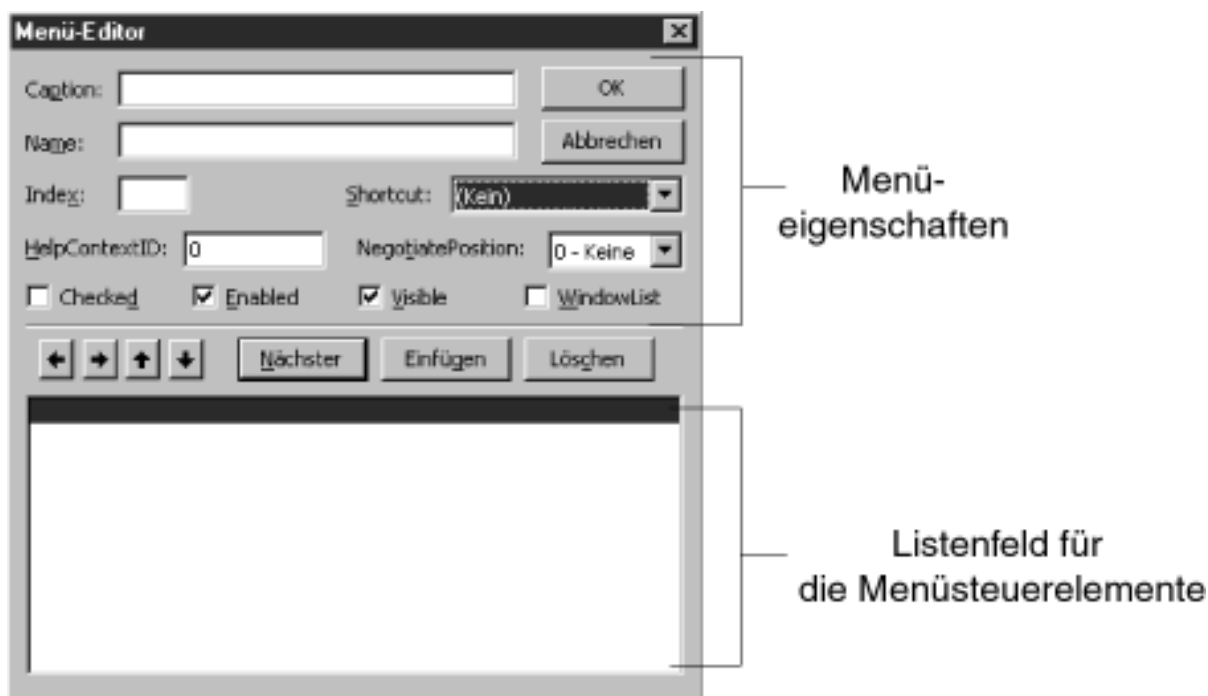


Abbildung 4.4:
Im Menü-Editor legen Sie Menüs und Untermenüs an.

In der oberen Hälfte des Menü-Editors, dem Abschnitt für die Menüeigenschaften, geben Sie die Steuerelementeigenschaften für eine Menüoption an. Diese Option erscheint in der Menüleiste oder in einem Untermenü, das aus der Menüleiste der Anwendung aufgeklappt wird. Das Listenfeld für die Menüsteuerelemente zeigt ein Baumdiagramm für das von Ihnen entwickelte Menü an.

Die Arbeit mit dem Menü-Editor

Die vielleicht beste Methode, die Arbeit mit dem Menü-Editor kennenzulernen, ist, eine einfache Anwendung mit Menüs zu erstellen. Sie müssen die dateibezogenen Operationen sowie andere komplexe Visual-Basic-Konzepte erst noch kennenlernen, deshalb ist das hier vorgestellte Beispielenü kaum mit den Standardmenüs unter Windows zu vergleichen. Nichtsdestotrotz macht diese Anwendung Sie mit der Methode vertraut, Menüs aufzubauen, und Sie erfahren, wie einfach die Bedienung des Menü-Editors ist.

Die Anwendung hat die folgenden Aufgaben:

- Anzeige eines Bezeichnungsfeldes in der Formlarmitte
- Bereitstellung einer Menüoption, mit der der Benutzer die Hintergrundfarbe des Bezeichnungsfeldes ändert
- Bereitstellung einer Menüoption, mit der der Benutzer den Text des Bezeichnungsfeldes ändert
- Bereitstellung der Menüoption **Beenden**, mit der der Benutzer das Programm beendet.

Wie Sie sehen, ist das Programm sehr einfach, aber Sie werden anhand des Beispiels lernen, wie man Menüs anlegt.

Starten Sie eine neue **Standard-EXE**-Anwendung im Menü **Datei, Neues Projekt**. Weisen Sie dem Formular den Namen `frmMenu` zu und setzen Sie den Titel in der `Caption` -Eigenschaft auf **Menü-Anwendung**. Richten Sie das Formular mit einer Höhe von 6030 (`Height`) und einer Breite von 8415 (`Width`) ein.

Fügen Sie dem Formular ein Bezeichnungsfeld mit den folgenden Eigenschaften hinzu:

- Name: lblMenu
- Alignment: 2 - Zentriert
- BackColor: Rot (klicken Sie in der Eigenschaft BackColor auf die Registerkarte **Palette** und wählen Sie das erste Rot aus)
- BorderStyle: 1 - Fest Einfach
- Caption: Wählen Sie eine Menüoption
- Schriftgröße: 24
- Schriftschnitt: Bold
- Height: 615
- Left: 1800
- Top: 2160
- Width: 4935



Ein *hexadezimalen Zahlensystem*, auch als System mit der Basis 16 bezeichnet, ist ein Zählsystem, das auf der Zahl 16 basiert. Für das normale Basis-10-Zahlensystem gibt es 10 eindeutige Ziffern, 0 bis 9. Im Basis-16- Zahlensystem gibt es 16 eindeutige Ziffern, 0 bis 9 und die Buchstaben A, B, C, D, E und F, die die restlichen »Ziffern« darstellen. Hexadezimale Zahlen werden immer durch ein vorangestelltes &H gekennzeichnet, so daß Visual Basic und der Programmierer erkennen, daß es sich um eine hexadezimale und nicht um eine dezimale Zahl handelt. Sie können einen numerischen Wert mit der Basis 10 oder mit der Basis 16 darstellen, aber für sehr große Werte stellt die Basis 16 ein kompakteres Format dar. Mit über einer Million möglicher Farbkombinationen kommt das Basis-16-Zahlensystem mit sehr viel weniger Ziffern für die Darstellung der einzelnen Farbschattierungen aus.



Ihr PC kann zwischen einigen hundert und mehr als einer Million Farben anzeigen. Jeder Farbe ist ein eindeutiger Codewert zugeordnet. Visual Basic verwendet für die Farbwerte das *hexadezimale Zahlensystem*. Die Auswahl von Rot in der Palette ist jedoch viel einfacher, als den genauen hexadezimalen Wert für die Farbe einzugeben, beispielsweise &H000000FF&.

Ihr Bildschirm sollte jetzt aussehen wie in Abbildung 4.5 gezeigt.

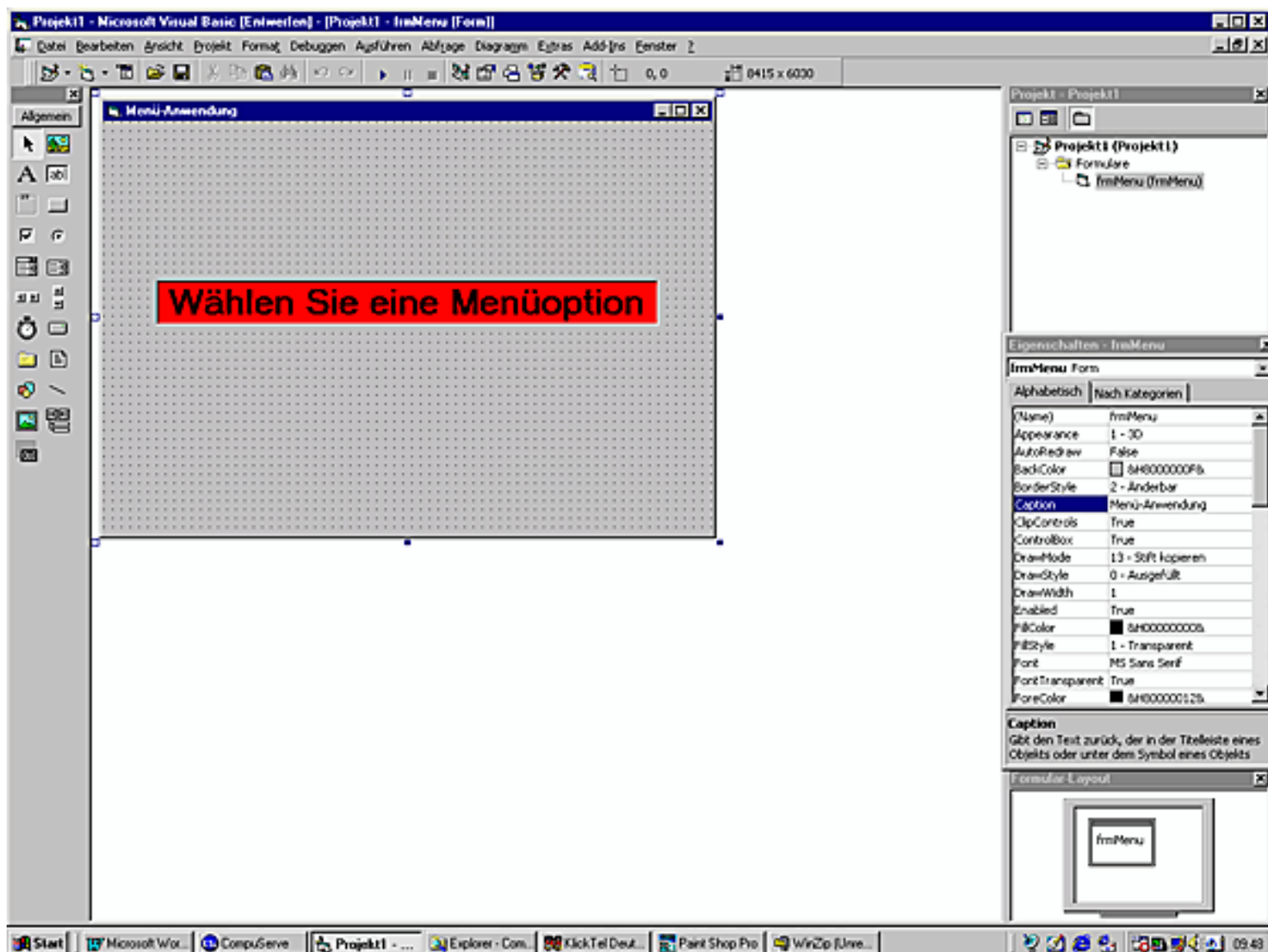


Abbildung 4.5:
Dieses Bezeichnungsfeld wird über Menüoptionen gesteuert.

Jetzt entwickeln Sie das Menü für Ihre Anwendung. Klicken Sie auf das Formular und drücken Sie (Strg)+(E), um den Menü-Editor anzuzeigen. Die meisten *Felder* des Menü-Editors sind leer, weil Sie bei der Einrichtung von Menüoptionen hier selbst Werte eingeben müssen.


Neuer Begriff

Ein *Feld*, häufig als Textfeld bezeichnet, ist ein generischer Name für eine Position, an der der Benutzer oder der Programmierer Werte in eine Anwendung eingeben. Der Menü-Editor enthält viele Felder, so wie die meisten Dialogfelder, in die Sie Werte eingeben.

In der Menüleiste, die Sie hier entwickeln werden, sollen die folgenden Einträge erscheinen:

- Datei
- Farbe
- Textausgabe

Beachten Sie, daß es für alle Optionen Beschleunigertasten (Shortcuts) geben soll, so daß der Benutzer die Optionen auch mit Hilfe der Tastatur auswählen kann. Wenn Sie dem Menü-Editor Elemente hinzufügen, erscheinen diese in der Menüleiste der Anwendung oder in einem Pulldown-Menü, abhängig davon, wie Sie die Werte eingeben. Gehen Sie wie folgt vor, um die Menüoption **Datei** in die

Menüleiste Ihrer Anwendung einzufügen:

1. Geben Sie in das Feld **Caption** &Datei ein. Wie überall in Visual Basic zeigt das Ampersand-Zeichen an, daß das (D) der Shortcut für die Menüauswahl sein wird. Während Sie den Titel eingeben, zeigt Visual Basic ihn sofort im Listenfeld für die Menüsteuerelemente an, das sich in der unteren Hälfte des Menü-Editors befindet.
2. Drücken Sie die (Tab)-Taste, um in das **Name**-Feld zu gelangen. Mit (Tab) und (Shift)+(Tab) bewegen Sie sich zwischen den Feldern des Menü-Editors.
3. Geben Sie für die erste Menüoption den Namen **mnuFile** ein.



Beachten Sie, daß Menüoptionen spezielle Steuerelemente sind, die wie alle anderen Steuerelemente Namen haben.

Was Sie tun sollten

Verwenden Sie für Menüoptionen eine konsistente Namenskonvention. Beginnen Sie alle Menünamen mit dem Präfix **mnu**, gefolgt von dem Namen. Wenn Sie Untermenüoptionen anlegen, verwenden Sie den Namen des übergeordneten Menüpunkts als Präfix, z.B. **mnuFileExit** für die Option **Datei, Beenden**.

4. Lassen Sie alle anderen Felder unverändert und klicken Sie auf die Schaltfläche **Nächster**, um weitere Menüoptionen einzugeben. Die Schaltfläche **Nächster** teilt dem Menü-Editor mit, daß Sie mit der ersten Option fertig sind und eine weitere Option eingeben möchten.
5. Geben Sie in das Feld **Caption** den Titel &Farbe ein, in das Feld **Name** **mnuColor**.
6. Klicken Sie auf die Schaltfläche **Nächster**.
7. Geben Sie in das Feld **Caption** für die letzte Menüoption den Titel &Textausgabe ein, in das Feld **Name** **mnuMessage**. Ihr Menü-Editor sollte jetzt aussehen wie in Abbildung 4.6 gezeigt.

Die drei Menüoptionen
für die Menüleiste

Abbildung 4.6:
Die ersten drei Optionen der
Menüleiste wurden angelegt.

Testen Sie Ihr Menü

Sie können Ihr Menü zu jedem Zeitpunkt der Entwicklung testen. Klicken Sie auf die **OK**-Schaltfläche, um den Menü-Editor zu schließen. Das Formular zeigt die neue Menüleiste an. Drücken Sie (F5), um Ihre Anwendung auszuführen. Weil Sie noch keine Untermenüs angelegt und keine Ereignisprozeduren geschrieben haben, passiert nichts, wenn Sie in der Menüleiste eine Auswahl treffen, aber Sie sehen bereits, wie einfach es sein kann, Menüs anzulegen. Klicken Sie auf die **Schliessen**-Schaltfläche für das Fenster (oder klicken Sie in der Symbolleiste von Visual Basic auf die Schaltfläche **Beenden**) und drücken Sie (Strg)+(E), um wieder in den Menü-Editor zurückzugelangen.

Ein Pull-down-Menü einfügen

Sie können die Pulldown-Menüs sofort oder später anlegen. Wenn Sie zuerst die vollständige Menüleiste anlegen, wie hier gezeigt, müssen Sie die Menüoptionen später an ihren jeweiligen Positionen anlegen. Dazu verwenden Sie im Menü-Editor die Schaltfläche **Einfügen**.



Die Schaltfläche **Einfügen** fügt eine leere Menüleisten- oder Menüposition an. Bevor Sie auf **Einfügen** klicken, markieren Sie die Menüoption hinter der Position, an der Sie eine Option einfügen möchten. Mit anderen Worten, um dem **Datei**-Menü eine Menüoption hinzuzufügen, markieren Sie im Listefeld für die Menüsteuerelemente den Eintrag **&Farbe** durch einen Klick und klicken dann auf **Einfügen**; Visual Basic fügt eine leere Zeile zwischen **&Datei** und **&Farbe** ein, wo Sie eine neue Menüoption für das **Datei**-Menü anlegen können.

Irgendwie muß Visual Basic zwischen einer Option in der Menüleiste und einer Option in einem Pull-down-Menü unterscheiden. Wenn Sie die **Beenden**-Option für das **Datei** -Menü mit derselben Technik wie in den vorhergehenden Abschnitten angelegt haben, zeigt Visual Basic die **Beenden**-Option in der Menüleiste zwischen **Datei** und **Farbe** an, nicht als Option des **Datei**-Menüs.

Bevor Sie eine neue Option einfügen, klicken Sie im Menü-Editor auf die Schaltfläche mit dem Pfeil nach rechts. Visual Basic fügt vor dem neuen Eintrag vier Punkte ein, die darauf hinweisen, daß das aktuelle Element eine Menüoption ist und nicht in der Menüleiste erscheint. Klicken Sie in das Textfeld **Caption** und geben Sie als Titel für die neue Option **&Beenden** ein. Während der Eingabe platziert der Menü-Editor die Option rechts eingerückt gegenüber den anderen Optionen im Listefeld. Diese Einrückung zeigt, daß das Element eine Option des vorhergehenden Menüs ist. Falls ein solcher Menüpunkt noch ein weiteres Untermenü hat, klicken Sie zweimal auf die Schaltfläche mit dem Pfeil nach rechts, um die Einrückungspunkte zweimal zu erhalten, die vor der Untermenü-Option erscheinen sollten.



Zu viele Untermenüebenen können verwirren. Verwenden Sie maximal zwei Menüebenen - ein Menü, das aus der Menüleiste heruntergeklappt wird, und möglicherweise noch ein Untermenü zu den dort aufgeführten Menüoptionen.

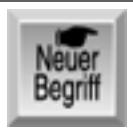
Geben Sie als Optionsnamen **mnuFileExit** ein. Damit haben Sie das **Datei**-Menü und sein

Dropdown-Menü fertig. Sie können das Programm ausführen und sehen, daß durch Auswahl von **Datei** jetzt ein Dropdown-Menü mit der Menüoption **Beenden** angezeigt wird. Es passiert jedoch noch nichts, wenn Sie diese Option auswählen, weil Sie noch keine Ereignisprozedur geschrieben haben.

Drei markierte Optionen einfügen

Das zweite Menü, **Farben**, erhält drei Optionen: **Blau**, **Grün** und **Rot**. Diese Farben schließen sich gegenseitig aus; das Bezeichnungsfeld kann nicht gleichzeitig rot, blau und grün sein. Diese Farbgebung ist also perfekt geeignet für die Demonstration *markierter Menüoptionen* (»checked«).

Abbildung 4.7 zeigt das Drop-down-Menü **Farben**. Wie Sie sehen, ist eine der Menüoptionen mit einem Häkchen gekennzeichnet. Die beiden anderen Optionen sind nicht gekennzeichnet. Beim Anlegen des **Farben**-Menüs legen Sie fest, daß beim Starten Rot als die aktive Option gesetzt wird, weil Sie beim Entwurf die Hintergrundfarbe auf Rot gesetzt haben. Der Benutzer sieht das rote Bezeichnungsfeld, bis er ihm eine neue Farbe zuweist.



Für eine *markierte Menüoption* wird ein Häkchen angezeigt, wenn sie aktiv ist.



Es ist möglich, daß mehrere Menüoptionen markiert werden, aber durch eine saubere Programmierung der Ereignisprozeduren für das Menü können Sie das verhindern. Wenn der Benutzer eine andere Option markiert, wird die ursprüngliche Markierung entfernt.

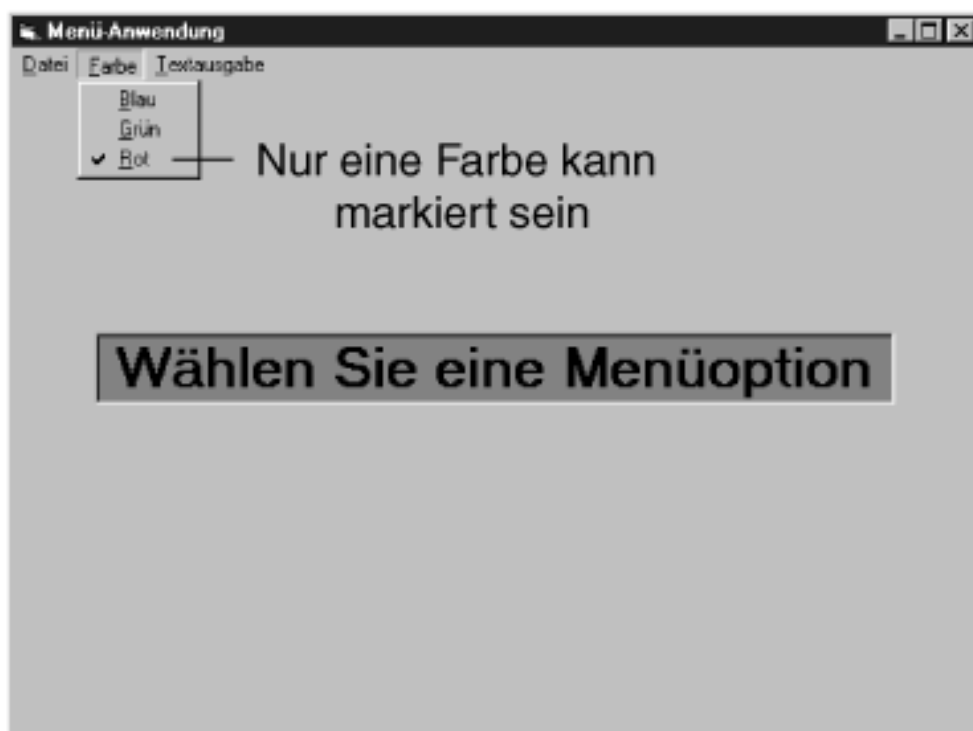


Abbildung 4.7:
Nur die markierte Menüoption ist aktiv.

Um dem Menü **Farben** Optionen hinzuzufügen, gehen Sie wie folgt vor:

1. Öffnen Sie den Menü-Editor.

2. Klicken Sie im Listenfeld für die Menüsteuerelemente auf die Option **&Textausgabe** .
3. Klicken Sie dreimal auf die Schaltfläche **Einfügen** und die Schaltfläche mit dem Pfeil nach rechts, um drei leere Zeilen für die Optionen im Menü **Farbe** einzufügen.
4. Klicken Sie in die erste leere Zeile, wo Sie die Option **Blau** einfügen.
5. Geben Sie als Titel **&Blau** ein, als Name `mnuColorBlue`. Wenn der Benutzer das Programm ausführt, ist die Option **Blau** nicht markiert, sie ist also nicht aktiv.
6. Klicken Sie auf **Nächster**, um die nächste Option einzufügen.
7. Geben Sie als Titel **&Grün** ein, als Name `mnuColorGreen`.
8. Klicken Sie auf **Nächster**, um die nächste Option einzufügen.
9. Geben Sie als Titel **&Rot** ein, als Name `mnuColorRed`.
10. Die Option **Rot** soll markiert sein, wenn der Benutzer das Programm aufruft. Klicken Sie also in das Feld **Checked**, um neben **Rot** ein Häkchen anzuzeigen.
11. Schließen Sie den Menü-Editor und führen Sie Ihre Anwendung aus, um das **Farbe** -Menü zu testen. Ihr Programmfenster sollte jetzt aussehen wie in Abbildung 4.7 gezeigt.

Sie können nicht nur markierte Menüoptionen bereitstellen, sondern die Menüoptionen auch aktiviert oder deaktiviert darstellen, entweder vom Menü-Editor aus, oder mit Hilfe der Programmanweisungen von Visual Basic. Abbildung 4.8 zeigt das **Debuggen** -Menü von Visual Basic mit mehreren aktivierten und mehreren deaktivierten Menüoptionen. Deaktivierte Menüoptionen werden grau dargestellt, so daß der Benutzer erkennt, daß er sie nicht aufrufen kann.

Mit der Option **Enabled** können Sie im Menü-Editor beim Menüentwurf festlegen, ob eine Option aktiviert dargestellt werden soll. Abhängig von der Anwendung aktivieren Sie Menüoptionen, wenn ihre Auswahl sinnvoll ist, und deaktivieren sie, wenn der Benutzer sie nicht auswählen soll. Viele Textverarbeitungen deaktivieren beispielsweise die Option **Einfügen** im **Bearbeiten**-Menü, wenn der Benutzer noch nichts in die Zwischenablage kopiert hat. Das Menü Ihrer Anwendung kann mit Steuerelementen verglichen werden, und mit Hilfe der verschiedenen Felder werden die Eigenschaften für diese Steuerelemente festgelegt. Ihr Visual-Basic-Code kann also die `Enabled`-Eigenschaft für ein Menüelement auf `True` oder `False` setzen, wie Sie in Kapitel 5 erfahren werden. Die Option kann aktiviert werden und dem Benutzer zur Verfügung stehen, oder deaktiviert und nicht mehr zur Verfügung stehen, bis das Programm sie wieder aktiviert.



Sie sehen, daß im **Debuggen**-Menü neben mehreren Optionen (die z.T. auch deaktiviert sind) ein Icon angezeigt wird. Diese Icons weisen auf die entsprechenden Schaltflächen in der Symbolleiste hin. Mit anderen Worten, für die Option **Debuggen, Einzelschritt** gibt es auch eine Schaltfläche in der Debuggen-Symbolleiste, die mit der hier angezeigten übereinstimmt. Diese Icons bieten dem Benutzer die Möglichkeit, zu lernen, welche Schaltfläche in der Symbolleiste dieselbe Aufgabe erfüllt wie die hier gezeigte Menüauswahl. Leider bietet Visual Basic keine Möglichkeit, in den Menüs Ihrer eigenen Anwendungen Icons anzuzeigen, aber es gibt einige Steuerelemente von Drittanbietern, die das erlauben.

Beachten Sie, daß es im **Debuggen**-Menü wie in vielen anderen Menüs Shortcuts gibt, die rechts neben den Menüoptionen angezeigt werden. (F8) ist der Shortcut für Einzelschritt. (Beschleunigertasten

werden auch oft als Shortcut-Tasten bezeichnet; sie lösen Menüoptionen aus, wie beispielsweise (Strg)+(C) für **Bearbeiten, Kopieren.**) Wenn Sie für eine Menüoption einen Shortcut festlegen möchten, wählen Sie die gewünschte Tastenkombination im Drop-down-Listefeld **Shortcut** aus. Die meisten Tastenkombinationen verwenden die (Strg)-Taste, beispielsweise (Strg)+(C).

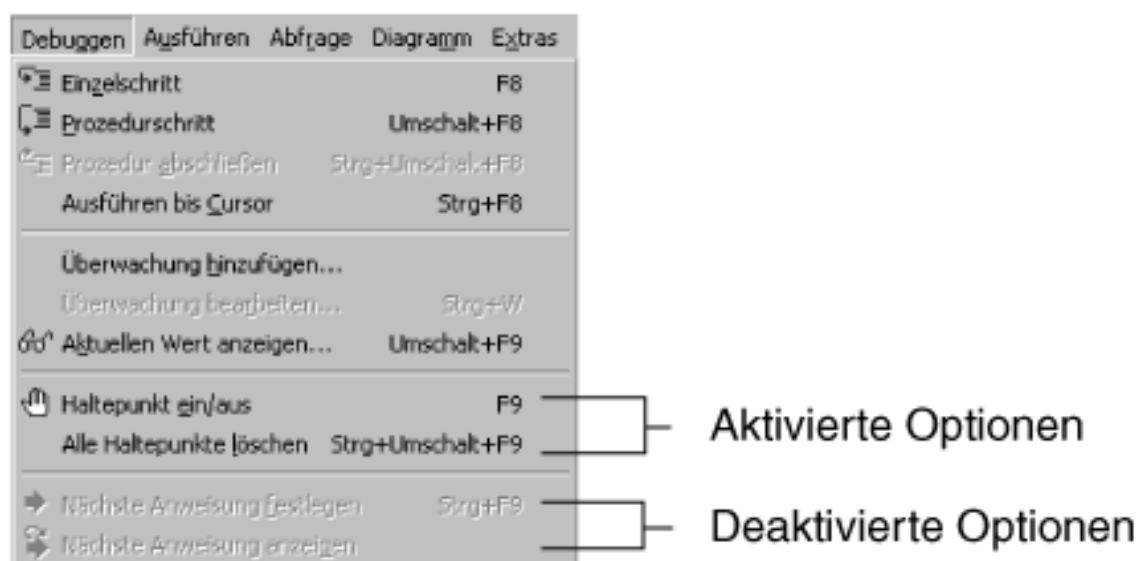


Abbildung 4.8:
Die Enabled-Eigenschaft legt fest, welche Menüoptionen aktiviert oder deaktiviert sind.

Das Untermenü Textausgabe fertigstellen

Die Option **Texteingabe** erzeugt ein Dropdown-Menü mit drei Optionen, wobei jeweils nur eine markiert ist, so wie im **Farben**-Menü. Die markierte Option zeigt die Meldung an, die im Bezeichnungsfeld ausgegeben wird. Der Übung halber formatieren Sie das Textausgabe-Menü anders als bisher. Das Menü Textausgabe sieht aus wie in Abbildung 4.9 gezeigt.

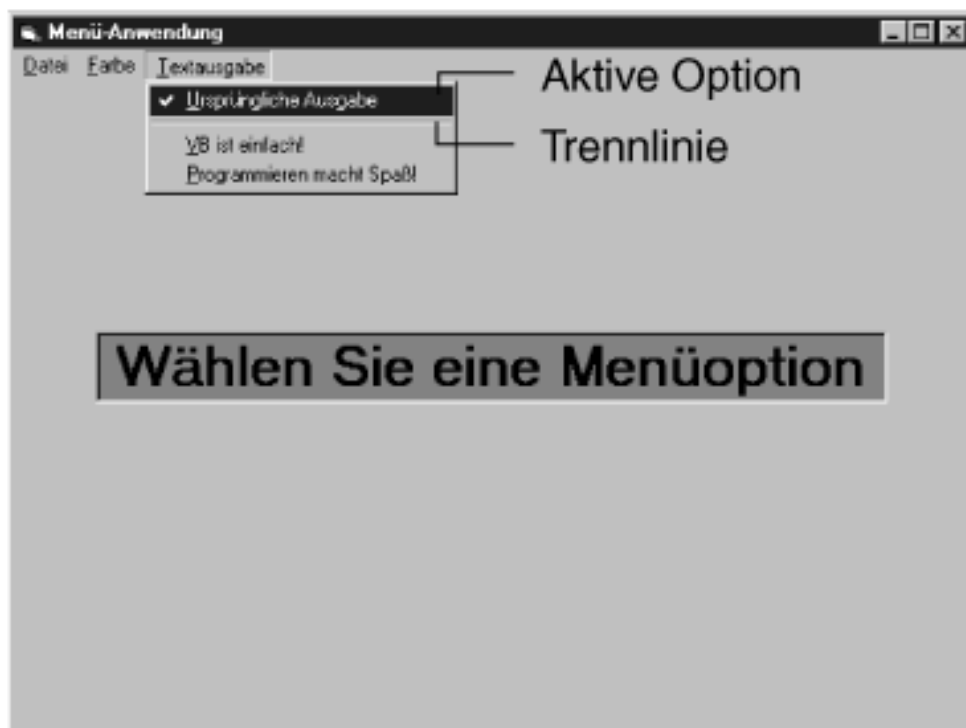


Abbildung 4.9:
Eine Trennlinie ermöglicht die Gruppierung von Menüoptionen.

Beachten Sie, daß es in diesem Menü eine Trennlinie gibt. Eine Trennlinie teilt Menüelemente von anderen Menüelementen ab. Die Trennlinie kann vom Benutzer nicht selektiert werden. Wenn der Benutzer die Taste (¼) drückt, während die Menüoption oberhalb der Trennlinie selektiert ist, wird die nächste Menüoption unterhalb der Trennlinie selektiert.

Gehen Sie wie folgt vor, um das Menü **Textausgabe** anzulegen und dort eine Trennlinie anzuzeigen:

1. Drücken Sie (Strg)+(E), um den Menü-Editor anzuzeigen.
2. Klicken Sie sich in der unteren Hälfte des Menü-Editors in die Zeile unterhalb von &Textausgabe, um die Eingabe der nächsten Option vorzubereiten.
3. Klicken Sie auf die Schaltfläche mit dem Pfeil nach rechts, um damit anzuzeigen, daß Sie eine Menüoption eingeben, die unterhalb der **Textausgabe**-Option in der Menüleiste aufgeklappt wird.
4. Geben Sie &Ursprüngliche Nachricht als Titel und mnuMessageInitial als Name für die neue Option ein.
5. Klicken Sie in das Feld **Checked**, um die Markierung beim Programmstart auf die erste Option zu setzen.
6. Klicken Sie auf **Nächster**, um das nächste Menüelement einzutragen. Beachten Sie, daß Visual Basic das nächste Element automatisch einrückt, weil auch das vorhergehende Element eingerückt war. (Wenn Sie nicht möchten, daß ein Element eingerückt dargestellt wird, klicken Sie auf die Schaltfläche mit dem Pfeil nach links, um die Punkte vor dem Eintrag zu entfernen. Die Schaltflächen mit dem Pfeil nach oben bzw. unten ermöglichen Ihnen, ein Element im Menübaum nach oben oder nach unten zu verschieben.)
7. Geben Sie für den Namen des nächsten Menüelements einen einzelnen Trennstrich (-) ein. Alle Trennlinien haben diesen Namen. Wenn Visual Basic den Trennstrich als Menünamen erkennt, wandelt es diesen in eine Trennlinie um.
8. Geben Sie als Namen für die Trennlinie mnuMessageSep1 ein. Trennlinien sind Objekte und sollten deshalb eindeutige Namen erhalten. Weitere Trennlinien im Menü numerieren Sie fortlaufend, also mnuMessageSep2, mnuMessageSep3 usw.



Trennlinien können nicht markiert, deaktiviert oder mit Shortcuts versehen werden.

9. Klicken Sie auf **Nächster**, um die nächste Menüoption einzugeben.
10. Geben Sie als Titel &VB ist einfach! ein, als Name mnuMessageSimple. Klicken Sie auf **Nächster**.
11. Geben Sie &Programmieren macht Spaß! als Titel und mnuMessageProgramming als Name für die nächste Option ein.

Ihr Menü-Editor sollte jetzt wie in Abbildung 4.10 gezeigt aussehen. Schließen Sie den Menü-Editor und führen Sie Ihr Programm aus, um das Menü anzuzeigen. Sie sind fast fertig. Jetzt brauchen Sie nur noch den Code einzufügen, der das Menü aktiviert.



Im Anwendungs-Assistenten können Sie ebenfalls Trennlinien einfügen. Wählen Sie die Option [Trennzeichen] aus, wenn Sie einem durch den Assistenten erzeugten Menü eine Trennlinie hinzufügen möchten.

Abbildung 4.10:
Ihr Menü ist fertig.

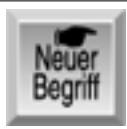
Code für das Menü

Jetzt fügen Sie dem Menü, das Sie in den letzten Abschnitten angelegt haben, Code hinzu, so daß es funktional wird. Wie in den vorherigen Kapiteln sollten Sie sich auch hier an dieser Stelle noch keine Gedanken über den Code machen. Konzentrieren Sie sich besser auf das Gesamtbild. Ein Menü und seine Optionen sind einfach nur Steuerelemente, die zur Laufzeit Ereignisse produzieren. Ihre Benutzer arbeiten mit dem Menü Ihrer Anwendung. Immer wenn sie eine Option auswählen, wird ein Click-Ereignis erzeugt. Für dieses wird die Click-Ereignisprozedur für diese Menüoption ausgeführt.

In Kapitel 3 haben Sie erfahren, daß die Click-Ereignisprozedur für die Menüoption `mnuFileExit` `mnuFileExit_Click()` heißt. Die Prozeduren für die anderen Menüoptionen erhalten entsprechende Namen. Listing 4.1 zeigt den vollständigen Code, der für die Anwendung erforderlich ist. Sie können den Code auf unterschiedliche Weise eingeben:

- Sie geben den Code für jede Prozedur einzeln ein. Wählen Sie im Formularfenster die Menüoption aus, der Sie Code hinzufügen möchten. Visual Basic öffnet das Codefenster und gibt die *Hüllzeilen* für Ihre Click-Ereignisprozedur vor. Anschließend füllen Sie den Rumpf für den Code aus. Schließen Sie das Codefenster und klicken Sie auf die nächste Menüoption, um

deren Coderumpf einzugeben, bis Sie auf diese Weise für alle Optionen eine Ereignisprozedur geschrieben haben.



Hüllzeilen sind die erste und die letzte Zeile der Prozedur. In Kapitel 3 haben Sie erfahren, welches Format die Hüllzeilen einer Ereignisprozedur haben. Sie wußten damals jedoch noch nicht, daß diese Zeilen als Hüllzeilen bezeichnet werden.

- Nachdem Sie die erste Ereignisprozedur nach der oben beschriebenen Methode erstellt haben, können Sie die nächste Menüoption auch direkt im Codefenster auswählen. Oben im Codefenster gibt es zwei Dropdown-Listenfelder, **Objekt** und **Prozedur**. Abbildung 4.11 zeigt die geöffnete **Objekt**-Liste. Wählen Sie die nächste Menüoption aus, für die Sie eine Prozedur schreiben möchten. Visual Basic platziert die Hüllzeilen für diese Ereignisprozedur unterhalb der vorhergehenden Ereignisprozedur im Codefenster. Geben Sie den Rumpf für die Prozedur ein. Fügen Sie die Ereignisprozeduren für alle Menüoptionen ein.



In dieser Anwendung ist jedes Objekt außer dem Bezeichnungsfeld ein Menüsteuerelement. Menüsteuerelemente unterstützen nur das **Click**-Ereignis, deshalb wird im Dropdown-Listenfeld **Prozedur** auch nur das **Click**-Ereignis angezeigt. Wenn Sie mit Steuerelementen arbeiten, die andere Ereignisse unterstützen, werden im Dropdown-Listenfeld des Codefensters auch andere Ereignisse bereitgestellt. Wenn Sie beispielsweise im **Objekt**-Listensfeld dieses Codefensters das Bezeichnungsfeld der Anwendung auswählen, sehen Sie im Prozedurfeld mehrere Ereignisnamen aufgelistet, weil das Bezeichnungsfeld mehrere Ereignistypen unterstützt.

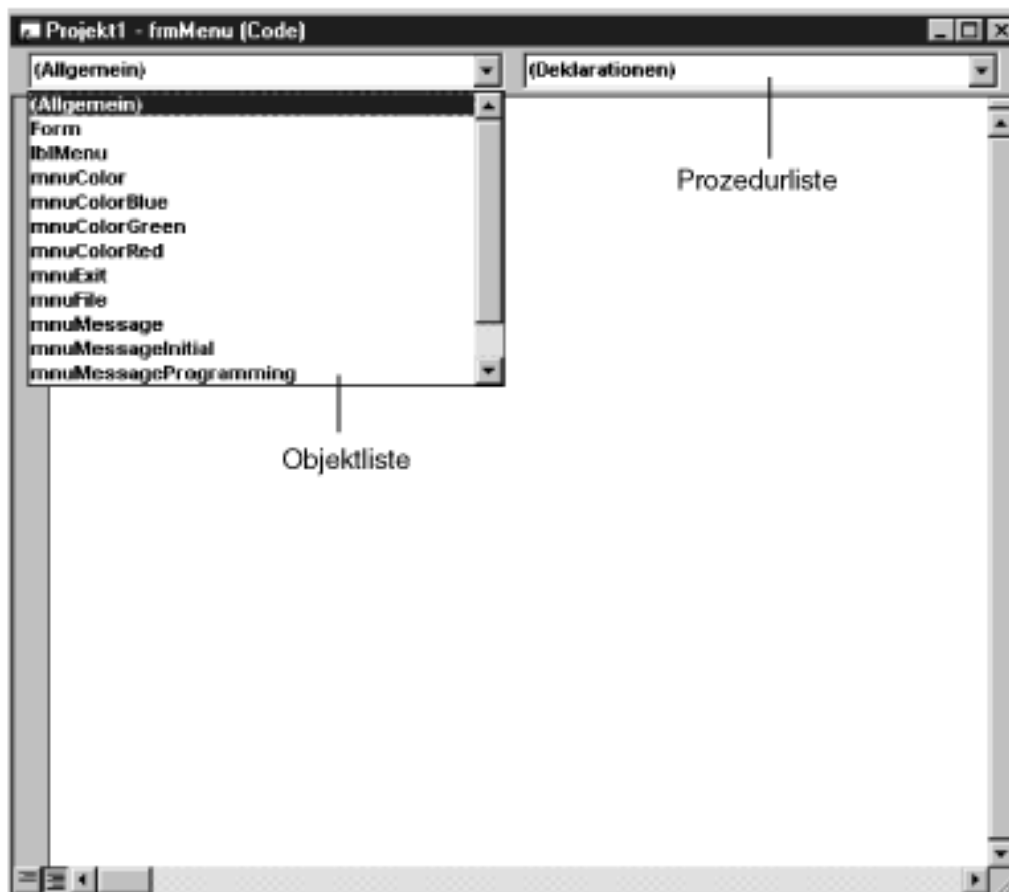


Abbildung 4.11:

Im Codefenster können Sie ein neues Paar Hüllzeilen anfordern.

- Sie öffnen das Codefenster und geben den Code von Anfang bis Ende ein. Diese Methode ist jedoch zeitaufwendiger, weil Sie die Hüllzeilen selbst tippen müssen.



Die Dropdown-Listenfelder **Objekt** und **Prozedur** im Codefenster sind praktisch, um Code zu finden, den Sie bereits eingegeben und jetzt ändern oder anzeigen möchten. Wenn eine Anwendung beispielsweise mehrere Ereignisprozeduren für verschiedene Steuerelemente enthält, und Sie die Doppelklick-Ereignisprozedur für eine bestimmte Schaltfläche schreiben möchten, wählen Sie die Schaltfläche aus dem Dropdown-Listenfeld **Objekt** aus. Anschließend wählen Sie im Dropdown-Listenfeld **Prozedur** **DbLClick** aus. Visual Basic sucht den Code für diese Ereignisprozedur und zeigt ihn im Codefenster an.

Listing 4.1: Der Menücode steuert, welche Farbe und welchen Inhalt das Bezeichnungsfeld hat.

```
1: Private Sub mnuColorBlue_Click()  
2: ' Bezeichnungsfeld Blau machen und die Menüoption  
3: ' Blau markieren. Die Optionen Rot und Grün  
4: ' dürfen nicht markiert sein  
5: lblMenu.BackColor = vbBlue  
6: mnuColorBlue.Checked = True  
7: mnuColorGreen.Checked = False  
8: mnuColorRed.Checked = False  
9: End Sub  
10:  
11: Private Sub mnuColorGreen_Click()  
12: ' Bezeichnungsfeld Grün machen und die Menüoption  
13: ' Grün markieren. Die Optionen Blau und Rot  
14: ' dürfen nicht markiert sein  
15: lblMenu.BackColor = vbGreen  
16: mnuColorBlue.Checked = False  
17: mnuColorGreen.Checked = True  
18: mnuColorRed.Checked = False  
19: End Sub  
20:  
21: Private Sub mnuColorRed_Click()  
22: ' Bezeichnungsfeld Rot machen und die Menüoption  
23: ' Rot markieren. Die Optionen Blau und Grün  
24: ' dürfen nicht markiert sein  
25: lblMenu.BackColor = vbRed  
26: mnuColorBlue.Checked = False  
27: mnuColorGreen.Checked = False  
28: mnuColorRed.Checked = True  
29: End Sub  
30:  
31: Private Sub mnuFileExit_Click()  
32: ' Programm beenden  
33: End  
34: End Sub
```

```

35:
36: Private Sub mnuMessageInitial_Click()
37: ' Wieder die Originalmeldung anzeigen und die
38: ' entsprechende Menüoption markieren.
39: ' Die anderen Optionen dürfen nicht markiert sein.
40: lblMenu.Caption = "Wählen Sie eine Menüoption aus"
41: mnuMessageInitial.Checked = True
42: mnuMessageProgramming.Checked = False
43: mnuMessageSimple.Checked = False
44: End Sub
45:
46: Private Sub mnuMessageProgramming_Click()
47: ' Ersatzmeldung anzeigen und die entsprechende
48: ' Menüoption markieren.
49: ' Die anderen Optionen dürfen nicht markiert sein.
50: lblMenu.Caption = "Programmieren macht Spaß!"
51: mnuMessageInitial.Checked = False
52: mnuMessageProgramming.Checked = True
53: mnuMessageSimple.Checked = False
54: End Sub
55:
56: Private Sub mnuMessageSimple_Click()
57: ' Ersatzmeldung anzeigen und die entsprechende
58: ' Menüoption markieren.
59: ' Die anderen Optionen dürfen nicht markiert sein.
60: lblMenu.Caption = "VB ist einfach!"
61: mnuMessageInitial.Checked = False
62: mnuMessageProgramming.Checked = False
63: mnuMessageSimple.Checked = True
64: End Sub

```

Auch hier sollten Sie sich keine Gedanken über die Details machen, sondern versuchen, die Arbeitsweise von Ereignisprozeduren zu verstehen, bevor Sie weiterlesen. In Kapitel 5 erhalten Sie erste Informationen über die Programmiersprache Visual Basic. Wenn Sie verstanden haben, wie Visual Basic arbeitet, werden Sie auch die Code-Details besser verstehen.

Führen Sie das Programm aus und testen Sie es. Wählen Sie die Menüoptionen **Farbe** und **Textausgabe** mehrere Male aus, um sicherzustellen, daß die Markierung dort angezeigt wird, wo Sie sie nach Ihrer Auswahl vermuten. Sie können die Farben ändern, bevor oder nachdem Sie den Text im Bezeichnungsfeld geändert haben.

Zusammenfassung

Dieses Kapitel hat Ihnen gezeigt, wie in Visual Basic Menüs angelegt werden. Der Menü-Editor verhält sich wie ein spezielles Eigenschaftenfenster, mit dem Sie die Werte für die Menüsteuerelemente einfach festlegen können. Im Menü-Editor verwalten Sie die Eigenschaften **Checked** und **Visible** für Menüoptionen und bestimmen außerdem Shortcuts für die verschiedenen Optionen.

In Kapitel 5 werden Sie die Programmiersprache Visual Basic kennenlernen. Sie erfahren, wie Visual

Basic Daten erkennt und speichert und wie es im Code Eigenschaftswerte zur Laufzeit zuweist.

Fragen & Antworten

Frage:

Warum erzeugt die Auswahl in einer Menüleiste kein Click-Ereignis?

Antwort:

Es wird erzeugt, es sei denn, durch die Menüoption wird ein Dropdown-Menü aufgerufen. Wenn für eine Menüoption kein Dropdown-Menü erscheint, erzeugt ihre Auswahl ein Click-Ereignis. Wenn dagegen ein Menü erscheint, hat das Menü Priorität gegenüber dem Click-Ereignis, und statt eines Click-Ereignisses wird das Dropdown-Menü angezeigt.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Wie heißt das Dialogfeld, in dem Sie Menüs anlegen können?
2. Richtig/Falsch. Menüoptionen sind Steuerelemente.
3. Welches Ereignis unterstützen alle Menüoptionen?
4. Wofür ist der Begriff »Shortcut« im allgemeinen reserviert?
5. Was bedeuten Shortcuts für die Arbeit mit Menüs?
6. Welche Ereignisse erzeugen Menüoptionen, wenn der Benutzer über einen Shortcut darauf zugreift?
7. Richtig/Falsch. Der Menü-Editor hilft, Menüs zu entwickeln und den Code für die Click-Ereignisprozedur für die Menüoptionen zu erzeugen.
8. Welche Aufgabe hat die Menüeigenschaft Checked?
9. Richtig/Falsch. Es ist möglich, mehrere Menüoptionen gleichzeitig zu markieren.
10. Was bedeuten in Listing 4.1 die Zeilen 57, 58 und 59?

Übungen

1. Beschreiben Sie den Unterschied zwischen der Eingabe einer Menüleistenoption und einer Dropdown-Menüoption im Menü-Editor.
2. *Fehlersuche:* Der Menümeister Manuel hat Probleme mit seinen Menüs, weil die Markierung von einer Menüoption nicht entfernt werden kann, wenn der Benutzer eine andere markierte Option auswählt. Können Sie Manuel einen ganz allgemeinen Ratschlag geben? (Sie müssen noch keinen Code schreiben.)

Fügen Sie Shortcuts für alle Menüoptionen in der heute entwickelten Anwendung ein. Stellen Sie sicher, daß zwei Optionen nicht denselben Shortcut haben.

❖Kapitel Inhalt Index ❖ Top Kapitel❖

Woche 1

Tag 5

Visual Basic - Grundlagen der Sprache

Mit diesem Kapitel beginnen Sie Ihren Streifzug durch die Programmierung mit der Sprache Visual Basic. Statt mit grafischen Objekten wie beispielsweise Befehlsschaltflächen zu arbeiten, geben Sie heute Text ein, der das Gehirn Ihrer Anwendung darstellt und dafür sorgt, daß etwas passiert. Hier werden Sie zunächst erfahren, wie Visual Basic Daten behandelt. Im nächsten Kapitel lernen Sie, wie man diese Daten mit Hilfe von Befehlen und Steuerelementen auf dem Formular manipuliert.

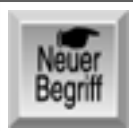
Heute lernen Sie die folgenden Dinge kennen:

- Das Codefenster
- Verschiedene Daten, die Sie deklarieren können
- Datentypen
- Speicheranforderungen der Visual-Basic-Daten
- Variablen - Deklaration und Wertzuweisungen
- Operatorprioritäten

Vorbemerkungen

Ein paar Sache setzen wir an dieser Stelle schon voraus: insbesondere sollten Sie wissen, wie Code in die Formulare und Steuerelemente einer Anwendung eingefügt wird. Sie wissen, daß Sie im Projektfenster alle Ihre Anwendungen verwalten und alle ihnen zugeordneten Dateien anzeigen können. Sie schreiben die Ereignisprozeduren zwar innerhalb des Codefensters, aber sie werden nicht in eigenen Dateien abgelegt, sondern zusammen mit ihren Steuerelementen. Ihre Projekte umfaßten bisher immer nur ein einziges Formular, und dieses Formular enthielt die Steuerelemente sowie die Ereignisprozeduren für diese Steuerelemente. Wie Sie sehen, kann ein Formular auch Code enthalten, bei dem es sich nicht um eine spezielle Ereignisprozedur handelt, sondern der auch allgemeineren Zwecken dient.

Einige Projekte enthalten auch anderen Code. In Kapitel 4 haben Sie erfahren, daß der gesamte Code in Prozeduren abgelegt ist, und daß Visual Basic zwischen Sub-Prozeduren und Funktionen unterscheidet. Ereignisprozeduren gehören zur Kategorie Sub-Prozeduren, aber im Laufe dieses Buchs werden Sie auch noch lernen, Funktionsprozeduren zu schreiben. Wenn eine Prozedur keine Ereignisprozedur für ein bestimmtes Steuerelement ist, erscheint dieser Code innerhalb eines separaten Codemoduls. Falls das Projekt ein Modul enthält, wird dieses im Projektfenster aufgelistet.



Ein *Modul* ist eine Datei, die Programmcode enthält. Die Datei, die das Formular und seinen Code enthält, wird als das *Formularmodul* bezeichnet, Sie haben also bereits mit Modulen gearbeitet.

Wenn ein Projekt mehrere Formulare enthält, enthält es auch mehrere Formularmodule, weil die Steuerelemente jedes Formulars auf Ereignisse reagieren müssen. Jedes Formular legt also seinen eigenen Code in einem Formularmodul ab. Einer der ersten Aspekte, den Sie bei der Verwendung mehrerer Formulare beachten sollten, ist, welches Formular als erstes auf dem Bildschirm erscheint, wenn der Benutzer die Anwendung startet. Das erste Formular, das Sie erzeugen, ist das Standard-Startobjekt, Sie können jedoch auch ein anderes Startobjekt bestimmen, indem Sie **Projekt | Eigenschaften von** <projektname> im Visual-Basic-Menü auswählen, wobei <projektname> den Namen des aktuellen Projekts darstellt. Visual Basic zeigt das Dialogfeld **Projekteigenschaften** an, wie in Abbildung 5.1 gezeigt. Während Ihrer Arbeit in den einzelnen Kapiteln lernen Sie, wie Sie Visual Basic so programmieren, daß gegebenenfalls ein zweites Formular erscheint.



Abbildung 5.1:
Im Dialogfeld Projekteigenschaften bestimmen Sie das Startformular.

Die Arbeit im Codefenster

Nachdem Sie wissen, was Ereignisprozeduren sind, werden Sie in den folgenden Kapiteln damit arbeiten, um so die Grundlagen der Programmiersprache Visual Basic kennenzulernen. Bevor Sie Details erfahren, sollten Sie jedoch verstehen, daß Formularmodule nicht nur Ereignisprozeduren enthalten, sondern auch einen *Deklarationsbereich*. Jedes Modul, das Code enthält, enthält auch einen Deklarationsbereich.



Ein *Deklarationsbereich* reserviert Platz für Datenbereiche und weist ihnen Namen zu. Sie werden im gesamten restlichen Modul verwendet. Sie brauchen im Deklarationsbereich keine Steuerelemente zu deklarieren, aber häufig ist es erforderlich, andere Visual-Basic-Datenspeicherbereiche dort zu deklarieren.

Der Deklarationsbereich erscheint immer ganz oben in einem Formular, das Code enthält. Er befindet

sich also vor allen Ereignisprozeduren in einem Formularmodul. Immer wenn Sie Code vor der ersten Ereignisprozedur eines Moduls eingeben, wird dieser Code als allgemein betrachtet und keinem bestimmten Steuerelement zugeordnet. Wenn Sie gerade erst beginnen, unter Visual Basic zu programmieren, deklarieren Sie in diesem Bereich Daten, können ihn also als reinen Datendeklarationsbereich betrachten. Später werden Sie in diesem Bereich auch allgemeine Prozeduren schreiben.

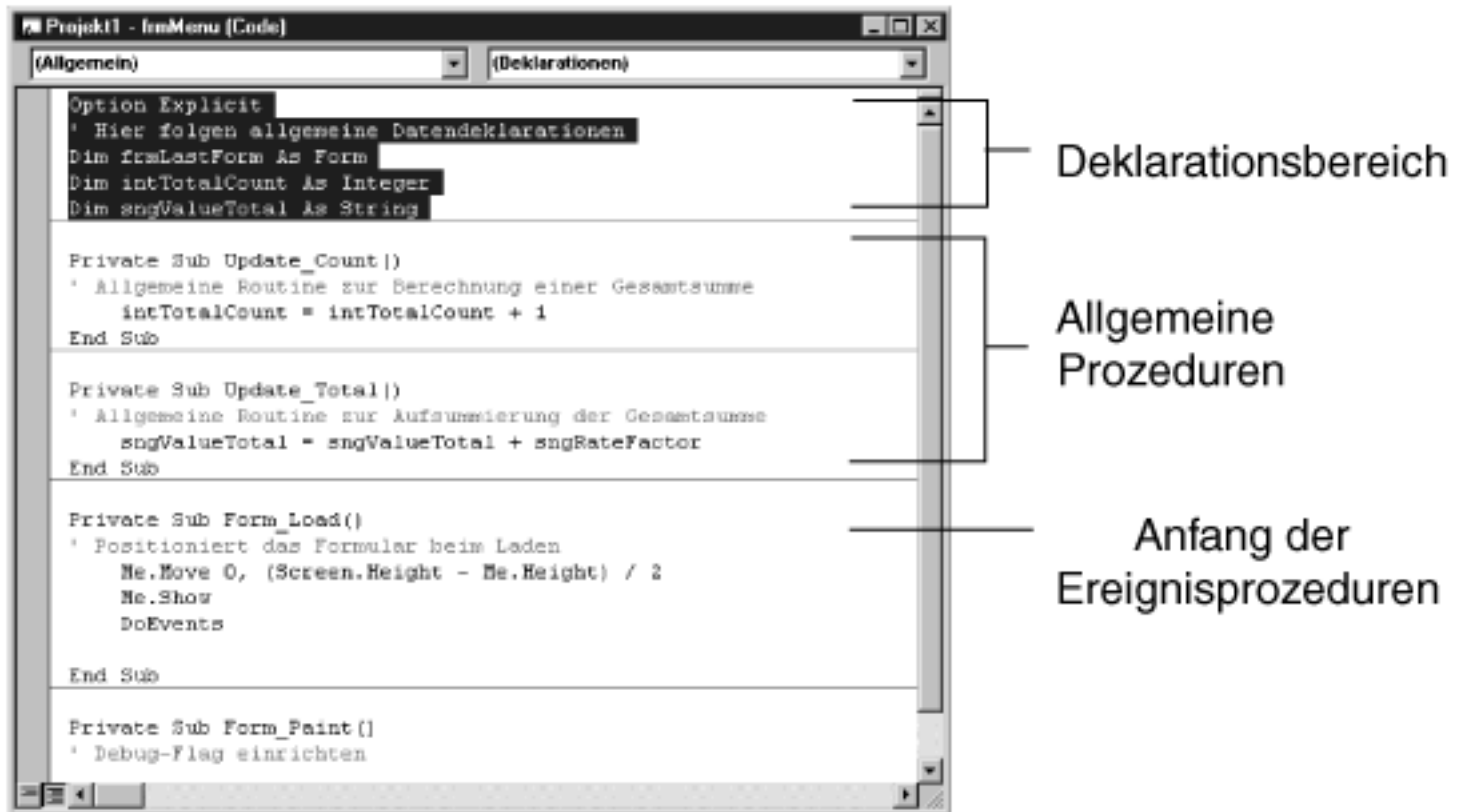


Abbildung 5.2:
Das Codefenster enthält verschiedene Abschnitte.

Betrachten Sie das Codefenster in Abbildung 5.2, um sich diese Information zu vergegenwärtigen. Die Details sind hier nicht wichtig - vielmehr sollten Sie das allgemeine Konzept verstehen. Der selektierte Text ist der Deklarationsbereich, beginnend mit **Option Explicit** als erste Anweisung. Beachten Sie, daß die Dropdown-Listenfelder **Objekt** und **Prozedur** die Einträge (Allgemein) und (Deklarationen) enthalten. Im Codefenster erkennen Sie anhand dieser Dropdown-Listenfelder, zu welchem Programmteil der betreffende Code gehört.

Die beiden nächsten Prozeduren sind keine Ereignisprozeduren, was Sie an ihren Namen erkennen. Jede Ereignisprozedur muß einen Unterstrich enthalten, der den Steuerelementnamen von dem Ereignisnamen abtrennt. Wenn Sie den Textcursor irgendwo in der Prozedur **Update_Count ()** platzieren, enthält das Dropdown-Listefeld **Objekt** weiterhin den Eintrag (Allgemein), weil der Code im allgemeinen Abschnitt des Moduls liegt. Das Listefeld **Prozedur** dagegen würde den Eintag **Update_Count** enthalten, weil das der Name der ausgewählten Prozedur ist. Das Codefenster listet die Namen für alle Prozeduren, die keine Ereignisprozeduren sind, im Listefeld **Prozedur** auf.

Genug mit den allgemeinen Ausführungen - widmen wir uns den Details!

Daten in Visual Basic

Bei der Programmierung unter Visual Basic verarbeiten Sie unterschiedliche Datentypen. Beispielsweise verwenden Sie Namen, Adressen, Geldbeträge, große Zahlen, kleine Zahlen und logische Daten, die True oder False (oder Richtig/Falsch, Ja/Nein usw.) sein können. Visual Basic unterstützt mehrere Datentypen, die allen Ihren Bedürfnissen genügen sollten.

Visual Basic ist wie die meisten anderen Programmiersprachen sehr heikel, was den Umgang mit Daten betrifft. Sie sollten also die Anforderungen, die Visual Basic an die Daten stellt, erfüllen. Bevor Sie mit einem Datenwert arbeiten, müssen Sie Visual Basic genau mitteilen, um welchen Datentyp es sich handelt. Sie sollten also zunächst lernen, welche Datentypen es gibt. Visual Basic unterstützt 12 Datentypen.

Numerische Daten

Im allgemeinen kann man alle numerischen Daten in zwei Kategorien einteilen:

- **Integer.** Integer sind ganze Zahlen ohne Dezimalpunkt, beispielsweise 614, 0, -934 oder 3918938. Integer können Zähler, Jahreszahlen, Altersangaben und andere ganzzahlige Werte aufnehmen.
- **Dezimalzahlen.** Zahlen mit Dezimalpunkt stellen Bruchwerte dar, beispielsweise 8.709, 0.005 oder -402.35534. Dezimalzahlen (manchmal auch als *Fließkommazahlen* bezeichnet) stellen beispielsweise Temperaturen, Geldbeträge oder Zinsraten dar. Für Dezimalzahlen muß ein Dezimalpunkt angegeben werden, auch wenn hinter dem Komma nur Null steht.



Numerische Werte werden auch als *Literale* oder *Konstanten* bezeichnet, weil sie sich nie ändern. Im Abschnitt »Die Arbeit mit Variablen« erfahren Sie, wie man Daten deklariert, die sich ändern können.



Integer und Dezimalzahlen werden in Visual Basic unterschiedlich gespeichert. Visual Basic behandelt sie unterschiedlich, auch wenn wir Menschen das nicht tun. Beispielsweise ist -8 für Visual Basic nicht dasselbe wie -8,00.

Einige Datentypen belegen sehr viel Speicher und sind ineffizient, während andere weniger Speicher verbrauchen und für Berechnungen schneller sind. Von außen sehen Sie es einer Zahl nicht an, wie viel Speicher sie belegt. Die Zahl 29999 beispielsweise belegt genau so viele Speicherpositionen wie die Zahl 701.



Bei der Vorstellung der verschiedenen Datentypen von Visual Basic erfahren Sie auch, wieviel Speicher dafür jeweils erforderlich ist. Die Speicheranforderungen sind zwar heute nicht mehr ganz so wichtig, wie sie es schon einmal waren, aber als Programmierer möchten Sie, daß Ihr Programm so effizient wie möglich ausgeführt wird. Wenn Sie also die Wahl zwischen zwei oder mehr Datentypen für einen Wert haben, verwenden Sie den Datentyp, der am wenigsten Speicher verbraucht.

Tabelle 5.1 beschreibt die sieben numerischen Datentypen von Visual Basic, ihre Speicheranforderungen und ihren Wertebereich. Entscheiden Sie anhand der Speicheranforderungen

und der Wertebereiche, welchen Datentyp Sie zur Deklaration von Daten verwenden. Wenn Sie beispielsweise negative Zahlen darstellen wollen, können Sie nicht den Datentyp **Byte** verwenden. Wollen Sie dagegen das Alter bestimmter Personen speichern, ist **Byte** der effizienteste und am besten geeignete Datentyp.



Ein *Byte* ist eine Speicherposition im Speicher des PC.

Tabelle 5.1: *Visual Basic* unterstützt sieben numerische Datentypen.

Typ	Speicher	Bereich
Byte	1 Byte	0 bis 255
Integer	2 Byte	-32768 bis 32767
Long	4 Byte	-2147483648 bis 2147483647
Single	4 Byte	-3.402823E+38 bis -1.401298E-45 für negative Werte; 1.401298E-45 bis 3.402823E+38 für positive Werte
Double	8 Byte	-1.79769313486232E+308 bis -4.94065645841247E-324 für negative Werte; 4.94065645841247E-324 bis 1.79769313486232E+308 für positive Werte
Currency	8 Byte	-922337203685477.5808 bis 922337203685477.5807 (die zusätzliche Genauigkeit stellt sicher, daß Berechnungen mit Geldbeträgen auf zwei Dezimalstellen genau ausgeführt werden)
Decimal	12 Byte	+/-79228162514264337593543950335 falls Sie keine Dezimalstellen verwenden; +/-7.9228162514264337593543950335 mit bis zu 28 Dezimalstellen (der Datentyp Decimal wird unter <i>Visual Basic</i> nicht vollständig unterstützt, wird aber zur Kompatibilität mit zukünftigen Versionen bereitgestellt)



Einige Datenwerte in Tabelle 5.1 sind in *wissenschaftlicher Notation* angegeben.



Die *wissenschaftliche Notation* stellt eine Abkürzung für die Darstellung extrem großer oder extrem kleiner Dezimalwerte dar. Das E steht für einen *Exponenten mit einfacher Genauigkeit* (als Datentyp **Single** gespeichert). Zahlen mit einer sehr exakten, wissenschaftlichen Notation verwenden ein D für einen *Exponenten mit doppelter Genauigkeit* (abgelegt als Datentyp **Double**). Für die Darstellungen können auch Kleinbuchstaben verwendet werden, e und d. Um einen Wert von der wissenschaftlichen Notation in sein numerisches Äquivalent umzuwandeln, multiplizieren Sie die Zahl links vom D oder E mit 10 in der durch die Zahl auf der rechten Seite bestimmten Potenz. Beispielsweise ist 5.83E+5 gleich 5.83 mal 10⁵, also 5.83 mal 100000 oder 583000, abgelegt als Wert mit einfacher Genauigkeit. Hier ist die wissenschaftliche Notation nicht kürzer als die Angabe 584000, aber offensichtlich ist das für 5.83E+125 (das wäre die Zahl 583 gefolgt von 123 Nullen) der Fall. Die Zahl -3.2D-6 stellt -3.2 dividiert durch 1000000 oder -.000032 dar, gespeichert als **Double**-Wert.

Der Aspekt der Datentypen erscheint Ihnen möglicherweise immer noch unwichtig, weil Sie erst noch die Datenspeicherbereiche kennenlernen müssen, aber das werden Sie im Abschnitt »Die Arbeit mit Variablen« gleich nachholen. Wenn Sie den Konstantenwert -8.3 in ein Programm eingeben, müssen Sie nicht angeben, daß er den Datentyp **Single** hat. Sie müssen sich jedoch darum kümmern, welchen Typ die Position hat, wo der Konstantenwert abgelegt wird. Sie können -8.3 nicht an einer Integer-Position ablegen und erwarten, daß Visual Basic den Wert richtig darstellt.

Manchmal wollen Sie bei der Verwendung einer Konstanten sicherstellen, daß Visual Basic einen speziellen Datentyp dafür verwendet. Angenommen, Sie möchten -8.3 zusammen mit **Double**-Werten in einer exakten mathematischen Berechnung verwenden. Wenn Visual Basic voraussieht, daß es sich bei -8.3 um den Datentyp **Single** handelt, kann die Berechnung nicht ausgeführt werden, weil in einigen Fällen sehr viele Dezimalstellen erforderlich sind. Sie können jedoch ein Datentyp-Suffix für Konstanten angeben, um sicherzustellen, daß Visual Basic ihnen einen bestimmten Datentyp zuordnet. Tabelle 5.2 zeigt diese Suffixe für die Datentypen. Wenn Sie in die Berechnung also `-8.3#` eingeben, erkennt Visual Basic, daß Sie die -8.3 von Anfang an als Zahl mit doppelter Genauigkeit verstanden wissen wollen - mit der höchstmöglichen Genauigkeit in den Dezimalstellen.

**Tabelle 5.2: Konstanten-Suffixe
für Visual Basic**

Suffix	Zugehöriger Datentyp
&	Long
!	Single
#	Double
@	Currency



Das E und das D in der wissenschaftlichen Notation repräsentiert die Datentypen **Single** und **Double**, Sie brauchen dort also keine Suffix-Zeichen, weil die Konstantentypen implizit deklariert sind.

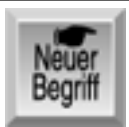


Obwohl dieser Abschnitt Ihnen sehr viel Theorie beigebracht hat, müssen Sie sich bei der Arbeit mit Konstanten keine besonderen Gedanken über Datentypen machen. Wenn Sie der Eigenschaft eines Steuerelements eine Zahl zuweisen möchten, dann tun Sie das einfach. Nur in ganz bestimmten Fällen, beispielsweise bei wissenschaftlichen und mathematischen Berechnungen mit großer Genauigkeit, müssen Sie sich selbst darum kümmern, ob eine Konstante die einfache oder die doppelte Genauigkeit verwendet. Achten Sie unbedingt darauf, keine Sonderzeichen innerhalb einer Zahl anzugeben. Schreiben Sie keine Punkte in eine lange numerische Konstante, es sei denn, Ihre Ländereinstellung von Windows ist entsprechend ausgelegt. Statt dessen verwenden Sie Kommata für Dezimalzahlen.

Visual Basic erkennt die Ländereinstellungen von Windows in Ihrem PC. Wenn Sie Ihren PC für ein europäisches Land eingerichtet haben, verwenden Sie statt des Dezimalpunkts ein Komma. Bei der Einrichtung für andere Länder wird der Dezimalpunkt verwendet.

Die restlichen Datentypen

Die nicht-numerischen Datentypen sind leichter zu verstehen als einige der numerischen Datentypen mit höchster Genauigkeit - falls Sie mathematisch nicht vorbelastet sind. Einer der Gründe, warum sich BASIC und seine Nachkommen so lange durchsetzen konnten, obwohl es mittlerweile »leistungsfähigere« Sprachen gibt, ist die Leistung, die BASIC in Bezug auf Text bereitstellt. BASIC und damit auch Visual Basic übertreffen in Hinblick auf die Verarbeitung von *Textstrings* fast alle anderen Sprachen.



Ein *String* ist eine Folge aus null oder mehr Zeichen. Ein String kann zwar Ziffern enthalten, wird aber nie in Berechnungen verwendet. Sie können Strings für Namen, Adressen, Codes, Sozialversicherungsnummern oder andere Datenwerte verwenden, mit denen Sie keine Berechnungen ausführen müssen. Verwenden Sie nur dann numerische Datentypen, wenn Sie Berechnungen ausführen möchten oder streng numerische Informationen aufnehmen müssen, beispielsweise Geldbeträge.

Neben den Stringdaten unterstützt Visual Basic auch noch andere Datentypen, beispielsweise Datumswerte oder *Boolesche* Daten. Tabelle 5.3 listet die nicht-numerischen Datentypen auf, die Visual Basic unterstützt.

Tabelle 5.3: Visual Basic unterstützt mehrere nicht-numerische Datentypen

Datentyp	Speicherbedarf	Bereich
String (feste Länge)	Länge des Strings	1 bis ca. 65400 Zeichen
String (variabel)	Länge + 10 Byte	0 bis 2 Milliarden Zeichen
Date	8 Byte	1. Januar 100 bis 31. Dezember 9999
Boolean	2 Byte	True oder False
Object	4 Byte	Ein eingebettetes Objekt
Variant (numerisch)	16 Byte	Beliebiger Wert; so groß wie Double

Variant (Text)	Länge plus 22 Byte	Entspricht einem variabel langen String

	<i>Boolesche Daten</i> , benannt nach dem Mathematiker George Boole, stellen Daten dar, die nur zwei verschiedene Werte annehmen können. Diese Werte werden normalerweise als True und False (Richtig und Falsch) angegeben, können aber auch wie Ja/Nein-Werte behandelt werden.

Stringkonstanten müssen immer in Anführungszeichen eingeschlossen werden. Strings können beliebige Zeichen enthalten. Hier ein paar Beispiele für gültige Stringkonstanten:

```
"Ohjemine"
"543-00-0234"
"Fabrikstr. 7"
""
```

Alles, was zwischen zwei Anführungszeichen geschrieben ist, ist ein String, sogar der *Nullstring* am Ende der Liste.

	Ein <i>Nullstring</i> , manchmal auch als String der Länge Null bezeichnet, ist null Byte lang und wird manchmal verwendet, um Stringdaten ohne einen Wert zu initialisieren. Der spezielle Visual-Basic-Wert Null stellt ebenfalls Nullstrings dar. Visual Basic unterstützt außerdem einen besonderen Stringwert, den leeren String, der durch das Schlüsselwort Empty dargestellt wird. Leere Strings stellen Strings ähnlich den Nullstrings dar, die Interpretation ist jedoch etwas anders: Eine Steuerelementeigenschaft, die das Schlüsselwort Empty enthält, gilt als nicht initialisiert, auch nicht mit einem Nullstring.

Der Unterschied zwischen Strings fester Länge und Strings variabler Länge wird Ihnen verständlich, sobald Sie mehr über die Datenspeichermethoden von Visual Basic gelernt haben.

	Vielleicht haben Sie schon von dem Problem mit dem Jahr 2000 gehört, auch als <i>Y2K-Fehler</i> bezeichnet. Jahrelang verwendeten die Programmierer zweistellige Jahreszahlen im Code, um Speicher zu sparen, den sie für die vollständige Jahresangabe gebraucht hätten. Diese Programme bekommen möglicherweise Probleme, wenn der Wechsel von 99 auf 00 erfolgt. Visual Basic ist auf das Jahr 2000 vorbereitet, d.h. die interne Datendarstellung berücksichtigt das nächste Jahrtausend. Am 31. Dezember 1999 um Mitternacht sollten Ihre Visual-Basic-Programme also keine Probleme haben, in das nächste Jahrtausend zu wechseln. Nichtsdestotrotz weichen einige Visual-Basic-Programmierer auf seltsame Tricks aus, die Zeit und Platz sparen sollen, deshalb muß nicht jeder Visual- Basic-Code notwendigerweise funktionieren. Als Neuling der Visual-Basic- Programmierung sollten Sie das Y2K-Programm berücksichtigen und in Ihren Programmen immer mit der vollständigen Jahresangabe arbeiten. Auf diese Weise bekommen Sie keine Probleme, wenn das Jahr 2000 endlich da ist.

Wenn Sie einen Datums- oder Zeitwert eingeben, schließen Sie diesen zwischen zwei Pfundzeichen (#) ein. Visual Basic erlaubt fast jedes Datums- und Zeitformat. Datum und Zeit können beliebigen

Ländereinstellungen Ihres PC entsprechen. Die folgenden Werte stellen gültige Datums- und Zeitwerte dar:

#4. Juli 1776#

#7:11 pm#

#19:11:22#

#1.2.2003#

#5. Dezember 99#

Der Datentyp `Boolean` kann genutzt werden, um einer Steuerelementeigenschaft Werte zuzuweisen, die nur `True` oder `False` sein können, beispielsweise `Enabled`.

Der Datentyp `Variant` kann beliebige Daten aufnehmen, außer Strings fester Länge. Dieser Datentyp wird für verschiedene Zwecke eingesetzt, insbesondere, wenn ein Datenspeicherbereich verschiedene Datentypen aufnehmen soll. Ein varianter Speicherbereich kann als temporärer Speicher für beliebige Datentypen dienen, die Sie später in einem spezifischeren Datenbereich ablegen werden.

Die Arbeit mit Variablen

Variablen können unterschiedliche Werte enthalten. Diese Werte können sich ändern, weil eine Variable nichts anderes ist als ein Speicherbereich - ähnlich einer Schachtel, die immer jeweils einen Gegenstand enthalten kann. Wenn Sie einen anderen Wert in einer Variablen ablegen, wird der alte Wert überschrieben. Der Konstantenwert 54 ändert sich nie, aber wenn Sie 54 innerhalb einer Variablen ablegen, enthält diese den Wert 54 nur so lange, bis dort ein anderer Wert abgelegt wird.



Eine *Variable* ist ein temporärer Speicherbereich in Ihrem Programm, der über einen Namen angesprochen wird, und in dem Sie Daten ablegen. Variablen können Zwischenberechnungen aufnehmen, oder Werte, die Sie Steuerelementen auf dem Formular zuweisen oder daraus lesen.

Sie sind für die Namenszuweisung aller Variablen in Ihrem Code verantwortlich. Zwei verschiedene Variablen können innerhalb einer Prozedur nicht denselben Namen haben, weil Visual Basic sie nicht unterscheiden kann. Anders als Steuerelementeigenschaften, die bereits einen Namen haben, gilt dies für Variablen nicht, bis Sie ihnen einen Namen zuordnen. Damit Sie eine Variable nutzen können, müssen Sie sie deklarieren, indem Sie Visual Basic ihren Namen und ihren Datentyp mitteilen. Variablen können nur Daten des Datentyps aufnehmen, für den sie deklariert ist. Eine als `Byte` deklarierte Variable kann keinen `String` aufnehmen. (Die Ausnahme zu dieser Regel ist eine Variable, die mit dem Datentyp `Variant` deklariert ist.)

Variablendeklarationen

Die `Dim`-Anweisung deklariert Variablen, indem sie ihnen einen Namen und einen Datentyp zuordnet. Damit Sie eine Variable nutzen können, müssen Sie sie mit `Dim` deklarieren. Sie können diese Regel locker handhaben, aber damit entsteht leicht eine schlampige Programmierung, die zu Fehlern führen kann. Rufen Sie das Dialogfeld **Extras, Optionen** auf. Auf der Registerkarte **Editor** markieren Sie die Option **Variablendeklaration erforderlich**. Auf diese Weise stellen Sie sicher, daß Visual Basic die Deklaration erzwingt. Der Deklarationsbereich Ihres Codes beinhaltet außerdem standardmäßig die Anweisung:

`Option Explicit`

Diese Anweisung teilt Visual Basic mit, daß der restliche Code in diesem Modul alle Variablen deklarieren muß, bevor sie verwendet werden können. Falls Sie dann irgendwo in Ihrem Programm einen Variablennamen falsch schreiben, erkennt Visual Basic diesen Fehler. Wenn Sie vor der Verwendung einer Variablen keine explizite Deklaration erzwingen, behandelt Visual Basic die falsch geschriebene Variable als nicht initialisierte Variable und verwendet möglicherweise einen fehlerhaften Wert, wenn diese Variable innerhalb von Berechnungen auftaucht.



Wenn Sie keine explizite Variablendeklaration erzwingen, setzt Visual Basic voraus, daß eine nicht deklarierte Variable den Datentyp `Variant` hat.

Hier das Format der `Dim`-Anweisung, mit der Sie Variablen deklarieren:

`Dim VarName As Datentyp`

`VarName` ist der Name, den Sie der Variablen zuweisen, und `Datentyp` ist einer der in den Tabellen 5.1 und 5.3 aufgelisteten Datentypen. Wenn Sie Variablen innerhalb einer Prozedur deklarieren (Ereignisprozedur oder nicht), deklarieren Sie sie unmittelbar hinter der ersten Hüllzeile. Die Variable steht dann innerhalb der gesamten Prozedur und nur in dieser Prozedur zur Verfügung. Kein anderes Programm kennt die Variable, sie ist also *lokal* für die Prozedur. Wenn Sie eine Variable im Deklarationsbereich eines Moduls deklarieren, hat jede Prozedur in diesem Modul Zugriff darauf. Die Variable ist dann global für das Modul, aber kein anderes Modul der Anwendung hat Zugriff darauf. Sie können Variablen auch global für ein gesamtes Projekt machen, aber je lokaler Sie Ihre Variablen machen, desto weniger besteht die Gefahr, daß Sie dieselbe Variable für unterschiedliche Zwecke verwenden.



Zwei Variablen können denselben Namen haben und doch unterschiedliche Variablen darstellen, so lange sie lokal innerhalb verschiedener Prozeduren deklariert sind. Darüber hinaus können zwei Prozeduren eine Variable, die lokal für eine davon ist, gemeinsam nutzen. Wie lokale Variablen gemeinsam genutzt werden, erfahren Sie in Kapitel 8.

Für die Namensgebung von Variablen sollten Sie einige Regeln beachten:

- Alle Variablennamen sollten mit einem Buchstaben beginnen.
- Innerhalb des Namens dürfen Buchstaben und Ziffern verwendet werden.
- Der Variablenname darf zwischen 1 und 255 Zeichen lang sein (verwenden Sie also keine Namen, die 256 Zeichen lang sind).
- Innerhalb des Namens sollten nur bestimmte Sonderzeichen verwendet werden. Am besten verwenden Sie in einem Variablennamen nur den Unterstrich (`_`). Wenn Sie sich auf Buchstaben, Ziffern und den Unterstrich begrenzen, müssen Sie sich nicht daran erinnern, welche Sonderzeichen nicht erlaubt sind. Insbesondere sollten Sie innerhalb eines Variablennamens keine Leerzeichen verwenden.

Die folgenden Regeln sind nicht zwingend, sollten aber bei der Namensgebung von Variablen befolgt werden:

- Stellen Sie den Variablennamen ein Präfix voraus, das den Datentyp beschreibt. Auf diese Weise müssen Sie nicht im Deklarationsbereich eines umfangreichen Programms nachlesen, um den Datentyp einer Variablen herauszufinden, und es ist weniger wahrscheinlich, daß Sie einen falschen Datentyp in einer Variablen ablegen. Tabelle 5.4 listet einige gebräuchliche

Präfixe für Variablennamen auf.



Es ist möglich, bestimmte Datentypen in Variablen abzulegen, die für einen anderen Typ deklariert wurden, falls diese Datentypen in Typ und Größe kompatibel sind. Beispielsweise können Sie einen **Byte**-Datenwert in einer **Integer**-Variablen ablegen, weil der Datentyp **Integer** einen größeren Bereich ganzer Zahlen unterstützt als der Datentyp **Byte**.

- Verwenden Sie sinnvolle Namen, also beispielsweise `curHighTotal` anstelle von mehrdeutigen Dingen wie etwa `a` oder `curX1`. Die Namen helfen, den Code zu dokumentieren.
- Verwenden Sie eine Kombination aus Groß- und Kleinbuchstaben, um die verschiedenen Teile des Variablennamens voneinander abzutrennen. (In diesem Buch werden für Variablennamen nur Groß- und Kleinbuchstaben verwendet. Viele Programmierer bevorzugen den Unterstrich, um Teile des Namen voneinander abzutrennen, beispielsweise `curHigh_Sales`.) Eine Kombination aus Groß- und Kleinbuchstaben wird auch als *Kamelnotation* bezeichnet, wegen der Höcker: `einKamelHöcker`.

Tabelle 5.4: Präfixe für Variablennamen, die den Datentyp einer Variablen beschreiben

Präfix	Datentyp	Beispiel
bln	Boolean	blnButtonEnabled
byt	Byte	bytLength
cur	Currency	curSales98
dte	Date	dteOverdue
dbl	Double	dblScientificAmt
int	Integer	intYear1998
lng	Long	lngWeatherDistance
obj	Object	objWorksheetAcct99
sng	Single	sngSales1stQte
str	String	strFirstName
vnt	Variant	vntValue

Hier einige mögliche Variablendeklarationen mit `Dim`:

```
Dim intTotal As Integer
Dim curSales99 As Currency
Dim dteFinal As Date
Dim strName As String
Dim blnIsChecked As Boolean
```

Geben Sie Variablen mit dem Datentyp `Boolean` Namen, die einer Ja/Nein-Frage entsprechen, beispielsweise `blnIsChecked`.

Innerhalb einer **Dim**-Anweisung können mehrere Variablendeklarationen kombiniert werden, getrennt durch Kommata, aber Sie müssen für jede Variable die Klausel **AS *DatenTyp*** angeben, beispielsweise wie folgt:

```
Dim intTotal As Integer, curSales99 As Currency
```

Wenn Sie die Klausel **AS *DatenTyp*** nicht angeben, deklariert Visual Basic die Variable als **Variant**. Die folgenden Anweisungen sind also äquivalent:

```
Dim vntControlVal As Variant  
Dim vntControlVal
```



Wenn Sie **Variant**-Variablen deklarieren, sollten Sie immer **AS Variant** angeben, um damit zu verdeutlichen, was Sie mit der Variablen vorhatten.

String-Deklaration

Die Deklaration von Strings wirft ein zusätzliches Problem auf, weil der Datentyp **String** für Strings variabler oder fester Länge verwendet werden kann. Der gebräuchlichste String-Datentyp ist der String mit variabler Länge; die Deklaration solcher Stringvariablen ist am einfachsten, weil dafür nur die **Dim**-Anweisung angegeben wird, so wie für andere Datentypen. Die folgenden Anweisungen deklarieren zwei Stringvariablen variabler Länge:

```
Dim strCityName As String  
Dim strStateName As String
```

Sowohl **strCityName** als auch **strStateName** können Strings beliebiger Länge aufnehmen. Wenn Sie in **strCityName** zunächst »Indianapolis« ablegen, können Sie später "Tulsa" im selben Namen unterbringen - die Variable paßt sich der neuen Stringlänge an. Größtenteils werden Sie mit Strings variabler Länge arbeiten; dieses Buch beschäftigt sich weniger mit Variablen fester Länge, es sei denn, die Länge spielt eine besondere Rolle, wie beispielsweise bei der Arbeit mit Dateien, die auf Diskette geschrieben werden sollen. Darüber hinaus sollten Sie die Anzahl der Zeichen für den Inhalt eines Bezeichnungsfelds oder eines anderen Steuerelements einschränken, indem Sie nur Strings fester Länge dafür erlauben.



Die Anführungszeichen werden nicht mit dem String abgelegt, sondern dienen nur dazu, den String in einer Stringkonstanten abzugrenzen.

Hier das Format der **Dim**-Anweisung, mit der Sie Strings fester Länge deklarieren:

```
Dim VarName As String * Länge
```

Die Option *** Länge** teilt Visual Basic mit, daß die **Dim**-Anweisung einen String fester *Länge* deklariert, der nie mehr Zeichen enthalten wird, als in *Länge* angegeben. Die folgende Anweisung deklariert eine Variable, die maximal fünf Zeichen aufnimmt:

```
Dim strZipcode As String * 5
```

Wenn sie versuchen, mehr Zeichen in einer Variablen fester Länge abzulegen, speichert Visual Basic nur so viele Zeichen, wie in der Deklaration angegeben, und verwirft den Rest. Solche Fehler sind häufig schwer zurückzuverfolgen.

Die Verwendung von Variablen

Nachdem Sie eine Variable deklariert haben, können Sie Daten darin ablegen. Dazu verwenden Sie am besten die *Zuweisung*. Die Zuweisung hat das folgende Format:

ElementName = *Ausdruck*

ElementName kann eine deklarierte Variable sein, was in diesem Kapitel auch größtenteils der Fall ist, aber auch ein Eigenschaftswert eines Steuerelements. *Ausdruck* kann eines der folgenden Dinge sein:

- Ein mathematischer Ausdruck
- Eine Konstante
- Eine Variable
- Ein logischer oder String-Ausdruck
- Der Eigenschaftswert eines Steuerelements (Steuerelementeigenschaften sind *Variant*, aber Visual Basic wandelt sie in einen Datentyp um, wenn Sie sie in Variablen ablegen)
- Ein mathematischer, logischer oder String-Ausdruck, der eine Kombination aus Konstanten, Variablen oder Steuerelementeigenschaften darstellt

Das Konzept der Ausdrücke erscheint Ihnen vielleicht jetzt noch verwirrend, aber ein Ausdruck kann fast alles sein, was einen Wert ergibt. Hier einige Beispiele für gültige Zuweisungen:

```
curSales = 5712.75
strFirstName = "Terry"
blnPassedTest = True
blnIsEnabled = lblTitle.Enabled
dblValue = 45.1#
intCount = intNumber
dteOld = #4-1-92#
sngOld97Total = sngNew98Total - 1000.00
```

Anhand der ersten Anweisung können Sie leicht erkennen, wie Anweisungen funktionieren. Der Wert 5712.75 wird in der Variablen *curSales* gespeichert. Sie können einer Konstanten ein Datentyp-Suffix mitgeben, wie in der fünften Zuweisung gezeigt, so daß beide Seiten der Zuweisung denselben Datentyp haben. In diesem Fall ist 45.1 jedoch kleiner als ein *Double*, so daß Visual Basic die Umwandlung automatisch vornimmt, wenn Sie das Suffix weglassen. Variablen mit dem Datentyp *Boolean* können die Werte *True* oder *False* zugewiesen werden, oder ein Eigenschaftswert, der *True* oder *False* ist. Beachten Sie, daß in der letzten Anweisung ein Minuszeichen enthalten ist. Die Schreibweise mathematischer Ausdrücke lernen Sie im nächsten Abschnitt kennen.



Visual Basic unterstützt ein älteres Zuweisungsformat, das mit dem Schlüsselwort *Let* beginnt. Die beiden folgenden Zuweisungen erledigen dasselbe:

```
Let intCount = 1
intCount = 1
```

In den ersten vier Kapiteln haben Sie erfahren, daß Sie die Eigenschaftswerte für Steuerelemente im Visual-Basic-Code setzen können. Dazu verwenden Sie die Zuweisung. Die folgende Anweisung ändert den Inhalt des Bezeichnungsfeldes `lblTitle`:

```
lblTitle.Caption = "Fertig!"
```

Alle Steuerelemente haben Standardeigenschaften. Wenn Sie keinen Eigenschaftsnamen angeben, setzt Visual Basic voraus, daß Sie diese Standardeigenschaft setzen möchten, wenn Sie keinen Namen angeben. Die Standardeigenschaft für Bezeichnungsfelder ist `Caption`, die folgende Anweisung ist also äquivalent mit der ersten:

```
lblTitle = "Fertig!"
```



Obwohl die Zuweisung mit Hilfe der Standardeigenschaft eines Steuerelements weniger Schreibarbeit erforderlich macht, gilt: je ausführlicher Sie Ihren Code formulieren, desto verständlicher ist er, und desto einfacher ist es für andere, ihn zu entziffern. Geben Sie immer die Eigenschaft an, der Sie einen Wert zuweisen, auch wenn das die Standardeigenschaft des Steuerelements ist. Bei der Wartung des Codes ist die Anweisung damit eindeutiger.

Wenn eine Anweisung einem Steuerelement einen Wert zuweist, wird dieses Steuerelement auf dem Formular unmittelbar aktualisiert. Der neue Titel erscheint also sofort auf dem Bildschirm, nachdem diese Zuweisung erfolgt ist.

Operatoren

Visual Basic unterstützt zahlreiche mathematische und String-Operatoren. Tabelle 5.5 listet die gebräuchlichsten Operatoren auf. Diese Operatoren werden in Ausdrücken verwenden, wo Sie Daten berechnen und damit arbeiten.



Operatoren manipulieren Daten, indem sie Ergebnisse kombinieren oder berechnen. Die meisten Operatoren werden durch Symbole dargestellt, andere dagegen sehen mehr wie Visual-Basic-Befehle aus, beispielsweise `Mod`.

Tabelle 5.5: Mit den Operatoren in dieser Tabelle können Sie Rechenoperationen ausführen und Daten manipulieren

Operator	Bedeutung	Beispiel	Ergebnis
<code>^</code>	Exponentiation	<code>2 ^ 3</code>	8
<code>*</code>	Multiplikation	<code>2 * 3</code>	6
<code>/</code>	Division	<code>6 / 2</code>	3
<code>+</code>	Addition	<code>2 + 3</code>	5
<code>-</code>	Subtraktion	<code>6 - 3</code>	3
<code>Mod</code>	Modulo	<code>11 Mod 3</code>	2
<code>\</code>	Ganzzahlige Division	<code>11 \ 3</code>	3

+ oder &	Stringverknüpfung	"Hallo," & " Du!"	"Hallo, Du!"

Bei der Exponentiation wird eine Zahl in eine Potenz erhoben. 2^3 ist also dasselbe wie 2^3 , also $2 \times 2 \times 2$, gleich 8. Sie können auch Bruchwerte in eine Potenz erheben, oder Werte in eine negative Potenz erheben, um eine n-te Wurzel zu berechnen. Die Operatoren für Multiplikation und Division funktionieren so, wie Sie es von ihnen gewöhnt sind. Der Ausdruck $10/3$ ergibt den Annäherungswert 3.3333, $10 * 3$ ergibt 30.

Der Mod-Operator gibt den Rest einer ganzzahligen Division zurück. Auf beiden Seiten von Mod dürfen nur ganzzahlige Werte stehen. Wenn Sie hier einen anderen Datentyp verwenden, versucht Visual Basic, diesen zu konvertieren und auf eine ganze Zahl zu runden, bevor es den Modulo berechnet. Beispielsweise gibt `11 Mod 3` das Ergebnis 2 zurück, weil 11 dividiert durch 3 gleich 3 Rest 2 ist. Der Operator für die ganzzahlige Division, `\` (beachten Sie, daß das der Backslash ist), gibt den ganzzahligen Wert einer Division zurück und verwirft den Rest. `11\3` ist also 3, weil $11/3$ gleich 3 Rest 2 ist. (Der normale Divisionsoperator, $11/3$, würde einen Bruchwert ergeben, also 3.666.)

Das Pluszeichen (+) ist ein *überladener Operator*, weil es zwei verschiedene Operationen darstellen kann, abhängig von den Daten auf beiden Seiten. Wenn Sie auf beiden Seiten des Pluszeichens oder des Ampersand-Zeichens Stringwerte angeben, konkateniert Visual Basic die Strings, verknüpft sie also, und behandelt die konkatenierten Strings wie einen einzigen String. Visual Basic fügt nichts zwischen den konkatenierten Strings ein; wenn Sie also möchten, daß ein Leerzeichen zwischen beiden erscheint, müssen Sie explizit ein Leerzeichen einfügen.



Ein *überladener Operator* ist ein Operator, der mehrere verschiedene Operationen ausführen kann, abhängig davon, in welchem Kontext er eingesetzt wird.



Bei der *Konkatenation* werden zwei oder mehr Strings zu einem einzigen, längeren String verknüpft.

Die folgende Zuweisung konkateniert die Werte aus zwei Bezeichnungsfeldern zu einem einzigen String, der in einer Variablen abgelegt wird. Zwischen den beiden Werten wird ein Leerzeichen eingefügt:

```
strCompleteName = lblFirst.Caption & " " & lblLast.Caption
```



Um eine Verwechslung mit dem Additionsoperator zu vermeiden, sollten Sie zur Konkatenation von Strings nur das Ampersand-Zeichen verwenden.

Die Reihenfolge der Operatoren

Visual Basic führt seine Berechnungen in einer streng definierten Reihenfolge aus, die Sie in Tabelle 5.6 sehen. Als erstes wird in einem Ausdruck die Exponentiation ausgeführt, anschließend die Multiplikation und die Division vor der Addition und der Subtraktion, es sei denn, Klammern überschreiben diese Reihenfolge.

Tabelle 5.6: Visual Basic geht bei der Berechnung von Ausdrücken in einer bestimmten Reihenfolge vor

Reihenfolge	Operatoren	Beispiel	Ergebnis
1	Klammern ()	$(2 + 3) * 7$	35
2	$\wedge$	$2 \wedge 3 + 1$	9
3	*, /, \, Mod	$2 + 3 * 7$	23
4	+, -	$10 - 4 * 2 + 1$	3

Wenn die Standardreihenfolge nicht durch Klammern geändert wird, berechnet Visual Basic immer die Zwischenergebnisse aller Multiplikationen und Divisionen in einem Ausdruck, bevor es die Additionen und die Subtraktionen ausführt. Die Exponentiation hat eine noch höhere Priorität.

Wenn Multiplikation und Division im selben Ausdruck vorkommen, führt Visual Basic sie von links nach rechts aus, es sei denn, diese Reihenfolge wird durch Klammern überschrieben. Der folgende Ausdruck erzeugt also das Ergebnis 15, weil Visual Basic zuerst 10 durch 2 dividiert, 5 erhält und das mit 3 multipliziert, so daß sich 15 ergibt. Auf dieselbe Weise werden auch Additionen und Subtraktionen von links nach rechts ausgeführt, wenn sie in einem Ausdruck vorkommen, ohne daß andere Operatoren oder Klammern ihr Verhalten ändern.

```
10 / 2 * 3
```

Wenn Sie Klammern innerhalb eines anderen Klammerpaars einbetten, berechnet Visual Basic zuerst die inneren Klammern. Im folgenden Ausdruck wird also zuerst (8-3) berechnet:

```
(10 + 2 - (8 - 3)) + 1
```

Zusammenfassung

Heute haben Sie einige vorbereitende Details über den Visual-Basic-Code erfahren. Als erstes haben Sie gelernt, welche Aufgabe der Deklarationsbereich in der Visual- Basic-Anwendung hat. Anschließend haben Sie Datentypen kennengelernt.

Visual Basic unterstützt unterschiedliche Datentypen. Sie müssen nicht nur wissen, wie man zwischen zwei oder mehr Datentypen unterscheidet, sondern auch, wie man die verschiedenen Datentypen deklariert, die man in einem Programm braucht. Variablen werden bei der Programmausführung als Zwischenspeicher verwendet, aber bevor Sie sie nutzen können, müssen Sie sie richtig deklarieren und ihnen einen Namen zuweisen. Nachdem Sie eine Variable deklariert haben, können Sie die mathematischen Operatoren von Visual Basic einsetzen, um Ergebnisse zu berechnen - und diese dann in den definierten Variablen abzulegen.

In Kapitel 6 gehen Sie einen Schritt weiter und lernen Operatoren kennen, mit denen Sie Daten vergleichen. Anschließend zeigen wir Ihnen neue Programmieranweisungen und Steuerelemente, die mit diesen Operatoren arbeiten.

Fragen & Antworten

Frage:

Warum berechnet Visual Basic nicht alle Operatoren von links nach rechts?

Antwort:

Visual Basic ist konform zur standardmäßigen und historisch untermauerten Hierarchie algebraischer Operatoren. Ärgern Sie sich also nicht über Visual Basic, sondern über die Mathematiker! Spaß beiseite. Durch die Reihenfolge der Operatoren werden Mehrdeutigkeiten in Ihrem Code verhindert, indem eine Reihenfolge bestimmt wird, die immer eingehalten wird. Sie müssen diese Reihenfolge nicht zwingend beibehalten, weil Sie sie durch die Verwendung von Klammern ändern können. Klammern dürfen auch dort gesetzt werden, wo sie nicht nötig sind. Die folgende Zuweisung würde dasselbe Ergebnis speichern, egal ob die Klammern gesetzt sind oder nicht, aber die Klammern verdeutlichen den Ausdruck, weil sie jede mögliche Mehrdeutigkeit eliminieren:

```
intValue = (8 * ) + intResult
```

Frage:

Warum sind lokale Variablen besser als globale Variablen?

Antwort:

Wenn Sie mehr über die Visual-Basic-Programmierung wissen, werden Sie besser verstehen, was der Unterschied zwischen lokalen und globalen Variablen bedeutet. Als Faustregel sollten Sie sich merken, daß lokal immer besser als global ist - mit ein paar Ausnahmen. Im allgemeinen sollte eine Prozedur immer nur das wissen, was sie unbedingt braucht. Sie sollte nur auf die Variablen Zugriff haben, die sie für ihre Aufgabe benötigt, und auf sonst nichts. Dadurch können schwer erkennbare Fehler vermieden werden, die bei globalen Variablen manchmal auftreten.

Vielleicht haben Sie schon erkannt, daß Steuerelemente für das gesamte Projekt global sind. Es muß möglich sein, daß alle Prozeduren den Eigenschaftswert eines Steuerelements verändern, so daß die Steuerelemente eines Formulars dem gesamten Code in einem Projekt zur Verfügung stehen.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welcher Code steht im Deklarationsbereich eines Programms?
2. Was können Sie tun, wenn zwei oder mehr Prozeduren auf die lokale Variable einer anderen Prozedur zugreifen müssen?
3. Richtig/Falsch. Der Wert einer Konstanten ändert sich nie.
4. Richtig/Falsch. Der Wert einer Variablen ändert sich nie.

5. Warum unterstützt Visual Basic zwei verschiedene Divisionsoperatoren?
6. Was ist ein überladener Operator?
7. Welcher Operator wird für die Konkatination von Stringausdrücken am besten verwendet?
8. Welcher Datentyp kann jeden anderen Datentyp aufnehmen?
9. Richtig/Falsch. Die Variablenpräfixe sind in Variablennamen zwingend erforderlich.
10. Wie können Sie sicherstellen, daß Visual Basic keine nicht deklarierten Variablen erlaubt (nennen Sie zwei Methoden)?

Übungen

1. Was macht Visual Basic mit der folgenden Variablendeklaration?

```
Dim intCnt As Integer, abc, curSales As Currency
```

2. **Fehlersuche:** Sally hat Probleme, mit dem folgenden Ausdruck einen richtigen Durchschnittswert zu berechnen. Können Sie ihr helfen?

```
sngAvg = sngGrade1 + sngGrade2 + sngGrade3 / 3
```

3. Welche Ergebnisse erzeugen die folgenden Formeln?

- a. $1 + 2 * 4 / 2$
- b. $(1 + 2) * 4 / 2$
- c. $1 + 2 * (4 / 2)$
- d. $9 \setminus 2 + 1$
- e. $(1 + (10 - (2 + 2)))$

4. Schreiben Sie Zuweisungen, die die folgenden Formeln in ihr Visual-Basic-Äquivalent umwandeln:

```
3 + 3
a = -----
4 + 4
```

```
b.
x = (a - b) * (a - 2)2
```

```
c. a1/2
f = ----
b1/2
```

5. Das Programm aus dem ersten Bonusprojekt enthielt die folgende Prozedur:

```
1: Private Sub cmdTest_Click()
2: ' Diese Ereignisprozedur wird ausgeführt, wenn
3: ' der Benutzer das eingegebene Kennwort testen will
4: If txtPassword.Text = "Sams" Then
5: ' Erfolg! Das Kennwort stimmt überein
6: Beep
7: Beep ' Bild anzeigen
8: imgPassword.Picture = LoadPicture("C:\Program Files\" _
9: & "Microsoft Visual Studio\Common\Graphics\MetaFile\" _
```



```
9: & "Business\coins.wmf")
  lblPrompt.Caption = "Zeig mir das Geld!"
10: Else
11:   lblPrompt.Caption = "Falsches Kennwort - Erneute Eingabe"
12:   txtPassword.Text = " ' Altes Kennwort löschen
13:   txtPassword.SetFocus ' Fokus auf das Textfeld
14: End If
15: End Sub
```

Anhand dieser Prozedur erkennen Sie, wie die Zuweisungen erfolgen. Verstehen Sie jetzt, warum lange Anweisungen, die mit einem Unterstrich fortgesetzt werden, wie beispielsweise die Zeilen 7, 8 und 9, ein Ampersand-Zeichen umfassen müssen, wenn Sie String-Konstanten zerlegen?

❖Kapitel	Inhalt	Index	❖Top	Kapitel❖

Woche 1

Bonusprojekt 2: Variablen und Ausdrücke

Der Code in diesem Bonusprojekt demonstriert die Verwendung von Variablendeklarationen, Zuweisungen und Ausdrücken. Nachdem Sie nun wissen, wie man ein Formular entwickelt und Steuerelemente anlegt, beschäftigen Sie sich mit dem Code. Sie sollen lernen, wie man die Steuerelemente einer Anwendung mit Hilfe von Programmanweisungen aktiviert. Ein sinnvoller Ausgangspunkt sind die Daten.

Der einfache Code in Listing BP2.1 wendet die Konzepte an, die Sie in Kapitel 5 kennengelernt haben. Hier wird kein visuelles Formularfenster beschrieben, Sie können sich also ganz auf den Code konzentrieren. Wenn Sie ein Formularfenster erstellen möchten, um den Code zu testen, legen Sie ein Formular mit den drei Bezeichnungsfeldern an `lblGrossPay`, `lblTaxes` und `lblNetPay`. Verwenden Sie die Befehlsschaltfläche `cmdCalcPay`, um den Code auszulösen. Sie brauchen dazu nur ein einfaches Formular, wie beispielsweise das in Abbildung BP2.1 gezeigte. Nach ein paar einfachen Berechnungen schreibt der Code die Ergebnisse in die drei Bezeichnungsfelder.

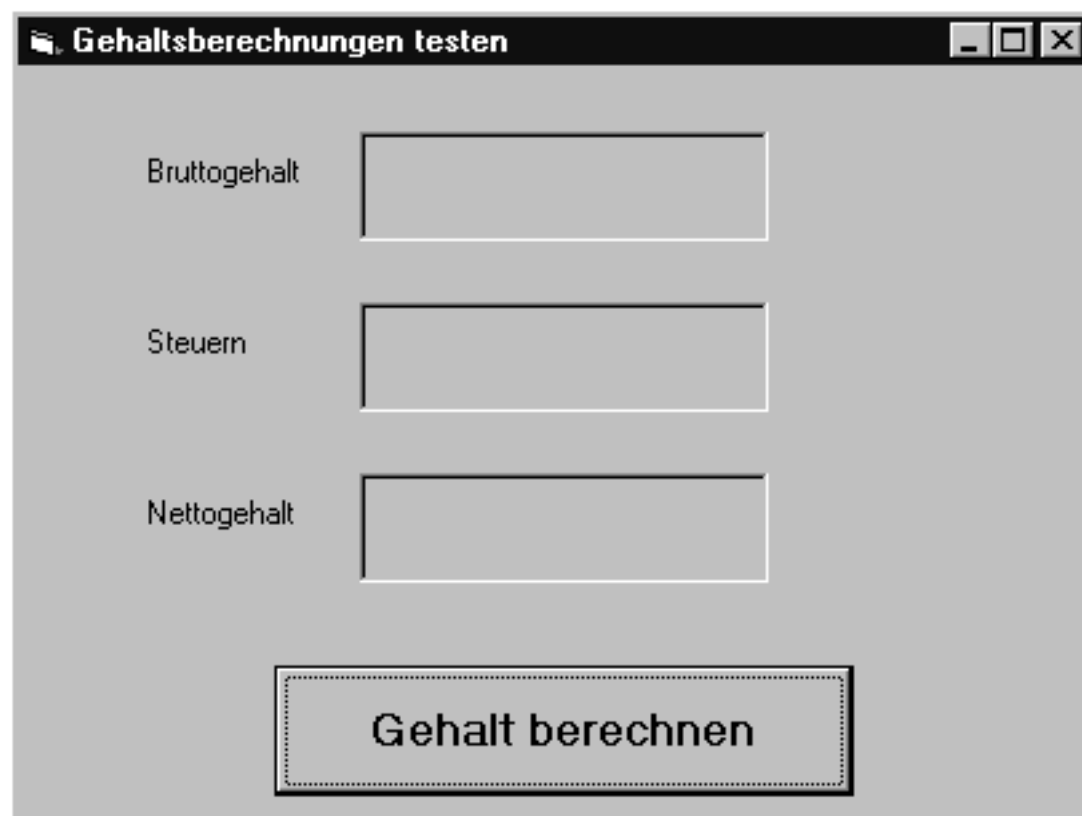


Abbildung BP2.1:
Diesen Code können Sie in einem ganz einfachen Formular testen.

Listing BP2.1: Dieser Code demonstriert die Verwendung von Variablen und Zuweisungen

```
1: Private Sub cmdCalcPay_Click()  
2: ' Drei Gehaltsvariablen berechnen  
3: Dim intHoursWorked As Integer  
4: Dim sngRate As Single, sngTaxRate As Single  
5: Dim curTaxes As Currency, curGrossPay As Currency  
6: Dim curNetPay As Currency  
7:  
8: ' Variablen initialisieren  
9: ' (Normalerweise kämen solche Daten vom Benutzer  
10: ' oder aus einer Datei)  
11: intHoursWorked = 40 ' Gesamtstundenzahl  
12: sngRate = 7.8 ' Stundenlohn  
13: sngTaxRate = 0.42 ' Steuersatz  
14:  
15: ' Beträge berechnen  
16: curGrossPay = intHoursWorked * sngRate  
17: curTaxes = sngTaxRate * curGrossPay  
18: curNetPay = curGrossPay - curTaxes  
19:  
20: ' Ergebnisse in den zugehörigen Bezeichnungsfeldern anzeigen  
21: lblGrossPay.Caption = curGrossPay  
22: lblTaxes.Caption = curTaxes  
23: lblNetPay.Caption = curNetPay  
24: End Sub
```

Analyse

Die Zeilen 1 und 24 sind die Hüllzeilen für die Ereignisprozedur der Befehlsschaltfläche. Zeilen wie 2 und 8 helfen, den Code zu dokumentieren, um so die Wartung zu vereinfachen. Die Zeilen 3 bis 6 deklarieren mehrere Variablen. Drei der Variablen haben den Datentyp Currency.

Die Zeilen 11, 12 und 13 weisen den drei Variablen Datenwerte zu. Normalerweise kommen die Gehaltsdaten vom Benutzer, der sie über die Tastatur eingibt, oder aus einer Datei. Weil Sie jedoch noch nicht wissen, wie man mit Eingaben von der Tastatur oder aus Dateien umgeht, wurden für dieses kurze Beispiel Zuweisungen verwendet. Beachten Sie, daß bei der Zuweisung einer Konstanten an eine Integer-Variable (Zeile 11) kein Dezimalpunkt verwendet wird. In den beiden Werten, die in den Zeilen 12 und 13 den beiden Variablen mit einfacher Genauigkeit Werte zuweisen, werden dagegen sehr wohl Dezimalpunkte verwendet.

Die Zeilen 16, 17 und 18 berechnen das Gehalt. Die Reihenfolge der Operatoren spielt in diesen Ausdrücken keine Rolle, weil sie so kurz sind. Sie sehen, daß die Datentypen in diesen Ausdrücken gemischt verwendet werden, was möglich ist, weil die Daten miteinander kompatibel sind.

Die Zeilen 21, 22 und 23 schließlich weisen den Bezeichnungsfeldern Variablenwerte zu. Sobald die Zuweisungen erfolgt sind, aktualisiert das betreffende Formular die Bezeichnungsfelder mit den berechneten und zugewiesenen Werten.



Eine Zuweisung kopiert einen Ausdruck auf der rechten Seite des Gleichheitszeichens in den Platzhalter (beispielsweise eine Variable oder ein Steuerelement) auf der rechten Seite des Gleichheitszeichens. Dabei handelt es sich nicht um eine Verschiebeoperation. Beispielsweise enthält die Variable `curGrossPay` nach Ausführung von Zeile 21 ihren Wert weiterhin, der jetzt auch im Bezeichnungsfeld angezeigt wird.

Woche 1

Tag 6

Programme steuern

In diesem Kapitel geht es wieder um Visual-Basic-Operatoren - die diesmal jedoch keine mathematischen Ergebnisse berechnen. Heute lernen Sie Vergleichs- und logische Operatoren, ebenso wie einige Steueranweisungen kennen, mit denen Sie Programme so schreiben können, daß bestimmte Codeabschnitte beliebig oft ausgeführt und verschiedene Bedingungen überprüft werden.

Im Überblick:

- Vergleichsoperatoren
- Logische Operatoren und ihre Kombination mit Vergleichsoperatoren
- Die If-Anweisung
- Schleifen
- Die vier Formate der Do-Schleife
- For- und Do-Schleifen im Vergleich

Vergleichsoperatoren

Angenommen, Sie schreiben eine Anwendung zur Verwaltung von Eingangsrechnungen. Die Anwendung berechnet, wie viel Geld Sie den Händlern schulden, und druckt die Schecks automatisch aus. Was passiert, wenn Sie bei einem der Hersteller seit dem letzten Zahlungslauf nichts gekauft haben? Soll das Programm dann einen Scheck über 0,00 DM ausstellen? Sicher nicht. Bisher hat der gesamte Programmcode innerhalb einer Prozedur eine Anweisung nach der anderen ausgeführt. Mit Hilfe der Vergleichsoperatoren und entsprechender Anweisungen, die Sie heute kennenlernen, können Sie ein Programm so schreiben, daß es seine Ausführungsreihenfolge ändert, wenn die Daten das erforderlich machen. Es ist also möglich, die Anwendung so zu schreiben, daß nur Schecks für Händler ausgestellt werden, für die Zahlungen anfallen.

Tabelle 6.1 listet mehrere neue Visual-Basic-Operatoren auf. Keiner dieser Operatoren führt mathematische Berechnungen aus, wie die, die Sie in Kapitel 5 kennengelernt haben. Statt dessen vergleichen diese *bedingten Operatoren* Daten. Sie machen Ihre Visual-Basic-Programme irgendwie intelligenter. Durch den Vergleich der Daten und die Analyse der Ergebnisse kann Ihr Visual-Basic-Programm entscheiden, welche Aktionen es ausführt. In Programmen mit bedingten Operatoren und Anweisungen überlassen Sie es Visual Basic, zur Laufzeit zu entscheiden, welche Anweisungen ausgeführt werden sollen.



Mit den *bedingten Operatoren* wird ein Visual-Basic-Datenwert mit einem anderen verglichen. Die bedingten Operatoren stellen fest, ob ein Wert größer, gleich oder kleiner als ein anderer Wert ist.

Tabelle 6.1: Visual Basic unterstützt sechs bedingte Operatoren

Operator	Beschreibung	Beispiel	Ergebnis
=	Gleich	7 = 2	False
>	Größer als	6 > 3	True
<	Kleiner als	5 < 11	True
>=	Größer oder gleich	23 >= 23	True
<=	Kleiner oder gleich	4 <= 21	True
<>	Ungleich	3 <> 3	False

Beachten Sie die Ergebnisspalte in Tabelle 6.1. Was ist das Ergebnis von $6 > 3$? Ist 6 größer als 3? Ja. Das Ergebnis dieses bedingten Ausdrucks ist also **True**. Wie Sie aus Kapitel 5 wissen, unterstützt Visual Basic den Datentyp **Boolean**, der die Werte **True** und **False** annehmen kann. Die Programmiersprache Visual Basic unterstützt die Schlüsselworte **True** und **False**, die Sie im Code verwenden können, um Booleschen Variablen und Steuerelementen Werte zuzuweisen.



In Tabelle 6.1 sehen Sie, daß es neben dem gestern gezeigten Pluszeichen noch einen überladenen Operator gibt. Das Gleichheitszeichen wird in Zuweisungen verwendet, um Variablen und Steuerelementen Ausdrücke hinzuzufügen. Außerdem wird es verwendet, um auf Gleichheit zu überprüfen. Visual Basic unterscheidet anhand des Kontexts für das Gleichheitszeichen zwischen den beiden Operatoren.

Bevor Sie diese Operatoren im Visual-Basic-Code einsetzen, sollten Sie wirklich verstanden haben, wie sie funktionieren. Der Ausdruck $23 \geq 23$ ergibt **True**, weil 23 größer *oder gleich* 23 ist. Sehen Sie sich in Tabelle 6.1 die Ergebnisspalte an, um die Vergleiche nachzuvollziehen.

Links und rechts von einem bedingten Operator können nicht nur Konstanten stehen. Sie können dort Konstanten, Variablen, Steuerelemente und Kombinationen davon verwenden. Visual Basic arbeitet mit unterschiedlichen Datenarten, und Ihre Programme müssen diese Daten vergleichen, um festzulegen, welcher Code am besten ausgeführt werden soll.

Eine Besonderheit tritt auf, wenn ein Wert auf einer Seite des bedingten Operators den **Null**-Wert enthält. In diesem Fall gibt Visual Basic als Ergebnis weder **True** noch **False** zurück, sondern **Null**. Sie müssen nach dem **Null**-Wert suchen, wenn Sie vermuten, daß er in einem der Werte stehen könnte, die Sie vergleichen wollen. In solchen Fällen suchen Sie nach drei möglichen Ergebnissen: **True**, **False** oder **Null**. Weil solche Ergebnisse Verwirrung stiften können, beinhaltet Visual Basic sogenannte interne Funktionen, die sind Werkzeuge, die Ihnen helfen, Null-Werte zu erkennen. Mehr darüber erfahren Sie in Kapitel 8. Darüber hinaus setzen die bedingten Operatoren voraus, daß der Wert **Empty** (d.h. die Komponente muß noch mit einem Wert initialisiert werden, wie Sie in Kapitel 5 erfahren haben) beim Vergleich von Strings dasselbe wie **Null** oder der **Nullstring** ist.

Mit bedingten Operatoren vergleichen Sie Strings genau wie numerische Werte. Im allgemeinen gelten dafür die folgenden Faustregeln:

- Großbuchstaben sind »kleiner« als Kleinbuchstaben, »ST. JOHN« liegt also vor »St. John«.
- Buchstaben sind gemäß ihrer alphabetischen Reihenfolge angeordnet, d.h. »A« ist kleiner als »B« und der Name »Walter« ist größer als (steht vor) »William«.
- Zahlen sind kleiner als Buchstaben, »3« ist also kleiner als »Drei«.

Wenn Ihnen diese allgemeinen Regeln verwirrend erscheinen, dann gehen Sie einfach davon aus, daß Visual Basic die meisten Stringdaten so sortiert, wie sie auch in Ihrem Telefonbuch sortiert sind. Durch den Vergleich von Stringdaten ist es möglich, daß Ihr Programm Namen alphabetisch anordnet, Kennwörter überprüft und Informationen analysiert.



Visual Basic unterstützt eine spezielle Anweisung im Deklarationsbereich eines Moduls:
Option Compare Text

Wenn Sie im Deklarationsbereich eines Moduls diese Anweisung angeben, vielleicht zusammen mit **Option Explicit**, die Sie in Kapitel 5 kennengelernt haben, werden Groß- und Kleinbuchstaben als gleichwertig betrachtet. Wenn Sie möchten, daß bei einem Vergleich die Groß-/Kleinschreibung nicht berücksichtigt wird, geben Sie **Option Compare Text** in Ihrem Modul an. Normalerweise ist es jedoch nicht sinnvoll, Groß- und Kleinbuchstaben gleich zu behandeln, weil man damit keine korrekte alphabetische Reihenfolge erzielen kann.



Visual Basic folgt für den Vergleich von Strings der ASCII-Reihenfolge, es sei denn, die **Option Compare Text** wird im Modul angegeben.



Eine *ASCII-Tabelle* listet die 256 auf dem PC verfügbaren Zeichen mit ihrer entsprechenden eindeutigen Nummer auf. Der Buchstabe A hat den ASCII-Wert 65, B hat 66 usw. Wenn Sie im Hilfe-Index von Visual Basic nach ASCII-Codes suchen, erscheint die ASCII-Tabelle auf dem Bildschirm, wie in Abbildung 6.1 gezeigt. Neben der Online-Hilfe, die immer zur Verfügung steht, finden Sie auch in Anhang C eine ASCII-Tabelle.

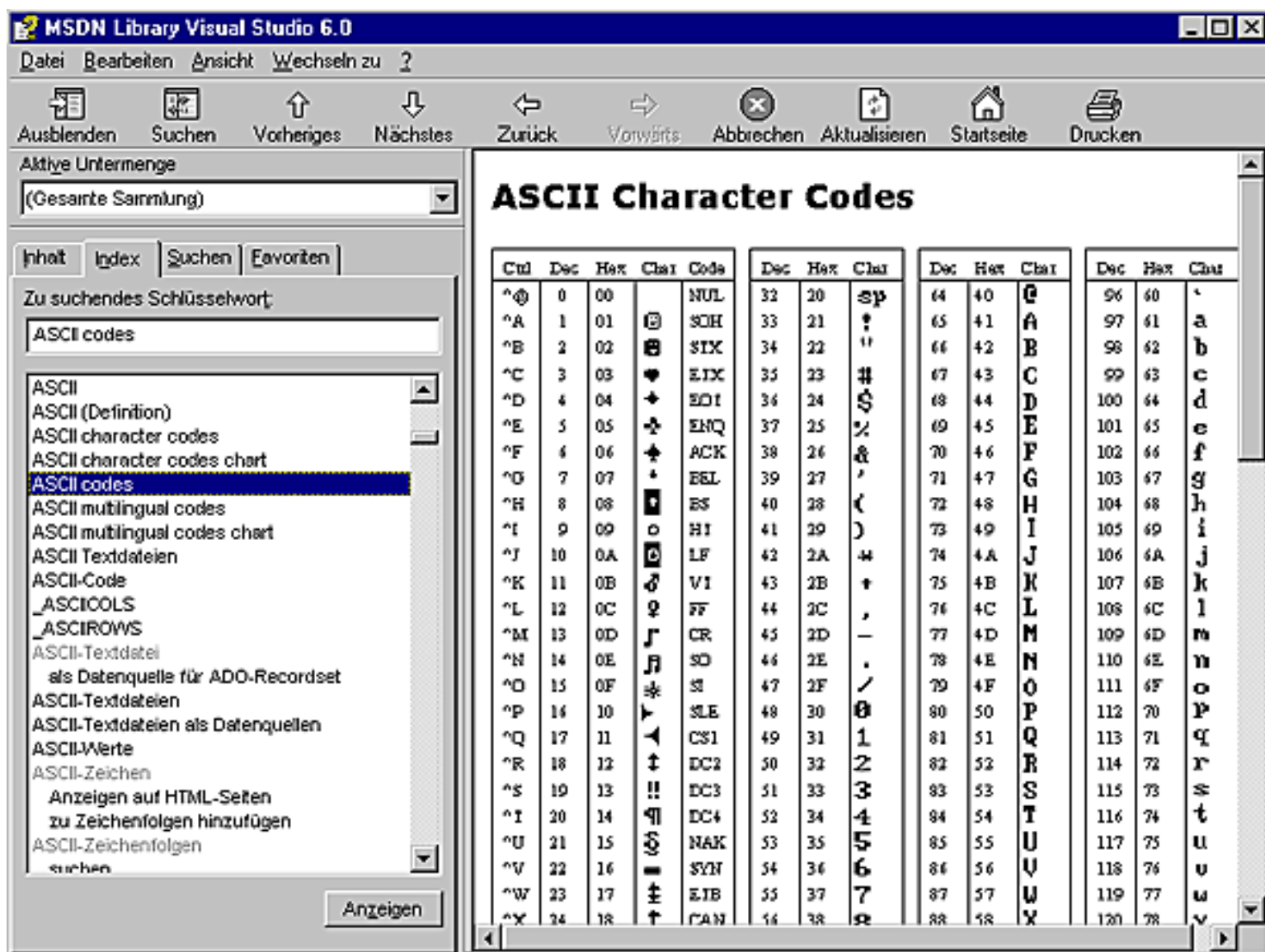


Abbildung 6.1:
Visual Basic verwendet für den Vergleich von Stringdaten die ASCII-Reihenfolge.

Betrachten Sie die folgenden Stringvergleiche:

- "abcdef" > "ABCDEF"
- "Ja!" < "Ja?"
- "Computer machen Spaß!" = "Computer machen Spaß!"
- "PC" <> "pc"
- "Bücher, Bücher, Bücher" >= "Bücher, Bücher"

Alle hier angezeigten Stringvergleiche haben das Ergebnis True.

Visual Basic unterstützt noch einen weiteren bedingten Operator, Like, der Werte basierend auf Wildcard-Übereinstimmungen vergleicht. Vielleicht haben Sie bei der Arbeit mit den Dateien die Wildcard-Zeichen * und ? bereits benutzt. * steht für null oder mehr Zeichen, ? steht für genau ein Zeichen. Darüber hinaus verwendet Like ein drittes Wildcard-Zeichen, #, das eine numerische Ziffer darstellt. Einige Beispiele sollten Like schnell erklären. Betrachten Sie die folgenden bedingten Ausdrücke, die Like verwenden und den Wert True zurückgeben:

- "Sams Publishing" Like "Sa*"
- "Qtr???" Like "QtrOne"
- "Data##" Like "Data98"
- "X" Like "[XYZ]"

Das letzte Beispiel zeigt eine spezielle Form eines `Like`-Ausdrucks. Wenn das Zeichen einem Zeichen innerhalb der Klammern entspricht, wird `True` zurückgegeben. Die Klammern stellen eine Methode dar, nur auf ein paar Zeichen zu vergleichen. Alle folgenden Vergleiche ergeben `True`, wenn Sie `Like` auf einen Stringwert von »`Code[12345]Red`« anwenden: "`Code1Red`", "`Code2Red`", "`Code3Red`", "`Code4Red`" und "`Code5Red`".



In der Praxis vergleichen Sie Variablen und Steuerelemente; die darin enthaltenen Daten können sich während der Programmausführung ändern. Diese Beispiele vergleichen nur Konstanten mit Konstanten, um zu demonstrieren, wie der Operator funktioniert.

Der bedingte Operator Gleich (=) ergibt für diese Ausdrücke `False`, weil = keine Wildcard-Zeichen erkennt.

Bedingte Daten

Vergleichen Sie Werte immer mit kompatiblen Datentypen. Sie können beispielsweise eine Zahl eines beliebigen numerischen Datentyps mit einer anderen Zahl vergleichen, um festzustellen, welche davon größer ist. Sie können Strings miteinander und Boolesche Werte miteinander vergleichen. Versuchen Sie jedoch nicht, einen String und eine Zahl zu vergleichen, weil die Ergebnisse normalerweise falsch sind.



Strings, Boolesche Werte, Währungsdaten, Datumswerte, Zeitwerte und die Integer-Datentypen (*Byte*, *Integer* und *Long*) können auf Gleichheit miteinander verglichen werden. Versuchen Sie jedoch nicht, zwei Werte mit einfacher oder mit doppelter Genauigkeit zu vergleichen. Aufgrund der Methode, wie Visual Basic Daten mit einer bestimmten Genauigkeit speichert, erscheinen zwei Werte mit einfacher Genauigkeit möglicherweise ungleich, weil Visual Basic intern eine Rundung für solche Werte vornimmt. Wenn Sie zwei Variablen mit gleicher Genauigkeit vergleichen möchten, subtrahieren Sie sie und vergleichen die Differenz, um zu prüfen, ob sie annähernd gleich sind. Diese Codierung ist mühsam, vermeiden Sie sie also, wo immer Sie können.

Visual Basic behandelt den `Variant`-Datentyp wohlwollend, wenn Sie einen bedingten Vergleich ausführen, weil Sie häufig die Werte eines Steuerelements, beispielsweise den `Text`-Wert eines Textfeldes, mit einer Konstanten oder einer Variablen vergleichen. Steuerelementeigenschaften werden normalerweise so verglichen, als handelte es sich um `Variant`-Daten. Wenn die `Variant`-Variable oder das Steuerelement Zeichen enthalten, die einer Zahl entsprechen, beispielsweise 234.56, und Sie diesen `Variant`-Wert mit einer numerischen Variablen vergleichen, nimmt Visual Basic einen numerischen Vergleich vor, indem es den `Variant`-Wert vorübergehend in eine Zahl umwandelt.

Wenn Sie dagegen eine `Variant`-Variable oder ein `Variant`-Steuerelement mit einem String vergleichen, wandelt Visual Basic diesen `Variant` in einen String um und führt einen zeichenweisen, ASCII-basierten Stringvergleich durch. Visual Basic übernimmt also einige müßige Details für Sie, die andernfalls Probleme beim Vergleich mit einem `Variant`-Datentyp verursachen würden.



Visual Basic verursacht einen unangenehmen Laufzeitfehler, wenn Sie einen **Variant**-Wert mit einem numerischen Wert vergleichen, der **Variant**-Wert aber nicht in eine Zahl umgewandelt werden kann. Sie sollten Ihre Daten kennen. Visual Basic unterstützt Sie mit einigen internen Funktionen, bestimmte Datentypen zu überprüfen, die Sie in Kapitel 8 genauer kennenlernen werden.

Kombination bedingter Operatoren mit logischen Operatoren

Technisch gesehen bieten die sechs bedingten Operatoren genügend Leistungsfähigkeit, jede mögliche Bedingung abzutesten, aber Sie können ihre Flexibilität noch erhöhen, indem Sie sie mit den *logischen Operatoren* von Visual Basic kombinieren. Tabelle 6.2 listet die logischen Operatoren auf.



Mit Hilfe der *logischen Operatoren* können Sie zwei oder mehr bedingte Vergleiche kombinieren. Für die logischen Operatoren werden Schlüsselwörter verwendet, so wie Sie das schon vom **Mod**-Operator her kennen.

Tabelle 6.2: Visual Basic unterstützt vier logische Operatoren.

Operator	Beschreibung	Beispiel	Ergebnis
And	Beide Seiten müssen True ergeben	(2 < 3) And (4 < 5)	True
Or	Eine Seite muß True ergeben	(2 < 3) Or (6 < 7)	True
Xor	Nur eine Seite darf True ergeben	(2 < 3) Xor (7 > 4)	False
Not	Negiert True	Not (3 = 3)	False

Die beiden Operatoren **And** und **Or** werden öfter als die anderen beiden logischen Operatoren verwendet. Der **Xor**-Operator hilft Ihnen, zwischen sich wechselseitig ausschließenden Optionen zu unterscheiden. Wenn mehrere Optionen in einer wechselseitig ausschließenden Situation **True** ergeben, wenn also beispielsweise ein Benutzer mehrere Monate für ein Datum oder ein Geburtsdatum angegeben hat, erkennen Sie an dem falschen Ergebnis von **Xor**, daß mehrere Optionen (oder überhaupt keine) ausgewählt wurden. Das **Not** schließlich negiert **True**- und **False**-Ausdrücke. Wenn Sie den **Not**-Operator zu häufig einsetzen, kann das zu verwirrendem Code führen, weil Sie dadurch beim Schreiben und Debuggen die Logik vertauschen müssen. Setzen Sie das **Not** also sparsam ein.

Betrachten Sie den folgenden Ausdruck, der bedingte Operatoren mit dem logischen Operator **And** kombiniert:

```
(aktVerkauf < aktMindVerkauf) And (intJahreAngestellt > 10)
```

Wenn die aktuellen Verkäufe kleiner als der geforderte Mindestwert sind und der Angestellte länger als 10 Jahre in der Firma beschäftigt ist (sehen Sie, wie Sie mit sinnvollen Variablennamen Ihren Code dokumentieren können?), ergibt der gesamte Ausdruck **True**. Sie könnten beide Bedingungen auch separat überprüfen, ohne sie zu kombinieren, aber das **And** ermöglicht Ihnen, das ganze innerhalb eines Ausdrucks unterzubringen.



Kombinieren Sie nicht zu viele bedingte Ausdrücke mit logischen Operatoren, weil Ihr Code dadurch leicht unübersichtlich werden kann. Zerlegen Sie komplexe Anweisungen wie etwa die folgende:

`(a > 6) And (b < 1) Or Not(1 = c) Xor (d = 4)`

Die Operatorreihenfolge bestimmt, wie bedingte und logische Operatoren plziert und ausgeführt werden. Betrachten Sie den folgenden Ausdruck:

`aktVerkauf * sngKommission > aktMaxVerkauf /10`

Welche Operation führt Visual Basic als erstes aus? Vergleicht es `sngKommission` mit `aktMaxVerkauf`, multipliziert die Antwort mit `aktVerkauf` und dividiert das Ganze durch 10? Diese Reihenfolge macht keinen Sinn, weil der Operator `>` `True` oder `False` zurückgibt, und für diese Werte ist die Anwendung mathematischer Operatoren nicht erlaubt.

Tabelle 6.3 zeigt eine vollständigere Reihenfolge der Operatoren als Kapitel 5. Dort sehen Sie, wie sich bedingte und logische Operatoren verhalten, die zusammen mit mathematischen Operatoren im selben Ausdruck verwendet werden.

Tabelle 6.3: Die vollständige Reihenfolge der Operatoren.

Reihenfolge	Operatoren
1	Klammern
2	$\wedge$
3	<code>*</code> , <code>/</code> , <code>\</code> , <code>Mod</code>
4	<code>+</code> , <code>-</code>
5	Bedingte Operatoren inklusive <code>Like</code>
6	Logischer Operator <code>Not</code>
7	<code>And</code>
8	<code>Or</code>
9	<code>Xor</code>



Um sicherzustellen, daß sich Ihre Programme so klar wie möglich präsentieren, sollten Sie um Ausdrücke Klammern setzen, um Mehrdeutigkeiten in der Ausführungsreihenfolge auszuschließen. Der oben gezeigte Ausdruck würde besser so dargestellt:

`(aktVerkauf * sngKommission) > (aktMaxVerkauf /10)`

Die If-Anweisung und bedingte Operatoren

Einer der gebräuchlichsten Befehle von Visual Basic ist **If**. Der Befehl **If** ist Teil einer mehrzeiligen Anweisung, der sogenannten **If-Anweisung**, die das folgende Format hat:

If bedingung Then

Block mit einer oder mehreren Visual-Basic-Anweisungen

End If

Die *bedingung* ist ein Ausdruck, der das Ergebnis **True** oder **False** erzeugt. Es kann sich also um eine Boolesche Variable handeln, ein Steuerelement, das den Wert *True* oder *False* hat, oder einen längeren Ausdruck, der möglicherweise einen oder mehrere bedingte Operatoren beinhaltet.



Visual Basic unterstützt auch die ältere Form der **If**-Anweisung, die in einer Zeile dargestellt wird:

If bedingung Then Visual-Basic-Anweisung

If-Anweisungen lösen fast immer mehrere Anweisungen aus, deshalb ist das mehrzeilige **If** heute sinnvoller. Auch wenn das **If** nur eine einzige Anweisung auslösen und in dieser Form funktionieren würde, sollten Sie ein mehrzeiliges **If** verwenden, um es später einfacher ergänzen zu können.

Was Sie tun sollten

Rücken Sie den Rumpf einer **If**-Anweisung ein, so daß Sie auf den ersten Blick erkennen, wo sie beginnt und wo sie endet. Alle mehrzeiligen **If**-Anweisungen haben gleich viele **If**- und **End If**-Zeilen. Das **End If** gehört immer zum letzten **If**, egal wie weit Sie den Code einrücken.

Die Menschen verwenden jeden Tag irgendwelche **If**-Anweisungen:

Wenn ich früher zur Arbeit gehe, dann kann ich früher nach Hause gehen.

Wenn ich mein Zimmer aufräume und den Müll runtertrage, dann kann ich Fußballspielen gehen.

Mit anderen Worten, wir verwenden im täglichen Leben dieselbe **If**-Anweisung wie Visual Basic. Die **If**-Anweisung funktioniert so: Wenn und nur wenn die *Bedingung* zutrifft, wird der Coderumpf der **If**-Anweisung ausgeführt. Lesen Sie die beiden oben gezeigten **If**-Anweisungen noch einmal. Dann und nur dann, wenn Sie früh zur Arbeit gehen, können Sie früher aufhören. Wenn Sie nicht früh zur Arbeit gehen, können Sie nicht früher aufhören. Die zweite Anweisung besagt, daß beide Bedingungen erfüllt sein müssen: Wenn Sie Ihr Zimmer aufräumen *und* den Müll runtertragen, nur dann können Sie Fußballspielen gehen.

Listing 6.1: Daten mit Hilfe von **If** vergleichen

```
1: If (curSales > curSalesGoal) Then
2: ' Der Verkäufer hat das Ziel überschritten
3: curSalaryBonus = 1000.00
4: lblSalesNote.Caption = "Sie sind der Beste!"
5: lblSalesNote.BackColor = Red
6: lblSalesNote.FontBold = True
7: End If
8: ' Hier folgt der weitere Programmcode
```

Wenn der Wert in `curSales` größer als der Wert in `curSalesGoal` ist, werden die vier Anweisungen (ohne den Kommentar) in den Zeilen 3 bis 6 ausgeführt. Wenn der Wert nicht größer ist (oder gleich), werden die Zeilen 3 bis 6 nicht ausgeführt. Das Programm wird danach auf jeden Fall mit Zeile 8 nach dem `If` fortgesetzt. Sehen Sie, wie die Daten die `If`-Anweisung steuern? Sehen Sie, wie das Programm die Entscheidung zur Laufzeit trifft? Diese Entscheidung legt fest, ob bestimmte Codeabschnitte ausgeführt werden, insbesondere der Code innerhalb des `If`-Rumpfs.



Die Klammern um die Bedingung sind in einer `If`-Anweisung nicht zwingend erforderlich, helfen aber, den Ausdruck zu verdeutlichen.

If mit Else

Die vorigen Abschnitte beschrieben bereits eine Form der `If`-Anweisung, aber die Programmierer verwenden häufig die erweiterte Form, die das folgende Format hat:

```
If bedingung Then  
  Block mit einer oder mehreren Visual-Basic-Anweisungen  
Else  
  Block mit einer oder mehreren Visual-Basic-Anweisungen  
End If
```

Wie für alle mehrzeiligen Anweisungen sollten Sie auch hier den Rumpf einrücken, es ist jedoch nicht zwingend erforderlich. Das erste Format für die `If`-Anweisung ermöglichte die Ausführung von Code, wenn eine bestimmte Bedingung zutraf, unterstützt aber keinen Code, der ausgeführt wird, wenn eine bestimmte Bedingung nicht zutrifft. Und genau das macht das `Else`. Die `If . . . Else`-Anweisung stellt zwei Code-Rümpfe bereit: einen, der ausgeführt wird, wenn die Bedingung zutrifft, und einen anderen, der ausgeführt wird, wenn die Bedingung nicht zutrifft. Egal, welche Bedingung gilt, das restliche Programm wird nach der Ausführung von `If . . . Else` fortgesetzt.

Im Bonusprojekt 1 haben Sie eine `If . . . Else`-Anweisung gesehen, mit der ein Kennwortfeld ausgewertet wurde. Listing 6.2 zeigt dieses `If . . . Else`.

Listing 6.2: Kennwortauswertung

```
1: If txtPassword.Text = "Sams" Then  
2: ' Richtig! Das Kennwort stimmt  
3: Beep  
4: Beep ' Bild anzeigen  
5: imgPassword.Picture = LoadPicture("C:\Program Files\" _  
6: & "Microsoft Visual Studio\Common\Graphics\MetaFile\" _  
7: & "Business\coins.wmf")  
8: lblPrompt.Caption = "Zeig mir das Geld!"  
9: Else  
10: lblPrompt.Caption = "Falsches Kennwort - Wiederholen"  
11: txtPassword.Text = "" ' Altes Kennwort löschen  
12: txtPassword.SetFocus ' Fokus in das Textfeld setzen  
13: End If
```

Zeile 1 prüft, ob das Textfeld das richtige Kennwort enthält. Wenn ja, wird der Code im Rumpf

unmittelbar hinter dem **If** ausgeführt, beginnend mit Zeile 2. Die **Beep**-Anweisung gibt einen Ton auf dem PC-Lautsprecher aus. Wenn das Kennwort korrekt eingegeben wurde, bewirken die Zeilen 3 und 4, daß der PC zweimal piept. Anschließend wird das Bild angezeigt, das in den Zeilen 5, 6 und 7 fortgesetzt spezifiziert wird. Das Bezeichnungsfeld zeigt das richtige Kennwort an. Nach der Abarbeitung des **If** wird das Programm in der Zeile fortgesetzt, die der **End If**-Anweisung folgt. Trifft die Bedingung dagegen nicht zu, wird der Rumpf unterhalb des Schlüsselworts **Else** ausgeführt und die Zeilen 10, 11 und 12 informieren den Benutzer darüber, daß das Kennwort nicht richtig eingegeben wurde.

Sie können die **If**s auch ineinander verschachteln, wie in Listing 6.3 gezeigt.

Listing 6.3: Verschachtelte If-Anweisungen machen den Datenvergleich leistungsfähiger

```
If (curSales > 10000.00) Then
If (intHrsWorked > 40) Then
curBonus = 750.00
Else
curBonus = 500.00
End If
lblBonus.Caption = "Gute Arbeit!"
End If
```

Wenn Sie die **If...Else**-Anweisungen so einbetten, gehört jedes **Else** und jedes **End If** jeweils zu dem vorhergehenden **If**. Die Einrückungen jedes weiteren **If**-Rumpfs helfen Ihnen zu erkennen, wo ein **If** anfängt und wo es endet.

Ein frühes Ende

Manchmal möchte man ein Ereignis oder eine andere Prozedur vorzeitig beenden. Dazu kombinieren Sie die Anweisungen **If** und **Exit**.

Die **Exit**-Anweisung hat das folgende Format:

```
Exit Sub | Function | Do | For
```

Die vertikalen Striche zwischen den Schlüsselwörtern bedeuten, daß **Exit** jeweils nur eines von ihnen folgen kann; welches Sie verwenden, hängt davon ab, was Sie verlassen bzw. beenden möchten. Um eine Ereignisprozedur zu verlassen (wobei es sich um eine Sub-Prozedur handelt, wie Sie in Kapitel 4 erfahren haben), verwenden Sie die Anweisung **Exit Sub**. Um eine Funktionsprozedur zu beenden, verwenden Sie **Exit Function**. Die Anweisungen **Exit Do** und **Exit For** werden später in diesem Kapitel erklärt.

Listing 6.4 beendet die Ereignisprozedur in Zeile 3, wenn die Bedingung der **If**-Anweisung zutrifft.

Listing 6.4: Mit Exit Sub beenden Sie eine Prozedur vorzeitig

```
1: Private Sub cmdCalc ()
2: If (txtSales.Text < 5000.00) Then
3: Exit Sub ' Prozedur beenden
4: Else
5: ' Wenn der Bonus mindestens $5,000 beträgt...
6: ' wird die nächste Anweisung ausgeführt, die
7: ' den Bonus als Prozentwert der Verkäufe
```

```

8: ' anzeigt
9: lblBonus.Caption = txtSales.Text * .05
10: End If
11: End Sub

```

Verschachtelte If...Else-Anweisungen

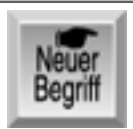
Wenn Sie eine If...Else-Anweisung in eine andere If...Else-Anweisung einbetten, verwenden Sie ElseIf, um die verschachtelte If-Anweisung zu beginnen. Betrachten Sie dazu den Code in Listing 6.5.

Listing 6.5: ElseIf ermöglicht, zwei oder mehr If...Else-Anweisungen zu verschachteln

```

1: If (intHours <= 40) Then
2:   curOverTime = 0.0
3:   ' Jetzt Stunden zwischen 40 und 50 überprüfen
4:   ' und das Eineinhalbfache zahlen
5:   ElseIf (intHours <= 50) Then
6:     curOverTime = (intHours - 40) * 1.5 * sngRate
7:   Else
8:     ' Doppelte Zahlung für die Zeit über 50 Stunden
9:     ' und das Eineinhalbfache für die Zeit zwischen
10:    ' 40 und 50
11:    curOverTime = ((intHours - 50) * 2 + (10 * 1.5)) * sngRate
12: End If

```



Eine *verschachtelte Anweisung* ist eine Anweisung, die im Rumpf einer anderen Anweisung enthalten ist.

Die ElseIf-Anweisung in Zeile 5 startet einen neuen If...Else-Block. Wenn die Stunden weniger oder gleich 40 sind, wie in Zeile 1 geprüft wird, müssen es mehr als 40 sein. Zeile 5 prüft, ob die Stunden zwischen 40 und 50 liegen (Zeile 5 wird nicht ausgeführt, wenn die Stunden nicht mindestens 40 sind). Für diese Überstunden wird die eineinhalbfache Zahlung berechnet. Wenn Zeile 5 nicht zutrifft, wurden mehr als 50 Stunden gearbeitet, wofür das Doppelte bezahlt wird. Zeile 11 enthält einen komplizierten Ausdruck, der für alle Stunden über 50 das Doppelte und für die 10 Stunden zwischen 40 und 50 das eineinhalbfache Gehalt berechnet.

Sind solche verschachtelte If...ElseIf...End If-Anweisungen nicht kompliziert und schwer zu debuggen? Natürlich. Dieses einfache Beispiel soll Ihnen nur zeigen, wie kompliziert sie sein können. Im nächsten Abschnitt lernen Sie eine bessere Alternative kennen: Select Case.

Auswahlen mit Select Case treffen

Die Anweisung Select Case ist besser dafür geeignet, mehrfache Bedingungen abzu prüfen. Mehr als drei oder vier eingebettete If...Else-Anweisungen machen ein Programm sehr kompliziert. Die Programmlogik wird verwickelt, wenn Sie schreiben »Wenn dies gilt, und wenn dies gilt, und wenn

noch irgend etwas gilt, dann mach dieses oder jenes, aber andernfalls wenn...«. Die `Select Case`-Anweisung hat das folgende Format:

```
Select Case Ausdruck
Case ausdruckÜbereinstimmung
Block mit einer oder mehreren Visual-Basic-Anweisungen
[ Case ausdruckÜbereinstimmung1
Block mit einer oder mehreren Visual-Basic-Anweisungen]
[ Case ausdruckÜbereinstimmung2
Block mit einer oder mehreren Visual-Basic-Anweisungen]
:
[ Case ausdruckÜbereinstimmungN
Block mit einer oder mehreren Visual-Basic-Anweisungen]
[Case Else
Block mit einer oder mehreren Visual-Basic-Anweisungen]
End Select
```

`Select Case` wählt eine von mehreren Bedingungen aus. Die Anzahl der Bedingungen, die durch [*Case ausdruckÜbereinstimmung# ...*] angegeben werden, ist davon abhängig, wie viele Bedingungen Sie abprüfen müssen. Wenn keine der Bedingungen erfüllt ist, wird der Code im `Case Else`-Rumpf ausgeführt, falls Sie einen solchen bereitgestellt haben.

Trotz des scheinbar komplizierten Formats ist die Verwendung von `Select Case` einfach. Betrachten Sie das Beispiel in Listing 6.6.

Listing 6.6: Die `Select Case`-Anweisung vergleicht mit mehreren Werten

```
1: ' Die Noten eines Schülers auswerten
2: Select Case txtGrade.Text
3: Case "A"
4: lblAnnounce.Caption = "Perfekt!"
5: Case "B"
6: lblAnnounce.Caption = "Großartig!"
7: Case "C"
8: lblAnnounce.Caption = "Mehr lernen!"
9: Case "D"
10: lblAnnounce.Caption = "Nahhilfe nehmen!"
11: Case "F"
12: lblAnnounce.Caption = "Noch einmal von vorne!"
13: Case Else
14: lblAnnounce.Caption = "Fehlerhafte Eingabe"
15: End Select
```



Der Datentyp von *Ausdruck* muß mit allen Vorgaben für *ausdruckÜbereinstimmung* übereinstimmen. Im Code von Listing 6.6 wird vorausgesetzt, das `txtGrade.Text` einen Buchstaben enthält. Aus diesem Grund prüfen die Zeilen 3, 5, 7 und 9, ob dieser Stringwert mit einem anderen Stringwert übereinstimmt.

Wenn das Textfeld `txtGrade.Text` den Buchstaben `A` enthält, wird der `Case`-Rumpf in Zeile 3 ausgeführt. Alle weiteren `Case`-Zweige werden von Visual Basic übersprungen. Nachdem das passiert ist, wird der Code beginnend ab Zeile 13 ausgeführt. Wenn das Textfeld `txtGrade.Text` den

Buchstaben *B* enthält, wird der **Case-Rumpf** in Zeile 5 ausgeführt usw. Der Rumpf eines **Case** kann mehrere Zeilen umfassen, auch wenn in diesem Beispiel jeweils nur eine Zeile gezeigt wird. Visual Basic weiß, daß nach einer Übereinstimmung im **Case-Zweig** der gesamte **Case-Rumpf** ausgeführt wird, bis zur nächsten **Case-Zeile**. Dort hat **Select Case** seine Aufgabe erledigt und das Programm kann fortgesetzt werden.

Wenn aus irgendeinem Grund eine andere Note als *A, B, C, D* oder *F* im Textfeld eingegeben wird, tritt **Case Else** in Kraft und zeigt an, daß eine falsche Eingabe vorgenommen wurde.

Visual Basic unterstützt noch eine Form von **Select Case**, wobei Sie mit dem Schlüsselwort **Is** einen bedingten Operator für jede *ausdruckÜbereinstimmung* angeben können. Listing 6.7 schreibt das zuvor gezeigte **Select Case** um, um die bedingte **Select Case**-Auswahl zu nutzen.

Listing 6.7: Sie können auch bedingte Select Case-Vergleiche verwenden

```
1: ' Notenauswertung eines Schülers
2: Select Case txtGrade.Text
3: Case Is >= 90
4: lblAnnounce.Caption = "Perfekt!"
5: Case Is >= 80
6: lblAnnounce.Caption = "Großartig!"
7: Case Is >= 70
8: lblAnnounce.Caption = "Mehr lernen!"
9: Case Is >= 60
10: lblAnnounce.Caption = "Nachhilfe nehmen!"
11: Case Else
12: lblAnnounce.Caption = "Noch einmal von Vorne!"
13: End Select
```

Bei diesem Format und der geänderten Beurteilung entspricht jeder **Case-Zweig** einem bestimmten Wertebereich, von 90 mit der besten, bis unter 60 mit der schlechtesten Beurteilung. Beachten Sie, daß keine Prüfung auf eine Beurteilung unterhalb von 60 gemacht wird, denn wenn die Beurteilung nicht zwischen 60 und 100 liegt, muß sie unter 60 liegen. (Dieses Beispiel setzt voraus, daß die Beurteilung zwischen 0 und 100 liegt; dabei soll demonstriert werden, wie **Case Else** funktioniert, und nicht, wie man fehlerhafte Daten auffängt.)

Es gibt noch ein letztes **Select Case**-Format in Visual Basic. Dabei wird mit dem Schlüsselwort **To** jeweils ein ganzer Wertebereich ausgewählt. Der Bereich legt fest, welcher **Case-Rumpf** ausgeführt wird. Verwenden Sie das bereichsbasierte **Select Case**, wenn Sie die Möglichkeiten sequentiell sortieren, wie in Listing 6.8 gezeigt.



Nicht für alle Vergleiche können **Select Case**-Anweisungen verwendet werden. Es gibt kein **Select Case**-Format, das logische Operatoren unterstützt, Sie können also kein **And**, **Or**, **Xor** oder **Not** in **Select Case** verwenden. In diesem Fall müssen Sie wieder auf eine verschachtelte **If...ElseIf...End If**-Anweisung zurückgreifen.

Listing 6.8: Beim Vergleich von ganzen Wertegruppen verwenden Sie ebenfalls Select Case

```
1: ' Notenauswertung eines Schülers
2: Select Case txtGrade.Text
3: Case 0 To 59
4: lblAnnounce.Caption = "Noch einmal von vorne"
```



```

5: Case 60 To 69
6: lblAnnounce.Caption = "Nachhilfe nehmen!"
7: Case 70 To 79
8: lblAnnounce.Caption = "Mehr lernen!"
9: Case 80 To 89
10: lblAnnounce.Caption = "Großartig!"
11: Case Else
12: lblAnnounce.Caption = "Perfekt!"
13: End Select

```

Beachten Sie, daß sich die Sortierreihenfolge der **Case**-Zweige in Listing 6.8 von dem zuvor verwendeten **Case**-Format unterscheidet. Das erste **Case** in Zeile 3 wertet den niedrigsten Bereich aus. Wenn die Beurteilung zwischen 0 und 59 liegt, erscheint die schlechteste Beurteilung im Bezeichnungsfeld. (Hier wird die Verwendung von **Integern** vorausgesetzt; wenn jemand den Wert 79.5 eingibt, würde ein Fehler auftreten; der Einfachheit halber wurden hier jedoch **Integer** verwendet.) Alle darauffolgenden Bereiche beinhalten stufenweise höhere Beurteilungen. Sie können auch mit **Stringbereichen** vergleichen, solange die gemäß der **ASCII-Tabelle** kleinsten **Strings** zuerst verglichen werden.



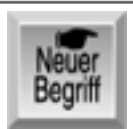
Die verschiedenen Formen der **Case**-Anweisung können in einer einzigen **Select Case**-Anweisung kombiniert werden. Das folgende **Case** verwendet alle Formate für den Vergleich mit einem Wert:

```
Case 101, 102, 201 To 205, Is > 300
```

Wenn der Ausdruck in der **Select Case**-Anweisung 101, 102, 201, 202, 203, 204, 205 oder größer als 300 ist, wird der Rumpf dieser **Case**-Anweisung ausgeführt.

Code mit Schleifen mehrfach ausführen

Visual Basic unterstützt *Schleifen*. Dabei werden unterschiedliche *Schleifenanweisungen* verwendet. Ihr PC ist schnell. Es kann sehr große Informationsmengen sehr schnell verarbeiten, beispielsweise die Datensätze für Gehaltsabrechnungen. Das Geheimnis, große Datenmengen zu verarbeiten oder sehr viele Werte zu vergleichen, ist die Verwendung von Schleifenanweisungen, so daß Ihr Programm alle dort vorgegebenen Daten verarbeitet.



Eine *Schleife* besteht aus einer oder mehreren Anweisungen, die wiederholt ausgeführt werden. Die Schleife wird ausgeführt, bis eine bestimmte Bedingung zutrifft.

Schleifen bilden die Grundlage für das nächste Kapitel, wo Sie lernen, wie Sie ohne die Verwendung von Textfeldern Eingaben vom Benutzer entgegennehmen. Einige Eingaben sind nicht für Textfelder geeignet. Häufig stellen Sie Ihrem Benutzer Fragen und übernehmen die Antworten, sobald der Benutzer die Eingabetaste drückt. Ein Textfeld ist für einfache Antworten, auf die Ihr Programm antworten soll, manchmal etwas kompliziert (Textfelder sind gut für die Eingabe von Formularinformationen geeignet, beispielsweise für Namen und Adressen für Gehaltsabrechnungen.)

Sie sollten das Konzept der Schleifen verstanden haben, bevor Sie sich mit Benutzereingaben beschäftigen, weil die Benutzer nicht immer die erwartete Antwort geben. Angenommen, Sie fragen

einen Benutzer, wie alt er ist, und Sie erhalten die Antwort 291. Das ist relativ unwahrscheinlich. Mit Hilfe einer Schleife stellen Sie dem Benutzer immer wieder dieselbe Frage, bis er eine vernünftige Antwort eingibt. Ihr Programm kann jedoch nicht vorhersehen, wann der Benutzer einen sinnvollen Wert eingibt - aber Sie können einfach weiterfragen, bis der Benutzer einen plausibleren Wert als 291 eingibt. Wie Sie sehen werden, können Schleifen einen beliebigen Codeblock wiederholen.

Die Do-Schleife

Die Do-Schleife von Visual Basic ist eine mehrzeilige Anweisung. So wie die If-Anweisung gibt es auch die Do-Schleife in unterschiedlichen Formaten:

Do While bedingung

Block mit einer oder mehreren Visual-Basic-Anweisungen

Loop

Do

Block mit einer oder mehreren Visual-Basic-Anweisungen

Loop While bedingung

Do Until bedingung

Block mit einer oder mehreren Visual-Basic-Anweisungen Loop

Do

Block mit einer oder mehreren Visual-Basic-Anweisungen

Loop Until bedingung

Die *bedingung* in einer DO-Schleife ist ein Ausdruck, ein Steuerelement oder ein Boolescher Wert, der True oder False ergibt. Welches Format Sie verwenden, ist von Ihren Anforderungen und Ihrem Stil abhängig. Dabei liegen die folgenden Unterschiede vor:

- Die Position des bedingten Vergleichs; wenn der bedingte Vergleich oben in der Schleife der DO-Anweisung erfolgt, kann es sein, daß der Schleifenrumpf nie ausgeführt wird. Wenn der bedingte Vergleich unten in der Schleife der Loop-Anweisung erscheint, wird der Rumpf immer mindestens einmal ausgeführt, weil Visual Basic die Prüfung erst nach dem Durchlauf des Schleifenrumpfs ausführt.
- Die Art des bedingten Vergleichs; die DO-Schleifen können ausgeführt werden, solange eine Bedingung zutrifft, oder bis eine Bedingung zutrifft. Im ersteren Fall wird der Schleifenrumpf so lange ausgeführt, wie die Bedingung zutrifft. Im letzteren Fall wird die Schleife ausgeführt, bis die Bedingung zutrifft.

Abbildung 6.2 zeigt eine DO-Schleife und wie sie wiederholt wird. Dieser Code erhöht einfach den in der Caption-Eigenschaft des Bezeichnungsfelds angegebenen Wert von 1 bis 10. Anschließend wird die Schleife abgebrochen. In der Realität wird der Code auf den heutigen PCs so schnell ausgeführt, daß das Bezeichnungsfeld so schnell aktualisiert wird, daß Sie die Schritte von 1 bis 10 überhaupt nicht wahrnehmen. Sie sollten die Arbeitsweise dieser Schleife jedoch unbedingt nachvollziehen.



Der Code in Abbildung 6.2 demonstriert eine spezielle Zuweisung, wo derselbe Variablenname auf beiden Seiten des Gleichheitszeichens erscheint. Bei dieser Zuweisungsform handelt es sich immer um die Aktualisierung des Variablenwerts. In diesem Fall bedeutet die Anweisung `intCtr = intCtr + 1`, daß zur Variablen `intCtr` bei jedem Schleifendurchgang 1 addiert wird.

Diese Anweisungen werden wiederholt, bis die Bedingung True ergibt.



```
'Code zur Demonstration  
'von Do-Schleifen  
Dim intCtr As Integer  
intCtr = 1'Zähler initialisieren  
Do  
    lblOut. Caption = intCtr  
    intCtr = intCtr + 1  
Loop Until (intCtr = 10)
```

Abbildung 6.2:
Eine Schleife wiederholt die in ihrem Rumpf angegebenen Anweisungen.

Der Code im Schleifenrumpf von Abbildung 6.2 wird 10mal wiederholt. Bei jedem Durchlauf addiert er 1 zur Variablen `intCtr`. Diese Do-Schleife verwendet das Format `Do . . . Loop Until`, so daß die Schleife wiederholt wird, bis `intCtr` gleich 10 ist. Listing 6.9 zeigt eine Schleife mit identischem Ergebnis, die das Format `Do While . . . Loop` verwendet.

Listing 6.9: Verwenden Sie die Do-Schleife, die Ihnen am besten gefällt

```
1: Do While intCtr <= 10  
2: ' Diese Schleife macht dasselbe wie die  
3: ' Schleife in Abbildung 6.2  
4: lblOut.Caption = intCtr  
5: intCtr = intCtr + 1  
6: Loop
```



Sie müssen die Bedingung, die im Schleifenrumpf ausgewertet wird, irgendwie ändern, sonst wird die Schleife endlos ausgeführt. Wenn Sie versehentlich eine Endlosschleife erzeugen, hängt Ihr Programm dort fest, bis Sie auf die **Ende**-Schaltfläche in der Symbolleiste von Visual Basic klicken, oder bis Sie das Programmfenster schließen. Irgend etwas innerhalb des Schleifenrumpfs muß dafür sorgen, daß sich die abgeprüfte Bedingung ändert, sonst wird die Schleife ewig ausgeführt.

Für einige Schleifen, insbesondere für Benutzereingaben, ist es erforderlich, daß der Schleifenrumpf mindestens einmal ausgeführt wird, so daß dadurch festgelegt wird, welcher Schleifentyp verwendet werden soll. Wenn Sie möchten, daß eine Schleife mindestens einmal ausgeführt wird, verwenden Sie eine Schleife, die die Bedingung am Schleifenende testet, wie in Listing 6.10 gezeigt.



Die folgenden Codebeispiele sind nicht vollständig. Statt der Anweisungen, die die Benutzereingabe und die Fehlermeldungen realisieren, wurden Kommentare eingefügt. Hier sollten Sie sich auf die Arbeitsweise der Schleifen konzentrieren. In Kapitel 7 erfahren Sie mehr über Eingaben und Ausgaben.

Listing 6.10: Ihr Benutzer braucht möglicherweise mehrere Anläufe, bevor er die richtigen Daten eingegeben hat

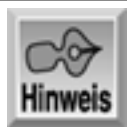
```
1: Dim strAns As String
```

```

2: '
3: ' Dem Benutzer wird eine Ja/Nein-Frage gestellt
4: lblPrompt.Caption = "Wollen Sie weitermachen (Ja oder Nein)?"
5: '
6: ' Eingaben in die Stringvariable
7: 'strAns übernehmen
8: ' Antwort überprüfen und weiterfragen, falls
9: ' der Benutzer nicht Ja oder Nein eingegeben hat
10: Do While (strAns <> "Ja" And strAns <> "Nein")
11: Beep ' Warnung
12: lblError.Caption = "Sie sollen mit Ja oder Nein antworten")
13: ' Eingaben wieder in die Stringvariable
14: 'strAns übernehmen
15: Loop
16: ' Fehlermeldung löschen
17: lblError.Caption = Null

```

Die DO-Schleife beginnt in Zeile 10. Wenn der Benutzer in den Zeilen nicht **Ja** oder **Nein** eingegeben hat (beachten Sie, daß hier statt der Eingabe Kommentare angezeigt werden), gibt die Schleife in Zeile 12 eine Fehlermeldung aus. Die Kommentarzeilen 13 und 14 simulieren die wiederholte Eingabe des Benutzers, und Zeile 15 schickt die Schleife zurück in Zeile 10, um diese Eingabe wieder zu überprüfen. Wie oft wird diese Schleife wiederholt? Entweder nie (wenn der Benutzer sofort Ja oder Nein eingibt) oder solange, bis der Benutzer eine falsche Antwort eingibt.



Natürlich kann die (^)-Taste des Benutzers gedrückt sein, und der Benutzer gibt möglicherweise JA oder NEIN ein, oder andernfalls ja oder nein. In diesem Fall schlägt Zeile 10 fehl, weil die beiden Strings nicht übereinstimmen. Mehr über den Vergleich von Strings, die sich nur in der Groß-/Kleinschreibung unterscheiden, erfahren Sie in Kapitel 8.

Wenn Sie eine Schleife verlassen wollen, bevor sie auf normale Weise beendet wird, verwenden Sie **Exit DO**. Beispielsweise könnten Sie innerhalb einer Schleife Eingangsrechnungen verarbeiten, bis alle Rechnungen erledigt sind. Tritt eine falsche Kundennummer auf, was mit Hilfe einer **If**-Anweisung in der Schleife abgetestet wird, können Sie eine Fehlermeldung anzeigen und die Schleife mit **Exit DO** verlassen.

For-Schleifen

Visual Basic unterstützt noch eine andere Schleife: die **For**-Schleife. **For**-Schleifen führen eine oder mehrere Anweisungen in einer festen Anzahl von Schleifendurchgängen aus, oder bis eine bestimmte Bedingung zutrifft. Die **For**-Schleife ist eine mehrzeilige Anweisung (so wie auch die **DO**-Schleifen), so daß die auszuführenden Anweisungen in einem Rumpf angegeben werden können. Die **For**-Anweisung verwendet das folgende Format:

```

For intZähler = intStart To intEnd [Step intInkrement]
Block mit einer oder mehreren Visual-Basic-Anweisungen
Next [intZähler]

```

intZähler muß eine numerische Variable sein, die den Schleifenrumpf steuert. Visual Basic initialisiert die Variable *intZähler* in *intStart* mit dem Ausgangswert, bevor die Schleife zum ersten Mal

durchlaufen wird (ein solcher Schleifendurchlauf wird auch als *Iteration* bezeichnet). Der *intStart*-Wert ist in der Regel 1, es kann sich jedoch auch um einen beliebigen anderen numerischen Wert (oder eine Variable oder einen Steuerelementwert) handeln, den Sie vorgeben. Bei jedem Durchlauf des Schleifenrumpfs wird die Variable *intZähler* um den Wert von *intInkrement* inkrementiert oder dekrementiert. Wenn Sie keinen *Step*-Wert angeben (beachten Sie, daß der *Step*-Zweig optional ist, weil er in der Formatspezifikation in eckigen Klammern dargestellt ist), geht *For* von einem *intInkrement*-Wert von 1 aus.



Eine *Iteration* ist ein Schleifendurchgang. Wenn eine Schleife dreimal wiederholt wird, gibt es drei Iterationen dieser Schleife.

IntEnd ist eine Zahl (oder eine Variable oder ein Steuerelementwert), die steuert, wann die Schleife enden soll. Wenn *intZähler* größer *intEnd* ist, wiederholt Visual Basic die Schleife nicht, sondern setzt die Programmausführung mit der Anweisung hinter *Next* fort. Mit *Next* kennzeichnet Visual Basic das Schleifenende. Wenn *intZähler* kleiner *intEnd* ist, inkrementiert Visual Basic *intZähler* um den Wert von *intInkrement*, und der Schleifenrumpf wird wiederholt. (Beachten Sie, daß Sie *intZähler* nach *Next* nicht wiederholen müssen, weil diese Anweisung optional ist und nur als Merkhilfe dient, welche *For*-Schleife die *Next*-Anweisung beendet.)

Trotz dieser umfangreichen Beschreibung ist die *For*-Schleife größtenteils sehr einfach und tut nichts anderes, als bei der Schleifenwiederholung mitzuzählen. Eine *For*-Schleife zählt nach oben, wenn der *Step*-Wert positiv ist, nach unten, wenn der *Step*-Wert negativ ist.

Die *For*-Schleife in Listing 6.11 entspricht der in Listing 6.9 gezeigten *DO*-Schleife. In der *For*-Schleife wird die Zählervariable für das Bezeichnungsfeld automatisch inkrementiert.

Listing 6.11: Wenn Visual Basic eine Zählervariable inkrementieren soll, verwenden Sie eine For-Schleife

```
1: For intCtr = 1 to 10
2: lblOut.Caption = intCtr
3: Next
```

Das ist doch ganz einfach! Die Schleife wird 10mal wiederholt. Wenn Zeile 1 zum ersten Mal wiederholt wird, wird *intCtr* der Wert 1 zugewiesen (der *intStart*-Wert). Der Schleifenrumpf, Zeile 2, wird unter Verwendung dieses Werts ausgeführt. Zeile 3 bewirkt, daß die Schleife wiederholt wird, indem sie *intCtr* um 1 inkrementiert (durch den impliziten *Step*-Wert, weil hier nichts anderes angegeben wurde), solange *intCtr* kleiner 10 ist (das ist der *intEnd*-Wert).



Die folgende Anweisung ist identisch mit Zeile 3 in Listing 6.11, weil die Variable in der *Next*-Anweisung optional ist:

```
Next intCtr
```

Ändern des Step-Werts

Die *For*-Schleife in Listing 6.12 beginnt mit 10 und wird in 5er-Schritten inkrementiert, bis die Schleifenvariable den Wert 100 erreicht hat.

Listing 6.12: Durch die Änderung des Inkrementwerts verhält sich die Schleife anders

```
1: For intCtr = 10 to 100 Step 5
2: lblOut.Caption = intCtr
3: Next
```

Die For-Schleife in Listing 6.13 beginnt mit 1000 und wird in 100er-Schritten dekrementiert, bis die Schleifenvariable gleich 0 ist.

Listing 6.13: Ein negativer Step-Wert dekrementiert die Schleifenvariable

```
1: For intCtr = 1000 to 0 Step -100
2: lblOut.Caption = intCtr
3: Next
```

Aus diesen kurzen Beispielen erkennen Sie, wie die Werte für *intStart*, *intEnd* und *intInkrement* die Schleife beeinflussen. (Wenn Sie einen negativen **Step**-Wert verwenden, muß der *intStart*-Wert größer als der *intEnd*-Wert sein; andernfalls wird der Schleifenrumpf nie ausgeführt.)



Verwechseln Sie Schleifen nicht mit der **If**-Anweisung. Beide Schleifentypen wie auch die **If**-Anweisungen verwenden bedingte Ausdrücke, aber Schleifen können den Rumpf so oft wiederholen, wie das erforderlich ist. Der Rumpf der **If**-Anweisung dagegen wird höchstens einmal wiederholt.



Verwenden Sie die Anweisung **Exit For**, wenn Sie eine Schleife verlassen möchten, bevor sie auf normale Weise beendet wird.

Verschachtelte For-Schleifen

Wie alle anderen Visual-Basic-Anweisungen können Sie auch **For**-Schleifen ineinander verschachteln. Wenn Ihr Programm eine Schleife mehrere Male wiederholen soll, verwenden Sie eine verschachtelte Schleife. Abbildung 6.3 zeigt eine Skizze einer verschachtelten **For**-Schleife. Sie können sich dabei vorstellen, daß sich die innere Schleife »schneller« dreht als die äußere Schleife. Die innere Schleife iteriert schneller, weil die Variable **In** innerhalb der Schleife von 1 bis 10 gezählt wird, und danach erst ist die erste Iteration der äußeren Schleife vollständig ausgeführt. Weil die äußere Schleife erst mit der Anweisung **Next Out** wiederholt wird, kann die innere Schleife einmal vollständig durchlaufen werden. Wenn die äußere Schleife ein zweites Mal iteriert, beginnt die innere Schleife erneut eine vollständige Ausführung.

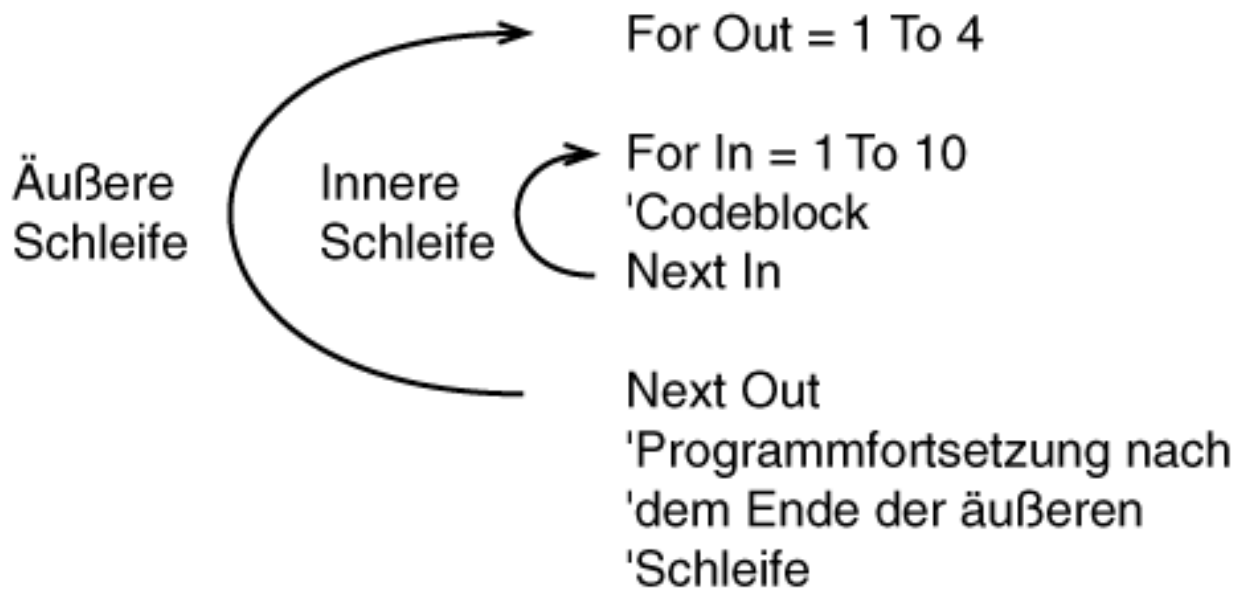


Abbildung 6.3:
Die äußere Schleife bestimmt, wie oft die innere Schleife ausgeführt wird.

Die innere Schleife in Abbildung 6.3 wird insgesamt 40mal durchlaufen: Die äußere Schleife iteriert viermal, die innere wird für jeden Durchgang der äußeren Schleife 10mal ausgeführt.

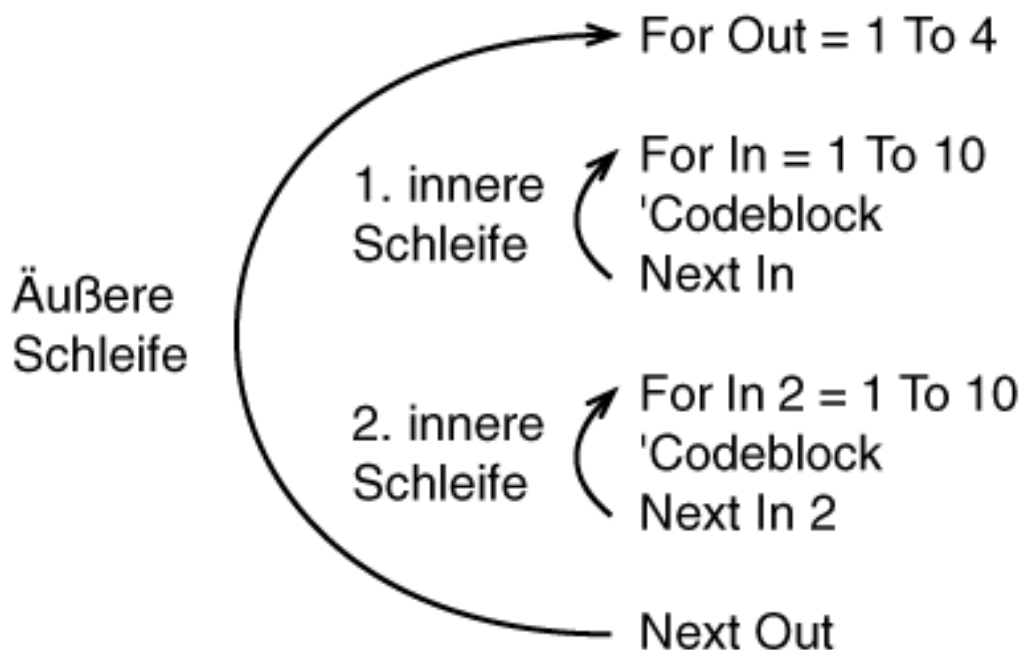


Abbildung 6.4:
Innerhalb einer Schleife können zwei oder mehr andere Schleifen verschachtelt werden.

Abbildung 6.4 zeigt, wie zwei Schleifen in einer äußeren Schleife verschachtelt sind. Beide Schleifen werden vollständig ausgeführt, bevor die äußere Schleife einmal iteriert. Wenn die äußere Schleife den zweiten Durchgang startet, werden die beiden inneren Schleifen wieder vollständig ausgeführt.

Die Codeblöcke in der innersten Schleife von Abbildung 6.4 werden je 40mal ausgeführt. Die äußere Schleife iteriert viermal, und jede innere Schleife wird bei jeder einzelnen Iteration der äußeren Schleife vollständig ausgeführt.

Was Sie tun sollten

Wenn Sie Schleifen verschachteln, ordnen Sie jeder **For**-Anweisung das richtige **Next** zu. Jedes **Next** gehört zu dem zuletzt im Code angegebenen **For**. Visual Basic gibt einen Fehler aus, wenn Sie ein Programm schreiben, dessen **Next**-Anweisung einer inneren Schleife nach der **Next**-Anweisung der äußeren Schleife erscheint. Wenn Sie die Variable für **Next** weglassen, ordnet Visual Basic das **Next** dem vorhergehenden **For** zu. Durch die Verwendung einer Variablen für die **Next**-Anweisung wird die Verwendung der Schleifen klarer dokumentiert und man erkennt schneller, wo eine Schleife beginnt und wo sie endet.

Zusammenfassung

Heute haben Sie gelernt, wie Sie Ihre Programme steuern. Mit Hilfe von bedingten Operatoren analysiert Visual Basic die ihm übergebenen Daten und reagiert auf die Werte von Variablen und Steuerelementen. Mit der **If**-Anweisung können Sie jetzt ein Programm schreiben, in dem Visual Basic den Wert einer Variablen oder eines Steuerelements auswertet und vom Ergebnis abhängig eine Aktion ausführt.

Neben der **If**-Anweisung unterstützt Visual Basic auch die **Select Case**-Anweisung, die statt verschachtelter **If**-Anweisungen verwendet werden kann und damit die Situation leichter verständlich macht. **Select Case** gibt es in drei Formaten, abhängig davon, wie Sie die verschiedenen Bedingungen auswerten möchten.

Wenn Sie bestimmte Abschnitte Ihres Codes mehrmals wiederholen möchten, verwenden Sie eines der in diesem Kapitel vorgestellten Schleifenkonstrukte. Die **Do**-Schleife iteriert, bis eine Bedingung erfüllt ist, oder solange eine Bedingung erfüllt ist, abhängig davon, welches Format Sie verwenden. Die **For**-Schleife führt eine feste Anzahl von Iterationen aus oder läuft, bis eine bestimmte Bedingung zutrifft. Anders als **Do** ändert die **For**-Schleife ihre Steuervariablen automatisch, indem sie diese bei jeder Iteration erhöht oder verringert.

In Kapitel 7 erfahren Sie, wie Sie den Benutzer in Eingabefeldern zu Eingaben auffordern. Sie lernen, wie man in Meldungsfeldern Nachrichten für den Benutzer anzeigt. Eingabe- und Meldungsfelder stellen einfache Methoden dar, mit Ihrem Benutzer zu kommunizieren.

Fragen & Antworten

Frage:

Warum soll ich den Not-Operator nicht verwenden?

Antwort:

Sie sollten den Not-Operator vermeiden, weil dadurch alles sehr kompliziert wird. Positive Anweisungen sind besser verständlich.

*Betrachten Sie den folgenden Ausdruck: **Not (A <= B)**. Wäre es nicht einfacher, **(A > B)** zu schreiben? Sie müssen **Not** natürlich nicht ganz vermeiden, weil es für Boolesche Datentypen durchaus sinnvoll sein kann, beispielsweise in Anweisungen wie **If Not (blnClearedScreen)**. Diese **If**-Anweisung bedeutet: Wenn der Bildschirm noch nicht gelöscht ist, soll das passieren, was hinter dem **If** angegeben ist. Sie können jedoch Ausdrücke mit **Not** fast immer umkehren, so daß sie verständlicher sind.*

Frage:

Ist es egal, welche äquivalenten *Do*- und *For*-Anweisungen ich für die Programmierung verwende?

Antwort:

*Es bleibt Ihnen überlassen, welches Schleifenkonstrukt Sie verwenden. Dabei besteht nicht nur eine Wahlmöglichkeit zwischen *For* und *Do*, sondern auch zwischen den verschiedenen Varianten der beiden Schleifenformen. Im allgemeinen kann man sagen, daß *For*-Schleifen praktisch sind, wenn Sie Werte zählen oder eine Schleife durchlaufen, bis oder solange eine bestimmte Bedingung erfüllt ist. Wenn Sie aufwärts oder abwärts zählen, ist die *For*-Schleife besser geeignet und außerdem etwas effizienter als eine äquivalente *Do*- Schleife.*

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welcher logische Operator gibt `True` zurück, wenn die Ausdrücke auf beiden Seiten `True` sind?
2. Was ist der Unterschied zwischen einem bedingten Operator und einem logischen Operator?
3. Was ist eine Schleife?

4. Beschreiben Sie, was die folgende Zuweisung bewirkt:

```
intTotal = intTotal - 10
```

5. Wie oft führt der folgende Code die `Beep`-Anweisung aus?

```
intN = 0
Do While (intN > 0)
intN = intN + 3
Beep
Loop
```

6. Warum sollte `Exit For` Teil einer `If`-Anweisung sein, statt im Rumpf einer `For`- Schleife zu erscheinen?
7. Richtig/Falsch? Es kann sein, daß beide Blöcke einer `If . . . Else`-Anweisung ausgeführt werden.
8. Richtig/Falsch? Eine `For`-Schleife wird abhängig von den Start- und Endewerten unter Umständen nie ausgeführt.
9. In welchen Situationen könnte es sinnvoll sein, `For`-Schleifen zu verschachteln?
10. Was ist der Unterschied zwischen einer entscheidungstreffenden Anweisung und einer Schleifenanweisung?

Übungen

1. Schreiben Sie eine `If`-Anweisung, die drei Zahlen auf Gleichheit überprüft.

2. Fehlersuche: Peters Schleife funktioniert nicht. Was stimmt damit nicht:

```
intN = 10
```

```
Do
```

```
Beep
```

```
Loop Until (intN > 100)
```

3. Richtig/Falsch? Die Uhr für ein Football-Spiel zählt 15 Minuten rückwärts. Das erfolgt viermal. Überlegen Sie, wie die vier Iterationen aussehen, und beschreiben Sie, mit welchen Visual-Basic-Anweisungen Sie diesen Vorgang simulieren.

Visual Basic ermöglicht Ihnen, die verschiedenen `Select Case`-Formate in einer einzelnen `Select Case`-Anweisung zu kombinieren. Sie können also ein normales `Case` mit einem bedingten `Case` und einem Bereichs-`Case` zusammen verwenden. Schreiben Sie das früher in diesem Kapitel gezeigte Beispiel für die Gehaltszahlung so um, daß dabei für die drei Fälle ein `Case` in mindestens zwei Variationen verwendet wird.

Woche 1

Tag 7

Eingabe/Ausgabe: Tastatur und Bildschirm

In diesem Kapitel erfahren Sie, wie Sie Eingaben vom Benutzer entgegennehmen und Informationen auf dem Bildschirm ausgeben. Sie haben bereits gesehen, wie Sie mit Textfeldern Eingaben anfordern, und Sie haben auch schon mit Bezeichnungsfeldern Informationen auf dem Bildschirm ausgegeben. Nichtsdestotrotz sind diese Steuerelemente nicht immer optimal dafür geeignet, dem Benutzer zur Laufzeit interaktiv Fragen zu stellen oder ihm Antworten bereitzustellen. Textfelder sind ausgezeichnet für Formulare und als Platzhalter für die Eingaben von Benutzern geeignet, aber manchmal muß die Antwort des Benutzers unmittelbarer erfolgen. Heute erfahren Sie, wie Sie diese Antwort mit relativ geringem Programmieraufwand erhalten.

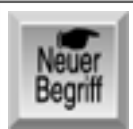
Im einzelnen lernen Sie die folgenden Dinge kennen:

- Interne Funktionen
- Die Funktion `MsgBox()`
- Optionale Funktionsparameter
- Die Funktion `InputBox()`
- Verarbeitung von Tastaturereignissen
- Kontrollkästchen
- Optionsfelder
- Optionsfeldgruppen in Rahmen-Steuerelementen

Interne Funktionen - Grundlagen

Funktionen sind eine Prozedurvariante, ähnlich den Sub-Prozeduren, die Sie in Kapitel 3 kennengelernt haben. Der Unterschied zwischen beiden ist, daß Funktionen nach der Ausführung ihres Codes einem anderen Programmteil einen Wert übergeben. In Kapitel 8 werden Sie erfahren, wie man Funktionen schreibt und wie sie arbeiten.

Hier sollten Sie zunächst verstehen, was *interne Funktionen* sind, weil Sie diese im restlichen Kapitel für den I/O verwenden werden. Eine interne Funktion verhält sich irgendwie wie eine Black Box, die einen Wert von Ihnen entgegennimmt und einen einzigen Wert an Ihr Programm zurückgibt. Abbildung 7.1 zeigt die Arbeitsweise einer Funktion.



Eine *interne Funktion*, manchmal auch als *integrierte Funktion* bezeichnet, wird in der Sprache Visual Basic bereitgestellt und übernimmt bestimmte Berechnungen oder I/O. Damit Sie eine interne Funktion benutzen können, müssen Sie ihren Namen und die jeweiligen Anforderungen kennen. Sie sehen den Code in der internen Funktion zwar nicht (sie sind ebenso Teil der Programmiersprache Visual Basic wie beispielsweise der Befehl `For`), können sie aber trotzdem in Ihren Programmen verwenden.



I/O steht für *Input/Output* (Eingabe/Ausgabe). Programmierer verwenden den Begriff I/O häufig generisch, um damit zu beschreiben, wie ein Formular Eingaben von einem Gerät (beispielsweise der Tastatur) entgegennimmt und Ausgaben an ein Gerät (beispielsweise den Bildschirm) sendet.

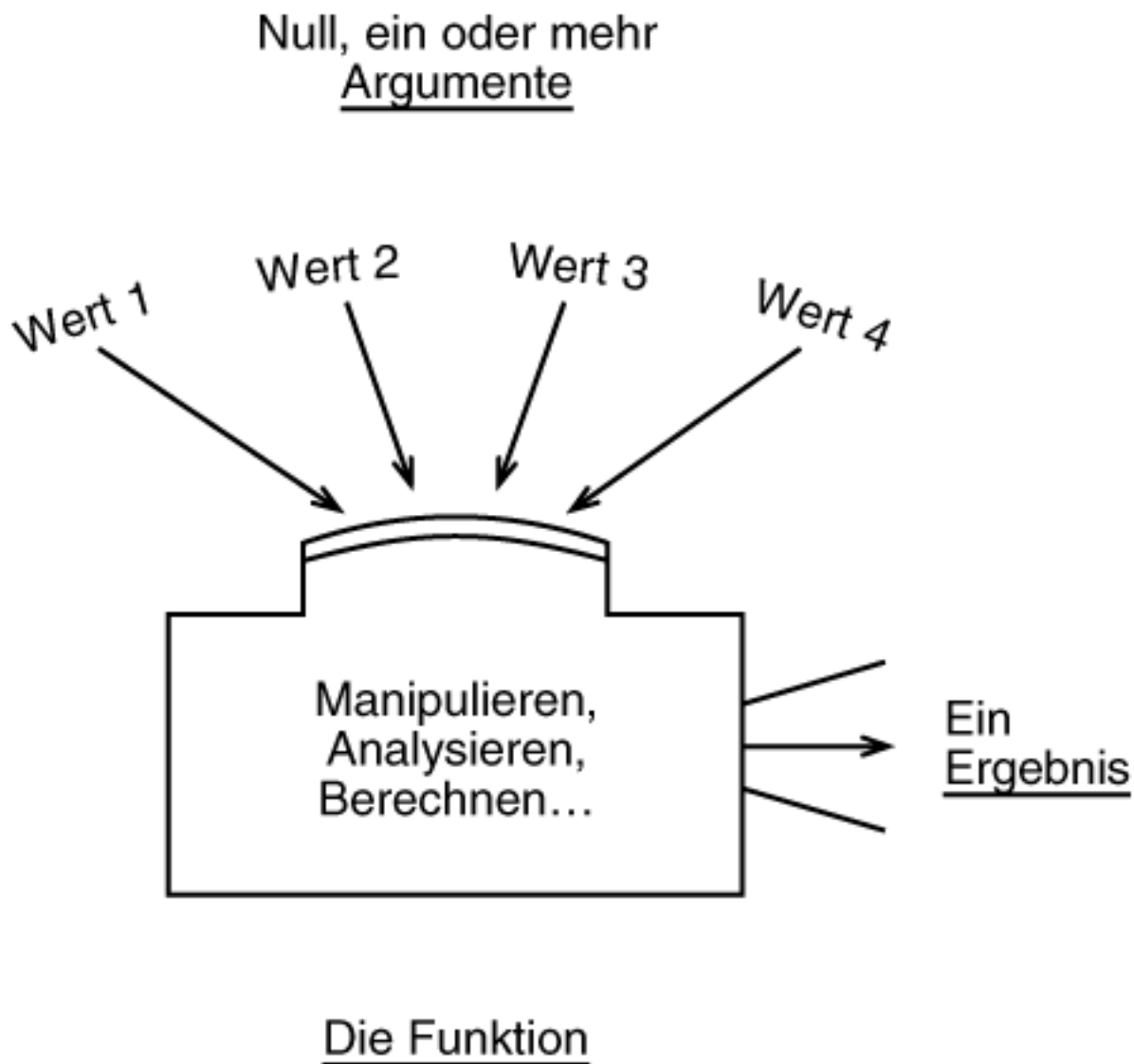


Abbildung 7.1:
Funktionen nehmen null, einen oder mehr Werte entgegen und geben einen einzigen Wert zurück.

Sie werden für die unterschiedlichsten Aufgaben Funktionen einsetzen. Je mehr Sie in diesem und insbesondere dem nächsten Kapitel darüber lernen, desto besser werden Sie sie verstehen. Hier sollten Sie die folgenden Dinge berücksichtigen:

- In der Regel werden einer Funktion ein oder mehrere Werte übergeben; es gibt nur wenige Funktionen, denen keine Parameter übergeben werden. Die übergebenen Werte werden auch als *Argumente* bezeichnet.



Ein *Argument* ist ein Wert, den Sie einer Funktion übergeben.

- Dem Funktionsnamen folgen immer Klammern (mit Ausnahme der Funktionen, für die keine Argumente übergeben werden).
- Die Argumente einer Funktion werden durch Kommas voneinander getrennt und stehen innerhalb der Klammern.

Sie haben bereits eine interne Funktion benutzt (im weiteren Kapitel werden wir nur noch den Begriff »Funktion« verwenden), nämlich in Kapitel 2, wo Sie ein Bild in das Anzeigefeld eines Formulars geladen haben. Hier folgt die Codezeile, die Sie dafür benutzt haben, wobei das Argument etwas gekürzt wurde, um die Beschreibung zu vereinfachen:

```
imgHappy.Picture = LoadPicture("\Happy.bmp")
```

Der Funktionsname ist `LoadPicture()`. (Wie allgemein üblich, werden auch in diesem Buch die Klammern hinter der Funktion angegeben, wenn wir darüber sprechen, um so zwischen den Namen von Funktionen, Variablen und Steuerelementen zu unterscheiden.) Diese Funktion nimmt nur ein Argument entgegen, einen String.



Für viele Funktionen sind ein oder mehrere Argumente erforderlich, die zum Teil jedoch optional sind. Für `LoadPicture()` ist ein Stringargument zwingend erforderlich, die anderen Argumente sind optional.



Der Datentyp und die Reihenfolge für ein Argument müssen eingehalten werden. Wenn für eine Funktion beispielsweise zwei Argumente erforderlich sind, ein Integer gefolgt von einem String, müssen Sie diese Argumente in genau dieser Reihenfolge eingeben.

Was übergibt dieser Code `LoadPicture()`? Einen String, der einen Dateinamen enthält. Was gibt `LoadPicture()` zurück? Das Bild, das sich in dieser Datei befindet. Die Zuweisung, die Sie in Kapitel 2 vorgenommen haben, weist dieses Bild der `Picture`-Eigenschaft des Anzeigefelds zu. Ohne die Funktion `LoadPicture()` könnte das Anzeigefeld das Bild nicht anzeigen. Für die `Picture`-Eigenschaft eines Anzeigefelds muß ein Bild angegeben werden, kein Pfadname (es sei denn, Sie geben zur Entwurfszeit ein Bild an, für das Sie im Eigenschaftfenster einen Pfad auswählen, womit Visual Basic die Grafik diesem Steuerelement zuweist).

Die Verwendung der Funktion `LoadPicture()` bedeutet einen großen Arbeitsaufwand für Visual Basic. Es muß Ihre Argumenteliste auswerten, um sicherzustellen, daß Sie die richtige Reihenfolge und die Datentypen eingehalten haben. Anschließend prüft es, ob der Pfad zur Datei korrekt ist. Danach muß es kontrollieren, ob Sie die Netzwerkzugriffsberechtigungen für diese Datei besitzen (falls Sie vernetzt sind). Schließlich lädt Visual Basic das Bild aus der Datei in das Anzeigefeld. Glücklicherweise stellt Visual Basic die Funktion `LoadPicture()` intern bereit, so daß Sie diese müßige Arbeit nicht selbst erledigen müssen. Und deshalb sind Funktionen auch so praktisch: sie ersparen Ihnen Arbeit. Funktionen ermöglichen Ihnen, sich auf das Wesentliche zu konzentrieren (die

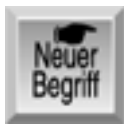
Besonderheiten Ihrer Applikation), und schaffen Ihnen die Routinearbeiten vom Hals.



Es gibt Funktionen für die Arbeit mit Grafiken, zur Berechnung gebräuchlicher mathematischer Formeln und für die Dateiverwaltung. Die Funktionen, die Sie in diesem Kapitel kennenlernen, sind einfach zu bedienen und sollen Ihnen helfen, sich mit dem Konzept vertraut zu machen. In Kapitel 8 werden Sie noch viele andere Funktionen kennenlernen.

Eingaben des Benutzers durch MsgBox()

Nachdem Sie nun wissen, was eine Funktion ist, wollen wir die Funktion `MsgBox()` genauer betrachten. `MsgBox()` ist eine Funktion, die ein Popup-Meldungsfeld anzeigt. Abbildung 7.2 zeigt ein Meldungsfeld. Wie Sie sehen, zeigt ein Meldungsfeld ein Icon und eine Meldung sowie mindestens eine Schaltfläche an. Mit dieser Schaltfläche kann der Benutzer bestätigen, daß er die Meldung gelesen hat.



Ein *Meldungsfeld* ist ein kleines Dialogfeld, in dem während der Programmausführungen Ausgaben angezeigt werden. Der Benutzer schließt ein Meldungsfeld, indem er auf eine Schaltfläche klickt. Das Meldungsfeld kann verschoben, nicht aber in der Größe geändert werden.

Die Argumente für die Funktion `MsgBox()` bestimmen, welches Icon, welche Meldung und wie viele Schaltflächen angezeigt werden. Der Programmierer kann damit genau steuern, wie sich dem Benutzer das Meldungsfeld präsentiert. Nachdem `MsgBox()` abgearbeitet ist, steht im Rückgabewert, welche Schaltfläche der Benutzer angeklickt hat. Ihr Programm wertet also den Rückgabewert von `MsgBox()` aus, falls das Meldungsfeld zwei oder mehr Schaltflächen angezeigt hat. Das Programm entscheidet mit Hilfe einer `If`-Anweisung, was für die Auswahl des Benutzers passieren soll.



Frühere Versionen von Visual Basic unterstützten die `MsgBox`-Anweisung. Anders als `MsgBox()` konnte die `MsgBox`-Anweisung nicht interpretieren, welche Schaltfläche der Benutzer gedrückt hatte. Die `MsgBox`-Anweisung wird heute als veraltet betrachtet, aber Visual Basic 6 unterstützt sie der Abwärtskompatibilität halber weiter.

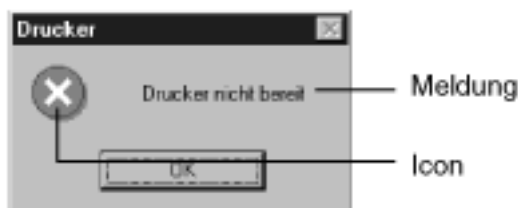


Abbildung 7.2:

Die Funktion `MsgBox()` zeigt eine Meldung an. Mit der Schaltfläche bestätigt der Benutzer, daß er die Meldung gelesen hat.

Die Funktion `MsgBox()` hat das folgende Format:

```
intAntwort = MsgBox(strAusgabe[, intStil][, strTitel])
```

`intAntwort` nimmt den Rückgabewert der Funktion entgegen. Das erste Argument ist ein String (oder eine Variable oder ein Steuerelement, die bzw. das einen String enthält), der als Meldung angezeigt wird. Das zweite Argument bestimmt, welche Schaltflächen angezeigt werden. Das dritte Argument enthält den Titel, der in der Titelleiste des Meldungsfelds angezeigt wird.

Alle Meldungsfelder zeigen mindestens eine Schaltfläche an. Ihr ausführendes Programm muß erkennen können, wann der Benutzer die Meldung gelesen hat. Die Programmausführung wird vorübergehend unterbrochen, bis der Benutzer auf eine der Schaltflächen im Meldungsfeld geklickt hat. Sobald der Benutzer die Schaltfläche anklickt, setzt das Programm seine Ausführung mit der Anweisung fort, die für die Schaltfläche vorgesehen ist.



Wenn Sie eine Meldung anzeigen, die zu lang für eine Zeile im Meldungsfeld ist, umbricht Visual Basic sie, und zeigt sie in zwei oder mehr Zeilen an. Der Umbruch erfolgt in den Wortzwischenräumen.

Angenommen, Sie wollen das Programm unterbrechen, so daß der Benutzer bestätigen kann, daß ein Bericht ausgedruckt werden soll. Dazu könnten Sie das folgende einfache Meldungsfeld verwenden:

```
intAntwort = MsgBox("Klicken Sie hier, wenn der Bericht ausgedruckt werden soll")
```

Dazu müssen Sie die Variable `intAntwort` irgendwo im Deklarationsbereich der Prozedur deklarieren (oder vielleicht auch im Deklarationsbereich des Moduls - auch wenn Sie noch nicht wissen, wie man globale Variablen deklariert). Wenn nicht anders angegeben (in dem zweiten Argument, das hier weggelassen wurde), zeigt Visual Basic eine Schaltfläche mit dem Text OK an, auf die der Benutzer klickt, nachdem er die Meldung gelesen hat. Weil das Meldungsfeld nur eine Schaltfläche enthält, ist die Integerzuweisung nicht besonders sinnvoll - aber Sie müssen irgend etwas mit dem Rückgabewert tun, ihn beispielsweise einer Variablen zuweisen. Abbildung 7.3 zeigt dieses einfache Meldungsfeld.

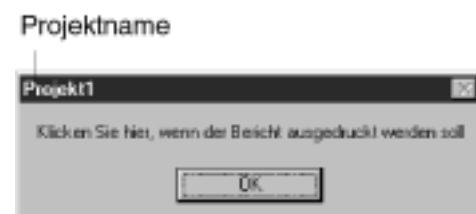


Abbildung 7.3:
Alle MsgBox ()-Funktionen zeigen eine Meldung und mindestens eine Schaltfläche an.

Beachten Sie noch etwas, was Visual Basic tut, wenn Sie nicht alle `MsgBox ( )`-Argumente angeben: es schreibt den Projekttitel in die Titelleiste des Meldungsfelds. Sie sollten einem Meldungsfeld also immer einen sinnvolleren Titel zuweisen. Wie das geht, erfahren Sie nach der Beschreibung des ersten optionalen Arguments.

Wenn Sie mehr Kontrolle über die Schaltflächen brauchen, geben Sie einen Integerwert (oder eine Variable oder ein Steuerelement) als erstes optionales Argument an. Wenn Sie nur eine Schaltfläche verwenden, ist der Rückgabewert nicht besonders wichtig, muß jedoch immer verarbeitet werden. Werden mehrere Schaltflächen verwendet, gibt der Rückgabewert an, welche Schaltfläche geklickt wurde. Diese Information kann in einer `If`-Anweisung oder in einem `Select Case` ausgewertet werden, um die entsprechende Verarbeitung für diese Schaltfläche vorzunehmen.

Tabelle 7.1 listet auf, welche Schaltflächen durch das erste optionale Argument der Funktion `MsgBox( )` erzeugt werden können.

Tabelle 7.1: Welche Schaltflächen in einem Meldungsfeld angezeigt werden, kann durch einen Integerwert angegeben werden.

Wert	Konstantenname	Beschreibung
0	vbOKOnly	OK
1	vbOKCancel	OK und Abbrechen
2	vbAbortRetryIgnore	Beenden , Wiederholen und Ignorieren
3	vbYesNoCancel	Ja , Nein und Abbrechen
4	vbYesNo	Ja und Nein
5	vbRetryCancel	Wiederholen und Abbrechen

Abbildung 7.4 zeigt das Meldungsfeld, das durch die folgende Meldung erzeugt wird:

```
intResponse = MsgBox("Jetzt drucken?", 1)
```

Die `1` legt fest, daß im Meldungsfeld zwei Schaltflächen erscheinen sollen, **OK** und **Abbrechen**. Diese Kombination ist praktisch, wenn Ihr Programm etwas ausführen soll, wofür Vorbereitungen erforderlich sind, beispielsweise wenn etwas gedruckt werden soll.

Wenn der Benutzer auf **OK** klickt, kann der Ausdruck vorgenommen werden, wenn er dagegen auf **Abbrechen** klickt, wird der Druckprozeß unterbrochen und das Programm wird an anderer Stelle fortgesetzt.

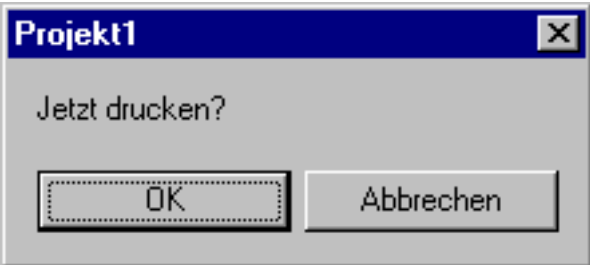


Abbildung 7.4:
Der Benutzer entscheidet, was als nächstes passieren soll.

Tabelle 7.2 zeigt die Rückgabewerte der Funktion `MsgBox( )`. Die folgende `If`-Anweisung könnte das oben beschriebene Meldungsfeld verarbeiten (die Details wurden hier durch Kommentare ersetzt, um das Ganze zu vereinfachen).

```
If (intResponse = 0) Then
' Hier steht der Code, der ausgeführt wird,
' wenn die OK-Schaltfläche angeklickt wurde
Else
' Hier steht der Code, der ausgeführt wird,
' wenn die Abbrechen-Schaltfläche angeklickt wurde
End If
```




Falls das Meldungsfeld weitere Schaltflächen anzeigt, müssen die entsprechenden Rückgabewerte in dieser **If**-Anweisung berücksichtigt werden. Möglicherweise ist es sinnvoll, eine **Select Case**-Anweisung dafür zu verwenden.

Tabelle 7.2: Diese Rückgabewerte müssen überprüft werden, um festzustellen, welche Schaltfläche der Benutzer angeklickt hat.

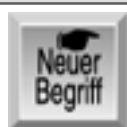
Wert	Konstantenname	Beschreibung
1	vbOK	Der Benutzer hat auf OK geklickt
2	vbCancel	Der Benutzer hat auf Abbrechen geklickt
3	vbAbort	Der Benutzer hat auf Beenden geklickt
4	vbRetry	Der Benutzer hat auf Wiederholen geklickt
5	vbIgnore	Der Benutzer hat auf Ignorieren geklickt
6	vbYes	Der Benutzer hat auf Ja geklickt
7	vbNo	Der Benutzer hat auf Nein geklickt



Wenn der Benutzer im Meldungsfeld auf die (Esc)-Taste gedrückt hat, reagiert Visual Basic, als hätte der Benutzer auf die **Abbrechen**-Schaltfläche geklickt und gibt den entsprechenden Wert zurück.

Benannte Konstanten

In Tabelle 7.1 und 7.2 sehen Sie die Spalte *Konstantenname*. Visual Basic unterstützt Hunderte von *benannten Konstanten*, die Sie in Prozeduren verwenden können.



Eine *benannte Konstante* ist ein Name, den Visual Basic in einer internen Werteliste verwaltet. Diese benannten Konstanten beginnen in der Regel mit dem Präfix **vb** (für Visual Basic). Der Wert der benannten Konstanten kann nicht geändert werden (deshalb heißen sie auch Konstanten), Sie können sie aber an jeder Stelle in einer Funktionsargumenteliste angeben, und auch an jeder anderen Stelle, wo die Verwendung von Konstanten möglich ist.

Benannte Konstanten machen Ihre Programme verständlicher. Beispielsweise sind die beiden folgenden Anweisungen identisch, aber in der zweiten Zeile wird offensichtlich, welche Schaltfläche verwendet wird:

```
intResponse = MsgBox("Jetzt drucken?", 1)
intResponse = MsgBox("Jetzt drucken?", vbOK)
```

Wenn Sie ein Programm schreiben, können Sie diese benannten Konstanten verwenden, ohne dafür in einem Handbuch oder der Online-Hilfe nachsehen zu müssen, und auch, ohne sich alle Namen selbst zu merken. Der Editor von Visual Basic zeigt während der Eingabe einer Funktion wie beispielsweise `MsgBox( )` eine Liste der benannten Konstanten an, aus der Sie einfach eine beliebige auswählen.

Später, wenn Sie das Programm warten und Änderungen vornehmen, können Sie dadurch sofort erkennen, wie das Meldungsfeld aussehen soll. Verwenden Sie statt der benannten Konstanten Konstantenwerte, müßten Sie den Wert nachschlagen, um zu erkennen, was das Meldungsfeld macht.

Was Sie tun sollten
Verwenden Sie die benannten Konstanten, wo immer das möglich ist. Dabei fällt keine zusätzliche Schreiarbeit an, weil Sie die Konstanten aus der Liste auswählen, die Visual Basic bei der Eingabe von Funktionsargumenten anzeigt.

Auslösen von Standardschaltflächen

Die erste Schaltfläche in einem Meldungsfeld ist immer die Standardschaltfläche. Visual Basic selektiert die erste (linke) Schaltfläche. Wenn der Benutzer die Eingabetaste drückt, ohne daß er auf eine Schaltfläche geklickt hat, wird die selektierte Schaltfläche ausgelöst.

Sie können bestimmen, welche Schaltfläche die Standardschaltfläche sein soll, indem Sie als Schaltflächenargument einen der in Tabelle 7.3 aufgelisteten Werte angeben.

Tabelle 7.3: Fügen Sie dem Schaltflächenargument einen dieser Werte hinzu, um die beim Start selektierte Standardschaltfläche zu bestimmen.

Wert	Konstantenname	Beschreibung
0	vbDefaultButton1	Die erste Schaltfläche ist die Standardschaltfläche
256	vbDefaultButton2	Die zweite Schaltfläche ist die Standardschaltfläche
512	vbDefaultButton3	Die dritte Schaltfläche ist die Standardschaltfläche

Trotz ihrer Länge sind die benannten Konstanten einfacher zu verwalten als die Zahlenkonstanten, Sie sollten Ihre Meldungsfelder also so beschreiben, wie in der nächsten Anweisung gezeigt:

```
intResponse = MsgBox("Ist der Drucker bereit?", vbYesNoCancel + vbDefaultButton1)
```



Wenn Sie ein Meldungsfeld anzeigen, das einen kritischen Prozeß steuert, beispielsweise das Löschen einer Datendatei, sollten Sie die **Abbrechen**- Schaltfläche zur Standardschaltfläche machen. Wenn der Benutzer versehentlich die Eingabetaste drückt, wird die **Abbrechen**-Schaltfläche ausgelöst und diese kritische Aktion wird nicht ausgeführt, was passiert, wenn Sie die **OK**-Schaltfläche als Standardschaltfläche beibehalten.

Das Icon für das Meldungsfeld

Durch die Angabe eines weiteren Werts im zweiten Argument wird das Icon festgelegt, das links im Meldungsfeld erscheint. Bisher wurde dieser Teil des Arguments noch nicht verwendet, die Meldungsfelder in den hier gezeigten Beispielen zeigten also noch keine Icons an.



Die Funktion `MsgBox( )` unterstützt mehrere zusätzliche und optionale Argumente, die jedoch hier nicht erklärt werden sollen, und die für die Entwicklung einfacher Programme nicht relevant sind.

Tabelle 7.4 zeigt die Konstantenwerte und die Icons, die sie erzeugen.

Tabelle 7.4: *Diese Werte erzeugen Icons in Ihrem Meldungsfeld.*

Wert	Konstantenname	Beschreibung	Icon
16	vbCritical	Kritische Meldung	
32	vbQuestion	Fragezeichen	
48	vbExclamation	Warnung	
64	vbInformation	Information	

Die folgende Anweisung erzeugt ein Meldungsfeld mit allen Komponenten, die Sie bisher kennengelernt haben. Alle Argumente werden übergeben. Abbildung 7.5 zeigt das resultierende Meldungsfeld.

```
intResponse = MsgBox("Ist der Drucker angeschaltet?", vbYesNoCancel + vbQuestion + vbDefaultButton2, "Eine Frage")
```



Abbildung 7.5:
Durch die Auswahl des Benutzers wird bestimmt, was als nächstes passiert.

Eingaben mit `InputBox()`

Mit der Funktion `MsgBox( )` werden Ihren Benutzern Meldungen angezeigt, auf die sie durch Auswahl der Schaltflächen reagieren können. Die nächste Aktion wird dadurch bestimmt, auf welche Schaltfläche sie klicken. Wenn die Meldung nur der Information des Benutzers dient, zeigen Sie ein Meldungsfeld mit einer einzigen Schaltfläche an, so daß der Benutzer das Feld schließen kann, nachdem er die Meldung gelesen hat. Anschließend wird das Programm fortgesetzt.

Wenn Sie dem Benutzer eine Frage stellen möchten und eine schnelle Antwort brauchen, so daß das

Textfeld für diese Form der Kommunikation nicht geeignet ist, verwenden Sie eine andere Funktion, die irgendwie mit `MsgBox()` verwandt ist: `InputBox()`. Die Funktion `InputBox()` zeigt ein Meldungsfeld an, in das der Benutzer einen Wert eingeben kann. Diese Abbildung 7.6 zeigt, wie so ein Eingabefeld aussieht.

	Ein <i>Eingabefeld</i> ist ein Meldungsfeld mit einem Feld, in das der Benutzer einen Wert eingeben kann, beispielsweise ein Wort oder einen Satz, die die im Titel gestellte Frage beantworten. Wie ein Meldungsfeld kann der Benutzer ein Eingabefeld verschieben oder schließen, aber seine Größe nicht ändern. Anders als bei Meldungsfeldern kann der Programmierer nicht bestimmen, welche Schaltflächen im Eingabefeld erscheinen. Eingabefelder enthalten nur die Schaltflächen OK und Abbrechen.

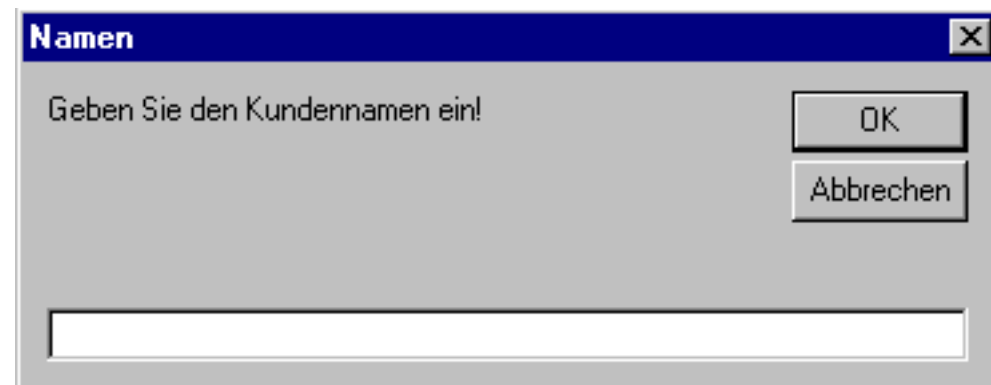


Abbildung 7.6:
Das Eingabefeld hat einen Titel und ein Feld zur Dateneingabe.

In Eingabefeldern können keine Icons angezeigt werden, wie das für Meldungsfelder möglich war. Hier das Format der Funktion `InputBox()`:

```
strAntwort = InputBox(strFrage[, strTitel][, strDefault][,
intXpos][, intYpos]
```

Die Funktion `InputBox()` gibt einen Wert mit dem Datentyp `Variant` zurück, den Sie immer als String interpretieren können; Sie können also die Funktion `InputBox()` einem String zuweisen oder sie überall dort einsetzen, wo ein Stringwert verarbeitet werden kann. (Weil der Rückgabewert den Datentyp `Variant` hat, kann er auch einer Steuerelementeigenschaft zugewiesen werden.) Der Rückgabewert stellt die Antwort des Benutzers dar, die dieser in das Eingabefeld eingegeben hat. Nur das erste Argument ist zwingend erforderlich. Die Argumente haben die folgenden Bedeutungen:

- *strFrage* - Dies ist die Meldung, die im Eingabefeld erscheint, beispielsweise eine Frage an den Benutzer. *strFrage* kann bis zu 1024 Zeichen umfassen. Formulieren Sie diese Meldung immer als Frage, so daß der Benutzer weiß, was er in das Eingabefeld eintragen soll.
- *strTitel* - Der Text für die Titelleiste des Eingabefelds. Falls kein Titel angegeben wird, verwendet Visual Basic statt dessen den Projektnamen.
- *strDefault* - Ein Standardwert, der im Eingabefeld erscheint. Ihr Benutzer kann diesen Vorgabewert übernehmen, der zurückgegeben wird, sobald der Benutzer das Eingabefeld schließt, er kann die Antwort aber auch abändern oder eine völlig andere Antwort eingeben. Wenn Sie wissen, welche Antwort der Benutzer voraussichtlich geben wird, stellen Sie Vorgabewerte bereit, die der Benutzer mit **OK** übernimmt.
- *intXpos*, *intYpos* - Die Twip-Koordinate, an der das Eingabefeld im Formular erscheinen soll. Damit bestimmen Sie die Position des Eingabefelds, falls es bestimmte Teile im

Formularfenster nicht überdecken soll. Wenn Sie keine Twip-Koordinaten vorgeben, plaziert Visual Basic das Eingabefeld in der Bildschirmmitte.



Ein Twip ist 1/1440 Inch und 1/567 Zentimeter.

Das oben gezeigte Eingabefeld wird durch die folgende Anweisung erzeugt:

```
strAntwort = InputBox("Geben Sie den Kundennamen ein!", "Namen")
```

Wenn Sie einen Standardwert und eine genaue Bildschirmposition für das Eingabefeld bereitstellen möchten, könnten Sie die folgende Anweisung ausführen:

```
strAntwort = InputBox("Geben Sie den Kundennamen ein!", "Namen",  
"Felix Krull", 500, 750)
```

Es muß eine Möglichkeit geben, zu erkennen, ob der Benutzer die **OK**-Schaltfläche gedrückt hat (oder die Eingabetaste, um damit **OK** als Standardschaltfläche auszulösen), oder ob er die **Abbrechen**-Schaltfläche gedrückt hat. Die Funktion `InputBox()` gibt einen String der Länge Null zurück, also "", wenn der Benutzer auf die **Abbrechen**-Schaltfläche geklickt hat, statt einen Wert einzugeben oder mit **OK** den Vorgabewert zu übernehmen. In Ihrem Code könnten Sie also die folgende Überprüfung verwenden:

```
If (strAnswer <> "") Then  
' Code, der die Eingaben vom Benutzer verarbeitet  
Else  
' Code, der das Abbrechen verarbeitet  
End If
```



Visual Basic unterstützt den speziellen Wert `Empty`, den Sie anstelle von "" verwenden können. Das Schlüsselwort `Empty` macht Ihren Code klarer. Sie können das oben gezeigte If-Konstrukt wie folgt umformulieren:

```
If (strAntwort <> Empty) Then
```

Angenommen, ein Benutzer möchte die Gesamtsumme für bestimmte Abteilungsdaten ermitteln. Sie könnten ihm ein Eingabefeld bereitstellen und den Benutzer nach der betreffenden Abteilung fragen. Nachdem er den Namen eingegeben hat, berechnen Sie die Gesamtsumme. Klickt der Benutzer dagegen auf die **Abbrechen**-Schaltfläche, nimmt das Programm an, er hätte es sich anders überlegt.

Die Tastatur

Ihre Programme können nicht alle Tastaturaktivitäten mit Hilfe von Steuerelementen und Eingabefeldern verarbeiten. Manchmal muß man unmittelbar bei der Eingabe auf einen bestimmten Tastendruck reagieren. Windows übergibt Ihren Anwendungen spezielle Tastaturereignisse, so daß Sie die Eingaben von der Tastatur überwachen können: `KeyPress`, `KeyDown` und `KeyUp`. Diese Ereignisse reagieren auf Tastenkombinationen wie beispielsweise (Alt)+(G) oder (°)+(P), ebenso wie auf einzelne Tasten. Wenn ein Tastaturereignis auftritt, können Sie prüfen, ob eine bestimmte Tastenkombination gedrückt wurde.

Nachdem Ihre Anwendung eine Tastatureingabe erhalten hat, kann sie die Eingabe modifizieren oder die gedrückte Taste ignorieren, falls es sich nicht um eine der unterstützten Tasten handelt. Die Prüfung der Tastatureingabe ist praktisch, um beispielsweise einen Eröffnungsbildschirm zu schließen, Eingaben auszuwerten oder bestimmte Spiele zu spielen.

Über die Tastatur ausgelöste Ereignisse

Das Ereignis `KeyPress` tritt auf, wenn der Benutzer eine Taste drückt, die einem der folgenden Zeichen entspricht:

- Groß- und Kleinbuchstaben
- Ziffern
- Interpunktionszeichen
- Eingabetaste, Tabulatortaste und Backspace.

Das `KeyPress`-Ereignis tritt für die meisten ASCII-Zeichen auf. `KeyPress` prüft nicht alle ASCII-Zeichen (beispielsweise den horizontalen Tabulator, Pfeiltasten und andere spezielle Steuerzeichen, die zwischen den ASCII-Werten 0 und 31 liegen), aber die meisten. Mit `KeyPress` können Sie z.B. genau feststellen, welche Taste gedrückt wurde. Beispielsweise gibt `KeyPress` ein `A` zurück, wenn der Benutzer diese Taste gedrückt hat.



Das `KeyPress`-Ereignis tritt auf, während die Taste gedrückt wird. Wenn der Benutzer die Taste gedrückt hält, tritt das Ereignis immer dann auf, wenn die Tastatur die Zeichen automatisch wiederholt.

Ein Ereignis ist immer einem Objekt zugeordnet, beispielsweise einer Schaltfläche oder dem Formular. Das `KeyPress`-Ereignis gehört immer zu dem Objekt, das den Fokus hatte, als der Benutzer die Taste drückte. Wenn kein Objekt den Fokus hat, wird das `KeyPress`-Ereignis dem Formular zugeordnet. (Abhängig von der `KeyPreview`-Eigenschaft kann hierfür eine Ausnahme auftreten, wie im Abschnitt »Prioritäten« später in diesem Kapitel noch erklärt wird.)



Verwenden Sie das Tastendruckereignis nicht, um einen Shortcut für ein Menü zu erkennen. Der Menü-Editor verwaltet die Shortcuts für Sie und richtet die Reaktion automatisch ein, indem er die `Click`-Ereignisprozedur für das Menüelement auslöst. Wenn Sie Tastendruckereignisse prüfen, kann Ihr Programm nicht auf Menüauswahlen reagieren.

Die `KeyPress`-Ereignisprozedur erhält immer ein Integer-Argument. Wenn Sie also ein `KeyPress`-Ereignis für ein Textfeld schreiben möchten, könnte die Ereignisprozedur wie folgt beginnen und enden:

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
    '
    ' Hier steht Code, der Tastencodes prüft und verarbeitet
    '
End Sub
```

Das Argument `KeyAscii` ist ein Integer, der den ASCII-Code des Zeichens darstellt, das der Benutzer gedrückt hat. Mit einer `If`- oder einer `Select Case`-Anweisung prüfen Sie, ob das

Zeichen den erwarteten Tastencode darstellt.

Eine der wichtigsten Aufgaben, die Sie mit **KeyPress** ausführen, ist die Modifizierung der Tastendrucke eines Benutzers. Das **KeyPress**-Ereignis tritt auf, sobald ein Benutzer die Taste drückt, und bevor ein Steuerelement den Tastencode erhält. Normalerweise zeigt beispielsweise ein Textfeld unmittelbar an, welche Taste der Benutzer gedrückt hat, während es den Fokus hatte. Wenn Sie dagegen eine **KeyPress**-Ereignisprozedur für ein Textfeld entwickeln, kann die **KeyPress**-Ereignisprozedur die Taste modifizieren, wie im folgenden Code gezeigt:

```
Private Sub txtTryIt_KeyPress(KeyAscii As Integer)
' Großes A in kleines B umwandeln
If KeyAscii = 65 Then ' 65 ist der ASCII-Code für A
KeyAscii = 66 ' 66 ist der ASCII-Code für B
End If
End Sub
```

Wenn das Textfeld **txtTryIt** den Fokus hat, nimmt das Textfeld alle Tastencodes entgegen und zeigt sie an, bis der Benutzer ein großes A mit dem ASCII-Wert 65 drückt. Die **If**-Anweisung ändert den **KeyAscii**-Wert in den Buchstaben B (ASCII 66), und das Textfeld zeigt das B an, nicht das A, weil das **KeyPress**-Ereignis den Tastencode erhält, bevor das Textfeld den **KeyAscii**-Wert erhält.



Lesen Sie in der Online-Hilfe von Visual Basic unter dem Stichwort *Tastencode-Konstanten* nach. Die Online-Hilfe beschreibt einige Konstanten, die Sie für die Tastatur prüfen können. Beispielsweise können Sie auf Backspace prüfen, indem Sie **KeyAscii** mit **vbKeyBack** vergleichen, auf die Eingabetaste, indem Sie es mit **vbKeyReturn** vergleichen, oder auf die Tabulatortaste, indem Sie es mit **vbKeyTab** vergleichen. (**KeyPress** prüft neben Buchstaben, Ziffern und Interpunktionszeichen nur auf diese drei Tasten.) Das Textfeld reagiert zwar auf andere Tastencodes wie (Pos1) oder (Ende), aber **KeyPress** reagiert nur auf die Eingabetaste, die Tabulatortaste und die Backspace-Taste zuverlässig.

Während **KeyPress** einen umfassenden Tastencodebereich überprüft, ist das **KeyDown**- Ereignis spezifischer. **KeyDown** tritt auf, wenn der Benutzer eine Taste gedrückt hat, wie **KeyPress**, aber **KeyDown** beschreibt einen detaillierteren - wenn auch etwas komplizierteren - Status Ihrer Tastatur. Beispielsweise gibt **KeyPress** die unterschiedlichen ASCII-Werte für das große T und das kleine t zurück. **KeyDown** gibt dafür denselben Wert zurück, aber außerdem einen Wert, der als Statusargument bezeichnet wird, und der den Status der (ª)-Taste beschreibt.



Das **KeyDown**-Ereignis tritt auf, wenn ein Benutzer eine Taste drückt. **KeyDown** und **KeyPress** können also gleichzeitig auftreten (vorausgesetzt, der Benutzer drückt eine ASCII-Taste).



Verwenden Sie **KeyPress**, wenn Sie auf einen ASCII-Tastencode prüfen möchten, weil **KeyPress** einfacher zu programmieren ist als **KeyDown**.

Hier die öffnenden und schließenden Anweisungen für eine **KeyDown**-Ereignisprozedur:

```
Private Sub txtTryIt_KeyDown(KeyCode As Integer, Shift As Integer)
```



```
'  
' Hier steht der Code zur Verarbeitung von Tastaturereignissen  
'
```

End Sub

KeyCode enthält den Tastencode, das Shift-Argument den Status der Steuerungstasten, wie beispielsweise (ª), (Strg) und (Alt). Der KeyCode entspricht immer dem Großbuchstaben der gedrückten Taste. Wenn der Benutzer also ein kleines t drückt, enthält das KeyCode-Argument den Wert 84 (das ist der ASCII-Wert für ein großes T).

	Seien Sie im Umgang mit KeyDown vorsichtig, weil es verwirrend sein kann, daß die Kleinbuchstaben ignoriert werden. Wenn Sie einen Tastencode für eine Ziffer erhalten, dann <i>müssen</i> Sie das Shift-Argument überprüfen. Wenn Shift anzeigt, daß der Benutzer gleichzeitig mit der Ziffer die (ª)-Taste gedrückt hat, dann wollte er damit nämlich das Sonderzeichen oberhalb der Ziffer eingeben (beispielsweise das & über der 6).

Der wichtigste Vorteil von KeyDown gegenüber von KeyPress ist, daß es zwar kompliziert in Hinblick auf den (ª)-Status ist, aber Sie damit fast jeden Tastencode abfragen können, unter anderem auch die Pfeiltasten, (Pos1), (Ende) usw. Auch hier finden Sie in der Online-Hilfe Informationen über Tastencode-Konstanten, die Visual Basic für diese speziellen Tastencodes verwendet.

Der *Shift-Status* gibt die Taste an - (ª), (Strg), (Alt) oder keine -, die der Benutzer zusammen mit der anderen Taste drückt. Das interne Binärmuster für das Shift-Argument bestimmt die Art des Shift-Status. Um den Shift-Status zu überprüfen, führen Sie eine And-Verknüpfung mit der Zahl 7 aus. (Diese Art des And wird als bitweises And bezeichnet, im Gegensatz zu dem allgemeinen logischen And, das als zusammengesetzter Vergleichsoperator arbeitet.) Der Code in Listing 7.1 ist die Shell, die die Prüfung auf den allgemeinen Shift-Status vornimmt.

Listing 7.1: Sie können Code schreiben, der den Shift-Status überprüft

```
1: Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)  
2: Dim intShiftState As Integer  
3: intShiftState = Shift And 7 ' Spezielles bitweises Und  
4: Select Case intShiftState  
5: Case 1  
6: ' Code für Shift-Kombinationen  
7: Case 2  
8: ' Code für Strg-Kombinationen  
9: Case 3  
10: ' Code für Alt-Kombinationen  
11: Case 4  
12: ' Code für Shift-Strg-Kombinationen  
13: Case 5  
14: ' Code für Shift-Alt-Kombinationen  
15: Case 6  
16: ' Code für Strg-Alt-Kombinationen  
17: Case 7  
18: ' Code für Shift-Strg-Alt-Kombinationen  
19: End Select  
20: End Sub
```


Das **KeyUp**-Ereignis tritt auf, wenn der Benutzer eine gedrückte Taste losläßt. Sie können prüfen, welche Taste losgelassen wurde (beispielsweise das (A), wenn der Benutzer die Hälfte des Tastencodes (^)+(A) losläßt), indem Sie das Argument analysieren, das **KeyUp()** übergeben wird. **KeyUp** tritt also sowohl nach **KeyDown** als auch nach **KeyPress** auf.

Der folgende Code zeigt eine Ereignisprozedur für ein Textfeld. Der Code wandelt die vom Benutzer eingegebenen Kleinbuchstaben im Textfeld in Großbuchstaben um:

```
1: Private Sub txtTry_KeyPress(KeyAscii As Integer)
2: ' Kleinbuchstaben in Großbuchstaben umwandeln
3: If (KeyAscii >= 97) And (KeyAscii <= 122) Then
4: KeyAscii = KeyAscii - 32 'Großbuchstaben erzeugen
5: End If
6: End Sub
```

Der ASCII-Wertebereich für Kleinbuchstaben geht von 97 (für a) bis 122 (für z), wie Sie in Anhang C nachlesen können. Die Differenz zwischen den ASCII-Werten für Kleinbuchstaben und den entsprechenden Großbuchstaben beträgt 32. Wenn die **KeyPress** -Ereignisprozedur einen ASCII-Wert für einen Kleinbuchstaben erhält, subtrahiert die Prozedur 32 von diesem Wert, um ihn in den entsprechenden Großbuchstaben umzuwandeln.



Verwenden Sie die Tastaturereignisse nicht, um ein Textfeld zu erstellen, das die Eingaben des Benutzers durch ein Vorgegebenes Zeichen verbirgt. Drücken Sie (Strg)+(T) und fügen Sie der Werkzeugsammlung das Microsoft Masked Edit Control 6.0 hinzu. (Weitere Informationen darüber, wie Sie der Werkzeugsammlung Werkzeuge hinzufügen, finden Sie in Kapitel 9.) Das Steuerelement Masked Edit ermöglicht Ihnen, Eingabefelder einzurichten, beispielsweise Telefonnummern mit Vorwahlen und automatische Klammern und Bindestriche. Wenn Sie eigene Routinen dafür schreiben, würden Sie das Rad neu erfinden - Zeit, in der Sie sich besser in die Sonne legen.

Tastencodes von Ihrem Programm aus senden

Die **SendKeys**-Anweisung sendet Tastencodes an Ihre Anwendung, so als ob der Benutzer sie eingeben würde. **SendKeys** ist praktisch, um die Platzierung des Textcursors zu steuern, weil Sie damit Tastencodes wie (Pos1) oder (Ende) senden können, um so den Textcursor in einem Textfeld oder einem anderen Steuerelement zur Dateneingabe zu positionieren. Die Syntax von **SendKeys** sieht wie folgt aus:

SendKeys *strTastencodes* [, *blnWait*]

strTastencodes ist häufig eine Stringkonstante, beispielsweise »Widgets, Inc.«, wenn Sie anstelle der Benutzer etwas übergeben möchten. Die Boolesche Option *blnWait* wird in der Regel weggelassen, und wenn sie **False** ist (das ist der Standardwert, wenn Sie *blnWait* nicht spezifizieren), geht die Steuerung an die ausführende Prozedur zurück, sobald die Tastencodes gesendet sind. Wenn *blnWait* gleich **True** ist, verarbeitet das System die Tastencodes, bevor der Code weiterverarbeitet wird, d.h. die Tastencode-Ereignisse sind während der Tasteneingabe aktiv.

Die folgenden Sonderzeichen müssen in geschweifte Klammern eingeschlossen werden ({}), wenn sie mit **SendKeys** versendet werden: Caret (^), Plus (+), Prozent (%), Tilde (~) und Klammern. Um beispielsweise die Eingabe von 7 + 6 zu simulieren, muß die **SendKeys**-Anweisung das Pluszeichen in geschweifte Klammern einbetten:

`SendKeys "7 {+} 6"`

Es gibt mehrere spezielle Tastencodes, wie beispielsweise die Funktionstasten oder (Pos1), für die ein `SendKeys`-Code und die Klammern erforderlich sind. Um beispielsweise den Tastencode für (Pos1) an eine Anwendung zu senden, verwenden Sie die Konstante `{Home}`:

`SendKeys "{Home}"`

Alle diese speziellen Tasten haben Codeäquivalente, die Sie benutzen können. Schlagen Sie `SendKeys` in der Online-Hilfe nach, um zu erfahren, welche Tastencodes für spezielle Tasten definiert sind:



Es ist nicht möglich, mit `SendKeys` den Tastencode `Drucken` an eine Applikation zu senden.

Prioritäten

Wenn Benutzer eine Taste drücken, erhalten das Formular oder das Steuerelement mit dem aktiven Fokus den Tastencode. Wenn kein Steuerelement den Tastencode besitzt, erhält das Formular das Tastencode-Ereignis. Wenn ein Steuerelement den Fokus hat, erhalten dieses Steuerelement oder das Formular den Tastencode, abhängig von dem Inhalt der `KeyPreview`-Eigenschaft des Formulars.

Wenn die `KeyPreview`-Eigenschaft `True` ist, erhält das Formular das Tastencode-Ereignis. Wenn Sie zwei Ereignisprozeduren namens `frmAcct_KeyDown()` und `txtEntry_KeyDown()` codiert haben, und die `KeyPreview`-Eigenschaft des Formulars `True` ist, wird die Ereignisprozedur `frmAcct_KeyDown()` ausgeführt, wenn der Benutzer eine Taste drückt. Wenn die `KeyPreview`-Eigenschaft des Formulars `False` ist, wird `txtEntry_KeyDown()` ausgeführt (vorausgesetzt, das Textfeld hat den aktuellen Fokus).

Zusätzliche Steuerelemente

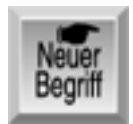
Eingabefelder und Meldungsfelder stellen eine bequeme Methode dar, Informationen anzuzeigen und Eingaben mit Hilfe von Popup-Fenstern entgegenzunehmen, die nach Bedarf angezeigt werden. Die Meldungs- und Eingabefelder sind als Ergänzung zu den Bezeichnungsfeldern und Textfeldern sehr praktisch, weil sie Ausgaben anzeigen und Eingaben entgegennehmen, und zwar auf andere Weise als diese Steuerelemente.

Es gibt aber noch weitere Steuerelemente, die Eingaben entgegennehmen und dem Benutzer ermöglichen, Auswahlen zu treffen. Das restliche Kapitel stellt Ihnen diese anderen Steuerelemente vor. Nachdem Sie dieses Kapitel gelesen haben, kennen Sie viele neue Steuerelemente für Ihre Visual-Basic-Anwendungen.

Kontrollkästchen

Ein Kontrollkästchen ermöglicht dem Benutzer die Auswahl einer Option. Es kann alleine aber auch zusammen mit mehreren anderen Kontrollkästchen angezeigt werden. Wenn ein Kontrollkästchen angeklickt wurde, zeigt es ein Häkchen an (d.h. der Benutzer hat die Option aktiviert, für die das Kontrollkästchen steht). Wenn der Benutzer erneut auf das Kontrollkästchen klickt, wird das Häkchen

wieder entfernt.



Ein *Kontrollkästchen* repräsentiert eine Option auf einem Formular. Es wird markiert, wenn die Option aktiviert ist und umgekehrt. Mit Hilfe von Kontrollkästchen bieten Sie dem Benutzer zweiwertige Auswahlen an, beispielsweise Richtig/Falsch oder An/Aus.

Ein Kontrollkästchen ist markiert oder nicht. Welchen Status es hat, wird in der *Value*-Eigenschaft abgelegt. Ist die *Value*-Eigenschaft gleich 1, ist das Kontrollkästchen selektiert und das Häkchen wird angezeigt. Ist die *Value*-Eigenschaft gleich 0, ist das Kontrollkästchen nicht selektiert und es erscheint kein Häkchen.



Ein einzelnes Kontrollkästchen bietet die Auswahl *True/False*, die durch die *Value*-Eigenschaft 1 oder 0 festgelegt wird. Wenn Sie auf eine Abfrage nur eine Ja/Nein-Antwort brauchen, dann stellen Sie nicht zwei Kontrollkästchen, sondern nur eines zur Verfügung. Kontrollkästchen sind besser für die Anzeige selektierter Optionen geeignet, als für die Beantwortung von Ja/Nein-Fragen. Abbildung 7.7 zeigt ein Formular mit drei Kontrollkästchen. Jedes der Kontrollkästchen kann markiert oder nicht markiert werden, und zwar durch den Benutzer oder durch den Wert, den der Programmierer beim Entwurf in der *Value*-Eigenschaft für das Kontrollkästchen festgelegt hat.



Wenn Sie in der *Caption*-Eigenschaft eine Beschleunigtaste spezifizieren, kann der Benutzer das Kontrollkästchen durch den entsprechenden Tastencode aktivieren oder deaktivieren, beispielsweise (Alt)+(F).



Abbildung 7.7:

Mit Hilfe von Kontrollkästchen können Ihre Benutzer verschiedene Optionen auswählen.

Mit Hilfe einer *If*-Abfrage prüfen Sie, ob ein Kontrollkästchen markiert wurde:

```
If (chkUnder.Value = 1) Then
```

```
' Code, der für das markierte Kontrollkästchen ausgeführt wird
```

```
Else
```

```
' Code, der für das nicht markierte Kontrollkästchen ausgeführt wird
```

```
End If
```

Ihr Formular kann ein, zwei oder mehr Kontrollkästchen beinhalten. Sie können unter Aufwand mühseliger Programmierung sicherstellen, daß zu jedem Zeitpunkt immer nur ein Kontrollkästchen

markiert wird, aber Visual Basic bietet eine bessere Methode, sich wechselseitig ausschließende Optionen bereitzustellen: Optionsfelder. Mehr darüber erfahren Sie im nächsten Abschnitt.

Optionsfelder

Optionsfelder ermöglichen dem Benutzer, genau eine von mehreren Optionen auszuwählen. Anders als Kontrollkästchen kann hier jeweils nur ein Optionsfeld selektiert sein. Abbildung 7.8 zeigt drei Optionsfelder, wovon eines selektiert ist. Wenn der Benutzer auf ein anderes klickt, deselektiert Visual Basic automatisch das zuvor selektierte Optionsfeld und selektiert das jetzt angeklickte Optionsfeld.



Ein *Optionsfeld* bietet dem Benutzer eine Auswahl zwischen verschiedenen Optionen auf einem Formular. Dabei kann jeweils nur ein Optionsfeld auf dem Formular selektiert sein. Sobald Sie ein anderes Optionsfeld anklicken, wird die Markierung für das erste aufgehoben.



Abbildung 7.8:
Ihre Benutzer können jeweils nur ein Optionsfeld selektieren.

Wenn Sie Optionsfelder auf Ihrem Formular anlegen, können Sie für alle in der Ereignisprozedur `Form_Load()` die `Value`-Eigenschaft zur Laufzeit auf `False` setzen. Das Formular wird angezeigt und keines der Optionsfelder ist selektiert. Nachdem der Benutzer eine Option ausgewählt hat, erscheint diese selektiert, bis der Benutzer ein anderes Optionsfeld anklickt. Wenn Sie der `Caption`-Eigenschaft für ein Optionsfeld eine Beschleunigtaste hinzufügen, kann es auch durch die entsprechende Tastenkombination selektiert werden.



Legen Sie nie nur ein einziges Optionsfeld auf einem Formular an, weil der Benutzer dieses zwar selektieren, aber nie mehr deselektieren kann.

Optionen im Rahmen gruppieren

Technisch betrachtet, kann der Benutzer mehrere Optionen gleichzeitig auswählen, wenn diese sich in separaten *Rahmen* befinden. Mit Hilfe von Rahmen gruppieren Sie Optionsfelder, so daß der Benutzer je ein Optionsfeld aus den verschiedenen Gruppen auswählen kann.



Ein *Rahmen*, manchmal auch als *Container-Steuererelement* bezeichnet, enthält Steuerelemente auf einer Ebene, die sich vom Formular als solchem unterscheidet. Sie können zwar jeweils nur eine Optionsfeld-Gruppe auf einem Formular anlegen, aber es ist möglich, mehrere solcher Gruppen in verschiedenen Rahmen anzulegen. Rahmen können aber mehr als nur Optionsfelder gruppieren. Ein Rahmen kann jedes beliebige Steuerelement aufnehmen, das Sie mit anderen Steuerelementen visuell gruppieren möchten.

Abbildung 7.9 zeigt eine Applikation mit zwei selektierten Optionsfeldern. Das ist möglich, weil die Optionsfelder im Formular und im Rahmen plazierte sind. Ohne den Rahmen könnte jeweils nur eines der fünf Optionsfelder selektiert werden.

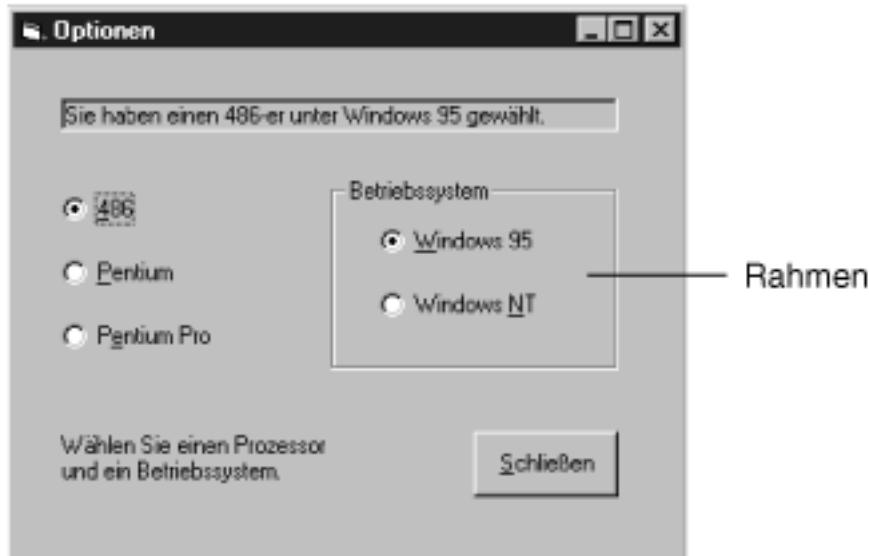


Abbildung 7.9:

Wenn Sie eine Gruppe Optionsfelder in einem Rahmen anlegen, kann der Benutzer ein Optionsfeld im Rahmen und eines im Formular selektieren.

Was Sie tun sollten

Verwenden Sie so viele Rahmen, wie Sie brauchen, um mehrere Gruppen mit Optionsfeldern auf einem Formular anzulegen.

Es ist ganz einfach, Rahmen anzulegen. Die folgenden Eigenschaften sind für die Verwendung von Rahmen interessant:

- **BorderStyle** - 0 - Kein oder 1 - Fest einfach. Wie jede Eigenschaft mit solchen Werten können Sie die **BorderStyle**-Eigenschaft sowohl zur Laufzeit als auch zur Entwurfszeit setzen, indem Sie entweder eine Zuweisung vornehmen oder den Wert im Eigenschaftenfenster auswählen. Wenn Sie den Wert 0 verwenden, ist der Rahmen unsichtbar und enthält keinen Titel und keine Linie, die ihn von dem Formular abheben. Ein Rahmen, der keinen Rahmen hat, kann zwar Optionsfelder gruppieren, aber Ihr Benutzer hat es schwer, zu erkennen, daß es vielleicht mehrere Optionsfeldgruppen auf dem Formular gibt.
- **Caption** - Der Text, der oben im Rahmen erscheint.
- **Font** - Legt die Schriftattribute für die Werte in der **Caption**-Eigenschaft fest.

Im nächsten Bonus-Projekt, »Variablen und Ausdrücke«, wird ein vollständiges Projekt entwickelt, in

dem ein Rahmen Optionsfelder aufnimmt. Sie werden lernen, wie man Steuerelemente in einem Rahmen platziert, nämlich indem man die Optionsfelder des Formulars innerhalb des Rahmens anlegt. Auf diese Weise erkennt Visual Basic, daß die Optionsfelder nicht Teil der Optionsfeldgruppe des Formulars, sondern der des Rahmens sind.

Zusammenfassung

In diesem Kapitel haben Sie die internen Funktionen von Visual Basic kennengelernt. Sie werden in diesem Buch noch mehr über interne Funktionen erfahren. Visual Basic unterstützt diese Funktionen, so daß Sie sie nur noch aufrufen, ihnen Argumente übergeben und die Rückgabewerte verarbeiten, ohne dieselbe Aufgabe mühselig selbst programmieren zu müssen.

Die Funktion `MsgBox( )` zeigt Ihrem Benutzer Meldungen in Popup-Fenstern an. Der Benutzer reagiert darauf, indem er auf eine Schaltfläche klickt. Der Benutzer kann dadurch entscheiden, ob ein bestimmter Prozeß fortgesetzt oder abgebrochen werden soll. Die Funktion `InputBox( )` dagegen stellt dem Benutzer Fragen und ermittelt Antworten, die Sie in Ihrem Programm weiterverwenden.

Andere Steuerelemente, wie beispielsweise Kontrollkästchen, Optionsfelder oder in einem Rahmen gruppierte Optionsfelder, bieten Ihren Benutzern eine Möglichkeit, Ihrem Programm bestimmte Informationen bereitzustellen. Anhand der selektierten Steuerelemente erkennt das Programm, welche Aktionen der Benutzer ausführen will.

In Kapitel 8 erfahren Sie mehr über die Struktur eines Visual-Basic-Programms. Sie lernen, was lokale und globale Variablen sind. Darüber hinaus lernen Sie fast alle weiteren internen Funktionen kennen, die Sie je brauchen.

Fragen & Antworten

Frage:

Warum kann eine interne Funktion nur einen Wert zurückgeben?

Antwort:

Eine interne Funktion wird zu ihrem Rückgabewert. Mit anderen Worten, sie verhält sich wie ein Ausdruck, der einen einzigen Wert produziert. Häufig übergeben Sie einer internen Funktion einen oder mehrere Werte, die diese irgendwie verarbeiten oder kombinieren soll. Der Rückgabewert stellt das Ergebnis der Verarbeitung oder der Kombination dar. Beispielsweise nimmt die Funktion `LoadPicture()` ein String-Argument entgegen, das den Pfadnamen für ein Bild enthält; der Rückgabewert ist das eigentliche Bild an dieser Position, das Sie einer grafischen Eigenschaft eines Steuerelements zuweisen können.

Interne Funktionen können überall dort eingesetzt werden, wo auch ihr Rückgabewert eingesetzt werden kann. Statt also eine String-Konstante oder -Variable in einem Meldungsfeld anzuzeigen, können Sie für den Stringwert im Meldungsfeld auch die Funktion `InputBox( )` verwenden. Auf diese Weise verschachteln Sie zwei Funktionen. Die innere Funktion, `InputBox( )`, wird zuerst ausgeführt. Sie ermittelt einen String vom Benutzer. Die Funktion `MsgBox( )` zeigt diesen String in einem Meldungsfeld an, beispielsweise wie folgt:

```
intAntw = MsgBox(InputBox("Wie ist Ihr Name?"))
```

Wie oft betten Sie eine `InputBox( )`-Funktion in eine `MsgBox( )`-Funktion ein? Vermutlich nie,

aber diese Zuweisung demonstriert, wie eine interne Funktion, in diesem Fall `InputBox()`, ihren Rückgabewert erzeugt, so daß ein anderer Code diesen unmittelbar weiterverarbeiten kann.

Frage:

Welche anderen internen Funktionen gibt es?

Antwort:

Es gibt interne Funktionen für Zahlen, Strings und andere Datentypen. Mehr darüber erfahren Sie in Kapitel 8.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist der Unterschied zwischen einem Argument und einer internen Funktion?
2. Richtig/Falsch. Sie können eine Standardschaltfläche für ein Meldungsfeld vorgeben.
3. Welches äquivalente Schlüsselwort gibt es für die leere Stringkonstante, " "?
4. Richtig/Falsch. Die Tabellen 7.1, 7.2 und 7.3 beschreiben verschiedene Argumente für die Funktion `MsgBox()`.
5. Was zeigt Visual Basic in der Titelleiste von Meldungs- und Eingabefeldern an, wenn Sie kein entsprechendes Argument vorgeben?
6. Was ist der wichtigste Unterschied zwischen einem Kontrollkästchen und einem Optionsfeld?
7. Richtig/Falsch. Man kann Optionsfelder auf einem Formular anzeigen, von denen keines selektiert ist.
8. Welcher Eigenschaftswert bestimmt, ob ein Kontrollkästchen selektiert ist?
9. Welcher Eigenschaftswert bestimmt, ob ein Optionsfeld selektiert ist?
10. Warum braucht man manchmal einen Rahmen, wenn man Optionsfelder auf einem Formular platziert?

Übungen

1. Beschreiben Sie, wie Ihr Code feststellt, ob der Benutzer einen Wert in ein Eingabefeld eingegeben hat (oder vielleicht den vorgegebenen Standardwert übernommen hat), oder ob er auf die **Abbrechen**-Schaltfläche geklickt hat.
2. Schreiben Sie die `MsgBox()`-Funktion für das in Abbildung 7.10 gezeigte Meldungsfeld.

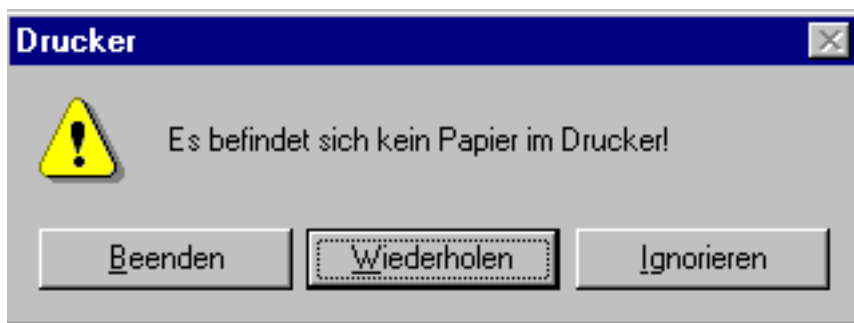


Abbildung 7.10:
Wie würden Sie dieses Meldungsfeld erzeugen?

3. Schreiben Sie eine Ereignisprozedur für eine Schaltfläche, die den Benutzer in zwei separaten Eingabefeldern auffordert, eine Stadt und ein Land einzugeben. Anschließend verknüpfen Sie die Namen, indem Sie ein Komma und ein Leerzeichen eingefügt haben, um Stadt und Land in einem Meldungsfeld anzuzeigen.
4. Schreiben Sie eine Applikation mit einem langen Formular, das fünf Schaltflächen enthält, die oben im Formular Optionsfelder simulieren. Geben Sie diesen fünf Schaltflächen die Namen der fünf Radiosender, die Sie am liebsten hören. Entwickeln Sie eine Ereignisprozedur für die einzelnen Schaltflächen, die anzeigt, welche Musik und welche Informationen diese Station überträgt. Zeigen Sie die Information in einem Meldungsfeld an.

Woche 1

Bonus-Projekt 3: Benutzereingaben und bedingte Logik

Der Code in diesem Bonusprojekt demonstriert die Verwendung von Kontrollkästchen, Optionsfeldern und Rahmen, so daß Sie den Umgang mit den in den vorigen Kapiteln vorgestellten Steuerelementen einüben können.

Für die Verwaltung so vieler Steuerelemente ist Visual-Basic-Code erforderlich. In diesem Bonusprojekt sehen Sie mehr Code, als Ihnen in den vergangenen Kapiteln begegnet ist. Hier die Aufgaben der Applikation:

- Sie zeigt mehrere Kontrollkästchen für den Benutzer an, so daß dieser ein oder mehrere Länder auswählen kann, um neben den Namen auch die Flaggen anzuzeigen.
- Sie zeigt dieselben Ländernamen unter Verwendung von Optionsfeldern an, so daß der Benutzer jeweils nur ein Land gleichzeitig auswählen kann.
- Sie zeigt in einem Meldungsfeld einen Fehler an, wenn der Benutzer nicht die richtige Eingabe bereitgestellt hat.
- Sie zeigt neben den Länder-Optionsfeldern eine weitere Optionsfeldgruppe an, in der der Benutzer festlegt, ob die Flagge in einem kleinen oder in einem großen Anzeigefeld dargestellt werden soll.
- Hier erfahren Sie nicht nur mehr über den Umgang mit den Steuerelementen, sondern lernen gleichzeitig ein neues Konzept kennen: mehrere Formulare in einem einzigen Projekt. Diese Applikation verwendet insgesamt drei Befehle. Sie werden lernen, zur Laufzeit jeweils das richtige Formular anzuzeigen.



Wie viele Applikationen in diesem Buch verwendet auch dieses Projekt Grafikdateien, die Sie zusammen mit Visual Basic erhalten haben. Abhängig davon, wie Sie Ihre Installation vorgenommen haben, haben Sie den Graphics-Ordner möglicherweise nicht in Ihrem Visual-Basic-Verzeichnis. In diesem Fall ändern Sie den Pfadnamen so, daß er auf Ihr CD-ROM-Laufwerk verweist, und legen Sie die erste Installations-CD für VB6 ein. Wenn Sie den Graphics-Ordner anlegen möchten, legen Sie die erste Visual- Basic-CD in das Laufwerk ein und wählen auf dem Bildschirm die Option **Hinzufügen/Ändern**, um Ihrer Festplatte die Grafik hinzuzufügen.

Das erste Formular

Abbildung BP3.1 zeigt das erste Formular, das Sie in diesem Bonusprojekt anlegen.

Tabelle BP3.1: *Legen Sie diese Steuerelemente auf Ihrem Formular an und setzen Sie die entsprechenden Eigenschaften.*

Steuerelement und Eigenschaft	Eigenschaftswert
Formular Name	frmSelect
Formular Caption	Flaggen-Auswahl
Formular Height	4035
Formular Width	6390
Bezeichnungsfeld Name	lblFlags
Bezeichnungsfeld BorderStyle	1 - Fest Einfach
Bezeichnungsfeld Caption	Flaggen
Bezeichnungsfeld Font	MS Sans Serif
Bezeichnungsfeld Schriftgröße	24
Bezeichnungsfeld Schriftschnitt	Fett
Bezeichnungsfeld Height	615
Bezeichnungsfeld Left	2400
Bezeichnungsfeld Top	600
Bezeichnungsfeld Width	1335
Optionsfeld #1 Name	optCheck
Optionsfeld #1 Caption	&Kontrollkästchen
Optionsfeld #1 Left	2280
Optionsfeld #1 Top	1920
Optionsfeld #1 Width	1575
Optionsfeld #2 Name	optOption
Optionsfeld #2 Caption	&Optionsfelder
Optionsfeld #2 Left	2280
Optionsfeld #2 Top	2520
Optionsfeld #2 Width	1695
Befehlsschaltfläche #1 Name	cmdSelect
Befehlsschaltfläche #1 Caption	Klicken Sie hier, wenn Sie eine Auswahl getroffen haben

Befehlsschaltfläche #1 Left	4560
Befehlsschaltfläche #1 Top	2040
Befehlsschaltfläche #2 Name	cmdExit
Befehlsschaltfläche #2 Caption	&Beenden
Befehlsschaltfläche #2 Left	4560
Befehlsschaltfläche #2 Top	2760



Abbildung BP3.1:

In diesem Formular wählt der Benutzer aus, welche Flagge angezeigt werden soll.

Der Code für dieses Formular ist zwar einfach, führt aber ein neues Konzept ein. Listing BP3.1 zeigt, wie Sie ein anderes Formular laden als das, das sich gerade auf dem Bildschirm befindet.

Listing BP3.1: Mit Hilfe der Optionsfelder stellen Sie fest, was der Benutzer angefordert hat

```

1: Private Sub cmdSelect_Click()
2: ' Fehlerüberprüfung und Anzeige des
3: ' vom Benutzer ausgewählten Formulars
4: Dim strMsg As String ' Rückgabewert des Meldungsfelds
5: If ((optCheck.Value = False) And (optOption.Value = False)) Then
6: strMsg = MsgBox("Wählen Sie eine andere Option", _
vbCritical, "Error!")
7: ElseIf (optCheck.Value = True) Then
8: frmFlagsCheck.Show ' Flaggen mit Kontrollkästchen
9: Else
10: frmFlagsOpt.Show ' Flaggen mit Optionsfeldern
11: End If
12: End Sub

```

```
13:
14: Private Sub Form_Load()
15: ' Alle Optionsfelder löschen
16: optCheck.Value = False
17: optOption.Value = False
18: End Sub
19: Private Sub cmdExit_Click()
20: ' Programm beenden
21: End
22: End Sub
```

Analyse des Startformulars

Die Zeilen 14 bis 18 legen fest, was beim Start der Applikation passiert, wenn das erste Formular geladen wird. (Das Dialogfeld **Projekt, Eigenschaften** sollte jetzt das Formular `frmSelect` als Startformular anzeigen.) Die Zeilen 16 und 17 setzen die Optionsfelder auf `False`. Dadurch wird der Benutzer gezwungen, eine davon auszuwählen.

Wenn der Benutzer auf die Schaltfläche klickt, ohne ein Optionsfeld ausgewählt zu haben, wird das in der zusammengesetzten bedingten Anweisung in Zeile 5 festgestellt. Solange die `Value`-Eigenschaft der beiden Optionsfelder `False` ist, zeigt Zeile 6 ein Meldungsfeld an, das den Benutzer darauf hinweist, daß er eine Auswahl treffen muß, bevor er auf die Schaltfläche klickt.

Wenn der Benutzer eines der Optionsfelder ausgewählt hat, stellen die Zeilen 7 bis 9 fest, welche Schaltfläche ausgewählt wurde, und das entsprechende Formular wird angezeigt. Beachten Sie, daß die Zeile das Formular mit `frmFlagsCheck.Show` aufruft. Das sieht nicht wie ein Befehl aus, sondern vielmehr wie ein Eigenschaftswert namens `Show`. Es gibt jedoch für Formulare keine Eigenschaft namens `Show`. `Show` ist eine *Methode*. Eine Methode ist kein Visual-Basic-Befehl (wie etwa `Next`), sondern ein Befehl, den Sie nur auf ein bestimmtes Objekt anwenden können. In diesem Fall ist das Objekt das Formular `frmFlagsCheck`. Die `Show`-Methode zeigt das Formular an, für das Sie diese Aufrufen. Sobald die Zeilen 8 oder 10 ausgeführt werden, sieht der Benutzer das entsprechende Formular auf dem Bildschirm.

Das Formular mit den Kontrollkästchen

Abbildung BP3.2 zeigt das nächste Formular, das Sie entwerfen werden.

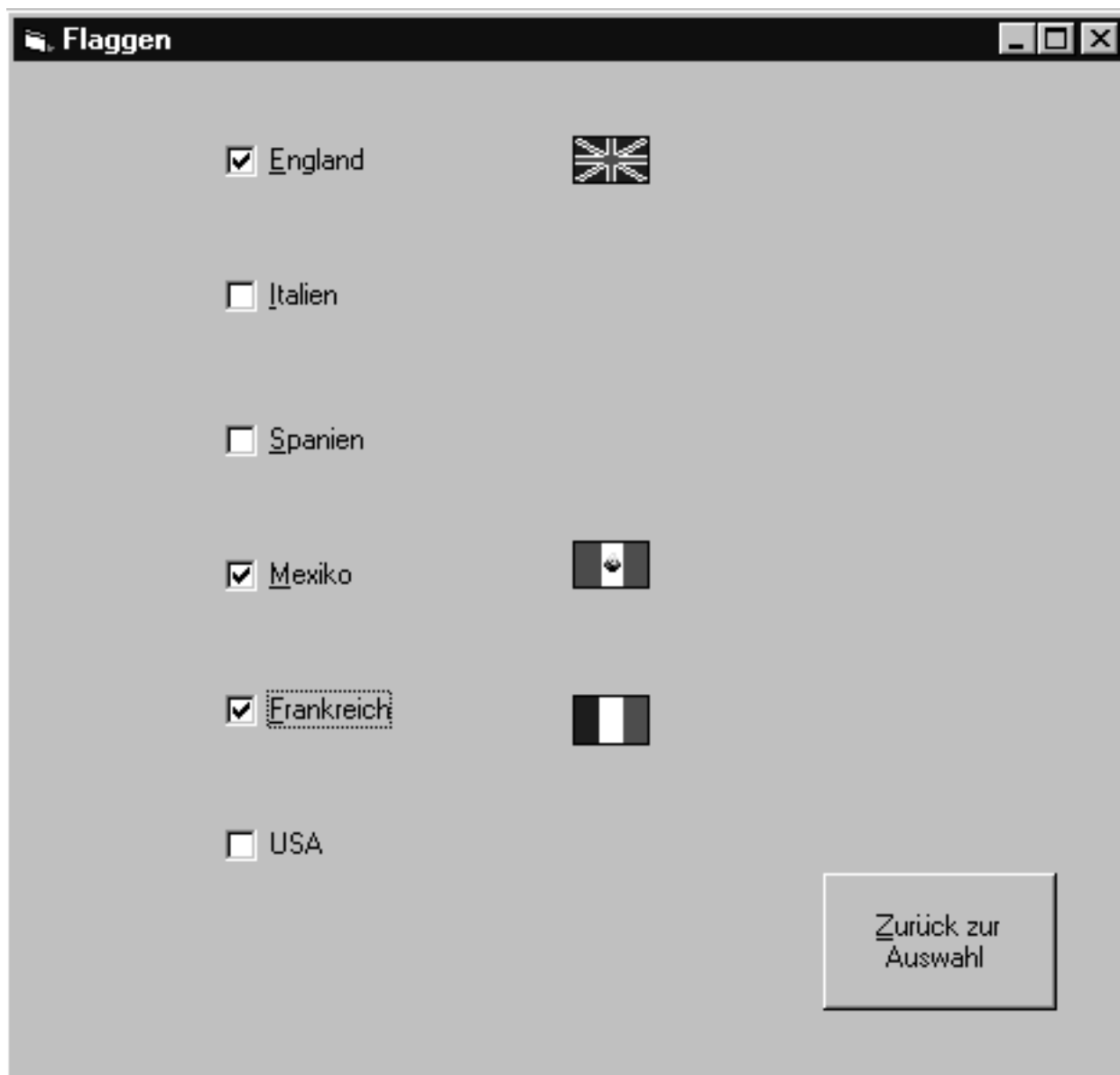


Abbildung BP3.2:
Der Benutzer kann die Flaggen verschiedener Länder anzeigen.

Das Formular enthält sechs Ländernamen, neben denen sechs Flaggen angezeigt werden können. Legen Sie ein neues Formular an und fügen Sie es dem aktuellen Projekt hinzu. Dazu gehen Sie wie folgt vor:

1. Wählen Sie in der Menüleiste **Projekt, Formular hinzufügen**.
2. Ein Dialogfeld mit Registerkarten wird angezeigt. Dort können Sie ein neues oder ein bereits existierendes Formular auswählen.
3. Doppelklicken Sie auf das Icon **Formular**, weil Sie für diese Applikation ein neues Formular brauchen. Das Formular erscheint innerhalb eines Arbeitsbereichs auf dem Bildschirm.

Tabelle BP3.2 listet die Eigenschaften für die Formularelemente auf. Beachten Sie, daß Sie den Pfadnamen für die **Picture**-Eigenschaften des Anzeigefelds so ändern müssen, daß Sie damit die Bilder auf Ihrem Computer ansprechen (möglicherweise auf Ihrer Visual Basic-CD, falls Sie den Ordner Graphics nicht installiert haben).

Tabelle BP3.2: Legen Sie diese Steuerelemente an und setzen Sie ihre Eigenschaften.

Eigenschaft des Steuerelements	Eigenschaftswert
Formular Name	frmFlagsCheck
Formular Caption	Flags
Formular Height	7035
Formular Width	7710
Kontrollkästchen #1 Name	chkEngland
Kontrollkästchen #1 Caption	&England
Kontrollkästchen #1 Left	2835
Kontrollkästchen #1 Top	420
Kontrollkästchen #2 Name	chkItaly
Kontrollkästchen #2 Caption	&Italien
Kontrollkästchen #2 Height	495
Kontrollkästchen #2 Left	2835
Kontrollkästchen #2 Top	1155
Kontrollkästchen #2 Width	1215
Kontrollkästchen #3 Name	chkSpain
Kontrollkästchen #3 Caption	&Spanien
Kontrollkästchen #3 Height	495
Kontrollkästchen #3 Left	2835
Kontrollkästchen #3 Top	1905
Kontrollkästchen #3 Width	1215

Kontrollkästchen #4 Name	chkMexico
Kontrollkästchen #4 Caption	&Mexiko
Kontrollkästchen #4 Height	495
Kontrollkästchen #4 Left	2835
Kontrollkästchen #4 Top	2595
Kontrollkästchen #4 Width	1215
Kontrollkästchen #5 Name	chkFrance
Kontrollkästchen #5 Caption	&Frankreich
Kontrollkästchen #5 Height	495
Kontrollkästchen #5 Left	2835
Kontrollkästchen #5 Top	3375
Kontrollkästchen #5 Width	1215
Kontrollkästchen #7 Name	chkUSA
Kontrollkästchen #7 Caption	&USA
Kontrollkästchen #7 Height	495
Kontrollkästchen #7 Left	2865
Kontrollkästchen #7 Top	4140
Kontrollkästchen #7 Width	1215
Anzeigefeld #1 Name	imgEngland
Anzeigefeld #1 Height	480

Anzeigefeld #1 Left	4440
Anzeigefeld #1 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flaguk
Anzeigefeld #1 Top	480
Anzeigefeld #1 Visible	False
Anzeigefeld #2 Name	imgItaly
Anzeigefeld #2 Height	480
Anzeigefeld #2 Left	4440
Anzeigefeld #2 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgitaly
Anzeigefeld #2 Top	1155
Anzeigefeld #2 Visible	False
Anzeigefeld #3 Name	imgSpain
Anzeigefeld #3 Height	480
Anzeigefeld #3 Left	4440
Anzeigefeld #3 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgspain
Anzeigefeld #3 Top	1890
Anzeigefeld #3 Visible	False
Anzeigefeld #4 Name	imgMexico
Anzeigefeld #4 Height	480
Anzeigefeld #4 Left	4440
Anzeigefeld #4 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgmex
Anzeigefeld #4 Top	2520
Anzeigefeld #4 Visible	False
Anzeigefeld #5 Name	imgFrance
Anzeigefeld #5 Height	480

Anzeigefeld #5 Left	4440
Anzeigefeld #5 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgfran
Anzeigefeld #5 Top	3315
Anzeigefeld #5 Visible	False
Anzeigefeld #6 Name	imgUSA
Anzeigefeld #6 Height	480
Anzeigefeld #6 Left	4440
Anzeigefeld #6 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgusa02
Anzeigefeld #6 Top	4080
Anzeigefeld #6 Visible	False
Befehlsschaltfläche Name	cmdReturn
Befehlsschaltfläche Caption	&Zurück zur Auswahl
Befehlsschaltfläche Left	5520
Befehlsschaltfläche Top	5040

Jetzt fügen Sie den Code für das Formular ein. Doppelklicken Sie auf das Formular frmFlagsCheck und geben Sie den in Listing BP3.2 aufgelisteten Code ein. Dieses Formular soll ein Bild einer Flagge anzeigen, wenn der Benutzer ein Kontrollkästchen anklickt. Sie müssen also für die Click-Ereignisse aller Kontrollkästchen eine Prozedur schreiben.

Listing BP3.2: Wenn der Benutzer ein Kontrollkästchen anklickt, wird eine Flagge angezeigt.

```

1: Private Sub chkEngland_Click()
2: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
3: If chkEngland.Value = 1 Then
4: imgEngland.Visible = True
5: Else
6: imgEngland.Visible = False
7: End If
8: End Sub
9: Private Sub chkItaly_Click()
10: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
11: If chkItaly.Value = 1 Then

```

```

12: imgItaly.Visible = True
13: Else
14: imgItaly.Visible = False
15: End If
16: End Sub
17: Private Sub chkSpain_Click()
18: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
19: If chkSpain.Value = 1 Then
20: imgSpain.Visible = True
21: Else
22: imgSpain.Visible = False
23: End If
24: End Sub
25: Private Sub chkMexico_Click()
26: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
27: If chkMexico.Value = 1 Then
28: imgMexico.Visible = True
29: Else
30: imgMexico.Visible = False
31: End If
32: End Sub
33: Private Sub chkFrance_Click()
34: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
35: If chkFrance.Value = 1 Then
36: imgFrance.Visible = True
37: Else
38: imgFrance.Visible = False
39: End If
40: End Sub
41: Private Sub chkUSA_Click()
42: ' Flagge anzeigen, wenn das Kontrollkästchen markiert ist
43: If chkUSA.Value = 1 Then
44: imgUSA.Visible = True
45: Else
46: imgUSA.Visible = False
47: End If
48: End Sub
49: Private Sub cmdReturn_Click()
50: ' Zurück zum Auswahlformular
51: frmFlagsCheck.Hide
52: frmSelect.Show
53: End Sub

```

Analyse für das Kontrollkästchen-Formular

Die Ereignisprozeduren sind für alle sechs Kontrollkästchen ähnlich. Sie haben die **Picture**-Eigenschaft der Bildfelder bereits beim Entwurf des Formulars mit seinen Steuerelementen gesetzt. Die Ereignisprozedur muß also nur noch die **Visible**-Eigenschaft auf **True** setzen, so daß

das Bild angezeigt wird. Es gibt jedoch ein Problem: Was ist, wenn der Benutzer noch einmal auf das Kontrollkästchen klickt, um die Markierung aufzuheben? Der Code muß die Anzeige des Bilds *ausschalten*.

Die Zeilen 43 bis 46 sind ein Beispiel dafür, wie der Code in den Ereignisprozeduren funktioniert. Zeile 43 wertet die `Value`-Eigenschaft des Kontrollkästchens aus. Enthält sie den Wert 1, hat der Benutzer auf das Kontrollkästchen geklickt und es markiert. Der Code aktiviert die Bildanzeige. Wenn `Value` gleich 0 ist, hat der Benutzer die Markierung des Kontrollkästchens aufgehoben. Der Code muß also die Bildanzeige ausschalten, was in Zeile 46 erfolgt.

Die `Click`-Ereignisprozedur für die Befehlsschaltfläche führt in den Zeilen 51 und 52 zwei Aufgaben aus. Zeile 51 zeigt eine neue Methode, `Hide`, die das Formular verbirgt, auf das Sie sie anwenden. (`Hide` ist das Gegenteil von `Show`.) Zeile 51 verbirgt das Formular mit den Kontrollkästchen, Zeile 52 zeigt das Startformular wieder an, so daß der Benutzer eine andere Option auswählen kann.

Das Formular mit den Optionsfeldern

Abbildung BP3.3 zeigt das nächste Formular, das Sie für die Applikation entwerfen. Abhängig von den vom Benutzer ausgewählten Optionen kann die Flagge groß oder klein angezeigt werden.



Beachten Sie, daß die Flagge hier groß dargestellt wird

Abbildung BP3.3:

Auf diesem Formular kann der Benutzer nur jeweils eine Flagge anzeigen und die Anzeigegröße festlegen.

Das Formular enthält sechs Optionsfelder neben sechs Ländernamen, und neben den Ländernamen kann eine von sechs Flaggen angezeigt werden. Darüber hinaus gibt es einen Rahmen mit zwei weiteren Optionsfeldern, die die Größe der angezeigten Flagge bestimmen.



Um einem Rahmen Optionsfelder hinzuzufügen, müssen Sie diese *auf* dem Rahmen zeichnen. Mit anderen Worten, wenn Sie auf das Werkzeug für die Optionsfelder doppelklicken und ein Optionsfeld in der Formularmitte erscheint, nimmt Visual Basic nicht an, daß dieses innerhalb des Rahmens angelegt werden soll. Um die Optionsfelder eines Rahmens von denen des Formulars abzutrennen, müssen Sie das Optionsfeld zeichnen, indem Sie in der Werkzeugsammlung *einmal* auf das entsprechende Werkzeug klicken und das neue Optionsfeld dann innerhalb des Rahmens aufspannen. Dieses Optionsfeld und alle anderen, die Sie auf diese Weise im Rahmen plazieren, werden als Teil des Rahmens betrachtet und von den anderen Optionsfeldern unterschieden.

Legen Sie ein neues Formular an und fügen Sie es dem aktuellen Projekt hinzu. Nachdem Sie das Formular eingefügt haben (das dritte Formular in diesem Projekt), weisen Sie ihm die in Tabelle BP3.3 aufgelisteten Steuerelemente und Eigenschaften zu.

Tabelle BP3.3: Mit den Optionsfeldern werden die Flaggen auf dem Formular anders angezeigt.

Eigenschaft des Steuerelements	Eigenschaftswert
Formular Name	frmFlagsOpt
Formular Caption	Flags
Formular Height	7335
Formular Width	8955
Optionsfeld #1 Name	optEngland
Optionsfeld #1 Caption	&England
Optionsfeld #1 Height	495
Optionsfeld #1 Left	2760
Optionsfeld #1 Top	360
Optionsfeld #1 Value	True
Optionsfeld #1 Width	1215
Optionsfeld #2 Name	optItaly
Optionsfeld #2 Caption	&Italien
Optionsfeld #2 Height	495
Optionsfeld #2 Left	2760
Optionsfeld #2 Top	1080

Optionsfeld #2 Width	1215
Optionsfeld #3 Name	optSpain
Optionsfeld #3 Caption	&Spanien
Optionsfeld #3 Height	495
Optionsfeld #3 Left	2760
Optionsfeld #3 Top	1800
Optionsfeld #3 Width	1215
Optionsfeld #4 Name	optMexico
Optionsfeld #4 Caption	&Mexiko
Optionsfeld #4 Height	495
Optionsfeld #4 Left	2760
Optionsfeld #4 Top	2520
Optionsfeld #4 Width	1215
Optionsfeld #5 Name	optFrance
Optionsfeld #5 Caption	&Frankreich
Optionsfeld #5 Height	495
Optionsfeld #5 Left	2760
Optionsfeld #5 Top	3240
Optionsfeld #5 Width	1215
Optionsfeld #6 Name	optUSA
Optionsfeld #6 Caption	&USA
Optionsfeld #6 Height	495
Optionsfeld #6 Left	2760
Optionsfeld #6 Top	3960

Optionsfeld #6 Width	1215
Rahmen Name	fraSize
Rahmen Caption	Flaggengröße
Rahmen Height	1215
Rahmen Left	1320
Rahmen Top	5040
Rahmen Width	1575
Rahmen Optionsfeld #1 Name	optLarge
Rahmen Optionsfeld #1 Caption	&Groß
Rahmen Optionsfeld #1 Height	255
Rahmen Optionsfeld #1 Left	360
Rahmen Optionsfeld #1 Top	360
Rahmen Optionsfeld #1 Width	1095
Rahmen Optionsfeld #2 Name	optSmall
Rahmen Optionsfeld #2 Caption	&Klein
Rahmen Optionsfeld #2 Height	255
Rahmen Optionsfeld #2 Left	360
Rahmen Optionsfeld #2 Top	720
Rahmen Optionsfeld #2 Width	1095
Anzeigefeld #1 Name	imgEngland
Anzeigefeld #1 Height	480
Anzeigefeld #1 Left	5280

Anzeigefeld #1 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flguk
Anzeigefeld #1 Stretch	True
Anzeigefeld #1 Top	2160
Anzeigefeld #1 Visible	True
Anzeigefeld #2 Name	imgItaly
Anzeigefeld #2 Height	480
Anzeigefeld #2 Left	5280
Anzeigefeld #2 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flg-italy
Anzeigefeld #2 Stretch	True
Anzeigefeld #2 Top	2160
Anzeigefeld #2 Visible	False
Anzeigefeld #3 Name	imgSpain
Anzeigefeld #3 Height	480
Anzeigefeld #3 Left	5280
Anzeigefeld #3 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flg-spain
Anzeigefeld #3 Stretch	True
Anzeigefeld #3 Top	2160
Anzeigefeld #3 Visible	False
Anzeigefeld #4 Name	imgMexico
Anzeigefeld #4 Height	480
Anzeigefeld #4 Left	5280
Anzeigefeld #4 Picture	\Program Files\Microsoft Visual Studio\Common\Gra phics\Icons\Flags\Flg\Flgmex
Anzeigefeld #4 Stretch	True

Anzeigefeld #4 Top	2160
Anzeigefeld #4 Visible	False
Anzeigefeld #5 Name	imgFrance
Anzeigefeld #5 Height	480
Anzeigefeld #5 Left	5280
Anzeigefeld #5 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgfran
Anzeigefeld #5 Stretch	True
Anzeigefeld #5 Top	2160
Anzeigefeld #5 Visible	False
Anzeigefeld #6 Name	imgUSA
Anzeigefeld #6 Height	480
Anzeigefeld #6 Left	5280
Anzeigefeld #6 Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgusa02
Anzeigefeld #6 Stretch	True
Anzeigefeld #6 Top	2160
Anzeigefeld #6 Visible	False
Befehlsschaltfläche Name	cmdReturn
Befehlsschaltfläche Caption	&Zurück zur Auswahl
Befehlsschaltfläche Height	495
Befehlsschaltfläche Left	4920
Befehlsschaltfläche Top	5400
Befehlsschaltfläche Width	1215

Listing BP3.3 zeigt den Code für das Formularmodul mit den Optionsschaltflächen. Und er ist ganz schön lang. Sie werden jedoch sehen, daß sich dieser Code immer wieder wiederholt und hauptsächlich aus sechs ähnlichen Routinen besteht, die den sechs Optionsfeldern zuzuordnen sind.

Listing BP3.3: Fügen Sie den Code ein, der die Flaggenanzeige mit den Optionsfeldern steuert

```
1: Private Sub optEngland_Click()  
2: ' Flagge anzeigen, wenn das Optionsfeld markiert ist  
3: If optSmall.Value = True Then  
4: imgEngland.Height = 480  
5: imgEngland.Width = 480  
6: Else ' Großes Bild  
7: imgEngland.Height = 2800  
8: imgEngland.Width = 2800  
9: End If  
10: imgEngland.Visible = True  
11: ' Anzeige aller anderen Flaggen deaktivieren  
12: imgItaly.Visible = False  
13: imgSpain.Visible = False  
14: imgMexico.Visible = False  
15: imgFrance.Visible = False  
16: imgUSA.Visible = False  
17: End Sub  
18: Private Sub optItaly_Click()  
19: ' Flagge anzeigen, wenn das Optionsfeld markiert ist  
20: If optSmall.Value = True Then  
21: imgItaly.Height = 480  
22: imgItaly.Width = 480  
23: Else ' Großes Bild  
24: imgItaly.Height = 2800  
25: imgItaly.Width = 2800  
26: End If  
27: imgItaly.Visible = True  
28: ' Anzeige aller anderen Flaggen deaktivieren  
29: imgEngland.Visible = False  
30: imgSpain.Visible = False  
31: imgMexico.Visible = False  
32: imgFrance.Visible = False  
33: imgUSA.Visible = False  
34: End Sub  
35: Private Sub optSpain_Click()  
36: ' Flagge anzeigen, wenn das Optionsfeld markiert ist  
37: If optSmall.Value = True Then  
38: imgSpain.Height = 480  
39: imgSpain.Width = 480  
40: Else ' Großes Bild  
41: imgSpain.Height = 2800  
42: imgSpain.Width = 2800  
43: End If
```

```
44: imgSpain.Visible = True
45: ' Anzeige aller anderen Flaggen deaktivieren
46: imgItaly.Visible = False
47: imgEngland.Visible = False
48: imgMexico.Visible = False
49: imgFrance.Visible = False
50: imgUSA.Visible = False
51: End Sub
52: Private Sub optMexico_Click()
53: ' Flagge anzeigen, wenn das Optionsfeld markiert ist
54: If optSmall.Value = True Then
55: imgMexico.Height = 480
56: imgMexico.Width = 480
57: Else ' Großes Bild
58: imgMexico.Height = 2800
59: imgMexico.Width = 2800
60: End If
61: imgMexico.Visible = True
62: ' Anzeige aller anderen Flaggen deaktivieren
63: imgItaly.Visible = False
64: imgSpain.Visible = False
65: imgEngland.Visible = False
66: imgFrance.Visible = False
67: imgUSA.Visible = False
68: End Sub
69: Private Sub optFrance_Click()
70: ' Flagge anzeigen, wenn das Optionsfeld markiert ist
71: If optSmall.Value = True Then
72: imgFrance.Height = 480
73: imgFrance.Width = 480
74: Else ' Großes Bild
75: imgFrance.Height = 2800
76: imgFrance.Width = 2800
77: End If
78: imgFrance.Visible = True
79: ' Anzeige aller anderen Flaggen deaktivieren
80: imgItaly.Visible = False
81: imgSpain.Visible = False
82: imgMexico.Visible = False
83: imgEngland.Visible = False
84: imgUSA.Visible = False
85: End Sub
86: Private Sub optUSA_Click()
87: ' Flagge anzeigen, wenn das Optionsfeld markiert ist
88: If optSmall.Value = True Then
89: imgUSA.Height = 480
90: imgUSA.Width = 480
91: Else ' Großes Bild
92: imgUSA.Height = 2800
```

```
93: imgUSA.Width = 2800
94: End If
95: imgUSA.Visible = True
96: ' Anzeige aller anderen Flaggen deaktivieren
97: imgItaly.Visible = False
98: imgSpain.Visible = False
99: imgMexico.Visible = False
100: imgFrance.Visible = False
101: imgEngland.Visible = False
102: End Sub
103: Private Sub cmdReturn_Click()
104: ' Zurück zum Auswahl-Formular
105: frmFlagsOpt.Hide
106: frmSelect.Show
107: End Sub
108: Private Sub optSmall_Click()
109: ' Alle Flaggen verbergen
110: ' Die folgenden Flaggen werden klein angezeigt
111: imgEngland.Visible = False
112: imgItaly.Visible = False
113: imgSpain.Visible = False
114: imgMexico.Visible = False
115: imgFrance.Visible = False
116: imgUSA.Visible = False
117: ' Optionsfelder zurücksetzen
118: optEngland.Value = False
119: optItaly.Value = False
120: optSpain.Value = False
121: optMexico.Value = False
122: optFrance.Value = False
123: optUSA.Value = False
124: End Sub
125: Private Sub optLarge_Click()
126: ' Alle Flaggen verbergen
127: ' Die folgenden Flaggen werden groß angezeigt
128: imgEngland.Visible = False
129: imgItaly.Visible = False
130: imgSpain.Visible = False
131: imgMexico.Visible = False
132: imgFrance.Visible = False
133: imgUSA.Visible = False
134: ' Optionsfelder zurücksetzen
135: optEngland.Value = False
136: optItaly.Value = False
137: optSpain.Value = False
138: optMexico.Value = False
139: optFrance.Value = False
140: optUSA.Value = False
141: End Sub
```

Analyse für das Formular mit den Optionsfeldern

Die Ereignisprozeduren verwenden alle fast denselben Code, nur der Ländername unterscheidet sich. In der ersten Prozedur (Zeilen 1 bis 17) sehen Sie, daß der Code zuerst prüft, ob das Optionsfeld **Gross** oder das Optionsfeld **Klein** ausgewählt wurde (Zeile 3). Wenn das Optionsfeld **Klein** ausgewählt wurde, werden in den Zeilen 4 und 5 die Eigenschaften **Height** und **Width** für das Steuerelement auf 480 Twips gesetzt. Wenn das Optionsfeld **Gross** ausgewählt wurde, setzt der Code die **Height**- und **Width**-Eigenschaften für die Flagge auf 2800 Twips.

Die restliche Ereignisprozedur zeigt die Flagge für das ausgewählte Land an, indem es in Zeile 10 die **Visible**-Eigenschaft auf **True** setzt und in den Zeilen 12 bis 16 die Anzeige der anderen Flaggen deaktiviert. Die folgenden sechs Ereignisprozeduren in den Zeilen 17 bis 102 sind dieser ersten Ereignisprozedur ganz ähnlich, zeigen aber jeweils die Flagge der anderen Länder an.

In Zeile 103 beginnt eine kurze Ereignisprozedur, die genau so arbeitet wie die Ereignisprozedur im Formular mit den Kontrollkästchen. Sie verbirgt das Formular mit den Optionsfeldern und zeigt das Auswahlformular wieder an, so daß der Benutzer ein anderes Formular auswählen oder das Programm beenden kann.

In den Zeilen 108 und 125 beginnen zwei fast identische Prozeduren, die die Anzeige aller Flaggen auf dem Bildschirm ausschalten. Natürlich ist immer nur eine Flagge sichtbar, aber statt zu ermitteln, welche gerade angezeigt wird, und deren **Visible**-Eigenschaft auf **False** zu setzen, werden einfach die **Visible**-Eigenschaften aller Bildfelder auf **False** gesetzt. Fünf der sechs sind zwar schon **False**, aber dieser Code stellt sicher, daß keine Flagge angezeigt wird.

Der Grund dafür, warum beim Anklicken der Schaltflächen für die größere oder kleinere Anzeige der Flaggen die Anzeige der Flaggen gelöscht wird, ist, daß die nächste Flagge, die der Benutzer anklickt, in der richtigen Größe angezeigt werden muß.



Zweifellos ist diese Applikation sehr müßig! Ihr Code wird für jedes der sechs Optionsfelder und Kontrollkästchen zweimal wiederholt. Später in diesem Buch werden Sie lernen, solchen Code zu optimieren. Wenn mehrere ähnliche Steuerelemente auf einem Formular angezeigt werden, die alle fast dieselbe Aufgabe haben, verwendet man ein *Steuerelementefeld*, das den Code verkürzt und seine Wartung leichter macht.

Herzlichen Glückwunsch! Die erste Woche Visual- Basic-Programmierung haben Sie geschafft! Und sie haben gesehen, daß Visual Basic Spaß macht. Außerdem vereinfacht die Programmierung. Vor Visual Basic mußten die Programmierer Code schreiben, der ihre Steuerelemente verwaltete, und gleichzeitig dafür sorgen, daß diese auf die Eingaben des Benutzers reagierten - Visual-Basic-Programmierer können diese trivialen Dinge ganz Visual Basic überlassen und sich auf die ganz speziellen Anforderungen ihrer Applikationen konzentrieren. Der Spaß hat gerade erst angefangen - in dieser Woche werden Sie noch mehr aus Ihren Applikationen machen.

Was Sie erwartet

In der nächsten Woche werden Sie weitere Fertigkeiten im Umgang mit Visual Basic erwerben. In den nächsten Tagen werden Sie fast die gesamte Programmiersprache Visual Basic kennenlernen, die Sie für Ihre Applikationen brauchen. In Kapitel 9 erfahren Sie, wie man auf die internen Funktionen von Visual Basic zugreift, um mit weniger Code mehr Aufgaben zu erfüllen. Mit Hilfe der internen Routinen überlassen Sie es Visual Basic, allgemeine Aufgaben auszuführen, wie beispielsweise Berechnungen oder die Datenverwaltung.

In dieser Woche geht es jedoch nicht nur um die Codierung. Sie erfahren, wie man Standarddialogfelder erzeugt, so daß sich Ihren Benutzern eine vertraute Oberfläche bietet, wenn sie beispielsweise eine Datei öffnen oder etwas an den Drucker schicken. Durch die Verwendung von Standarddialogfeldern erlernen die Benutzer Ihre Programme schneller.

In der ersten Woche haben Sie erfahren, wie man über Steuerelemente auf dem Bildschirm mit den Benutzern kommuniziert. In dieser Woche lernen Sie neue Methoden kennen, wie Sie mit dem Benutzer zusammenarbeiten können. Sie erfahren, wie man Tastencodes auffängt und auswertet, um damit mehr Kontrolle über die vom Benutzer eingegebenen Antworten zu haben. Außerdem lernen Sie, wie man auf Mausklicks, Doppelklicks und Mausbewegungen reagiert, so daß Ihre Programme so gut wie möglich mit dem Benutzer kommunizieren. In Kapitel 10 lernen Sie, daß es ganz einfach ist, einem Steuerelement eine Drag&Drop-Operation hinzuzufügen.

Ihre Programme werden in dieser Woche nicht nur durch die Dateneingabe verbessert. Die Druck-Funktionen machen sie flexibler und praktischer. Aus den Daten, die Sie verarbeiten, können Sie Berichte erzeugen. In Kapitel 14 lernen Sie, Ihren Bildschirm mit Grafik und sogar Multimedia-Präsentationen interessanter zu machen. Visual Basic beinhaltet Steuerelemente, die Grafik und Multimedia unterstützen. Mit den Eigenschaften, Ereignissen und Methoden dieser Steuerelemente wird Ihre Applikation lebendig.

Woche 2

Tag 8

So funktionieren komplexe Programme

In diesem Kapitel lernen Sie, wie komplexe Visual-Basic-Programme aussehen können, die mehrere Module und Prozeduren enthalten und gemeinsam Daten nutzen. Wenn Sie sehr große Applikationen entwickeln, sollten Sie dafür sorgen, daß Daten von Prozeduren und Modulen gemeinsam genutzt werden, indem Sie die Variablen entsprechend deklarieren und die Prozeduren so schreiben, daß andere Prozeduren darauf zugreifen können.

Sie schreiben nicht nur viele eigene Prozeduren, sondern werden auch die internen Funktionen von Visual Basic einsetzen, um eine allgemeine Analyse auszuführen und Strings, Zahlen und andere Daten zu verarbeiten. In diesem Kapitel erfahren Sie alles über die internen Funktionen von Visual Basic, was Sie für Ihre Arbeit brauchen.

Heute lernen Sie die folgenden Dinge kennen:

- Saubere Programmstruktur
- Allgemeine Prozeduren gegen duplizierten Code
- Gültigkeitsbereiche von Variablen
- Argumentelisten
- Numerische Funktionen
- Funktionen zur Datenanalyse
- Funktionen zur Stringmanipulation
- Funktionen für Zeit und Datum

Die richtige Programmstruktur

Sie besitzen bereits eine allgemeine Vorstellung, wie Visual-Basic-Programme funktionieren. Wenn der Benutzer mit Steuerelementen arbeitet, treten Ereignisse auf. Wenn die Applikation eine Ereignisprozedur enthält, die einem Steuerelement und einem Ereignis entspricht, wird diese ausgeführt, sobald das Ereignis auftritt. Der Programmcode von Visual Basic setzt sich größtenteils aus Ereignisprozeduren zusammen. Am Codeanfang gibt es einen Deklarationsbereich.



Sie wissen, daß es für jedes Formular Code im Formularmodul gibt, der es steuert. Wenn Sie ein Formular innerhalb des Arbeitsbereichs im Formularfenster anzeigen, wird also der Code für dieses Formular angezeigt, wenn Sie im Projektfenster auf die Schaltfläche **Code anzeigen** klicken.

Jetzt sollten Sie sich anderen Programminhalten widmen, die eine Visual-Basic-Applikation enthalten kann. Ereignisprozeduren sind nicht die einzigen Prozeduren, die im Code erscheinen können. Abbildung 8.1 beschreibt den Code, der in einem Formularmodul stehen kann. Neben den Deklarationen und

Ereignisprozeduren könnte das Formularmodul *allgemeine Prozeduren* und *Klassenprozeduren* enthalten.

Das Formularmodul

Deklarationen
Private Sub Gen_Proc1() ' Rumpf der ersten allgemeinen Prozedur End Sub
Private Sub Gen_Proc2() ' Rumpf der zweiten allgemeinen Prozedur End Sub
Private Sub Event_Proc1() ' Rumpf der ersten Ereignisprozedur End Sub
Private Sub Event_Proc2() ' Rumpf der zweiten Ereignisprozedur End Sub
Private Sub Class_Proc() ' Rumpf der ersten Klassenprozedur End Sub

Abbildung 8.1:
Ein Formularmodul kann unterschiedliche Codeabschnitte enthalten.

	Eine <i>allgemeine Prozedur</i> ist eine Prozedur, die keinem Steuerelement-Ereignis zugeordnet ist, sondern allgemeine Berechnungen und Verarbeitungen ausführt.
	Eine <i>Klassenprozedur</i> definiert ein spezielles Objekt in Visual Basic, das Sie selbst anlegen. In gewisser Weise definiert eine Klasse einen neuen Datentyp oder eine Variable, die Sie in Ihrem Programm benutzen können.

Sie kennen bereits die beiden Drop-down-Listenfelder im Codefenster, die das Objekt und die Prozedur anzeigen, das bzw. die Sie gerade im Codefenster bearbeiten. Wenn Sie das Drop-down-Listenfeld **Objekt** aufklappen, sehen Sie für jedes Steuerelement des aktuellen Formulars einen Eintrag (das ist das Formular, dessen Formularmodul Sie gerade ansehen). Darüber hinaus sehen Sie oben in der Liste ein spezielles Objekt (**Allgemein**). (Die Klammern um den Namen weisen darauf hin, daß es sich hier nicht um das Steuerelement mit dem Namen **Allgemein** handelt, sondern um einen speziellen Abschnitt.) Der allgemeine Abschnitt enthält sowohl den Deklarationscode, der oben in einem Codefenster erscheint, als auch alle allgemeinen Prozeduren, die Sie schreiben.

Aufruf allgemeiner Prozeduren

Allgemeine Prozeduren können Funktions- oder Sub-Prozeduren sein. Wozu braucht man eine allgemeine Prozedur? In Bonusprojekt 2 (»Variablen und Ausdrücke«) haben Sie in zwei Ereignisprozeduren den folgenden Code gesehen (optLarge_Click() und optSmall_Click()):

```
imgEngland.Visible = False
imgItaly.Visible = False
imgSpain.Visible = False
imgMexico.Visible = False
imgFrance.Visible = False
imgUSA.Visible = False
' Optionsfelder zurücksetzen
optEngland.Value = False
optItaly.Value = False
optSpain.Value = False
optMexico.Value = False
optFrance.Value = False
optUSA.Value = False
```

Das ist eine Menge Tipparbeit, die Sie hier zweimal haben. Natürlich können Sie auch die Befehle **Kopieren** und **Einfügen** im Codefenster verwenden, um den Code nicht zweimal eingeben zu müssen, aber da gibt es ein Problem. Was ist, wenn Sie eine Änderung an dem Code vornehmen möchten? Sie müßten sich also merken, wo überall Sie den Code eingefügt haben. Doppelt verwendeter Code ist nicht nur wegen der Tipparbeit schlecht, sondern erschwert auch die Wartung.

In so einer Situation legen Sie den Code am besten in einer eigenen allgemeinen Prozedur ab, ähnlich der im folgenden Code. Beachten Sie, daß allgemeine Prozeduren einen Namen erhalten müssen. Prozedurnamen werden nach demselben Prinzip wie Variablennamen erzeugt. Darüber hinaus kann es sich bei einer allgemeinen Prozedur um eine Funktion oder um eine Sub-Prozedur handeln.

```
1: Private Sub Clear_Flags()
2: ' Alle angezeigten Flaggen ausblenden
3: ' Die folgenden Flaggen werden klein angezeigt
4: imgEngland.Visible = False
5: imgItaly.Visible = False
6: imgSpain.Visible = False
7: imgMexico.Visible = False
8: imgFrance.Visible = False
9: imgUSA.Visible = False
10: ' Optionsfelder zurücksetzen
11: optEngland.Value = False
12: optItaly.Value = False
13: optSpain.Value = False
14: optMexico.Value = False
15: optFrance.Value = False
16: optUSA.Value = False
17: End Sub
```

Wenn Sie in einer anderen Prozedur an eine Position gelangen, wo Sie diesen Code brauchen, *rufen* Sie die allgemeine Prozedur auf, etwa wie folgt:

```
Call Clear_Flags()
```




Eine Applikation *aufrufen* bedeutet, eine Prozedur innerhalb einer anderen Prozedur auszuführen.

Die Anweisung `Call` weist Visual Basic an, die aktuell ausgeführte Prozedur vorübergehend zu unterbrechen (egal, ob es sich um eine Ereignisprozedur oder um eine allgemeine Prozedur handelt) und den Code der aufgerufenen Prozedur auszuführen. Nachdem die aufgerufene Prozedur abgearbeitet ist, wird der Code der aufrufenden Prozedur weiter ausgeführt, und zwar ab der Zeile hinter der `Call`-Anweisung.

Private und öffentliche Prozeduren

Die Verwendung von `Call` kann sehr zeitsparend sein. Außerdem werden Ihre Programme dadurch sehr viel einfacher wartbar, weil Sie allgemeinen Code in einer Prozedur ablegen und diese dann überall im Programm aufrufen, wenn der Code ausgeführt werden soll. Möglicherweise haben Sie auch eine Routine in einer Applikation, die Sie irgendwo in einer anderen Applikation verwenden möchten. Vielleicht haben Sie einen Briefkopf entwickelt, der den Namen und die Adresse Ihrer Firma enthält, und der auf allen Berichten erscheinen soll, die Sie in anderen Applikationen ausdrucken.

Wenn sich die Prozedur im allgemeinen Abschnitt eines Formularmoduls befindet, kann keine andere Applikation ohne dieses Formularmodul darauf zugreifen. Sie können diese Prozedur also innerhalb eines Codemoduls ablegen. Mit der Zeit entwickeln Sie auf diese Weise ein spezielles Codemodul für bestimmte Aufgaben (hier für den Ausdruck von Berichten), das mehrere verschiedene Routinen enthalten kann. Jede Applikation, die Berichte erzeugt, kann diese Prozeduren nutzen, ohne daß Sie sie überall neu schreiben müßten. Sie klicken einfach nur mit der rechten Maustaste in das Projektfenster der Applikation und wählen im Pop-up-Menü **Hinzufügen, Modul** aus, um Ihr allgemeines Prozedurmodul in die Applikation zu kopieren, die den Code nutzen will.



Wenn Sie allgemeine Prozeduren schreiben und sie in andere Applikationen einfügen, bauen Sie letztlich eine eigene Bibliothek interner Funktionen auf. Eigentlich sind sie nicht *intern*, weil sie nicht Teil des Visual-Basic- Systems sind, aber Sie können sie allen Applikationen hinzufügen, in die Sie diese Prozeduren laden. Sie müssen diesen Code nie wieder schreiben, um dieselben Prozeduren auszuführen. Sie brauchen sie nur von einer beliebigen Applikation aus aufzurufen.

Code innerhalb eines Formularmoduls kann den Code aus einem hinzugefügten Codemodul verwenden. Sie brauchen die Prozedur nur vom Formular aus aufzurufen, mit einer Ausnahme: Sie können außerhalb des aktuellen Moduls nur öffentliche Prozeduren aufrufen. Betrachten Sie die folgende Anweisung zur Prozedurdeklaration:

```
Private Sub ReportIt()
```

Diese Prozedur kann nur innerhalb des Moduls aufgerufen werden, in dem sie sich befindet. Wenn Sie die Prozedur als öffentliche Prozedur schreiben, indem Sie sie wie folgt deklarieren, kann jedes Modul in dieser Applikation sie aufrufen:

```
Public Sub ReportIt()
```

Alle Ihre allgemeinen Prozeduren sollten öffentlich sein, wenn sie von anderen Modulen aus aufgerufen werden sollen.

Damit verstehen Sie die folgenden Regeln:

- Eine als **Private** deklarierte Prozedur kann nur innerhalb ihres eigenen Moduls verwendet werden.
- Eine als **Public** deklarierte Prozedur kann von jeder Prozedur innerhalb ihrer Applikation verwendet werden.

Variablen-Gültigkeitsbereiche

Nicht nur Code kann öffentlich oder privat sein - auch Variablen haben einen öffentlichen oder einen privaten *Gültigkeitsbereich*, werden aber von den Programmierern normalerweise als *global* und *lokal* bezeichnet. Der Unterschied besteht darin, wie die Variablen dem umschließenden Code zur Verfügung gestellt werden.



Eine *lokale Variable* kann nur von Code verwendet werden, für den sie sichtbar ist.

Eine *globale Variable* kann von Code außerhalb ihres Deklarationsbereichs verwendet werden.

Der Begriff *Gültigkeitsbereich* bezieht sich auf die Verfügbarkeit einer Variablen im Programmcode.



Alle Steuerelemente sind für den gesamten Code innerhalb einer Prozedur immer *sichtbar* und *öffentlich*. Steuerelemente auf dem Formular sind dem Code nie verborgen.

Angenommen, Sie arbeiten als Programmierer für das Videogeschäft am Ort. Sie schreiben eine allgemeine Prozedur, die die Mehrwertsteuer berechnet, wozu Sie die spezifischen Werte für diese Berechnung bereitstellen. Der Code berechnet die Mehrwertsteuer für den jeweiligen Gesamtpreis. Irgendwann stellen Sie fest, daß die Steuerberechnung eine allgemeine Aufgabe ist, weil Sie Steuern für Videos wie auch für andere Dinge, die im Geschäft verkauft werden, berechnen, beispielsweise für Getränke, die an einer anderen Theke angeboten werden. Statt dieselbe Berechnung in der Applikation zweimal vorzunehmen, legen Sie sie in einem Codemodul ab, das Sie für die Video-Applikation angelegt haben. Die Mehrwertsteuer-Prozedur ist öffentlich, so daß jede Prozedur in jeder Applikation, die Sie dem Modul hinzufügen, sie aufrufen und damit Steuern berechnen kann.

Es gibt jedoch ein Problem. Prozeduren können nicht immer Daten gemeinsam nutzen. Betrachten Sie den folgenden Codeausschnitt:

```
Private Sub GetSalesTotal()  
' Sub-Prozedur, die die Preise für alle Artikel addiert  
' und dann die Steuer für den Gesamtpreis berechnet  
Dim curTotal As Currency  
'  
' Hier wird die restliche Prozedur fortgesetzt
```

Die Prozedur `GetSalesTotal()` addiert alle Preise, die in den Textfeldern auf einem Formular angezeigt werden, und legt die Summe in der Variablen `curTotal` ab. Anschließend soll die Prozedur für die Mehrwertsteuer die Steuer für das `curTotal` dieser Prozedur berechnen. Das geht mit den Methoden, die Sie bisher kennen, noch nicht, weil `curTotal` eine lokale Variable ist.



Nur Code innerhalb einer Prozedur kann eine lokale Variable nutzen, die Sie in dieser Prozedur deklarieren. Die Variable ist nur innerhalb der Prozedur *sichtbar*, die sie deklariert hat.



Der Begriff *sichtbar* bezieht sich auf die Verwendung einer Variablen. Eine Variable ist nur für Prozeduren sichtbar, die Zugriff auf diese Variable haben. Eine lokale Variable beispielsweise ist nur innerhalb der Prozedur sichtbar, die sie deklariert hat. Eine globale Variable ist im gesamten Modul sichtbar, in dem Sie sie deklariert haben.

Sie können `curTotal` jedoch als Variable deklarieren, die global für das Modul ist, indem Sie die `Dim`-Anweisung aus der Prozedur herausziehen und die Variable im Deklarationsbereich des Formularmoduls

deklarieren, etwa wie folgt:

Option Explicit

' Hier erfolgen die Deklarationen aller öffentlichen Variablen

```
Dim curTotal As Currency
```

Der gesamte Code im Formularmodul kann jetzt auf die Variable `curTotal` zugreifen. Hat Sie das weitergebracht? Die Antwort überrascht Sie vielleicht. Sie haben die Situation nämlich sogar verschlechtert. Aus Kapitel 5 wissen Sie, daß lokale Variablen in der Regel besser als globale Variablen sind, weil Prozeduren nur auf die Daten Zugriff erhalten sollen, die sie brauchen. Sobald Sie die Variable global machen, was passiert, wenn Sie die Variable im Deklarationsbereich eines Moduls deklarieren (im Abschnitt (Allgemein) des Codefensters), haben Sie diese Variable für alle Prozeduren im Modul sichtbar gemacht, sogar für all diejenigen, die die Variable gar nicht brauchen. Und dennoch ist die Variable noch nicht sichtbar für andere Module!



Für Visual-Basic-Anwendungen, die mehrere Module umfassen, definieren die Begriffe *lokal* und *global* nicht alle Situationen. Vielmehr sollten Sie eine Variable als *lokal* bezeichnen, wenn sie für eine Prozedur sichtbar ist, in der Sie sie deklariert haben, als *Variable auf Modulebene*, wenn sie in dem Modul sichtbar ist, in dem Sie sie deklariert haben, und als *global* - oder *public* -, wenn sie von jeder anderen Position der Applikation aus sichtbar ist.

Um eine Variable wirklich global für das gesamte Projekt zu machen, verwenden Sie bei der Deklaration das Schlüsselwort `Public` anstelle von `Dim`. Außerdem müssen Sie die `Public`-Deklaration im Deklarationsbereich eines Moduls platzieren, damit die Variable öffentlich ist - und damit global für die gesamte Applikation.



Zwei öffentliche Variablen können denselben Namen haben. Sie deklarieren sie womöglich in zwei separaten Modulen als `Public`. Um sicherzustellen, daß Sie die richtige öffentliche Variable verwenden, falls Sie je eine solche brauchen, qualifizieren Sie den Variablennamen mit dem Modulnamen. `MyLibrary.curSales` beispielsweise bezeichnet die Variable `curSales` im Codemodul `MyLibrary`. Wenn es in einem anderen Modul eine Variable `curSales` gibt, garantiert die Verwendung von `MyLibrary.curSales`, daß Visual Basic die korrekte Variable verwendet.

Betrachten Sie Abbildung 8.2, die diese theoretischen Erklärungen verdeutlichen soll. Dort sehen Sie, wie sich öffentliche Variablen, Variablen auf Modulebene und lokale Variablen zueinander und innerhalb der Applikation verhalten. Jedes Rechteck innerhalb der beiden Module stellt eine Prozedur dar. Die Beschriftungen erklären, welche Variablen in welchen Prozeduren zur Verfügung stehen.

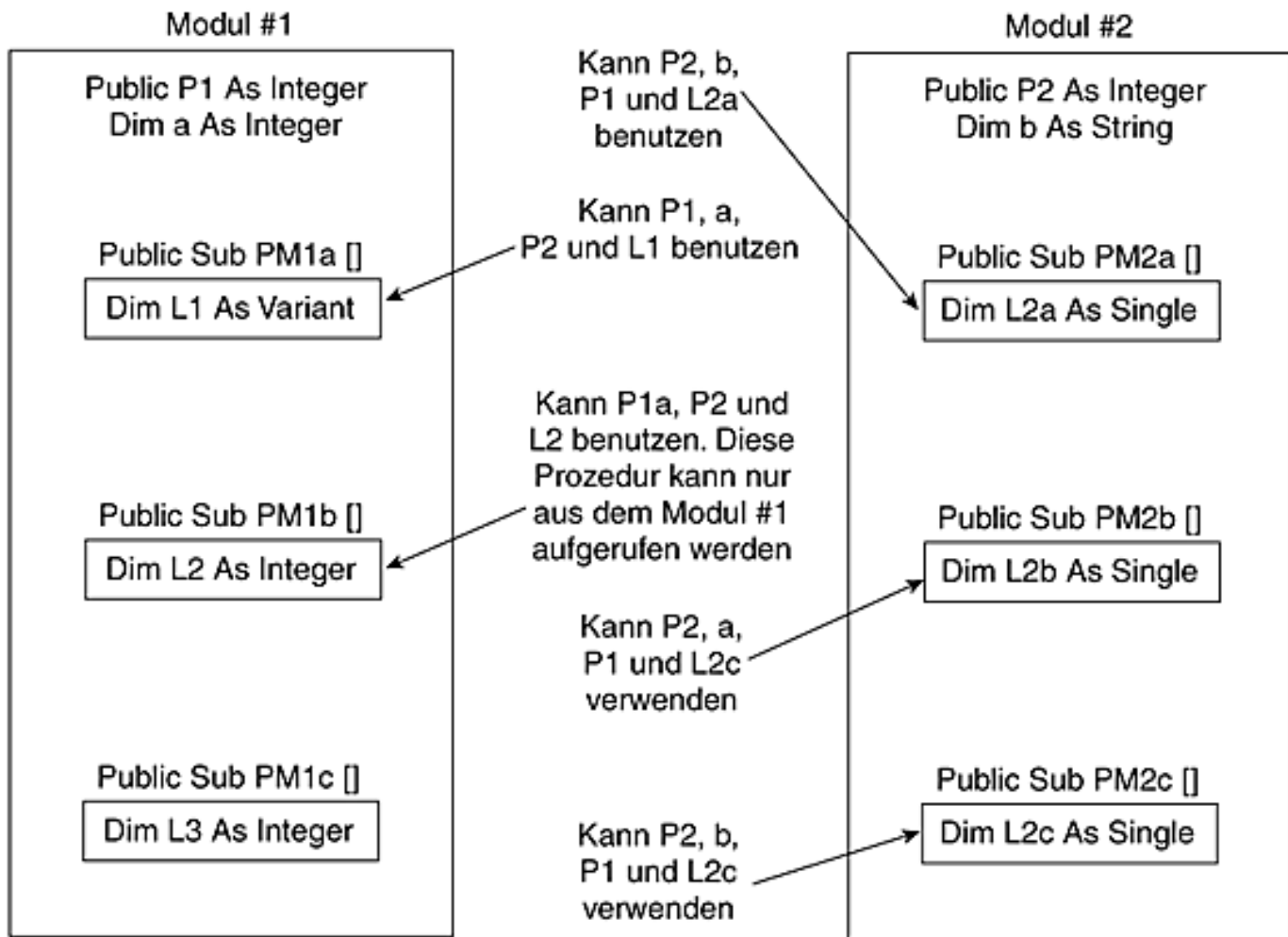


Abbildung 8.2:
Der Gültigkeitsbereich einer Variablen bestimmt, welche Prozeduren diese Variable verwenden können.

All dieses Gerede über öffentliche Variablen macht die eingefleischten Programmierer ganz nervös, weil sie wartbaren Code fordern. Irgendwann werden sie Ihnen erzählen: »Wenn Variablen auf Modulebene schon nicht gut sind, dann sind öffentliche Variablen, die innerhalb einer gesamten Applikation zur Verfügung stehen, noch schlechter, weil sie von allen Prozeduren aller Module der Applikation gesehen werden können.«

Vielleicht erkennen Sie das Dilemma. Außer in dem seltenen Fall, daß dieselbe Variable in fast jeder Prozedur eines Moduls verwendet wird, sollten Sie nur lokale Variablen deklarieren - mit **Dim** innerhalb ihrer Prozeduren. Andere Prozeduren brauchen diese Variablen manchmal, insbesondere allgemeine Prozeduren, die sich außerhalb von Codemodulen befinden. Es muß eine Möglichkeit geben, lokale Daten gemeinsam zu nutzen. Visual Basic stellt eine solche Möglichkeit bereit - wie Sie im nächsten Abschnitt erfahren werden.

Datenübergabe

Wenn zwei Prozeduren Daten gemeinsam nutzen müssen, kann die aufrufende Prozedur ihre lokalen Variablen an die aufgerufene Prozedur senden. Um die Variablen zu übergeben, listen Sie sie in den Klammern der Prozedur auf. Sie haben bereits Argumente an Prozeduren übergeben, beispielsweise bei der Verwendung von **InputBox()**. Bei der Eingabeaufforderung innerhalb der Klammern von **InputBox()** handelte es sich um lokale Daten der Prozedur, die **InputBox()** aufrief, und **InputBox()** nahm diese Daten entgegen, um sie zu verarbeiten.

Die aufgerufene Prozedur muß alle innerhalb der Klammern übergebenen Argumente deklarieren. Ein einfaches Beispiel wird alles erklären. Listing 8.1 enthält zwei Prozeduren. Die eine sendet zwei Werte, eine Gesamtsumme und einen Rabatt, an die zweite Prozedur, die die Steuer für die Gesamtsumme minus dem Rabatt berechnet. Die Prozedur für die Mehrwertsteuer verarbeitet die Daten, die ihr von der ersten Prozedur

übergeben wurden, und zeigt ein Meldungsfeld mit der Gesamtsumme an.

Listing 8.1: Die erste Prozedur sendet Argumente an die zweite Prozedur

```
1: Private Sub GetTotal()  
2: ' Diese Prozedur addiert eine Gesamtsumme auf dem Formular  
3: ' und sendet sie zusammen mit einem Rabattsatz  
4: ' an eine Prozedur zur Mehrwertsteuerberechnung  
5: Dim curTotal As Currency  
6: Dim sngDisc As Single ' Spezieller Rabatt  
7: '  
8: ' Summe aus den Textfeldern des Formulars berechnen  
9: curTotal = txtSale1.Text + txtSale2.Text + txtSale3.txt  
10: '  
11: ' Gesamtsumme an die Mehrwertsteuer-Prozedur senden  
12: Call SalesTax(curTotal, sngDisc)  
13: End Sub  
14:  
15: Public Sub SalesTax(curTotal As Currency, sngRateDisc As Single)  
16: ' Mehrwertsteuer berechnen und Rabatt abziehen  
17: Dim curSalesTax As Currency  
18: Dim intMsg As Integer ' Für MsgBox()  
19: '  
20: ' Die folgenden Berechnungen verwenden einen Steuersatz  
21: ' von 3% plus einem Zuschlag von 1/2 Prozent  
22: curSalesTax = (curTotal * .03) + (curTotal * .005)  
23: '  
24: ' Rabatt abziehen  
25: curSalesTax = curSalesTax - (sngRateDisc * curTotal)  
26: '  
27: ' Steuer anzeigen  
28: intMsg = MsgBox("Die Mehrwertsteuer beträgt " & curSalesTax)  
29: '  
30: ' Prozeduren kehren automatisch zur aufrufenden Prozedur  
31: ' zurück, nachdem sie abgearbeitet sind  
32: End Sub
```



Beachten Sie, daß die Namen der übergebenen Variablen und des empfangenen Arguments nicht übereinstimmen müssen (wie beispielsweise `sngDisc` in Zeile 13 und `sngRateDisc` in Zeile 15), sehr wohl aber die Anzahl der Argumente und ihre Datentypen. Die aufgerufene Prozedur verwendet die Argumente unter dem Namen, unter dem sie sie kennt.

Die Prozedur `SalesTax()` könnte sich in einem anderen Codemodul als `GetTotal()` befinden, die sich im allgemeinen Abschnitt eines Formularmoduls befinden könnte. Zeile 12 ruft die Prozedur auf, die die Steuer berechnet. Sie beginnt in Zeile 15. Beachten Sie, daß die beiden übergebenen Variablen lokal für `GetTotal()` sind; `GetTotal()` stellt sie `SalesTax()` bereit, indem sie die Variablen als Argumente übergibt. Die Argumente sind erforderlich, weil `SalesTax()` zwei Argumente des angegebenen Datentyps braucht.

Nachdem der Prozedur `SalesTax()` die beiden Werte übergeben wurden, kann sie sie behandeln, als handelte es sich um ihre eigenen lokalen Variablen. Die Werte werden für Berechnungen verwendet, und Zeile 28 zeigt die berechnete Steuer in einem Meldungsfeld an.

Abbildung 8.3 zeigt, wie `GetTotal()` seine Werte an `SalesTax()` übergibt.

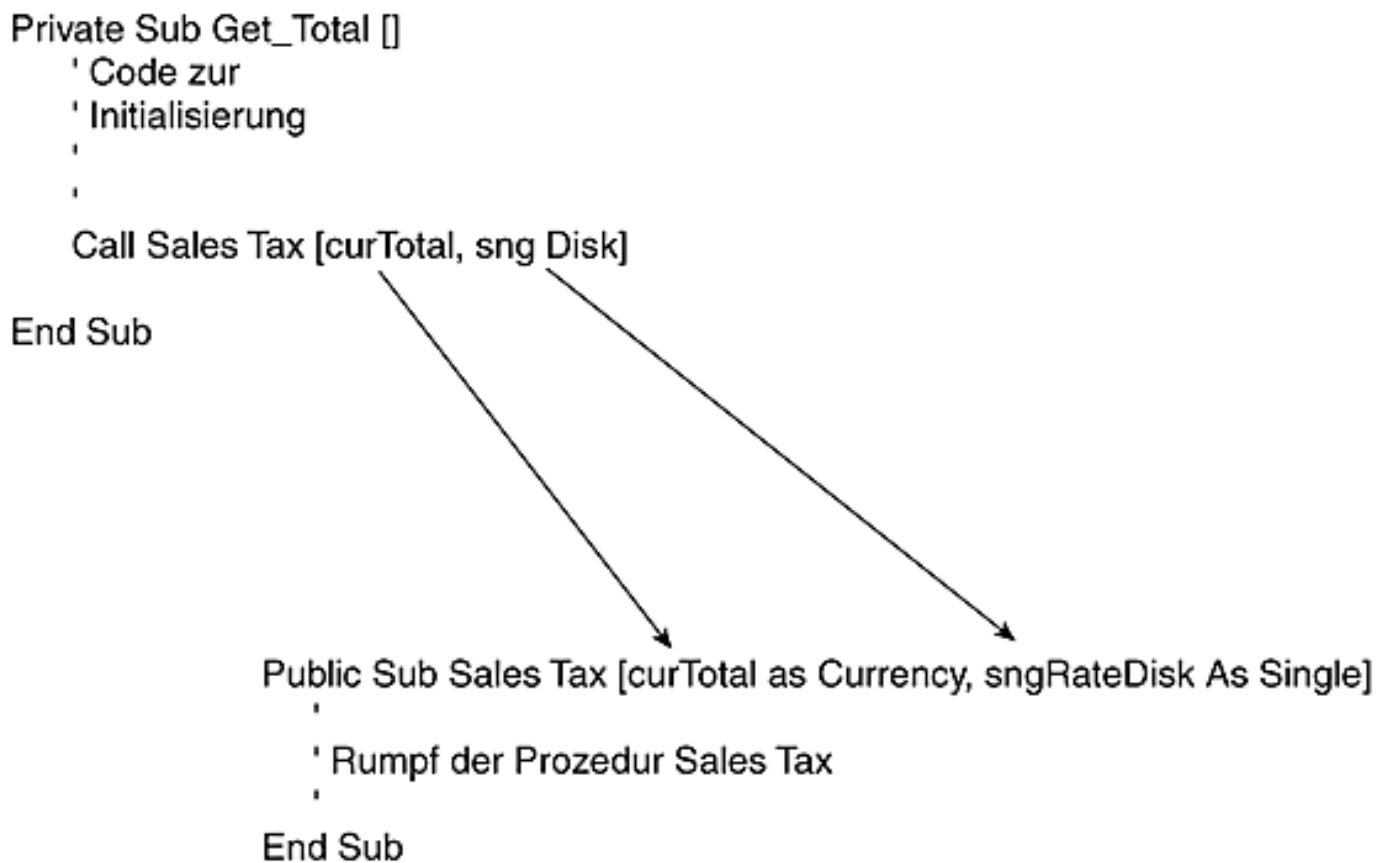


Abbildung 8.3:
GetTotal() sendet zwei Argumente an SalesTax(), wo sie weiter verarbeitet werden.

	In Zeile 28 scheint keine Übereinstimmung der Datentypen vorzuliegen, aber Visual Basic kann das kompensieren. Obwohl MsgBox () als erstes Argument ein String übergeben werden muß, und obwohl der Operator & Strings konkateniert, wandelt Visual Basic eine Zahl in einen String um, wenn Sie versuchen, einen String mit einer Zahl (oder umgekehrt) zu konkatenieren.

Was Sie tun sollten

Verwenden Sie **Exit Sub**, wenn Sie eine Sub-Prozedur verlassen müssen, bevor die normale **End Sub**-Anweisung ausgeführt wird. (**End Function** macht dasselbe für Funktionsprozeduren.) Verwenden Sie **Exit Sub** immer innerhalb einer **If**-Anweisung oder in einer anderen bedingten Anweisung, weil Sie nicht möchten, daß es jedesmal ausgeführt wird, und weil sonst der Code hinter **Exit Sub** nie ausgeführt würde.

Visual Basic unterstützt noch eine Methode, Sub-Prozeduren aufzurufen. Sie können dabei sowohl die **Call**-Anweisung als auch die Klammern weglassen. Die beiden folgenden Anweisungen sind äquivalent:

```

Call PrintTitle (Title)
PrintTitle Title

```

Als Referenz und als Wert

Listing 8.1 übergibt seine Variablen als Referenz, d.h. die aufgerufene Prozedur kann die Argumente in der aufrufenden Prozedur ändern. Wenn Sie den Argumenten in der Argumenteliste der empfangenden Funktion das Schlüsselwort **ByVal** vorausstellen, nimmt Visual Basic die Argumente als Wert entgegen, d.h. die aufgerufene Prozedur kann die Argumente in der aufrufenden Prozedur nicht ändern. (**ByRef** ist ein optionaler Bezeichner, weil alle Argumente standardmäßig als Referenz übergeben werden.)

Hätte beispielsweise die Prozedur `SalesTax()` in Listing 8.1 den Wert in den Argumenten `curTotal` oder `sngRateDisc` verändert, würden diese Variablen auch in `GetTotal()` verändert. Die Standardmethode für die Übergabe von Argumenten als Referenz bewirkt, daß die Argumente der aufrufenden Prozedur nicht davor geschützt sind, von der aufgerufenen Prozedur geändert zu werden.

Wäre Zeile 15 in Listing 8.1 geschrieben wie die folgende, wären die Variablen der aufrufenden Prozedur sicher, weil nichts, was `SalesTax()` macht, die ursprünglichen Argumentwerte ändern kann, nachdem die Programmsteuerung in die Prozedur `GetTotal()` übergegangen ist.

```
Public Sub SalesTax(ByVal curTotal As Currency, ByVal sngRateDisc As _
Single)
```

Funktionsprozeduren im Gegensatz zu Sub-Prozeduren

Eine Funktion unterscheidet sich von einer Sub-Prozedur nicht nur in der ersten und letzten Anweisung (den Hüllzeilen), sondern auch, weil eine Funktion einen Wert an die aufrufende Prozedur zurückgibt. Im vorigen Abschnitt haben Sie Code gesehen, der eine Sub-Prozedur aufrief. Der Aufruf einer Sub-Prozedur ist ganz ähnlich. Sie rufen eine Funktionsprozedur wie eine interne Funktion auf - Sie geben den Funktionsnamen und die Argumente an, verwenden aber keine `Call`-Anweisung. Der Funktionsaufruf wird damit zum Rückgabewert und kann in einem Ausdruck oder einer Anweisung weiterverwendet werden.

Die erste Zeile einer Funktionsprozedur, auch als *Funktionsdeklaration* bezeichnet, muß das folgende Format haben:

```
Public|Private Function FunkName([Arg As datenTyp][, ...]) As datenTyp
```

Der vertikale Strich zwischen `Public` und `Private` zeigt, daß eine Funktion privat für ihr Modul oder öffentlich für die gesamte Applikation sein kann. Eine Funktion muß keine Argumente haben. In diesem Fall können Sie die Klammern weglassen. Die meisten Funktionen nehmen aber mindestens ein Argument entgegen. `As datenTyp`, das es in Sub-Prozeduren nicht gibt, deklariert den Datentyp des Funktionsrückgabewerts. Eine Funktion kann nur einen einzigen Wert zurückgeben, und der Datentyp dieses Werts wird durch den hier angegebenen Datentyp festgelegt.

Wenn die Prozedur `SalesTax()` als Funktion geschrieben wird, gibt sie die berechnete Steuer zurück. Sie hätte dann die folgende Deklarationszeile:

```
Public Sub SalesTax(curTotal As Currency, sngRateDisc As Single) As
_Currency
```



Wie den Sub-Prozeduren können Sie auch Funktionsprozeduren Argumente als Wert oder als Referenz übergeben, abhängig davon, wie weit Sie die Argumente der aufrufenden Funktion schützen möchten.

Irgendwo in der Funktion müssen Sie einer Variablen den Rückgabewert zuweisen. Diese Variable hat denselben Namen wie die Funktion hat. Sie wird nicht deklariert, kann aber dennoch benutzt werden. Wenn die letzte Anweisung in `SalesTax()` einer Variablen namens `SalesTax` einen Wert zuweist, ist dies der Rückgabewert der Funktion. Nachdem die Funktion beendet ist, entweder bei `End Function` oder durch die Anweisung `Exit Function`, wird der Wert zurückgegeben, der sich gerade in dieser Variablen befindet.

Die aufrufende Prozedur muß deshalb einen Platz für den Rückgabewert bereitstellen. Im allgemeinen weisen Sie den Rückgabewert einer Variablen zu. In einer aufrufenden Prozedur findet man nicht selten eine Anweisung wie die folgende:

```
curDailyNet = CalcNetAdj(curGrSls, sngTaxRt, curGrRate, curStrExp)
```

`CalcNetAdj()` ist eine Funktion; die vier Werte werden ihr übergeben, berechnet, und der Variablen `CalcNetAdj` wird irgendwo innerhalb von `CalcNetAdj()` ein Wert zugewiesen. Dieser Wert wird der Variablen `curDailyNet` in der aufrufenden Prozedur zugewiesen, sobald die Funktion beendet ist.

Der Übungsabschnitt am Kapitelende gibt Ihnen Gelegenheit, die Prozedur `CalcTax()` aus Listing 8.1 von einer Sub-Prozedur in eine Funktionsprozedur umzuschreiben. Der Übung halber betrachten Sie hier Listing 8.2, das Ihnen helfen soll, Ihr Verständnis für die Arbeitsweise von Funktionen zu vertiefen.

Listing 8.2: Funktionen geben einen einzigen Wert an die aufrufende Prozedur zurück

```
1: ' Aufrufende Prozedur
2: Private Sub CP()
3: Dim varR As Variant ' Lokale Variablen, mit deren Hilfe der
4: Dim varV As Variant ' Rückgabewert der Funktion verarbeitet wird
5: Dim intI As Integer ' Nimmt den Rückgabewert auf
6:
7: varR = 32 ' Ausgangswerte
8: varV = 64
9:
10: intI = RF(varR, varV) ' Funktion aufrufen und varR und varV
11: ' übergeben. intI nimmt den Rückg.wert auf
12: MsgBox("Nach der Rückgabe enthält intI " & intI)
13: MsgBox("Nach der Rückgabe enthält varR" & varR)
14: MsgBox("Nach der Rückgabe enthält varV" & varV)
15:
16: End Sub
17: ' Empfangende Funktion
18: Public Function RF (varR As Variant, ByVal varV As Variant) As Integer
19: ' Eines als Referenz, das andere als Wert
20: varR = 81 ' Beide Argumente ändern
21: varV = varV + 10
22: ' Rückgabewerte setzen
23: RF = varR + varV
24: End Function
```

Zeile 10 übergibt `varR` und `varV` (32 und 64) an die Funktion. Die Funktion ändert die Argumente in den Zeilen 20 und 21 sofort in 81 und 74. Weil `varR` als Referenz übergeben wurde, ist `varR` in der aufrufenden Prozedur (`CP()`) jetzt ebenfalls 81. Zeile 23 addiert die Funktionsargumente 81 und 74 und weist die Summe dem Funktionsnamen zu, der den Rückgabewert darstellt. Nach Ausführung von `End Function` weist Zeile 10 in der aufrufenden Prozedur `intI` den Wert 155 zu. Zeile 12 zeigt den Wert an. Zeile 13 zeigt, daß `var` von der aufgerufenen Funktion geändert wurde und gibt in einem Meldungsfeld den Wert 81 aus. Zeile 14 zeigt im Meldungsfeld den Wert 64 an, weil es davor geschützt war, von der aufgerufenen Funktion geändert zu werden.

Steuerelemente als Argumente übergeben

Variablen sind nicht die einzigen Daten, die Sie zwischen Prozeduren übergeben können. Sie können auch Steuerelement-Werte übergeben. Angenommen, Sie schreiben eine Prozedur, die mit dem Wert eines Steuerelements arbeitet, aber Sie müssen wissen, welche Art Steuerelement übergeben wurde. Vielleicht wird die Prozedur von mehreren verschiedenen Modulen aufgerufen und es werden unterschiedliche Steuerelemente übergeben, abhängig von der erforderlichen Verarbeitung. In so einem Fall verwenden Sie die Anweisung `If TypeOf`, um zu prüfen, welchen Datensteuertyp das Argument hat.

Hier das Format der typischen `If`-Anweisung mit `.TypeOf`:

```
If TypeOf objekt Is objektTyp then
Block mit einer oder mehreren Visual-Basic-Anweisungen
Else
Block mit einer oder mehreren Visual-Basic-Anweisungen
```


End If



Eine *Steuervariable*, manchmal auch als *Objektvariable* bezeichnet, ist eine Variable, die als Steuerelement deklariert ist. Variablen können nicht nur die Datentypen **Integer**, **String** und **Currency** annehmen, sondern Sie können eine Variable auch als **Object** deklarieren das ein beliebiges Objekt in Visual Basic darstellen kann, auch Steuerelemente. Die folgende Variablendeklaration deklariert Steuervariablen:

```
Dim objCmdFirst As CommandButton
Dim objNameList As ListBox
Dim objPhoto As Image
```

Darüber hinaus könnte auch eine Argumenteliste eine Steuervariable enthalten:

```
Public Function FixControls (objIncoming As Object)
```

Wenn eine Funktion ein Argument erhält, das mit dem Datentyp **Object** deklariert wurde, können Sie die Art des Steuerelements mit dem folgenden Code feststellen:

```
If TypeOf objIncoming Is CommandButton Then
MsgBox("Sie haben eine Schaltfläche übergeben")
ElseIf TypeOf objIncoming Is CheckBox Then
MsgBox("Sie haben ein Kontrollkästchen übergeben")
ElseIf TypeOf objIncoming Is TextBox Then
MsgBox("Sie haben ein Textfeld übergeben")
End If
```

Interne Funktionen

Sie haben bereits drei interne Funktionen kennengelernt: **LoadPicture()**, **MsgBox()** und **InputBox()**. Visual Basic unterstützt zahlreiche weitere interne Funktionen. Im restlichen Kapitel lernen Sie die wichtigsten internen Funktionen kennen, mit denen Sie leistungsfähigere Programme entwickeln können. Nachdem Sie das Kapitel durchgearbeitet haben, wissen Sie nicht nur, wie man eigene Sub-Prozeduren und Prozeduren schreibt, sondern Sie werden auch die meisten internen Funktionen von Visual Basic kennen. Die Werkzeuge, die Sie heute entwickeln, werden Ihre Programmierfähigkeiten wesentlich verbessern. Sie können diese Fertigkeiten sofort einsetzen, um komplexe Applikationen zu schreiben.



Nachdem Sie dieses Kapitel gelesen haben, werden Sie so viel über die Visual-Basic-Programmierung wissen, daß Sie eigene Programme schreiben können. In den folgenden Kapiteln lernen Sie zwar noch neue Befehle kennen, aber das Wichtigste, was Programmieranfänger und Fortgeschrittene wissen müssen, haben Sie bereits gelernt. Das ist doch eine Perspektive! Die meisten der folgenden Kapitel beschreiben Steuerelemente und Eigenschaften, mit denen Sie Ihre Formulare weiter verbessern.

Numerische Funktionen

Am besten beginnen wir unsere Erklärung mit den internen Funktionen zur Integer- Umwandlung. Das Format für die beiden gebräuchlichsten sieht so aus:

```
Int(numerischerWert)
Fix(numerischerWert)
```

numerischerWert kann eine beliebige numerische Konstante, eine Variable oder ein Ausdruck sein, aber auch eine weitere eingebettete Funktion enthalten, die eine Zahl zurückgibt. Egal, welchen numerischen Datentyp Sie übergeben, beide Funktionen geben diesen Datentyp zurück, den Sie als Integer verwenden

können.

Was Sie lassen sollten

Übergeben Sie `Int()` und `Fix()` keine nichtnumerischen Argumente, weil Visual Basic für eine solche Anweisung einen Laufzeitfehler erzeugt.

Beide Funktionen runden ihre Argumente zur nächsten ganzen Zahl hin ab. Sie unterscheiden sich darin, wie sie negative Werte behandeln. In den folgenden Anweisungen zeigt der Kommentar den Rückgabewert der Funktionen:

```
intAns1 = Int(6.8) ' 6
intAns2 = Fix(6.8) ' 6
intAns3 = Int(-6.8) ' -7
intAns4 = Fix(-6.8) ' -6
```



Beachten Sie, daß weder `Int()` noch `Fix()` Werte aufrufen. `Int()` gibt den nächstkleineren Integer-Anteil seines Arguments zurück. Negative Zahlen werden also zur nächstkleineren negativen Zahl abgerundet. `Fix()` gibt den gekürzten Integer-Anteil des Arguments zurück und behält die ganze Zahl des Arguments bei.



Der Begriff »gekürzt« bedeutet, etwas wird entfernt. `Fix()` schneidet den Dezimalanteil seines Arguments ab. Eine somit gekürzte 5.23 würde zu 5 und eine gekürzte -5.23 zu -5.

Die Funktion für den *Absolutwert* ist praktisch, wenn Sie die Differenz zwischen Werten wie beispielsweise Abständen und Temperaturen berechnen möchten.



Der *Absolutwert* ist der positive Wert einer Zahl. Der Absolutwert von 19 ist 19, der Absolutwert von -19 ist ebenfalls 19.

Die Funktion `Abs()` ist die interne Funktion zur Berechnung von Absolutwerten. Angenommen, Sie möchten den Altersunterschied zwischen zwei Angestellten wissen. Dazu berechnen Sie den Absolutwert des Altersunterschieds:

```
intAgeDiff = Abs(intEmpAge1 - intEmpAge2)
```

Diese Anweisung stellt sicher, daß in der Variablen `intAgeDiff` ein positiver Altersunterschied abgelegt wird, egal, welcher Angestellte älter ist. Ohne die Funktion `Abs()` würde die Berechnung einen negativen Wert ergeben, wenn der erste Angestellte jünger als der zweite ist.

Die Funktion `Sqr()` gibt die Quadratwurzel für eine positive Zahl zurück. Die Kommentare in den folgenden Anweisungen zeigen die Ergebnisse der Funktionsaufrufe:

```
intVal1 = Sqr(4) ' 2
intVal2 = Sqr(64) ' 8
intVal3 = Sqr(4096) ' 16
```



`Sqr()` berechnet auch die angenäherte Quadratwurzel von Dezimalzahlen.

Visual Basic unterstützt mehrere komplexe trigonometrische Funktionen. Die folgende Liste beschreibt ein paar davon:

- `Exp()` gibt die Basis eines *natürlichen Logarithmus* zurück (dieser Wert wird auch als *e* bezeichnet und ist annäherungsweise 2,718282), erhoben in die durch das Argument übergebene Potenz.
- `Log()` gibt den natürlichen Logarithmus des Arguments zurück.
- `Atn()` gibt den Arkustangens des Arguments in Radianen zurück.
- `Cos()` gibt den Cosinus des Arguments in Radianen zurück.
- `Sin()` gibt den Sinus des Arguments in Radianen zurück.
- `Tan()` gibt den Tangens des Arguments in Radianen zurück.



Wenn Sie trigonometrische Funktionen brauchen, die ein Argument verwenden, das in Grad und nicht in Radianen angegeben wird, multiplizieren Sie das Argument mit $(\pi / 180)$. π ist annäherungsweise 3.14159.

Datentypfunktionen

Visual Basic unterstützt mehrere Funktionen, die mit dem Datentyp ihrer Argumente arbeiten, und nicht mit den eigentlichen Argumentwerten:

- Die Funktionen, die prüfen, um welche Daten es sich handelt: `IsDate()`, `IsNull()`, `IsNumeric()` und `VarType()`
- Die Shortcut-Funktionen `IIf()` und `Choose()`
- Die Funktionen zur Datentypumwandlung

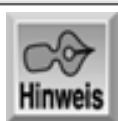
Funktionen zur Überprüfung der Daten

Diese Funktionen überprüfen Datentypen und spezielle Variableninhalte. Ihre Programme verwenden die verschiedensten Daten, und manchmal weiß man nicht im voraus, welcher Datentyp zur Verarbeitung ansteht. Bevor Sie beispielsweise eine Berechnung ausführen, sollten Sie sicherstellen, daß es sich um numerische Daten handelt.

Tabelle 8.1 listet die `Is()`-Datenprüffunktionen auf und beschreibt, wie sie arbeiten. Jede Funktion nimmt ein Argument mit dem Datentyp `Variant` entgegen.

Tabelle 8.1: Die `Is...()`-Datenprüffunktionen prüfen den Inhalt von Variablen und Steuerelementen

Funktionsname	Beschreibung
<code>IsDate()</code>	Ermittelt, ob es sich bei dem Argument um einen Datumstyp handelt (oder ob die Daten in gültige Datumswerte umgewandelt werden können)
<code>IsEmpty()</code>	Prüft, ob das Argument initialisiert wurde
<code>IsNull()</code>	Prüft, ob das Argument den <code>Null</code> -Wert enthält
<code>IsNumeric()</code>	Prüft, ob das Argument eine Zahl enthält (oder ob die Daten in eine gültige Zahl umgewandelt werden können)



Alle `Is...()`-Funktionen nehmen den Datentyp `Variant` entgegen, weil sie in der Lage sein müssen, beliebige Daten zu betrachten und ihren Typ zu ermitteln.

Der Codeausschnitt in Listing 8.3 ist einfach, demonstriert aber, was passiert, wenn Sie die Funktion `IsEmpty()` auf Variablen anwenden, die initialisiert wurden, und auf solche, die nicht initialisiert wurden. Mit `IsEmpty()` stellen Sie fest, ob der Benutzer einen Wert in ein Feld eingegeben hat.

Listing 8.3: Test auf leere Variablen

```
1: ' Code der die Is()-Funktionen demonstriert
2: Dim var1 As Variant, var2 As Variant,
3: Dim var3 As Variant, var4 As Variant
4: Dim intMsg As Integer ' Rückgabe in einem Meldungsfeld
5: ' Variablen mit Beispielwerten belegen
6: var1 = 0 ' 0-Wert
7: var2 = Null ' Null-Wert
8: var3 = "" ' Null-String
9: ' Alle Is()-Funktionen aufrufen
10: If IsEmpty(var1) Then
11: intMsg = MsgBox("var1 ist leer", vbOKOnly)
12: End If
13: If IsEmpty(var2) Then
14: intMsg = MsgBox("var2 ist leer", vbOKOnly)
15: End If
16: If IsEmpty(var3) Then
17: intMsg = MsgBox("var3 ist leer", vbOKOnly)
18: End If
19: If IsEmpty(var4) Then
20: intMsg = MsgBox("var4 ist leer", vbOKOnly)
21: End If
```

Dieser Code gibt nur ein Meldungsfeld mit dem folgenden Inhalt aus:

var4 ist leer

Dieses Ergebnis wird erzeugt, weil alle anderen Variablen irgendwelche Daten enthalten (sie wurden initialisiert).



Mit `IsNull()` prüfen Sie, ob ein Steuerelement oder ein Feld in einem Bericht oder einem Formular Daten enthalten. `IsEmpty()` sollte nur für Variablen verwendet werden.

`IsNull()` überprüft sein Argument und gibt `True` zurück, wenn es den `Null`-Wert enthält. Der Wert `Null` ist ein spezieller Wert, den Sie einer Variablen zuweisen, um entweder anzuzeigen, daß es keine Daten gibt, oder daß ein Fehler vorliegt. (Wie Ihr Programm den `Null`-Wert interpretiert, hängt davon ab, wie Sie das festlegen.)



Angenommen, Sie weisen einer Variablen einen Null-Wert zu (`varA = Null`), dann sind Sie vielleicht versucht, diesen wie folgt abzuprüfen:

```
If (varA = Null) Then ...
```

Ein solches `If` schlägt immer fehl. Die einzige Möglichkeit, einen `Null`-Wert in einer Variablen zu erkennen, ist die Verwendung von `IsNull()`.

Die Überprüfung auf Daten ist einfach. Wenn Ihre Visual-Basic-Prozedur wissen muß, ob das Textfeld `txtHoursWorked` Daten enthält, führt sie die folgende `If`-Anweisung aus:

```
If IsNull(txtHoursWorked) Then
intMsg = MsgBox("Sie haben keine Stunden eingegeben!", vbOKOnly)
Else ' Danke für die gute Arbeit
intMsg = MsgBox("Danke für die Eingabe!", vbOKOnly)
End If
```

Diese If-Anweisung prüft, ob der Benutzer etwas in das Feld eingegeben hat, bevor das Programm fortgesetzt wird.

IsNumeric() prüft, ob das Argument eine Zahl ist. Jeder Variant-Wert, der in eine Zahl umgewandelt werden kann, gibt True zurück. Die folgenden Datentypen können in Zahlen umgewandelt werden:

- Empty (wird in Null umgewandelt)
- Integer
- Long Integer
- Einfache Genauigkeit
- Doppelte Genauigkeit
- Währungswerte
- Datum (ergibt immer False)
- Strings (wenn sie wie eine gültige Zahl »aussehen«)

Der folgende Code fordert die Benutzer auf, ihr Alter einzugeben. Dazu verwendet er eine Variant-Variable. Das Programm zeigt einen Fehler an, wenn der Benutzer einen anderen Wert als eine Zahl eingibt.

```
1: Dim varAge As Variant
2: Dim intMsg As Integer ' MsgBox() Rückgabe
3: varAge = InputBox("Wie alt sind Sie?", "Geben Sie Ihr Alter ein")
4: If IsNumeric(varAge) Then
5: intMsg = MsgBox("Danke!", vbOKOnly)
6: Else
7: intMsg = MsgBox("Was möchten Sie verbergen?", _
vbOKOnly+vbQuestion)
8: End If
```

Zeile 4 stellt sicher, daß der Benutzer ein numerisches Alter eingegeben hat, und nicht etwa ein Wort oder einen Satz.

Wenn Sie wissen müssen, welchen Datentyp eine Variable hat, rufen Sie die Funktion VarType() auf. Tabelle 8.2 listet die 16 möglichen Rückgabewerte der Funktion VarType() auf.

Tabelle 8.2: Die Rückgabewerte von VarType() zeigen, welcher Datentyp vorliegt

Rückgabewert	Konstantenname	Datentyp
0	vbEmpty	Empty
1	vbNull	Null
2	vbInteger	Integer
3	vbLong	Long
4	vbSingle	Single
5	vbDouble	Double
6	vbCurrency	Currency
7	vbDate	Date
8	vbString	String
9	vbObject	Object
10	vbError	Ein Fehlerwert

11	vbBoolean	Boolean
12	vbVariant	Variant (weitere Informationen über Variant-Arrays finden Sie in Kapitel 10)
13	vbDataObject	Ein Datenzugriffsobjekt
14	vbDecimal	Decimal
17	vbByte	Byte
8192	vbArray	Ein Array (VB addiert 8192 zum Datentyp; 8194 steht also für ein Integer-Array)

Die Prozedur in Listing 8.4 verwendet die Anweisung `Select Case`, um den übergebenen Datentyp auszudrucken.

Listing 8.4: Mit `VarType()` ermitteln Sie, welcher Datentyp übergeben wurde

```

1: Private Sub PrntType(varA) ' Variant, falls nicht anders angegeben
2: Dim intMsg As Integer ' MsgBox() Rückgabewert
3: Select Case VarType(varA) ' VarType() gibt einen Integer zurück
4: Case 0
5: intMsg = MsgBox("Das Argument ist leer")
6: Case 1
7: intMsg = MsgBox("Das Argument ist Null")
8: Case 2
9: intMsg = MsgBox("Das Argument ist Integer")
10: Case 3
11: intMsg = MsgBox("Das Argument ist Long")
12: Case 4
13: intMsg = MsgBox("Das Argument ist Single")
14: Case 5
15: intMsg = MsgBox("Das Argument ist Double")
16: Case 6
17: intMsg = MsgBox("Das Argument ist Currency")
18: Case 7
19: intMsg = MsgBox("Das Argument ist Date")
20: Case 8
21: intMsg = MsgBox("Das Argument ist String")
22: Case 9
23: intMsg = MsgBox("Das Argument ist ein Object")
24: Case 10
25: intMsg = MsgBox("Das Argument ist ein Fehler")
26: Case 11
27: intMsg = MsgBox("Das Argument ist Boolean")
28: Case 12
29: intMsg = MsgBox("Das Argument ist ein Variant-Array")
30: Case 13
31: intMsg = MsgBox("Das Argument ist ein Datenzugriffsobjekt")
32: Case 14
33: intMsg = MsgBox("Das Argument ist Decimal")
34: Case 17
35: intMsg = MsgBox("Das Argument ist Byte")
36: Case Else
37: intMsg = MsgBox("Das Argument ist ein Array")

```

```
38: End Select
39: End Sub
```

Die Shortcut-Funktionen IIf() und Choose()

Es gibt ein bequemes Äquivalent zu der `If . . . Else`-Anweisung mit einzeiligem Rumpf. Die Funktion `IIf()` arbeitet ähnlich wie die Funktion `@If()` aus den gebräuchlichen Tabellenkalkulationen. Das Format für `IIf()` sieht wie folgt aus:

`IIf(bedingung, TrueRumpf, FalseRumpf)`

`IIf()` funktioniert nur für kurze `If . . . Else`-Anweisungen, wie beispielsweise:

```
If (curSales < 5000.00) Then
curBonus = 0.00
Else
curBonus = 75.00
End If
```

Jeder der beiden Rümpfe dieser `If . . . Else`-Anweisung umfaßt nur eine Zeile. Sie können dieses Konstrukt also auch mit einer kürzeren `IIf()`-Funktion darstellen und `curBonus` den Rückgabewert zuweisen:

```
curBonus = IIf(curSales < 5000.00, 0.00, 75.00)
```

Abbildung 8.4 zeigt, wie dieses `IIf()` funktioniert.

	Obwohl <code>IIf()</code> kürzer als ein äquivalentes mehrzeiliges <code>If . . . Else</code> ist, ist <code>IIf()</code> nicht so deutlich wie das mehrzeilige Konstrukt. Wenn Sie jemals einen der Rümpfe der <code>IIf()</code>-Funktion erweitern, müssen Sie diese Funktion in ein mehrzeiliges <code>If . . . Else</code> umwandeln. Größtenteils ist es also sinnvoller, von vorneherein das mehrzeilige Format der <code>If . . . Else</code>-Anweisung zu verwenden.

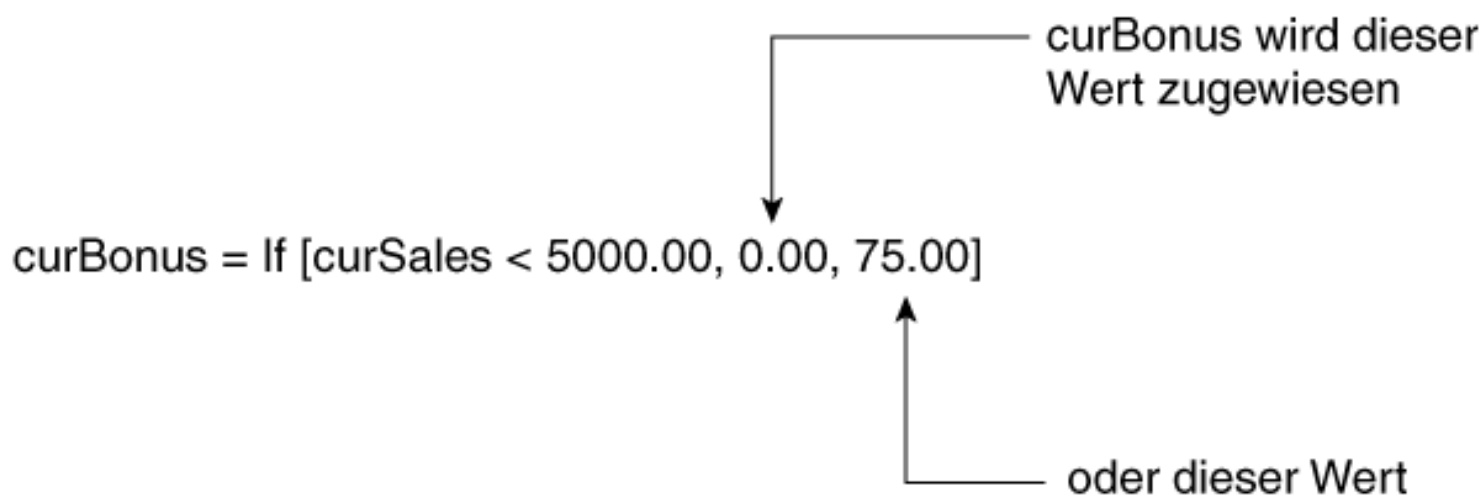


Abbildung 8.4:
Der Variablen auf der linken Seite des `IIf()` wird einer von zwei Werten zugeordnet.

Sie können nicht durch Null dividieren, weil die Division durch Null in der Mathematik nicht definiert ist. Die folgende `IIf()`-Funktion gibt einen durchschnittlichen Verkaufspreis zurück, oder einen `Null`-Wert, falls eine Division durch Null auftreten würde:

```
curAveSales = IIf(intQty > 0, curTotalSales / intQty, Null)
```



Visual Basic interpretiert Nullwerte in allen Situationen als `False`. Damit können Sie die oben gezeigte Anweisung auch wie folgt schreiben:

```
curAveSales = IIf(intQty, curTotalSales / intQty, Null)
```

Die `Choose()`-Funktion stellt eine Abkürzung für bestimmte `Select Case`-Anweisungen dar. `Choose()` kann viele Argumente haben - mehr als andere eingebaute Funktionen. Abhängig vom ersten Argument gibt `Choose()` eines der weiteren Argumente zurück. Hier das Format von `Choose()`:

```
Choose(intIndexNum, ausdruck[, ausdruck] ...)
```

Nach dem zweiten Argument (`ausdruck`) können Sie beliebig viele Argumente angeben. `intIndexNum` ist eine Variable oder ein Feld, die eine Zahl zwischen 1 und der Anzahl der Ausdrücke in der Funktion sein kann.

Wenn Sie beispielsweise ein kleine Tabelle mit Preiscodes, Abkürzungen oder Produktcodes anlegen möchten, ist die Verwendung von `Choose()` häufig praktischer als eine `If`- oder `Select Case`-Anweisung. `Choose()` hat jedoch einen begrenzteren Gültigkeitsbereich als `If`, weil es nur für Integer-Werte angewendet werden kann, nicht für allgemeinere Vergleiche.



`Choose()` gibt Null zurück, wenn `intIndexNum` nicht zwischen 1 und der Anzahl der Ausdrücke liegt.

Das erste Argument von `Choose()` kann ein Ausdruck sein. Sie müssen also das erste Argument so anpassen, daß es innerhalb des Bereichs der nachffolgenden Argumente liegt. Wenn die für den Index möglichen Werte beispielsweise zwischen 0 und 4 liegen, addieren Sie 1 zum Index, so daß sich der Bereich zwischen 1 und 5 erstreckt und eine korrekte Auswahl aus der `Choose()`-Liste treffen kann.

Angenommen, ein Formular enthält ein Bezeichnungsfeld mit einem Preiscode. Wenn die Benutzer ein neues Produkt eingeben, sollen sie gleichzeitig einen Preiscode zwischen 1 und 5 eingeben, der den folgenden Codes entspricht:

1	Voller Preis
2	5% Rabatt
3	10% Rabatt
4	Sonderauftrag
5	Postbestellung

Die folgende `Choose()`-Funktion weist einem Bezeichnungsfeld abhängig von dem Preiscode die korrekte Beschreibung zu.

```
Descript = Choose(lblProdCode, "Voller Preis", "5% Rabatt", "10% Rabatt", "Sonderauftrag", "Postbestellung")
```

Funktionen zur Datentypumwandlung

Tabelle 8.3 beschreibt die Funktionen zur Datentypumwandlung, gekennzeichnet durch den vorgestellten Buchstaben *C* (für *Convert*). Diese Funktionen wandeln ihr Argument von einem Datentyp in einen anderen um.

Tabelle 8.3: Die Funktionen zur Datentypumwandlung konvertieren einen Datentyp in einen anderen

Funktionsname	Beschreibung
CBool()	Wandelt das Argument in den Datentyp Boolean um
CByte()	Wandelt das Argument in den Datentyp Byte um
CCur()	Wandelt das Argument in den Datentyp Currency um
CDate()	Wandelt das Argument in den Datentyp Date um
Cdbl()	Wandelt das Argument in den Datentyp Double um
CDec()	Wandelt das Argument in den Datentyp Decimal um
CInt()	Wandelt das Argument in den Datentyp Integer um
CLng()	Wandelt das Argument in den Datentyp Long um
CSng()	Wandelt das Argument in den Datentyp Single um
CStr()	Wandelt das Argument in den Datentyp String um
CVar()	Wandelt das Argument in den Datentyp Variant um



Es muß grundsätzlich möglich sein, das Argument in den Zieldatentyp umzuwandeln. Die Zahl 123456789 beispielsweise kann mit CByte() nicht in den Datentyp Byte umgewandelt werden, weil der Datentyp Byte keine so großen Zahlen aufnehmen kann.

Anders als Int() und Fix() gibt CInt() den gerundeten Integer zurück, der am nächsten bei dem Argument liegt. Die Kommentare rechts von den folgenden Anweisungen geben an, was in den Variablen gespeichert wird:

```
intA1 = CInt(8.5) ' Speichert 8 in intA1
intA2 = CInt(8.5001) ' Speichert 9 in intA2
```

Der folgende Code deklariert eine Variable mit vier verschiedenen Datentypen und wandelt die Argumente in diese Datentypen um. Beachten Sie, daß Sie diesen Funktionen auch Ausdrücke übergeben können, die numerische Ergebnisse erzeugen, so daß Sie die Datentypen für die von Ihnen berechneten Ergebnisse festlegen können, bevor Sie sie in einem Feld oder in einer Variablen ablegen:

```
curVar1 = CCur(123) ' Wandelt 123 in den Datentyp Currency um
dblVar2 = Cdbl(123) ' Wandelt 123 in den Datentyp Double Precision um
sngVar3 = CSng(123) ' Wandelt 123 in den Datentyp Single Precision um
varVar4 = CVar(123) ' Wandelt 123 in den Datentyp Variant um
```

String-Funktionen

Die String-Funktionen manipulieren und analysieren String-Daten. Einer der größten Vorteile von Visual Basic gegenüber anderen Programmiersprachen ist die umfassende Unterstützung von Strings - dank dem BASIC-Hintergrund.

Len() ermittelt die Länge

Len() ist eine der wenigen Funktionen, denen man einen numerischen Wert oder einen String als Argument übergeben kann - obwohl Len() hauptsächlich für Stringdaten verwendet wird. Len() gibt die Anzahl der Speicherbytes zurück, die benötigt werden, um das Argument aufzunehmen. Hier das Format von Len():

Len(*Ausdruck*)



`Len()` nimmt beliebige String-Werte entgegen (Variablen, Konstanten oder Ausdrücke). Als Argument von `Len()` können nur numerische Variablen, nicht jedoch numerische Konstanten oder Ausdrücke angegeben werden.

`Len()` gibt die Länge (die Anzahl der Zeichen) der String-Variablen, der String-Konstanten oder des String-Ausdrucks innerhalb seiner Klammern zurück. Die folgende `MsgBox()`-Funktion erzeugt die Ausgabe 6.

```
intMsg = MsgBox(Len("abcdef"))
```



Wenn der String `Null` enthält, gibt `Len()` den Wert 0 zurück. Durch den Test auf einen Null-String erkennen Sie, ob ein Benutzer Daten in ein Eingabefeld oder ein Steuerelemente eingegeben hat.

Strings umwandeln

Es gibt mehrere Funktionen zur Umwandlung von String-Daten. Tabelle 8.4 beschreibt die Funktionen, die in den folgenden Beispielen verwendet werden.

Tabelle 8.4: Die Funktionen zur String-Umwandlung

Funktionsname	Beschreibung
<code>CStr()</code>	Wandelt ein Argument in einen String um
<code>Str()</code>	Wandelt das numerische Argument in einen String um (genauer gesagt, in einen <code>Variant</code> -Datentyp, den Sie als String benutzen können)
<code>Val()</code>	Wandelt ein String-Argument in eine Zahl um, vorausgesetzt, Sie übergeben einen String, der »aussieht« wie eine Zahl

`CStr()` und `Str()` wandeln ihre Argumente in String-Werte um. Der einzige Unterschied ist, daß `CStr()` anders als `Str()` vor positiven Zahlen, die in Strings umgewandelt wurden, kein führendes Leerzeichen einfügt. Listing 8.5 demonstriert den Unterschied zwischen `CStr()` und `Str()`.

Listing 8.5: `Str()` fügt vor den positiven Zahlen ein führendes Leerzeichen ein, `CStr()` macht das nicht

```
1: Private Sub convStr ()
2: Dim str1 As String, s2 As String
3: Dim intMsg As Integer ' Für angeklickte Schaltfläche
4: str1 = CStr(12345)
5: str2 = Str(12345)
6: intMsg = MsgBox("****" & str1 & "****")
7: intMsg = MsgBox("****" & str2 & "****")
8: End Sub
```

Zeile 6 erzeugt ein Meldungsfeld, das `***12345***` anzeigt. Zeile 7 zeigt `*** 12345***` an. Beachten Sie das Leerzeichen vor der Zahl, die `Str()` erzeugt hat.

Die ASCII-Funktionen

Mit `Chr()` und `Asc()` wandeln Sie Strings in ihre numerischen ASCII-Werte um und umgekehrt. Die ASCII-Tabelle listet alle auf dem PC verfügbaren Zeichen auf und weist ihnen sequentielle Nummern (den *ASCII-Code*) zu.

Durch die Übergabe einer Zahl als Argument für `Chr ( )` erzeugen Sie das Zeichen, das dieser Nummer in der ASCII-Tabelle entspricht. Mit `Chr ( )` erzeugen Sie Zeichen für Variablen und Steuerelemente, die nicht auf der Tastatur Ihres PC bereitgestellt werden, aber in der ASCII-Tabelle enthalten sind.

Die Funktion `Asc ( )` ist das genaue Gegenteil von `Chr ( )`. Während `Chr ( )` ein numerisches Argument entgegennimmt und ein String-Zeichen zurückgibt, nimmt `Asc ( )` ein String-Argument entgegen und wandelt dieses Argument in die entsprechende Nummer der ASCII-Tabelle um.

Die folgende Anweisung legt also ein A in `strVar` ab, weil der ASCII-Wert von A gleich 65 ist:

```
strVar = Chr(65) ' Speichert ein A in aVar
```

Es ist natürlich viel sinnvoller, A direkt in `strVar` abzulegen. Aber wie ist es, wenn Sie in einem Meldungsfeld eine spanische Frage stellen möchten? Spanische Fragen beginnen immer mit einem umgekehrten Fragezeichen, aber auf Ihrer Tastatur gibt es dieses Zeichen nicht. Um dieses Zeichen zu erzeugen, setzen Sie `Chr ( )` ein:

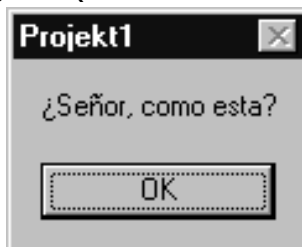
```
' Chr(241) erzeugt ein n mit einer Tilde  
strMyQuest = Chr(191) & "Se" & Chr(241) & "or, como esta?"  
intMsg = MsgBox(strMyQuest)
```

Abbildung 8.5 zeigt das Meldungsfeld für diesen Code.

`Asc ( )` gibt den ASCII-Wert für das übergebene Zeichenargument zurück. Sie finden die ASCII-Zeichen in der Online-Hilfe von Visual Basic. Das Argument muß ein String mit einem oder mehreren Zeichen sein. Wenn Sie `Asc ( )` einen String mit mehreren Zeichen übergeben, gibt es den ASCII-Wert des ersten Zeichens im String zurück.

Der folgende Code demonstriert einen sinnvollen Verwendungszweck für `Asc ( )`:

```
strAns = InputBox("Möchten Sie den Namen sehen?")  
If ((Asc(strAns) = 89) Or (Asc(strAns) = 121)) Then  
b = MsgBox("Der Name ist " + aName)
```



```
End If
```

Abbildung 8.5:
Mit Hilfe von ASCII-Zeichen werden Zeichen angezeigt,
die sich nicht auf der Tastatur befinden.

Der Benutzer beantwortet die Frage mit y, Y, yes oder Yes. Der `If . . . Then`-Test funktioniert für alle Eingabewerte, weil 89 der ASCII-Wert für Y ist, und 121 der ASCII- Wert für y. `Asc ( )` gibt den ASCII-Wert des ersten Buchstabens des String-Arguments zurück.

Teilstring-Funktionen

Die Teilstring-Funktionen geben bestimmte String-Abschnitte zurück. `Right ( )` gibt Zeichen von rechts zurück. `Left ( )` gibt Zeichen von links zurück. `Mid ( )` setzt an, wo `Right ( )` und `Left ( )` nicht taugen - es ermöglicht Ihnen, Zeichen aus der Mitte des Strings zu ermitteln.

Hier die Formate der Teilstring-Funktionen:

```
Left(stringWert, numerischerWert)  
Right(stringWert, numerischerWert)  
Mid(stringWert, startPosition[, länge])
```

Der folgende Codeabschnitt demonstriert die Verwendung von `Left()`:

```
strA = "abcdefg"
partSt1 = Left(strA, 1) ' Speichert a
partSt2 = Left(strA, 3) ' Speichert abc
partSt3 = Left(strA, 20) ' Speichert abcdefg
```



Wenn Sie versuchen, mehr Zeichen von links im String zurückzugeben, als es überhaupt gibt, gibt `Left()` den gesamten String zurück.

`Right()` funktioniert ähnlich, außer daß es die Zeichen von der rechten Seite eines Strings zurückgibt:

```
strA = "abcdefg"
partSt1 = Right(strA, 1) ' Speichert g
partSt2 = Right(strA, 3) ' Speichert efg
partSt3 = Right(strA, 20) ' Speichert abcdefg
```

`Mid()` schafft, was `Left()` und `Right()` nicht tun - es gibt Zeichen aus der Mitte des Strings zurück. `Mid()` verarbeitet drei Argumente: einen String, gefolgt von zwei Integer. Der erste Integer legt fest, wo `Mid()` anfangen soll, Zeichen aus dem String zu ermitteln (die Position, beginnend bei 1), und der zweite Integer bestimmt, wie viele Zeichen von dieser Position aus zurückgegeben werden sollen. Wenn Sie die beiden Integer nicht angeben, verwendet `Mid()` 1 als Startposition

`Mid()` kann eine beliebige Anzahl Zeichen von einer beliebigen Position im String ermitteln. Das folgende Beispiel demonstriert die Arbeitsweise von `Mid()`:

```
strA = "Visual Basic FORTRAN COBOL C Pascal"
lang1 = Mid(strA, 1, 12) ' Speichert Visual Basic
lang2 = Mid(strA, 14, 7) ' Speichert FORTRAN
lang3 = Mid(strA, 22, 5) ' Speichert COBOL
lang4 = Mid(strA, 28, 1) ' Speichert C
lang5 = Mid(strA, 30, 6) ' Speichert Pascal
```

Wenn Sie das Argument `länge` nicht angeben, gibt Visual Basic alle Zeichen rechts von der Anfangsposition zurück. Ist die Länge länger als der restliche String, ignoriert Visual Basic das Argument `länge`.



`Mid()` kann sowohl als Befehl als auch als Funktion eingesetzt werden. Es verhält sich wie ein Befehl, wenn es links vom Gleichheitszeichen einer Zuweisung erscheint, und wie eine Funktion, wenn es an anderer Stelle erscheint. Es hat das folgende Format:

`Mid(string, start[, länge])`

Mit der Anweisung `Mid()` können Sie den Inhalt des Strings ändern, der innerhalb der Klammern angegeben ist. Der folgende Code initialisiert einen String mit drei Wörtern und ändert dann das mittlere Wort mit Hilfe von `Mid()`:

```
strSentence = "Rain in Spain"
' Das mittlere Wort ändern

id(strSentence, 6, 2) = "on"
' Nach der Veränderung gibt
intMsg = MsgBox("After change: " & strSentence)
' "Rain on Spain" aus
```

Umwandlung in Groß- und Kleinbuchstaben

Die Funktion `UCase()` gibt das übergebene Argument in Großbuchstaben zurück. `LCase()` gibt das übergebene Stringargument in Kleinbuchstaben zurück. Die folgende `MsgBox`-Funktion zeigt `VISUAL BASIC` an:

```
intMsg = MsgBox(UCase("Visual Basic"))
```

Strings ausrichten und kürzen

`LTrim()` und `RTrim()` kürzen Leerzeichen am Anfang oder am Ende eines Strings. `LTrim()` gibt den als Argument übergebenen String ohne führende Leerzeichen zurück. `RTrim()` gibt den als Argument übergebenen String ohne nachfolgende Leerzeichen zurück. Die Funktion `Trim()` entfernt führende und folgende Leerzeichen.

Hier die Formate der Funktionen zum Kürzen von Strings:

```
LTrim(stringAusdruck)
RTrim(stringAusdruck)
Trim(stringAusdruck)
```

Die folgenden Anweisungen kürzen Leerzeichen vom Anfang, vom Ende oder von beiden Seiten des Strings:

```
st1 = LTrim(" Hello") ' Speichert Hello
st2 = RTrim("Hello ") ' Speichert Hello
st3 = Trim(" Hello ") ' Speichert Hello
```

Ohne diese Funktionen würden zusätzlich zum Wort *Hello* auch die Leerzeichen in die Zielvariablen kopiert.

`Str()` wandelt positive Zahlen in Strings mit einem führenden Leerzeichen (wo das imaginäre Pluszeichen stehen würde) um. Sie können `LTrim()` mit `Str()` kombinieren, um das führende Leerzeichen zu entfernen. Die erste der beiden folgenden Anweisungen speichert das führende Leerzeichen in `str1`. Die zweite Anweisung verwendet `LTrim()`, um das Leerzeichen zu entfernen, bevor der String in `str2` gespeichert wird:

```
str1 = Str(234) ' Speichert " 234"
str2 = LTrim(Str(234)) ' Speichert "234"
```

Die Funktion `ReverseIt()` wendet einige der in diesem Kapitel beschriebenen Funktionen zur String-Verarbeitung an. Diese Funktion, die Sie in Listing 8.6 sehen, kehrt die Reihenfolge einer bestimmten Anzahl von Zeichen innerhalb eines Strings um.

Listing 8.6: Diese Funktion kehrt die Reihenfolge der Zeichen in einem String um. Dazu werden die String-Funktionen benutzt.

```
1: Public Function ReverseIt (strS As String, ByVal n As Integer) As _
String
2: ' Übergabewerte: Ein String und ein Integer, die angeben, wie viele
3: ' Zeichen in der Reihenfolge umgekehrt werden
4: ' Aufgabe: Kehrt die Reihenfolge der Zeichen in dem vorgegebenen
5: ' String um
6: ' Rückgabe: der veränderte String
7:
8: ' Kehrt die ersten n Zeichen in s um.
9:
10: Dim strTemp As String, intI As Integer
11:
12: If n > Len(strS) Then n = Len(strS)
13: For intI = n To 1 Step -1
```

```

14: strTemp = strTemp + Mid(strS, intI, 1)
15: Next intI
16: ReverseIt = strTemp + Right(strS, Len(strS) - n)
17: End Function

```

Angenommen, die Funktion `ReverseIt()` wird mit der folgenden Anweisung aufgerufen:
`newStr = ReverseIt ("Visual Basic", 6)`

Wenn alles funktioniert, enthält der String `newStr` die Zeichen `lausiV Basic` (die ersten sechs Zeichen werden in umgekehrter Reihenfolge dargestellt). Zeile 10 deklariert zwei lokale Variablen; die erste ist die String-Variable `strTemp`, die den umgekehrten String aufnimmt. Die zweite Variable, `intI`, wird in der `For`-Schleife verwendet.



Beginnend mit Version 6, beinhaltet Visual Basic eine String-Funktion namens `StrReverse()`, die den umgekehrten String des einzigen String-Arguments zurückgibt. Listing 8.6 ist vielleicht nicht so effizient wie `StrReverse()`, aber es kann die Verwendung der Funktion `Mid()` demonstrieren.

Die `If`-Anweisung in Zeile 12 stellt sicher, daß der `ReverseIt()` übergebene Integer nicht größer als die Länge des übergebenen Strings ist. Es ist unmöglich, mehr Zeichen umzukehren, als es im String gibt. Wenn mehr Zeichen übergeben werden, stellt die `If`-Anweisung sicher, daß der gesamte String umgekehrt wird, indem die Länge mit der `Len()`-Funktion auf die String-Länge gesetzt wird.

In Zeile 13 zählt die `For`-Schleife von der Position, an der die Umkehrung beginnen soll (in `n` gespeichert) herunter bis 1. Mit Hilfe von `Mid()` in Zeile 14 konkateniert Visual Basic ein Zeichen von der String-Position `n` mit dem neu aufzubauenden String. Ist `n` gleich 1, wird der gesamte String umgekehrt angezeigt.

Spezielle Funktionen

Visual Basic stellt Ihnen Funktionen zur Verfügung, mit denen Sie Datums- und Zeitwerte auswerten und manipulieren können. Diese Funktionen sind wichtig, um genau aufzeichnen zu können, wann ein Feld bearbeitet wurde - für Sicherheits- oder Prüfzwecke. Außerdem sollten alle ausgedruckten Berichte eine Zeitangabe beinhalten (häufig auch als *Datums- und Zeitstempel* bezeichnet). Damit wird genau festgehalten, wann ein Bericht erzeugt wurde. Wenn man sehr viele ähnliche Berichte hat, wird einem dadurch die Unterscheidung erleichtert.

Neben den Datums- und Zeitfunktionen unterstützt Visual Basic eine spezielle Funktion zur Datenformatierung, mit der Sie formatierte Strings erzeugen.

Datum und Zeit

Ihre Windows-Einstellungen bestimmen die Rückgabewerte von `Date` und `Time`. Beispielsweise gibt die `Date`-Funktion auf vielen Systemen das Systemdatum als `Variant` zurück, und zwar im folgenden Format:
`tt-mm-jjjj`

Dabei ist `mm` die Nummer für den Monat (von 01 bis 12), `tt` ist eine Nummer für den Tag (von 01 bis 31), und `jjjj` ist eine Jahreszahl (von 1980 bis 2099). Für `Date` sind keine Klammern erforderlich, weil es sich dabei um eine der wenigen Funktionen handelt, die keine Argumente entgegennehmen.

`Time` gibt die Systemzeit als Datentyp `Variant` zurück, wozu das folgende Format verwendet wird:
`hh:mm:ss`

`hh` ist die Stunde (von 00 bis 23), `mm` ist die Minute (von 00 bis 59) und `ss` ist die Sekunde (von 00 bis 59).

`Time` verwendet ein 24-Stunden-Format.

Now kombiniert die Funktionen `Date` und `Time`. Es gibt den Datentyp `Variant` zurück, und zwar im folgenden Format (falls Sie den `Variant`-Rückgabewert von `Now` in einem Meldungsfeld ausgeben, sehen Sie dieses Format):

`tt.dd.jj hh:mm:ss`

Die Platzhalter entsprechen dabei denen der `Date`- und `Time`-Funktionen.

Sie sollten sich zu allen drei Datums- und Zeitfunktionen merken, daß sie Werte zurückgeben, die intern in doppelter Genauigkeit abgelegt sind (das bietet ausreichend viel Genauigkeit, um sicherzustellen, daß alle Datums- und Zeitwerte exakt gespeichert werden können). Am besten formatieren Sie Datums- und Zeitwerte mit `Format ( )`, das Sie am Ende dieses Kapitels kennenlernen werden.

Angenommen, es ist 9:45. Die Anweisung

```
currentTime = Time
```

speichert den Wert 9:45:00 in der Variablen `currentTime`. Wenn das Datum der 23.3.99 ist, speichert die Anweisung

```
currentDate = Date
```

den Wert 23.3.99 in der Variablen `currentDate`. Die Anweisung

```
currentDateTime = Now
```

speichert 23.3.99 9:45:00 in der Variablen `currentDateTime`.



Wenn Sie Datums- und Zeitwerte eingeben, müssen Sie sie in Pfundzeichen einschließen, beispielsweise wie folgt:

```
#21.11.1993#
```

Weil es mehrere Datumsformate gibt, erkennt Visual Basic fast jede Methode, wie Sie das Datum angeben. So lange Sie das Datum zwischen Pfundzeichen eingeben, können Sie für ein Datum die verschiedensten Formate verwenden:

```
tt.mm.jj
```

```
tt-mm-jjjj
```

```
tt/mm/jj
```

```
mm/tt/jjjj
```

```
tt. monatsname jjjj tt. mmm jjjj (dabei ist mmm eine Abkürzung für  
den Monatsnamen, beispielsweise Dez).
```

Hier einige Methoden, wie Sie die Zeit formulieren können:

```
hh
```

```
hh:mm
```

```
hh:mm:ss
```

Für `Time` wird das 24-Stunden-Format verwendet.

Berechnung der Zeit zwischen zwei Ereignissen

Die `Timer`-Funktion gibt die Anzahl der Sekunden zurück, die auf der internen Uhr des Computers seit Mitternacht vergangen sind. Das Format von `Timer` ist einfach:

```
Timer
```

Wie Sie sehen, nimmt die Funktion `Timer` keine Argumente entgegen. Sie brauchen also hinter dem Funktionsnamen keine Klammern anzugeben. `Timer` ist perfekt geeignet für das Timing eines Ereignisses. Beispielsweise können Sie den Benutzern eine Frage stellen und dann feststellen, wie lange sie für die Antwort

gebraucht haben. Speichern Sie dazu den Wert von `Timer`, bevor Sie die Benutzer befragen. Anschließend subtrahieren Sie diesen Wert vom `Timer`-Wert nach der Antwort. Die Differenz zwischen beiden `Timer`-Werten ist die Anzahl der Sekunden, die der Benutzer für eine Antwort gebraucht hat. Listing 8.7 zeigt eine Prozedur, die genau das erledigt.

Listing 8.7: Sie können die Zeit messen, die der Benutzer für eine Antwort braucht

```
1: Public Sub CompTime ()
2: ' Prozedur, die die Antwortzeit des Benutzers mißt.
3: Dim intMsg As Integer ' MsgBox() Rückgabe
4: Dim varBefore, varAfter, varTimeDiff As Variant
5: Dim intMathAns As Integer
6: varBefore = Timer ' Zeit vor der Frage speichern
7: intMathAns = Inputbox("Was ist 150 + 235?")
8: varAfter = Timer ' Zeit nach der Antwort speichern
9: ' Die Differenz zwischen den Zeitwerten ist, wie viele Sekunden
10: ' der Benutzer für die Antwort gebraucht hat
11: varTimeDiff = varAfter - varBefore
12: intMsg = MsgBox("Dazu haben Sie" + Str(varTimeDiff) & " Sekunden
gebraucht!")
13: End Sub
```

Zeile 6 speichert die Anzahl der Sekunden seit Mitternacht, bevor eine Frage gestellt wurde. Zeile 7 stellt eine Frage. Nachdem der Benutzer eine Antwort eingegeben hat, speichert Zeile 8 die Anzahl der Sekunden seit Mitternacht zu diesem Zeitpunkt. Die Differenz, die in Zeile 11 berechnet wird, gibt an, wie lange der Benutzer für die Antwort gebraucht hat.

`Timer` ermittelt die Anzahl der Sekunden zwischen Zeitwerten, aber nur für Zeitwerte innerhalb eines Tages. Wo `Timer` nicht mehr funktioniert, setzen `DateAdd()`, `DateDiff()` und `DatePart()` ein. Tabelle 8.4 listet die drei Funktionen der Datumsarithmetik und ihre Beschreibungen auf.

Tabelle 8.5: Die Funktionen für die Datumsarithmetik vergleichen Datumswerte

Funktionsname	Beschreibung
DateAdd()	Gibt ein neues Datum zurück, nachdem Sie einen Wert zu einem Datum addiert haben
DateDiff()	Gibt die Differenz zwischen zwei Datumswerten zurück
DatePart()	Gibt eine Komponente (ein Element) eines bestimmten Datums zurück

Diese drei arithmetischen Funktionen arbeiten mit den in Tabelle 8.5 aufgelisteten Datumskomponenten. Tabelle 8.5 listet die Datumskomponenten auf, mit denen diese Funktionen arbeiten, ebenso wie ihre Intervallwerte, die die einzelnen Komponenten beschriften. Mit Hilfe der Intervallwerte werden innerhalb der Datumsarithmetik- Funktionen Teile eines Datums oder einer Zeit ermittelt.

Tabelle 8.6: Die Datumsarithmetik-Funktionen arbeiten mit diesen Zeitabschnitten

Intervallwert	Zeitabschnitt
jjjj	Jahr
q	Quartal
m	Monat
j	Tag im Jahr
t	Tag

w	Wochentag (Sonntag ist 1, Montag ist 2 usw. für <code>Day()</code> , <code>Month()</code> , <code>Year()</code> und <code>DateDiff()</code>)
ww	Woche
h	Stunde
n	Minute (beachten Sie, daß das kein m ist)
s	Sekunde

Trotz seines Namens kann `DateAdd()` sowohl für Datums- als auch für Zeitwerte verwendet werden (wie alle Datumsfunktionen), weil das Datum, das `DateAdd()` übergeben wird, im Datentypformat `Date` erscheinen muß. Hier das Format von `DateAdd()`:

`DateAdd(intervall, anzahl, altesDatum)`

`intervall` muß ein Wert (in Stringformat) aus Tabelle 8.5 sein. Das Intervall bestimmt, welche Zeitdauer addiert oder subtrahiert wird (ein Sekundenwert, ein Minutenwert oder was auch immer). Der Wert `anzahl` bestimmt, wie viele Intervallwerte Sie addieren. Wenn Sie etwas zu einem Datum addieren wollen, verwenden Sie einen positiven Wert, wenn Sie etwas von einem Datum subtrahieren wollen, verwenden Sie einen negativen Wert. Der Wert `altesDatum` ist das Datum oder die Zeit, von dem bzw. der Sie ausgehen (das Datum oder die Zeit, zu der Sie etwas addieren oder von dem Sie etwas subtrahieren). `altesDatum` ändert sich nicht. Die Funktion `DateAdd()` gibt das neue Datum zurück.

Angenommen, Sie kaufen heute etwas mit einer Kreditkarte, die nach 25 Tagen abgerechnet wird. Die folgende Anweisung addiert 25 Tage zum heutigen Datum und legt das Ergebnis in `intStarts` ab:

`intStarts = DateAdd("y", 25, Now)`

Das Datum in `intStarts` liegt 25 Tage nach dem heutigen Datum.



Sie können »y«, »d« oder »w« für das Intervall verwenden, wenn Sie Tage zu einem Datum addieren.

Angenommen, Sie arbeiten für eine Firma, wo Sie 10 Jahre angestellt sein müssen, um eine Berechtigung auf eine Betriebsrente zu haben. Die folgende Anweisung addiert 10 Jahre zu Ihrem Einstellungsdatum und speichert dieses Datum in `vested`:

`vested = DateAdd("yyyy", 10, hired)`

Beachten Sie, daß der Intervallstring festlegt, was zum Datum addiert wird.



Die Funktionen zur Datumsarithmetik nehmen das aktuelle Jahr an, falls nichts anderes angegeben ist (das Jahr, das auf der Systemuhr eingestellt ist).

`DateDiff()` gibt die Differenz zwischen zwei Datumswerten zurück. Wenn Sie sicherstellen möchten, daß Sie einen positiven Wert erhalten, betten Sie die Funktion in die Funktion `Abs()` ein. Die Differenz wird in dem Intervall ausgedrückt, das Sie vorgegeben haben. Hier das Format von `DateDiff()`:

`DateDiff(intervall, datum1, datum2)`

Die folgende Anweisung ermittelt, wie viele Jahre ein Angestellter für eine Firma gearbeitet hat:

`angestelltSeit = Abs(DateDiff("yyyy", anstellungsDatum, Now))`

`DatePart()` gibt eine Komponente eines Datums zurück (den Teil, den Sie im Intervall spezifiziert haben).

Mit `DatePart()` finden Sie heraus, welchen Wochentag, welchen Monat, welche Woche oder welche Stunde (oder was immer Sie für ein Intervall wählen) das Datum darstellt. Hier das Format von `DatePart()`:

Die folgende Anweisung speichert den Tag im Monat, an dem ein Angestellter seine Arbeit aufgenommen hat.

```
DatePart("w", einstellungsDatum)
```

Die Datums- und Zeitfunktionen, die Sie hier kennengelernt haben, arbeiten mit seriellen Werten. Diese Werte werden mit doppelter Genauigkeit abgelegt, um sicherzustellen, daß Datum und Zeit vollständig gespeichert werden können, und daß eine exakte Datumsarithmetik erfolgt.



Ein *serieller Wert* ist die interne Darstellung eines Datums oder einer Zeit, abgelegt in einem `VarType 7` (dem Datentyp `Date`) oder in einem `Variant`-Datentyp.

Die Funktion `DateSerial()` hat das folgende Format:

```
DateSerial(jahr, monat, tag)
```

`jahr` ist ein Integer mit der Jahreszahl (00 bis 99 oder 1900 bis 1999 oder eine vierstellige Jahreszahl) oder ein Ausdruck. `monat` ist ein Integer mit der Monatsnummer (1 bis 12) oder ein Ausdruck. `tag` ist ein Integer mit der Tagesnummer (1 bis 31) oder ein Ausdruck. Wenn Sie für eines dieser Integer-Argumente einen Ausdruck angeben, spezifizieren Sie dabei die Anzahl der Jahre, Monate oder Tage seit einem oder bis zu einem bestimmten Datum. Um die Ausdrücke für das serielle Argument zu verdeutlichen, betrachten Sie die beiden folgenden Funktionsaufrufe für `DateSerial()`, die denselben Wert zurückgeben:

```
d = DateSerial(1998, 10, 6)
```

und

```
d = DateSerial(1988+10, 12-2, 1+5)
```

Die `DateSerial()`-Funktionen stellen sicher, daß Ihre Datumsargumente nicht außerhalb der Grenzen liegen. Beispielsweise war 1996 ein Schaltjahr, deshalb hatte der Februar 29 Tage. Der folgende Aufruf von `DateSerial()` scheint ein ungültiges Datum zu erzeugen, weil der Februar nicht einmal in Schaltjahren 30 Tage haben kann:

```
d = DateSerial(1996, 2, 29+1)
```

Dieser Funktionsaufruf ist richtig, weil `DateSerial()` das angegebene Datum weiterzählt und damit den 1. März 1996 enthält, also einen Tag nach dem letzten Tag im Februar. Die in Listing 8.8 gezeigte Funktion enthält eine interessante Verwendung der Funktion `DateSerial()`.

Listing 8.8: Dieser Code berechnet den nächsten Wochentag nach einem bestimmten Datum

```
1: Function DueDate (dteAnyDate) As Variant
2: ' Argumente: Ein Datumswert
3: ' Aufgabe: Berechnet den ersten Tag des Monats, der nicht an einem
4: ' Wochenende liegt, und der nach dem angegebenen Datum liegt.
5: ' Rückgabe: Das berechnete Datum
6:
7: Dim varResult As Variant
8:
9: If Not IsNull(dteAnyDate) Then
10: varResult = DateSerial(Year(dteAnyDate), Month(dteAnyDate) + 1, 1)
11: If Weekday(varResult) = 1 Then ' Sonntag -> Tag dazu.
12: DueDate = Result + 1
13: ElseIf Weekday(varResult) = 7 Then ' Samstag -> 2 Tage dazu
14: DueDate = varResult + 2
```

```

15: Else
16: DueDate = varResult
17: End If
18: Else
19: varResult = Null
20: End If
21: End Function

```

Wenn diese Funktion aufgerufen wird, wird ihr ein Datumswert übergeben, der als `VarType 7 Date` oder `Variant` abgelegt ist. Wie Sie dem Kommentar entnehmen, berechnet die Funktion den ersten Wochentag (2 für Montag bis 6 für Freitag) des nächsten Monats (den ersten Arbeitstag des Monats, der nach dem im Argument angegebenen Datum liegt).

Die Funktion `DateValue()` ist ähnlich `DateSerial()`, außer daß sie ein String-Argument entgegennimmt, wie das folgende Format zeigt:

```
DateValue(stringDatumsAusdruck)
```

`stringDatumsAusdruck` muß ein String sein, den Visual Basic als Datum erkennt (beispielsweise wie diejenigen, die für die `Date`-Anweisung in diesem Kapitel bereits beschrieben wurden). Wenn Sie den Benutzer auffordern, ein Datum in einzelnen Komponenten einzugeben (also zuerst das Jahr, dann den Monat und dann den Tag abfragen), verwenden Sie `DateValue()`, um diese Werte in ein internes seriellles Datum umzuwandeln. Wenn der Benutzer ein vollständiges Datum eingeben soll (das Sie in einer String-Variablen ablegen), wie beispielsweise 19. Oktober 1999, wandelt `DateValue()` diesen String in das serielle Format um, das man für Datumswerte benötigt.

Die Funktionen `TimeSerial()` und `TimeValue()` arbeiten ähnlich wie ihre Datums-Gegenstücke. Wenn Sie drei einzelne Werte für eine Tageszeit vorgeben, wandelt `TimeSerial()` dieser Werte in ein internes Zeitformat um (`Variant` oder `VarType 7`). Hier das Format von `TimeSerial()`:

`TimeSerial()` nimmt auch Ausdrücke für seine Argumente entgegen und zählt diese ggf. weiter, so wie `DateSerial()`.

Wenn Sie einen String mit einem Zeitwert haben (vielleicht hat der Benutzer eine Zeit eingegeben), wandelt `TimeValue()` diesen String in einen Zeitwert mit dem folgenden Format um:

```
TimeValue(stringZeitAusdruck)
```

`Day()`, `Month()` und `Year()` wandeln ihre Datumsargumente (mit den Datentypen `Variant` oder `VarType 7`) in eine Tageszahl, Monatszahl oder Jahreszahl um. Diese drei Funktionen sind einfach:

```

Day(datumsArgument)
Month(datumsArgument)
Year(datumsArgument)

```

`Weekday()` gibt für das übergebene Datumsargument den Wochentag an (siehe Tabelle 8.5).

Übergeben Sie `Day()`, `Month()` und `Year()` das heutige Datum (das Sie mit `Now` ermitteln):

```

d = Day(Now)
m = Month(Now)
y = Year(Now)

```

Der Monatstag (siehe Tabelle 8.5), die Monatszahl und das Jahr werden in drei verschiedenen Variablen abgelegt.

Die Funktion Format()

Eine der leistungsfähigsten und komplexesten Funktionen ist `Format ( )`. Sie gibt ihr Argument in einem anderen Format zurück, als es übergeben wurde. Hier das Format dieser Funktion:

`Format(ausdruck, format)`

`Format ( )` gibt den Datentyp `Variant` zurück, den Sie fast immer als String verwenden. Der Ausdruck kann ein numerischer oder ein String-Ausdruck sein. Sie können alle Daten formatieren - Zahlen, Strings, Datums- oder Zeitwerte -, um ihr Aussehen zu verändern. Beispielsweise könnten Sie Scheckbeträge mit Kommata und Dollarzeichen versehen.

Das Format ist eine String-Variable oder ein Ausdruck, der eines oder mehrere der in den Tabellen 8.7 bis 8.9 gezeigten Formatzeichen enthält. Welche Tabelle Sie verwenden, hängt von den Daten ab (String, numerisch oder Datumswerte), die Sie formatieren. Die Tabellen sind umfangreich, aber nachdem Sie ein paar Beispiele betrachtet haben, werden Sie wissen, wie Sie diese Formatzeichen verwenden.

Tabelle 8.7: Diese Zeichen formatieren String-Ausgaben

Symbol	Beschreibung
@	An der @-Position erscheint in der Ausgabe ein Zeichen. Wenn sich an der Position @ im String kein Zeichen befindet, erscheint ein Leerzeichen. Das @ wird von rechts nach links gefüllt (falls es mehrere gibt).
&	Dieses Zeichen verhält sich wie @, außer daß nichts angezeigt wird, wenn es in dem auszugebenden String kein Zeichen an der Position von & gibt.
!	Das Ausrufezeichen erzwingt, daß alle Platzhalterzeichen (@ und &) von links nach rechts aufgefüllt werden.
<	Kleiner erzwingt, daß alle Zeichen in Kleinbuchstaben dargestellt werden.
>	Größer erzwingt, daß alle Zeichen in Großbuchstaben dargestellt werden.

Tabelle 8.8: Diese Zeichen formatieren numerische Ausgaben

Symbol	Beschreibung
Nullstring, ""	Dieser String zeigt die Zahl unformatiert an.
0	An der Position von 0 erscheint in der Ausgabe eine Ziffer, falls die zu formatierende Zahl eine Ziffer enthält. Wenn sich an der Position von 0 keine Ziffer befindet, erscheint eine 0. Wenn es nicht so viele Nullen für die zu formatierende Zahl gibt, wie Nullen im Formatfeld stehen, werden führende oder folgende Nullen ausgedruckt. Wenn die Zahl mehr numerische Positionen enthält, erzwingt 0, daß alle Ziffern rechts vom Dezimalpunkt so gerundet werden, daß sie dem Anzeigemuster entsprechen, und alle Ziffern links vom Dezimalpunkt werden ausgegeben, wie sie sind. Dieses Anzeigeformat verwenden Sie hauptsächlich, wenn führende oder folgende Nullen angezeigt werden sollen.
#	Das Pfundzeichen verhält sich wie 0, außer daß nichts angezeigt wird, wenn die zu formatierende Zahl nicht so viele Ziffern enthält wie das Anzeigeformat Pfundzeichen (#) vorgibt.
.	Der Punkt gibt an, wie viele Ziffern (in Kombination mit 0 oder #) links und rechts eines Dezimalpunkts erscheinen sollen.

%	Die zu formatierende Zahl wird mit 100 multipliziert und das Prozentzeichen (%) wird an seiner Position innerhalb des Anzeigeformatstrings ausgegeben.
,	Wenn ein Komma zwischen #-Zeichen oder 0-Zeichen erscheint, sind die Tausendergruppen einfacher zu lesen, weil das Komma jeweils drei Ziffern gruppiert (es sei denn, die Zahl ist kleiner als 1000). Wenn Sie zwei Kommas nebeneinander schreiben, wird die Zahl durch 1000 dividiert (die Zahl wird damit skaliert).
E-, E+, e-, e+	Die Zahl wird in wissenschaftlicher Notation formatiert, wenn das Format gleichzeitig mindestens eine 0 oder ein # enthält.
:	Der Doppelpunkt bewirkt, daß zwischen den Stunden-, Minuten- und Sekundenwerten eines Zeitwerts Doppelpunkte angezeigt werden.
/	Der Schrägstrich bewirkt, daß zwischen den Tag-, Monats- und Jahreswerten ein Schrägstrich angezeigt wird.
-, +, \$, Leerzeichen	Diese Zeichen erscheinen an der entsprechenden Position innerhalb des formatierten Strings.
\	Das Zeichen, das dem Backslash folgt, erscheint an der betreffenden Position im formatierten String.

Tabelle 8.9: Diese Zeichen formatieren Datumsausgaben

Symbol	Beschreibung
c	Zeigt das Datum (wie das Symbol ddddd, wenn nur ein Datum angezeigt wird), die Zeit (wie ttttt, wenn nur die Zeit angezeigt wird) oder beides an, wenn beide Werte vorhanden sind.
d	Zeigt die Tageszahl an, zwischen 1 und 31.
dd	Zeigt die Tageszahl mit führender Null an, von 01 bis 31.
ddd	Zeigt eine Abkürzung für den Tagesnamen an, von Son bis Sam.
dddd	Zeigt den vollständigen Tagesnamen an, von Sonntag bis Samstag.
dddddd	Zeigt das Datum (Monat, Tag, Jahr) gemäß Ihrer Ländereinstellungen in der Systemsteuerung für das Kurzformat an (normalerweise t/m/jj).
ddddddd	Zeigt das Datum (Monat, Tag, Jahr) gemäß der Ländereinstellungen in Ihrer Systemsteuerung für das ausführliche Format an (normalerweise tt mmmm jjjj).
w, ww	Siehe Tabelle 8.6.
m	Zeigt die Monatszahl zwischen 1 und 12 an. Das m steht auch für Minute, wenn es einem h oder einem hh folgt.
mm	Zeigt die Monatszahl mit führender Null an, zwischen 01 und 12. Das mm steht auch für Minute, wenn es einem h oder hh folgt.
mmm	Zeigt den abgekürzten Monatsnamen an, von Jan bis Dez.
mmmm	Zeigt den vollständigen Monatsnamen an, von Januar bis Dezember.
q	Zeigt das Quartal des Jahres an.
y	Zeigt den Tag im Jahr an, von 1 bis 366.
yy	Zeigt das Jahr zweistellig an, von 00 bis 99 (zum Jahrtausendwechsel zeigt yy für 2000 weiterhin das zweistellige Format an).

yyyy	Zeigt die vollständige Jahreszahl an, von 1000 bis 9999.
h, n, s	Siehe Tabelle 8.6.
tttt	Zeigt die Zeit gemäß Ihrer Ländereinstellungen in der Systemsteuerung an (normalerweise h:nn:ss).
AMPM	Verwendet das 12-Stunden-Format und zeigt AM und PM an.
ampm	Verwendet das 12-Stunden-Format und zeigt am und pm an.
AP	Verwendet das 12-Stunden-Format und zeigt A oder P an.
ap	Verwendet das 12-Stunden-Format und zeigt a oder p an.

Die folgenden Anweisungen demonstrieren die Verwendung der Anzeigeformate. Die Kommentare beschreiben, was in der Variablen für formatierten Date (das sind die Variablen links vom Gleichheitszeichen) abgelegt wird:

```
strS = Format("AbcDef", ">") ' ABCDEF
strS = Format("AbcDef", "<") ' abcdef
strS = Format("2325551212", "(@@@) @@@-@@@@") ' (232) 555-1212
```

Wie die letzte Anweisung zeigt, können Sie String-Daten in jedem beliebigen Format anzeigen. Auch wenn die zu formatierenden Daten, wie beispielsweise die Telefonnummer in der letzten Zeile, aus dem Textfeld einer Tabelle stammen, kann Format wie gewohnt eingesetzt werden.

Angenommen, man kann die Ortsnetzvorwahl für eine Telefonnummer weglassen, die Sie drucken möchten. Format () füllt von rechts nach links auf, die Anweisung

```
strS = Format("5551212", "(@@@) @@@-@@@@")
```

speichert also in strS das folgende:

```
( ) 555-1212
```

Wenn Sie die Ortsnetzvorwahl angeben, wird sie innerhalb der Klammern ausgegeben.

Verwenden Sie das Ausrufezeichen (!) nur, wenn das Auffüllen von der anderen Richtung her erfolgen soll (wenn für das Ende des zu formatierenden Strings möglicherweise keine Daten mehr bereitstehen). Die Anweisung

```
strS = Format("5551212", "!(@@@) @@@-@@@@")
```

speichert den folgenden, falschen Wert in strS:

```
(555) 121-2
```

Listing 8.9 zeigt, wie die numerische Formatierung funktioniert. Der Kommentar rechts von den Anweisungen beschreibt, wie die Daten formatiert werden.

Listing 8.9: Anhand der Beispiele erkennen Sie, wie Format() funktioniert

```
1: strS = Format(9146, "|#####|") ' |9146|
2: strS = Format(2652.2, "00000.00") ' 02652.20
3: strS = Format(2652.2, "#####.##") ' 2652.2
4: strS = Format(2652.216, "#####.##") ' 2652.22
5: strS = Format(45, "+###") ' +45
6: strS = Format(45, "-###") ' -45
7: strS = Format(45, "###-") ' 45-
8: strS = Format(2445, "$#####.##") ' $2445.
9: strS = Format(2445, "$#####.00") ' $2445.00
```

```
10: strS = Format(2445, "00Hi00") ' 24Hi45
```

Listing 8.10 demonstriert, wie die Formatierung für Datum und Zeit funktioniert. Der Kommentar rechts zeigt, wie die Daten formatiert wurden.

Listing 8.10: Mit Format() formatieren Sie Datums- und Zeitwerte

```
1: Dim varD As Variant
2: varD = Now ' Vorausgesetzt ist, das Datum ist der
3: ' 21. Mai 1999 14:30
4: strND = Format(varD, "c") ' 5/21/99 2:30:02 PM
5: strND = Format(varD, "w") ' 6 (für Freitag)
6: strND = Format(varD, "ww") ' 22
7: strND = Format(varD, "dddd") ' Freitag
8: strND = Format(varD, "q") ' 2
9: strND = Format(varD, "hh") ' 14
10: strND = Format(varD, "h AM/PM") ' 2 PM
11: strND = Format(varD, "hh AM/PM") ' 02 PM
12: strND = Format(varD, "d-mmmm h:nn:ss") ' 21-Mai 14:30:02
```

Zusammenfassung

In diesem Kapitel haben Sie viel über die Struktur von Visual-Basic-Programmen erfahren. Bei der Entwicklung von Programmen mit mehreren Prozeduren und Modulen müssen Sie den Gültigkeitsbereich der Variablen beachten, so daß Prozeduren Zugriff auf alle Daten haben, die sie brauchen. Variablen sollten in der Regel lokal sein, so daß die Argumente zwischen Prozeduren, die Zugriff auf die Daten benötigen, explizit übergeben werden. Prozeduren können Sub-Prozeduren und Funktionsprozeduren sein. Wenn Sie solche Prozeduren entwickeln, legen Sie eine eigene Routinenbibliothek an, die Sie auch in anderen Applikationen laden können.

Neben Ihren eigenen Prozeduren bietet Visual Basic eine große Sammlung interner Funktionen, die Zahlen, Strings und andere Datentypen analysieren und verarbeiten. Die internen Funktionen stehen jederzeit in jedem Modul zur Verfügung, so daß Sie sie überall nutzen können.

In Kapitel 9 kehren wir zurück zu der visuellen Natur von Visual Basic. Es zeigt Ihnen, wie man einer Applikation Standarddialogfelder hinzufügt.

Fragen und Antworten

Frage:

Warum gibt es keine lokalen und globalen Steuerelemente?

Antwort:

Alle Steuerelemente müssen im gesamten Code zur Verfügung stehen, deshalb sind alle Steuerelemente öffentlich für den gesamten Code. Die Steuerelemente sind öffentlich, weil sie vom Code abgetrennt sind. Wenn Sie keine Steuervariablen erzeugen und den Inhalt der Steuerelementeigenschaften dort ablegen, brauchen Sie sich nie Sorgen über den Gültigkeitsbereich zu machen.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welche Variable hat den größten Gültigkeitsbereich - lokal, auf Modulebene oder öffentlich?
2. Welche Variable hat den kleinsten Gültigkeitsbereich - lokal, auf Modulebene oder öffentlich?
3. Richtig/Falsch. Das Schlüsselwort **ByRef** ist optional.
4. Wie viele Werte kann eine Sub-Prozedur zurückgeben?
5. Nennen Sie zwei Funktionen, die eine Abkürzung für die **If**-Anweisung darstellen.
6. Was passiert, wenn das erste Argument von **Choose()** kleiner als 1 ist?
7. Was macht **Abs()**?
8. Was wird in den folgenden Anweisungen in der Variablen **strS** gespeichert?
 - a. `strS = Format("74135", "#####-#####")`
 - b. `strS = Format(d, "h ampm")`
 - c. `strS = Format(12345.67, "#####.###")`
9. Welchen Wert enthält **intN** nach der Ausführung der folgenden Anweisung (sehen Sie nicht in einer ASCII-Tabelle nach):
`intN = Asc(Chr(192))`
10. Was ist der Unterschied zwischen den Funktionen **Now** und **Time**?

Übungen

1. Schreiben Sie Listing 8.1 so um, daß **SalesTax()** eine Funktionsprozedur ist, die die berechnete Steuer an die aufrufende Prozedur zurückgibt. Verwenden Sie in der aufrufenden Prozedur **GetTotal()** eine **MsgBox()**-Funktion, um die von **SalesTax()** berechnete Steuer auszugeben.
 2. Schreiben Sie die folgende **If**-Anweisung als **IIf()**-Funktion:

```
If (intTotal >1000) Then  
strTitle = "Gute Arbeit!"  
Else  
strTitle = "Leider falsch."  
End If
```
 3. Schreiben Sie die folgende **If**-Anweisung als **Choose()**-Funktion:

```
If (ID = 1) Then  
intBonus = 50  
ElseIf (ID = 2) Then  
intBonus = 75  
ElseIf (ID = 3) Then  
intBonus = 100  
End If
```
 4. Welche Werte werden in diesen Anweisungen zugewiesen?

```
intN = Int(-5.6)  
intO = Fix(-5.6)  
intP = CInt(-5.6)
```
-

Woche 2

Tag 9

Dialogfelder

Dieses Kapitel zeigt Ihnen, wie Sie Ihrer Applikation Dialogfelder hinzufügen. Sie sollten jedoch nicht einfach beliebige Dialogfelder entwickeln. Statt dessen sollten Sie die sogenannten Standarddialogfelder nutzen, ein Steuerelement, das sechs verschiedene Dialogfelder für Ihre Applikationen bereitstellt. Wenn Ihr Benutzer etwas aus einer Dateiliste auswählen soll, oder wenn er einen Bericht aus Ihrer Visual-Basic-Applikation ausdrucken soll, stellen Sie ihm mit Hilfe von Dialogfeldern eine vertraute Oberfläche bereit.

Heute lernen Sie die folgenden Dinge:

- Standarddialogfelder zur besseren Akzeptanz durch den Benutzer
- Standarddialogfelder platzieren
- Methoden zum Anlegen von Standarddialogfeldern
- Eigenschaften zur Einrichtung der richtigen Dialogfeldoptionen
- Eingaben in einem Dialogfeld
- Die Anweisung `On Error Goto`, mit deren Hilfe Sie erkennen, ob der Benutzer auf die **Abbrechen**-Schaltfläche des Dialogfelds geklickt hat

Wozu Sie ein Standarddialogfeld brauchen

Je besser Ihre Applikation das Look&Feel gebräuchlicher Windows-Applikationen nachbildet, beispielsweise Microsoft Word, desto schneller werden Ihre Benutzer sich an sie gewöhnen. Wenn Sie Software schreiben, die Sie verkaufen möchten, dann ist es wichtig, daß Ihre Benutzer diese Software akzeptieren, insbesondere, wenn Sie erwarten, daß die Benutzer auch spätere Upgrades kaufen. Wenn Sie Ihre Programme nicht verkaufen, sondern Anwendungen innerhalb Ihrer Firma schreiben, dann bedeuten glückliche Benutzer weniger Wartungsanrufe und mehr Produktivität.

Wenn Sie also eine Applikation schreiben, in der eine Datei geöffnet werden soll, oder die Ausgaben auf den Drucker anbietet, haben Sie die folgenden Möglichkeiten:

- Sie bilden den Stil der Dialogfelder anderer Applikationen nach, die dieselbe Aufgabe erfüllen.
- Sie schreiben eigene Dialogfelder, in der Hoffnung, die Standarddialogfelder damit zu verbessern.

Sie könnten das Aussehen eines Dialogfelds, das erscheint, wenn Sie **Datei, Öffnen** auswählen,

möglicherweise verbessern, aber der Versuch wäre nicht klug. Zum einen entspricht Ihre Applikation dann *nicht* dem Standard. Ihre Benutzer haben eine viel steilere Lernkurve, wenn sie versuchen müssen, in einem Dialogfeld das zu tun, was sie von den meisten Windows-Applikationen bereits kennen. Darüber hinaus ist mehr Programmieraufwand erforderlich, weil es länger dauert, Programme zu schreiben, die für die allgemeinen Aufgaben keine Standarddialogfelder verwenden.

Was Sie tun sollten

Entwickeln Sie Applikationen, die dem Standard entsprechen und dieselben Menüs und Dialogfelder verwenden wie die meisten anderen Windows-Applikationen.

Der Grund dafür, warum unter Visual Basic weniger Entwicklungsaufwand entsteht, ist die Bereitstellung von Standarddialogfeldern, mit denen Sie Ihren Applikationen die im folgenden aufgelisteten Dialogfelder hinzufügen, die Sie nicht selbst entwickeln zu brauchen. Diese Dialogfelder sehen aus wie die Dialogfelder in den Standard-Windows-Programmen, und die sich auch genau so verhalten:

- **Farbauswahl** - Zeigt ein Dialogfeld an, in dem der Benutzer eine Farbe aus einer Palette auswählt oder Farben anpaßt.
- **Schriftauswahl** - Zeigt ein Dialogfeld an, in dem der Benutzer Schriftart und Schriftgröße auswählt.
- **Datei öffnen** - Zeigt ein Dialogfeld an, in dem der Benutzer einen Dateinamen auswählt, um diese Datei in Ordnern, auf Laufwerken und sogar auf Netzwerk- PCs zu öffnen.
- **Druckauswahl** - Zeigt ein Dialogfeld an, in dem der Benutzer einen Drucker und die Druckereinstellungen für alle Windows-Drucker auswählt.
- **Speichern unter** - Zeigt ein Dialogfeld an, in dem der Benutzer einen Dateinamen angibt, unter dem eine Datei in Ordnern, auf Laufwerken und auch auf Netzwerk-PCs gespeichert wird.
- **Hilfe** - Startet die Windows-Hilfemaschine und zeigt ein Dialogfeld an, in dem der Benutzer die Hilfe auswählt, die Sie für Ihre Applikation bereitstellen.

Das Standarddialog-Steuererelement hinzufügen

Zunächst finden Sie das Standarddialog-Steuererelement nicht in der Werkzeugsammlung. Visual Basic zeigt nicht alle Steuererelemente sofort auf der Werkzeugsammlung an, weil diese dadurch zu umfangreich würde. Außerdem brauchen Sie nicht immer alle Steuererelemente, die in Visual Basic zur Verfügung stehen. Wenn Sie in Ihrer Applikation jedoch Standarddialogfelder verwenden möchten, die den gewohnten Oberflächen entsprechen, fügen Sie Ihrer Werkzeugsammlung das Standarddialog-Steuererelement hinzu. Dazu gehen Sie wie folgt vor:

1. Drücken Sie (Strg)+(T) (der Shortcut für **Projekt, Komponenten**), um das Dialogfeld **Komponenten** aufzurufen, wie in Abbildung 9.1 gezeigt.

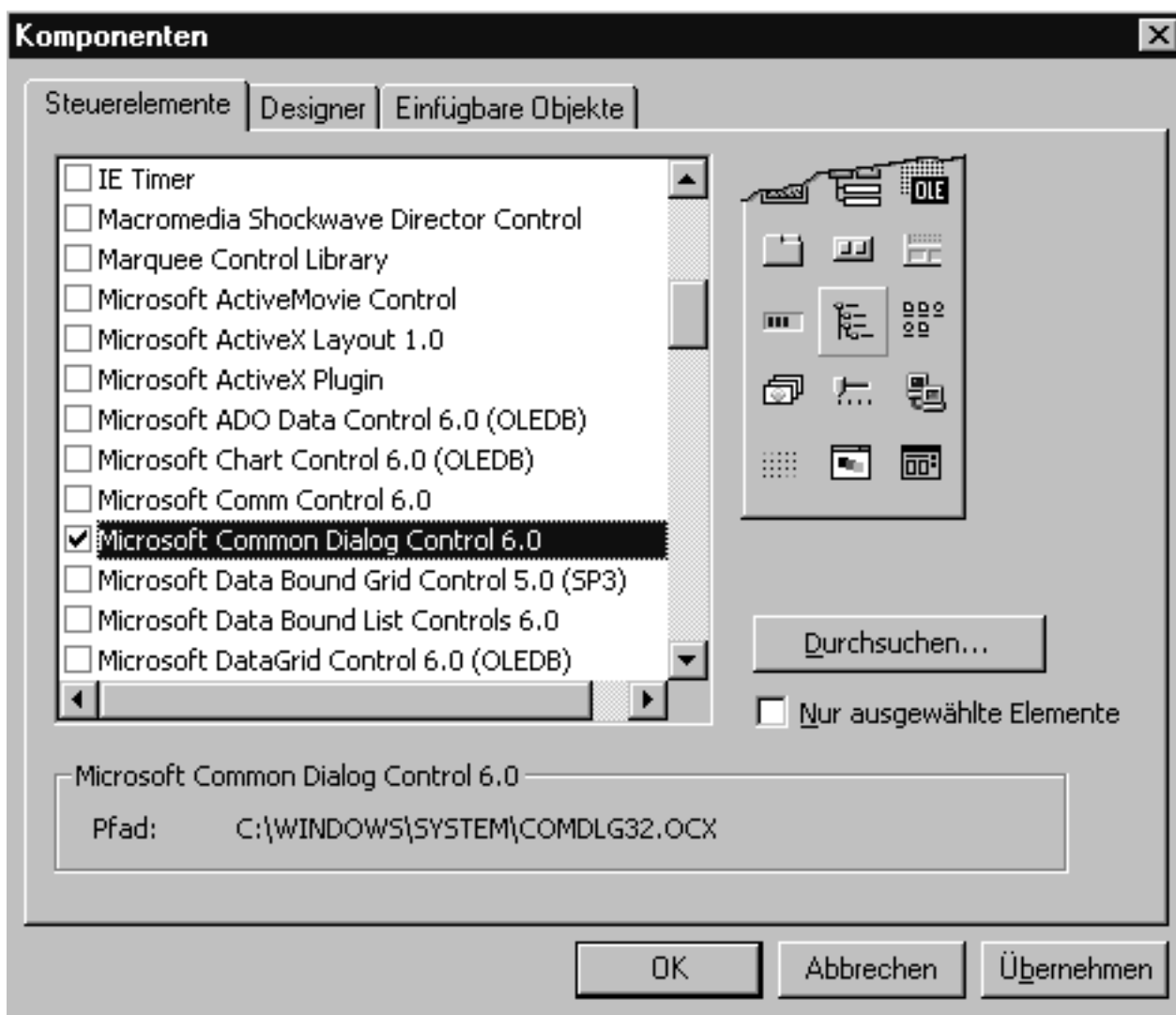


Abbildung 9.1:

Im Dialogfeld Komponenten sehen Sie die auf Ihrem System verfügbaren Komponenten, die Sie Ihrer Werkzeugsammlung hinzufügen können.

2. Blättern Sie nach unten bis zum Eintrag *Microsoft Common Dialog Control 6.0*.
3. Markieren Sie diesen Eintrag und klicken Sie auf **OK**. Jetzt wird ganz unten in Ihrer Werkzeugsammlung das Standarddialog-Steuerelement angezeigt.



Betrachten Sie die Einträge im Dialogfeld **Komponenten**. Vielleicht finden Sie weitere Steuerelemente, die Sie brauchen können. Es handelt sich dabei ausschließlich um ActiveX-Steuerelemente. Die meisten Namen sind so aussagekräftig, daß Sie daran erkennen, was Sie erwartet. Beispielsweise ist *Microsoft Calendar Control 8.0* ein Steuerelement, mit dem Sie Ihrer Applikation Kalenderoperationen hinzufügen, beispielsweise wenn Sie ein Programm für eine Zeitabrechnung oder einen Zeitplaner entwickeln möchten. Prüfen Sie nach dem Hinzufügen die Eigenschaften der betreffenden Steuerelemente, um zu sehen, was alles zur Verfügung steht. Lesen Sie in der Online-Dokumentation nach, welche Ereignisse und Methoden sie unterstützen. Weitere Steuerelemente finden Sie in Computer-Zeitschriften und im Internet. In Kapitel 17 erfahren Sie, wie Sie eigene ActiveX-Steuerelemente programmieren, die Sie Ihrer Werkzeugsammlung hinzufügen.

Die Arbeit mit dem Standarddialog-Steuererelement

Abbildung 9.2 zeigt das Standarddialog-Steuererelement auf einem Formular. Wie Sie sehen, sieht das Steuererelement überhaupt nicht aus wie die zuvor aufgelisteten Dialogfelder. Statt dessen ist es irgendwie viel zu klein, als daß es für irgend etwas genutzt werden könnte. Wenn Sie versuchen, es zu vergrößern, ignoriert Visual Basic dies, obwohl um das Steuererelement acht Griffe für die Größenänderung bereitgestellt werden.

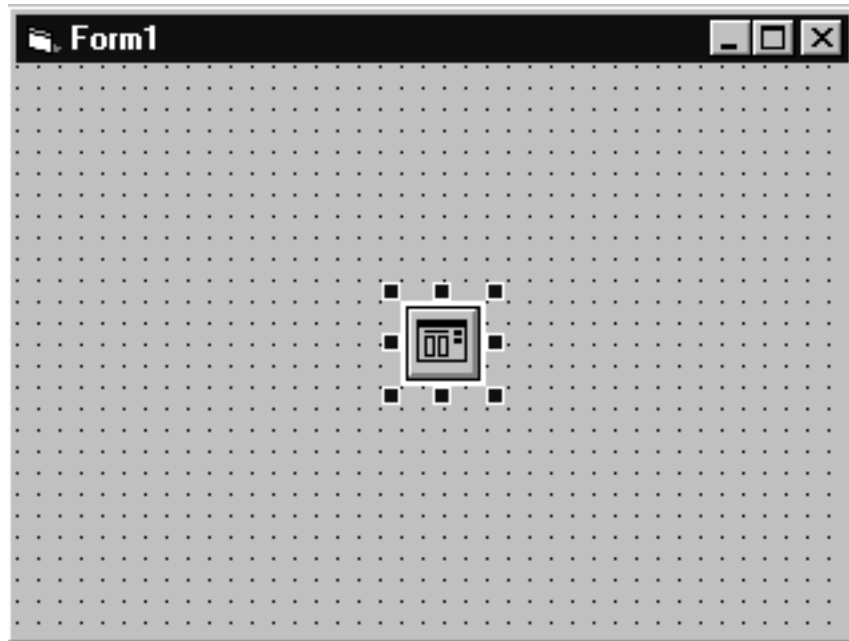


Abbildung 9.2:
Das Standarddialog-Steuererelement sieht auf Ihrem Formular nicht nach viel aus.

Wenn Sie das Programm ausführen, sieht das Standarddialogfeld aus wie eines der zuvor aufgelisteten Dialogfelder. Visual Basic zeigt dieses Dialogfeld immer in der Bildschirmmitte an, unabhängig davon, wo Sie das Steuererelement auf dem Formular angelegt haben. Sie können es also an einer beliebigen Stelle platzieren, wo es nicht stört. Wenn das Dialogfeld schließlich aufgerufen wird, zeigt Visual Basic es automatisch in der Bildschirmmitte an. Bis das Standarddialogfeld angezeigt wird, wird es von Visual Basic verborgen, so daß es nicht über anderen Steuerelemente auf Ihrem Formular erscheint.

Wenn Ihr Programm eines der spezifischen Dialogfelder des Standarddialogs aufruft, erscheint dieses auf dem Bildschirm, falls Sie in Ihrem Visual-Basic-Programm das korrekte Dialogfeld ausgewählt haben.

Welches Formular des Standarddialogs erscheint, wird durch die entsprechenden Eigenschaften und die Ausführung der jeweiligen Methode für die Anzeige des Dialogfelds bestimmt. Wie Sie aus früheren Kapiteln wissen, ist eine Methode eine interne Prozedur, die Sie auf ein bestimmtes Steuererelement anwenden können. Um ein bestimmtes Dialogfeld anzuzeigen, führen Sie eine der folgenden Methoden auf das Standarddialog-Steuererelement aus:

- ShowColor zeigt das Dialogfeld **Farbe** an.
- ShowFont zeigt das Dialogfeld **Schriftart** an.
- ShowHelp zeigt das Dialogfeld **Hilfe** an.
- ShowOpen zeigt das Dialogfeld **Datei öffnen** an.
- ShowPrinter zeigt das Dialogfeld **Drucken** an.

- ShowSave zeigt das Dialogfeld **Speichern unter** an.

Wenn Sie einem Standarddialog den Namen **dbFont** geben, könnten Sie die Methode **ShowFont** dafür aufrufen, so daß das Steuerelement das Dialogfeld **Schriftart** anzeigt, nachdem Sie einige Eigenschaften gesetzt haben:

```
dbFont.ShowFont ' Dialogfeld Datei öffnen anzeigen
```

Wie Sie sehen, brauchen Sie nur ein einziges Standarddialog-Steuerelement, um ein oder mehr Standarddialogfelder zu erzeugen. Die Methode löst das Steuerelement aus, aber Sie sollten die entsprechenden Eigenschaften setzen, bevor Sie eines der sechs Dialogfelder aktivieren, die Sie in den nächsten Abschnitten kennenlernen werden.



Wenn Sie mehrere der Dialogfelder in Ihrer Applikation anzeigen möchten, legen Sie ein Standarddialog-Steuerelement an und wenden die verschiedenen Methoden darauf an, die Sie hier kennenlernen. Sie können aber auch mehrere Standarddialog-Steuerelemente anlegen und sie nur für jeweils ein ganz bestimmtes Dialogfeld verwenden. Ein Steuerelement ist zwar einfacher zu verwalten als mehrere, aber vielleicht möchten Sie mehrere Steuerelemente verwenden, um die Eigenschaftswerte nur einmal setzen zu müssen. Wenn Sie nur ein einziges Steuerelement einsetzen, müssen Sie die Eigenschaften jedesmal ändern, wenn Sie ein anderes Dialogfeld anzeigen möchten.

Alle vom Standarddialog-Steuerelement erzeugten Dialogfelder tun nichts weiter, als dem Benutzer eine Auswahl zur Verfügung zu stellen. Mit anderen Worten, wenn der Benutzer eine Schrift aus dem Dialogfeld **Schriftart** auswählt und dann auf **OK** klickt, ändert sich die Bildschirmschrift nicht. Das Standarddialog-Steuerelement soll nur eine allgemeine Oberfläche für die Standarddialogfelder bereitstellen und die Eigenschaften gemäß der Auswahlen des Benutzers setzen. Nachdem der Benutzer das Dialogfeld geschlossen hat, werten Sie in Ihrem Visual-Basic-Code diese Eigenschaftswerte aus und führen die entsprechenden Aktionen aus.

Das Dialogfeld **Farbe**

Weil Windows so viele Farben unterstützt, stellt das Dialogfeld **Farbe** eine einfache Möglichkeit dar, Ihren Benutzern eine Auswahl daraus zu bieten. Wenn Ihre Benutzer beispielsweise in der Lage sein sollen, die Hintergrundfarbe eines Formulars zu ändern, ist es nicht sinnvoll, Ihrem Benutzer das folgende Eingabefeld anzuzeigen:

```
strAns = InputBox("Welche Farbe soll der Hintergrund haben?")
```

Beachten Sie, daß die Eigenschaft **BackColor** als hexadezimaler Farbcode angegeben wird (wie alle anderen farbbezogenen Eigenschaften, beispielsweise **ForeColor**). Wenn Ihr Benutzer als Antwort in das Eingabefeld **Rot** eingibt, dann können Sie der **BackColor** -Eigenschaft diese Antwort nicht zuweisen:

```
frmTitle.BackColor = strAns ' Das funktioniert NICHT!
```

Das Dialogfeld **Farbe** bietet Ihrem Benutzer nicht nur eine Standardmethode, eine Farbe auszuwählen, sondern wandelt die ausgewählte Farbe gleichzeitig in ihren Hexadezimalwert um. Nachdem der Benutzer das Dialogfeld geschlossen hat, weisen Sie der **BackColor**-Eigenschaft eine der Eigenschaften des Dialogfelds zu, um den Formularhintergrund entsprechend zu ändern.

Um ein Standarddialog-Steuerelement in das Dialogfeld **Farbe** umzuwandeln und eine Farbauswahl

zu ermöglichen, gehen Sie wie folgt vor:

1. Setzen Sie die Eigenschaft `DialogTitle` des Standarddialogfelds auf den Titel, der in der Titelleiste der Farbauswahl erscheinen soll, beispielsweise »Wählen Sie eine Hintergrundfarbe aus«.
2. Setzen Sie die `Flags`-Eigenschaft des Standarddialogfelds auf einen der Werte aus Tabelle 9.1. Die Tabelle beschreibt, wie das Dialogfeld für die Farbauswahl aussehen soll. Wenn Sie mehrere der in Tabelle 9.1 beschriebenen Flags setzen möchten, wie beispielsweise eine Ausgangsfarbe oder die Bereitstellung einer **Hilfe**- Schaltfläche, können Sie die Flag-Werte addieren.
3. Erzeugen Sie das Dialogfeld **Farbe**, indem Sie im Code für das Standarddialog- Steuerelement die Methode `ShowColor` aufrufen.

Tabelle 9.1: Sie können einen oder mehrere dieser Flags-Werte kombinieren, um die Darstellung des Dialogfelds für die Farbauswahl zu bestimmen.

Konstantenname	Flags-Wert	Beschreibung
<code>cdlCCRGBInit</code>	1	Die Ausgangsfarbe setzen.
<code>cdlCCFullOpen</code>	2	Zeigt das gesamte Dialogfeld an, auch den Abschnitt für eine benutzerdefinierte Farbe.
<code>cdlCCPreventFullOpen</code>	4	Verhindert, daß Benutzer eigene Farben definieren.
<code>cdlCCHelpButton</code>	8	Zeigt eine Hilfe-Schaltfläche im Dialogfeld an.

Angenommen, Sie möchten das Auswahlfeld **Farbe** anzeigen, dem Benutzer eine Farbauswahl ermöglichen und im Dialogfeld eine **Hilfe**-Schaltfläche anzeigen. Wenn Sie in Ihrem Formular das Standarddialog-Steuerelement `cdbColor` angelegt haben, könnten Sie das Farbauswahlfeld mit dem folgenden Code erzeugen:

```
' Flags-Eigenschaft setzen
cdbColor.Flags = cdlCCFullOpen + cdlCCHelpButton ' Farbdialog
erzeugen
' Farbdialog anzeigen
cdbColor.ShowColor
```

Abbildung 9.3 zeigt das Dialogfeld.

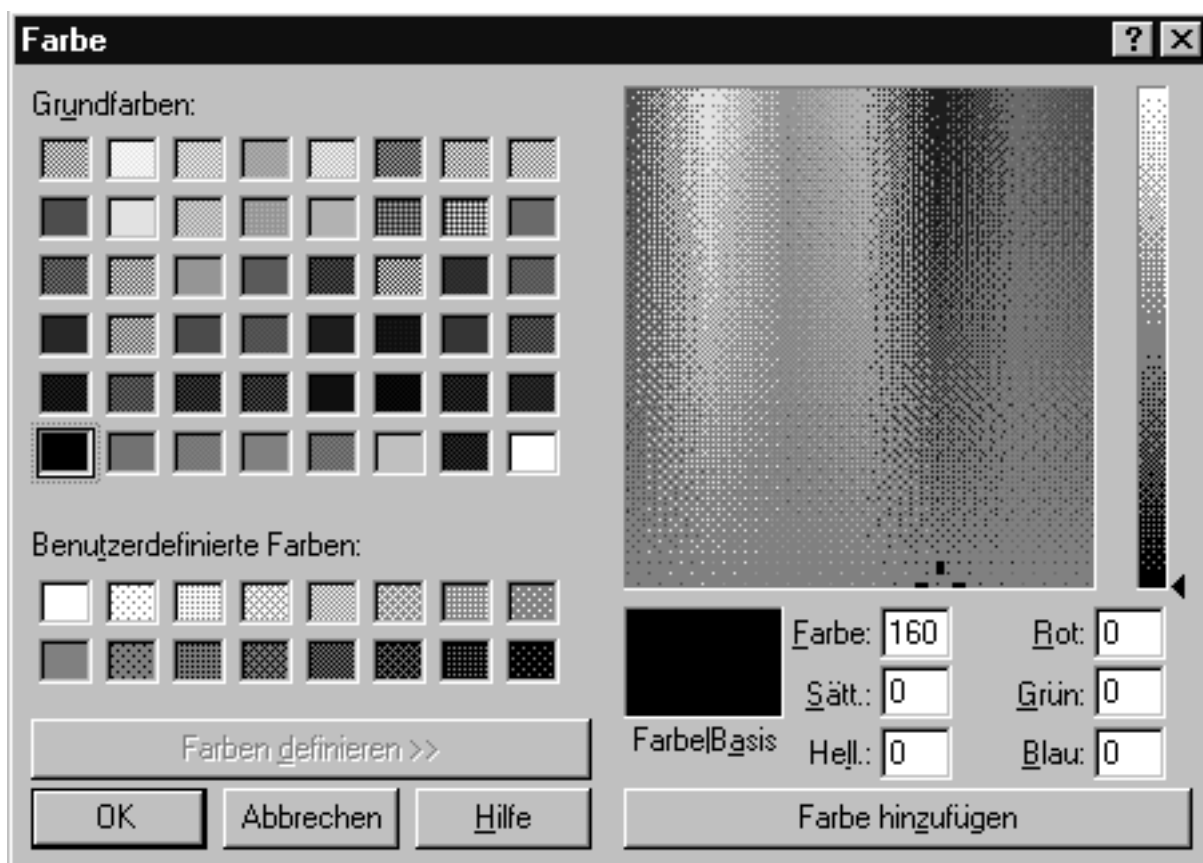


Abbildung 9.3:
Das Dialogfeld **Farbe** wird mit Hilfe der Methode **ShowColor** angezeigt.

Falls Sie dem Benutzer nur einen kleineren Farbbereich anzeigen möchten, wo er die Schaltfläche **Farben definieren** anklicken muß, um Farben zu definieren, lassen Sie das `cdlCCFullOpen`-Flag weg. Abbildung 9.4 zeigt das kleinere Dialogfeld **Farbe**.

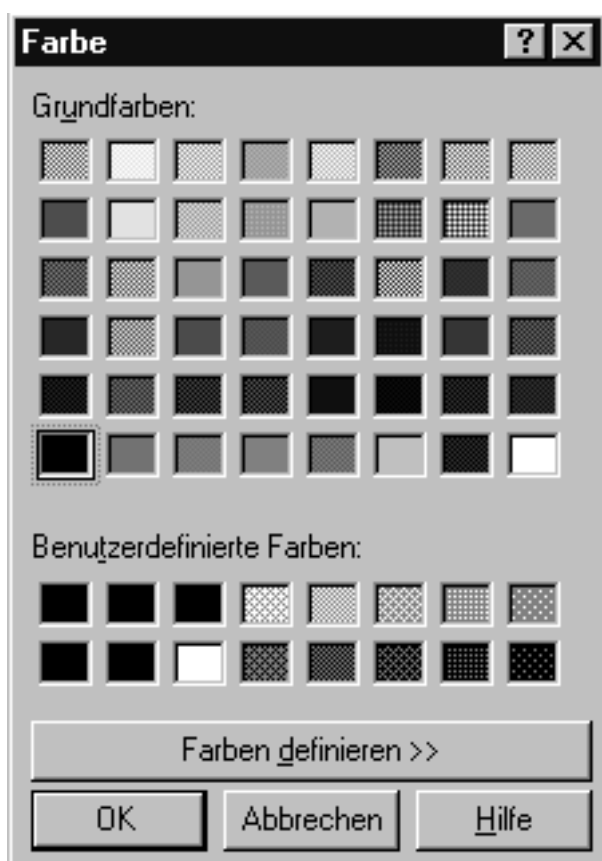


Abbildung 9.4:

Sie können das Dialogfeld Farbe verkleinern.

Wenn der Benutzer das Dialogfeld schließt, werden seine Eigenschaften so gesetzt, wie der Benutzer sie ausgewählt hat. Die wichtigste Eigenschaft ist `Color`, die den Hexadezimalcode für die vom Benutzer ausgewählte oder definierte Farbe enthält. Sie könnten beispielsweise den folgenden Code verwenden:

```
' Die Hintergrundfarbe eines Formulars auf die im Dialogfeld  
' Farbe ausgewählte Farbe setzen  
frmTitle.ForeColor = cdbColor.Color
```

Die Abbrechen-Schaltfläche

Ihr Code muß feststellen können, ob der Benutzer eine Farbe ausgewählt und auf die **OK**-Schaltfläche geklickt hat, oder ob er auf die **Abbrechen**-Schaltfläche geklickt hat, d.h. der Benutzer wollte keinen Farbwert ändern. Sie müssen nicht nur im Dialogfeld **Farbe** in der Lage sein, das Anklicken der **Abbrechen**-Schaltfläche zu erkennen, sondern auch in allen anderen Standarddialogfeldern.

Um zu prüfen, ob der Benutzer auf die **Abbrechen**-Schaltfläche geklickt hat, brauchen Sie einen neuen Visual-Basic-Befehl - die Anweisung `On Error Goto`. Diese Anweisung springt innerhalb der Programmausführung an ein Code-Label, falls während der nachfolgenden Anweisungen ein Fehler auftritt. Die Anweisung

```
On Error Goto dbErrorHandler
```

weist Visual Basic an, zum Code mit dem Label `dbErrorHandler` zu springen, wenn in einer der Zeilen, die der Anweisung `On Error Goto` folgen, ein Fehler auftritt (das gilt bis zum Ende der Prozedur).

Ein Code-Label ist ein Label im Code, das Sie nach denselben Regeln bezeichnen wie Variablen. Ein Label muß jedoch mit einem Doppelpunkt enden, um es von einer Variablen zu unterscheiden. Bei dem oben gezeigten Beispiel brauchen Sie im Code irgendwo hinter der Anweisung `On Error Goto` das folgende Code-Label (im allgemeinen plazieren die Programmierer dieses Code-Label irgendwo am Prozedurende):

```
dbErrorHandler:
```

Die Anweisungen hinter dem Code-Label werden ausgeführt, wenn in der Prozedur ein Fehler auftritt; eine `Exit`-Anweisung kann die Prozedur vorzeitig beenden. Visual Basic löst eine Fehlerbedingung aus, wenn der Benutzer auf die **Abbrechen**-Schaltfläche klickt und Sie die `CancelError`-Eigenschaft auf `True` gesetzt haben. Das Anklicken von **Abbrechen** ist zwar kein echter Fehler, aber wenn Sie es wie eine Fehlerbedingung behandeln, könnten Sie beispielsweise Code wie in Listing 9.1 gezeigt schreiben, um die Situation zu verarbeiten.

Listing 9.1: Sie bestimmen, was passiert, wenn der Benutzer auf die Abbrechen-Schaltfläche klickt

```
1: Private Sub mnuViewColor_Click()  
2: cdbColor.CancelError = True ' Erzwingt einen Fehler, wenn der  
3: ' Benutzer auf Abbrechen klickt  
4: On Error Goto dbErrorHandler ' Sprung im Fehlerfall  
5:  
6: ' Color-Flags-Eigenschaft setzen.
```

```

7: cdbColor.Flags = cdlCCFullOpen + cdlCCHelpButton ' Farben-
8: ' Dialogfeld erzeugen
9: ' Dialogfeld anzeigen.
10: cdbColor.ShowColor
11:
12: ' Hintergrundfarbe eines Formulars auf die im
13: ' Dialogfeld ausgewählte Farbe setzen
14: frmTitle.ForeColor = cdbColor.Color
15: Exit Sub ' Die reguläre Prozedur ist fertig
16: dbErrHandler:
17: ' Der Benutzer hat auf Abbrechen geklickt, deshalb wird
18: ' die Prozedur ignoriert und die Farbe nicht geändert
19: Exit Sub
20: End Sub

```

Wenn der Benutzer eine Farbe auswählt und auf OK klickt, weist Zeile 14 dem Formularhintergrund diese Farbe zu. Statt die Prozedur zu beenden, wenn der Benutzer auf **Abbrechen** klickt, könnten Sie auch Standardwerte (zwischen den Zeilen 16 und 19) für den Formularhintergrund setzen, statt die aktuellen Werte beizubehalten und die Prozedur zu verlassen.



Die Fehlerverarbeitung in Zeile 16 wird ausgeführt, wenn ein Fehler auftritt, nicht nur, wenn die **Abbrechen**-Schaltfläche angeklickt wird. In Kapitel 16 erfahren Sie, wie man das Systemobjekt `Err` auswertet, um anhand der Fehlernummer genau festzustellen, welcher Fehler aufgetreten ist.

Das Dialogfeld **Schriftart**

Das Standarddialog-Steuerelement erzeugt das Dialogfeld **Schriftart**, das Sie von Ihren anderen Windows-Applikationen bereits kennen. Sie sollten dieses Dialogfeld nicht nur deshalb verwenden, weil es sich dabei um den Standard handelt. Sie wissen nicht genau, welche Schriftarten der PC enthält, auf dem Ihre Applikation ausgeführt wird. Das Standarddialog-Steuerelement **Schriftart** durchsucht das System des Benutzers nach allen Schriften auf dem Computer und zeigt die gefundenen Schriften im Dialogfeld **Schriftart** an.

Abbildung 9.5 zeigt ein typisches **Schriftart**-Dialogfeld, das erscheint, wenn Sie die `ShowFont`-Methode auf das Standarddialog-Steuerelement anwenden.

Wie für das Dialogfeld **Farbe** müssen Sie auch für das Dialogfeld **Schriftart** die `Flags`-Eigenschaft auf einen bestimmten Wert setzen. Die `Flags`-Eigenschaften unterscheiden sich von denen für das Dialogfeld **Farben**, weil das Dialogfeld **Schriftart** viel komplexer ist.



Die `Flags`-Werte des **Schriftart**-Dialogfelds können sehr groß werden. Deshalb weisen die Visual-Basic-Programmierer der `Flags`-Eigenschaft entweder Konstantennamen oder hexadezimale Werte zu. Tabelle 9.2 listet die `Flags`-Eigenschaften auf. Wie fast überall in der Visual-Basic-Programmierung wird die Programmwartung einfacher, wenn Sie Konstantennamen verwenden, weil die Namen Aufschluß über die Aufgaben geben, während die Hexadezimalwerte nicht selbstdokumentierend sind.

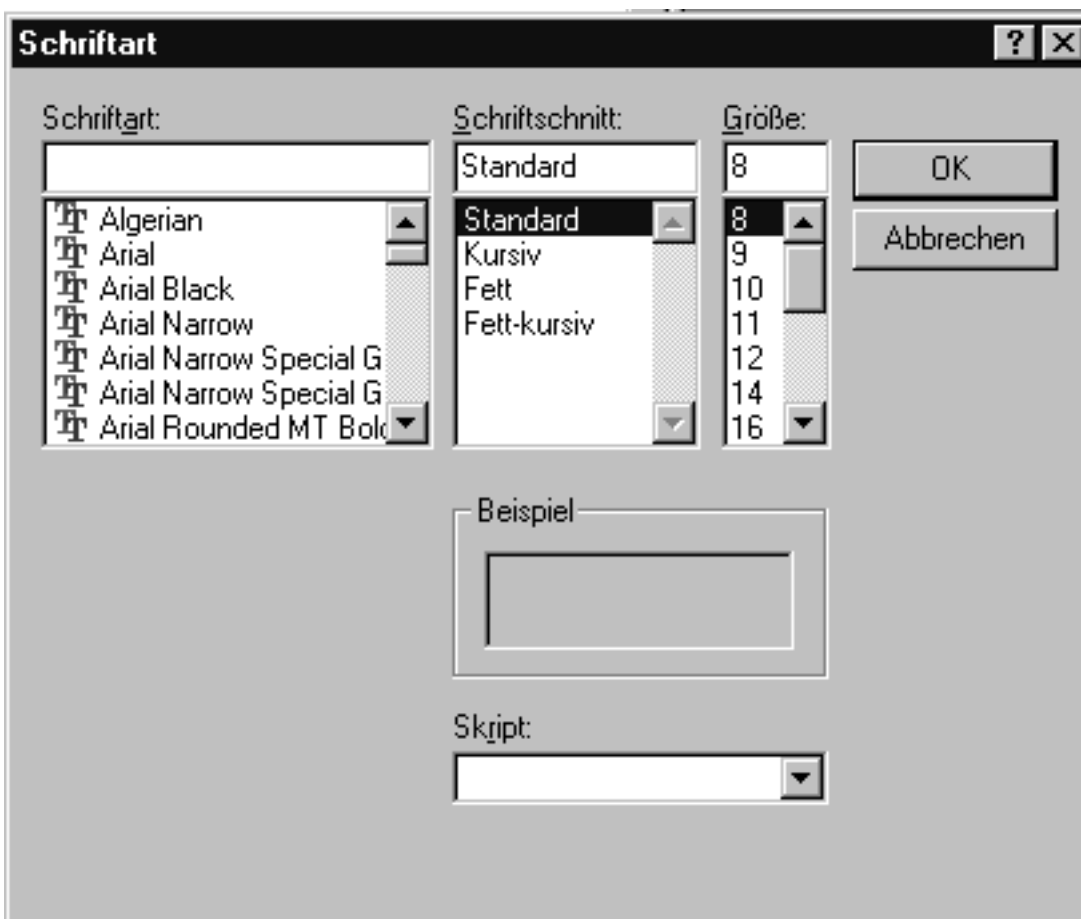


Abbildung 9.5:
 Die Benutzer wählen aus dem Dialogfeld Schriftart einen Schriftstil und einen Schriftschnitt aus.

Tabelle 9.2: Setzen Sie die Flags-Werte für das Dialogfeld Schriftart, bevor Sie die Methode ShowFont aufrufen

Konstantenname	Flags-Wert	Beschreibung
cdlCFANSIOnly	&H400	Stellt sicher, daß das Dialogfeld nur Schriften aus dem Windows-Zeichensatz anbietet, und keine symbolbasierten Schriften.
cdlCFApply	&H200	Aktiviert die Schaltfläche Anwenden .
cdlCFBoth	&H3	Listet die verfügbaren Drucker- und Bildschirmschriften im Dialogfeld auf. Die hDC-Eigenschaft gibt den Gerätekontext für den Drucker an.
cdlCFEffects	&H100	Bietet im Dialogfeld die Auswahlen Durchgestrichen , Unterstrichen und Farbeffekte an.
cdlCFFixedPitchOnly	&H4000	Stellt sicher, daß das Dialogfeld nur Schriften mit festem Durchschuß anbietet.
cdlCFForceFontExist	&H10000	Zeigt eine Fehlermeldung an, wenn der Benutzer versucht, eine Schrift oder einen Stil auszuwählen, die bzw. den es nicht gibt.
cdlCFHelpButton	&H4	Zeigt die Hilfe -Schaltfläche für das Dialogfeld an.

<code>cdlCFLimitSize</code>	<code>&H2000</code>	Stellt sicher, daß im Dialogfeld nur Schriftgrößen angeboten werden, die in dem durch die Eigenschaften <code>Min</code> und <code>Max</code> vorgegebenen Bereich liegen.
<code>cdlCFNoFaceSel</code>	<code>&H80000</code>	Standardmäßig ist kein Schriftname ausgewählt.
<code>cdlCFNoSimulations</code>	<code>&H1000</code>	Deaktiviert GDI-Schriftsimulationen (Graphic Device Interface).
<code>cdlCFNoSizeSel</code>	<code>&H200000</code>	Standardmäßig ist keine Schriftgröße ausgewählt.
<code>cdlCFNoStyleSel</code>	<code>&H100000</code>	Standardmäßig ist kein Schriftstil ausgewählt.
<code>cdlCFNoVectorFonts</code>	<code>&H800</code>	Verhindert, daß Vektorschriften ausgewählt werden.
<code>cdlCFPrinterFonts</code>	<code>&H2</code>	Listet nur Schriften auf, die der Drucker unterstützt, wie in der Eigenschaft <code>hDC</code> vorgegeben.
<code>cdlCFScalableOnly</code>	<code>&H20000</code>	Erlaubt nur die Auswahl von skalierbaren Schriften.
<code>cdlCFScreenFonts</code>	<code>&H1</code>	Listet nur die vom System unterstützten Bildschirmschriften auf.
<code>cdlCFTTOnly</code>	<code>&H40000</code>	Erlaubt nur die Auswahl von TrueType-Schriften.
<code>cdlCFWYSIWYG</code>	<code>&H8000</code>	Bestimmt, daß im Dialogfeld nur die Schriften ausgewählt werden können, die sowohl für den Drucker als auch für den Bildschirm zur Verfügung stehen (wenn Sie dieses Flag setzen, sollten Sie auch <code>cdlCFBoth</code> und <code>cdlCFScalableOnly</code> setzen).

 Warnung	Sie müssen mindestens eine der drei folgenden Flag-Eigenschaften setzen, damit das Dialogfeld Schriftart erscheint: <code>cdlCFScreenFonts</code>, <code>cdlCFPrinterFonts</code> und <code>cdlCFBoth</code>. Wenn Sie keine dieser Eigenschaften setzen, gibt Visual Basic eine Fehlermeldung aus, sobald Sie die <code>ShowFont</code>-Methode anwenden.

Listing 9.2 zeigt, wie Sie das Dialogfeld **Schriftart** einrichten, anzeigen und auf Eingaben reagieren.

Listing 9.2: Zeigen Sie das Dialogfeld Schriftart an, wenn Ihre Benutzer einen Schriftstil und eine Schriftgröße auswählen sollen

```

1: ' Flags-Eigenschaften für das Dialogfeld Schriftart setzen.
2: CdbFont.Flags = cdlCFBoth Or cdlCFEffects
3: CdbFont.ShowFont ' Schriftart-Dialogfeld anzeigen
4: ' Die Eigenschaften eines Bezeichnungsfelds auf die vom
5: ' Benutzer ausgewählten Schrifteigenschaften setzen
6: LblMessage.Font.Name = CdbFont.FontName
7: LblMessage.Font.Size = CdbFont.FontSize
8: LblMessage.Font.Bold = CdbFont.FontBold
9: LblMessage.Font.Italic = CdbFont.FontItalic
10: LblMessage.Font.Underline = CdbFont.FontUnderline
11: LblMessage.Font.Strikethru = CdbFont.FontStrikethru
12: LblMessage.ForeColor = CdbFont.Color

```

Achten Sie auf die mehrteiligen Anweisungen in den Zeilen 6 bis 10. Sie haben bisher noch nie gesehen, daß Eigenschaftsnamen durch zwei Punkte voneinander abgetrennt waren. Wenn Sie im Eigenschaftenfenster für die Eigenschaft **Font** auf die Schaltfläche mit den drei Punkten klicken, erscheint das Dialogfeld **Schriftart**, in dem Sie verschiedene Werte setzen können. Eine **Font**-Eigenschaft enthält also mehrere Werte und Sie müssen den Eigenschaftswert weiter qualifizieren, um den Wert zu beschreiben, den Sie setzen möchten. Es gibt mehrere **Font**-Eigenschaftswerte, die jeweils verschiedene Schriftstile, Schriftgrößen oder Farben usw. festlegen. Diese mehrteiligen Namen werden von rechts nach links gelesen. Betrachten Sie die folgende Anweisung aus Zeile 8:

```
LblMessage.Font.Bold = CdbFont.FontBold
```

Zeile 8 teilt Visual Basic mit, die **FontBold**-Eigenschaft des Dialogfelds (die entweder **True** oder **False** ist) dem **Bold**-Attribut der **Font**-Eigenschaft des Bezeichnungsfelds **lblMessage** zuzuweisen.

Hilfe bei der Einrichtung von Steuerelementen

Sie können alle Eigenschaften des Standarddialog-Steuerelements zur Laufzeit setzen, aber Visual Basic bietet eine geniale Möglichkeit, viele Eigenschaften schon zur Entwurfszeit zu setzen.

Das Standarddialog-Steuerelement enthält die Eigenschaft (**Benutzerdefiniert**). Wenn Sie auf die Schaltfläche mit den drei Punkten klicken, die für diese Eigenschaft angezeigt wird, zeigt Visual Basic das Dialogfeld **Eigenschaftenseiten** an (siehe Abbildung 9.6).

Auf diesen Eigenschaftenseiten ist es ganz einfach, Eigenschaften für ein Dialogfeld zu setzen. Hier sehen Sie die wichtigsten Eigenschaften für die verschiedenen Standarddialogstile. Wenn Sie beispielsweise möchten, daß im Dialogfeld **Schriftart** als Vorgabe eine Schrift mit 12 Punkt Fett angezeigt wird, geben Sie in das Textfeld **FontSize** 8 ein und markieren das Kontrollkästchen **Bold**.



Abbildung 9.6:

Zur Entwurfszeit werden Eigenschaften im Dialogfeld Eigenschaftenseiten gesetzt.

Der Standarddialog zum Öffnen von Dateien

Tabelle 9.3 zeigt die `Flags`-Eigenschaften, die Sie setzen können, bevor Sie die Methode `ShowOpen` anwenden. Das Dialogfeld **Datei öffnen**, das Sie in Abbildung 9.7 sehen, bietet Ihren Benutzern eine Standardoberfläche zum Öffnen von Dateien. Das Dialogfeld übernimmt die Auswahl des Ordners und des Netzwerks, falls das vor der Dateiauswahl erforderlich ist.



Sie brauchen Tabelle 9.3 auch für die `Flags`-Eigenschaften des Dialogfelds **Speichern unter**.

Tabelle 9.3: Die `Flags`-Werte für die Dialogfelder **Datei öffnen** und **Speichern unter**

Konstantenname	Flags-Wert	Beschreibung
<code>cdlOFNAllowMultiselect</code>	<code>&H200</code>	Ermöglicht, daß im Listenfeld Dateiname mehrere Dateien ausgewählt werden. Die <code>FileName</code> -Eigenschaft gibt einen String zurück, der alle ausgewählten Dateinamen enthält (die Namen sind im String durch Leerzeichen voneinander getrennt).
<code>cdlOFNCreatePrompt</code>	<code>&H2000</code>	Fragt den Benutzer, ob eine Datei angelegt werden soll, die noch nicht existiert. Dieses Flag setzt automatisch die Flags <code>cdlOFNPathMustExist</code> und <code>cdlOFNFileMustExist</code> .
<code>cdlOFNExplorer</code>	<code>&H80000</code>	Verwendet das dem Explorer ähnliche Dialogfeld zum Öffnen von Dateien.
<code>cdlOFNExtensionDifferent</code>	<code>&H400</code>	Weist darauf hin, daß sich die Dateinamenerweiterung des zurückgegebenen Dateinamens von der in der <code>DefaultExt</code> -Eigenschaft angegebenen Erweiterung unterscheidet. Dieses Flag wird nicht gesetzt, wenn die <code>DefaultExt</code> -Eigenschaft <code>Null</code> enthält, wenn die Erweiterungen übereinstimmen, oder wenn die Datei keine Erweiterung hat. Sie können den Wert dieses Flags überprüfen, nachdem das Dialogfeld geschlossen wurde.

cdlOFNFileMustExist	&H1000	Die Benutzer dürfen nur Dateinamen eingeben, die existieren. Wenn dieses Flag gesetzt ist und der Benutzer gibt einen ungültigen Dateinamen ein, wird eine Warnung angezeigt. Dieses Flag setzt automatisch das Flag cdlOFNPathMustExist.
cdlOFNHelpButton	&H10	Zeigt die Hilfe -Schaltfläche für das Dialogfeld an.
cdlOFNHideReadOnly	&H4	Verbirgt das Kontrollkästchen Mit Schreibschutz öffnen .
cdlOFNLongNames	&H200000	Erlaubt lange Dateinamen.
cdlOFNNoChangeDir	&H8	Zwingt das Dialogfeld, das aktuelle Verzeichnis so zu setzen, wie es beim Öffnen des Dialogfelds gesetzt war.
cdlOFNNoDereferenceLinks	&H100000	Verbietet die Dereferenzierung von <i>Shell-Links</i> (auch als <i>Shortcuts</i> bezeichnet). Standardmäßig bewirkt die Auswahl eines Shell-Links, daß dieser von der Shell dereferenziert wird.
cdlOFNNoLongNames	&H40000	Verbietet lange Dateinamen.
cdlOFNNoReadOnlyReturn	&H8000	Spezifiziert, daß die zurückgegebene Datei das Attribut Read-Only nicht gesetzt hat und sich nicht in einem schreibgeschützten Verzeichnis befindet.
cdlOFNNoValidate	&H100	Erlaubt ungültige Zeichen im zurückgegebenen Dateinamen.
cdlOFNOverwritePrompt	&H2	Bewirkt, daß das Dialogfeld Speichern unter eine Warnung erzeugt, wenn der angegebene Dateiname bereits existiert. (Die Benutzer können dann wählen, ob die Datei überschrieben werden soll.)
cdlOFNPathMustExist	&H800	Die Benutzer dürfen nur gültige Pfade eingeben. Wenn dieses Flag gesetzt ist und die Benutzer einen ungültigen Pfad eingeben, erscheint eine Warnung.
cdlOFNReadOnly	&H1	Markiert das Kontrollkästchen Mit Schreibschutz öffnen , wenn das Dialogfeld erzeugt wird. Dieses Flag gibt außerdem den Status des Kontrollkästchens Mit Schreibschutz öffnen nach dem Schließen des Dialogfelds an.
cdlOFNShareAware	&H4000	Zeigt an, daß mögliche Freigabe-Fehler ignoriert werden.



Abbildung 9.7:
Das Standarddialog-Steurelement zeigt das Dialogfeld Datei öffnen an.

In einem dateibezogenen Dialogfeld wie **Datei öffnen** findet man häufig einen Filter für die Dateinamenerweiterungen, so daß das Dialogfeld nur Dateien mit einer bestimmten Dateinamenerweiterung anzeigt, beispielsweise alle Dateien, die mit der Wildcard-Auswahl *.doc übereinstimmen. Der Benutzer kann den Standardfilter überschreiben, indem er einen anderen Filter oder *.* eingibt und damit alle Dateien im Dialogfeld anzeigt. Mit der Zuweisung der Filter-Eigenschaft setzen Sie den Ausgangswert für diesen Filter. Sie hat das folgende Format:

FilterBeschreib1 | erweiterung1 | FilterBeschreib2 | erweiterung2
 |
 FilterBeschreib3 |erweiterung3"

Die folgende Anweisung beispielsweise weist einen Filter zu, der nur Word- und Excel- Dokumente anzeigt, wenn das Dialogfeld **Datei Öffnen** angezeigt wird:

```
cdbFiles.Filter = "Word Docs (*.doc)|*.doc|Excel Docs (*.xls)|*.xls"
```

	Verwechseln Sie die Dateinamenerweiterungen in der Beschreibung nicht mit den eigentlichen Erweiterungen im Filter. In dem Beispiel ist Word Docs (*.doc) der Text, der den Benutzern angezeigt wird, und die Angabe *.doc, die dem ersten Pipe-Symbol folgt, ist die erste Filteranweisung des Dialogfelds.

Sie können mehrere Filter anlegen, indem Sie mehrere Strings für die **Filter**-Eigenschaft bereitstellen. Wenn Sie mehrere Filter bereitstellen, müssen Sie die **FilterIndex** -Eigenschaft für den Filter setzen, den Sie für das aktuelle Dialogfeld **Datei öffnen** verwenden möchten. Der erste Filter hat den **FilterIndex** 1. Diese Zahl wird für jeden zusätzlichen Filter hochgezählt.

Die **FileName**-Eigenschaft des Standarddialogs enthält den ausgewählten Dateinamen, nachdem der Benutzer das Dialogfeld geschlossen hat.

Das Dialogfeld Speichern unter

Das Dialogfeld **Speichern unter** ist fast identisch mit dem Dialogfeld **Datei öffnen**, außer in Hinblick auf den Titel und einige andere Optionen, wie beispielsweise die Titel der Schaltflächen. Bei einer MDI-Applikation könnten Sie dem Benutzer erlauben, mehrere Dateien im Dialogfeld **Datei öffnen** auszuwählen, weil jede Datendatei in einem eigenen Dokumentfenster angezeigt werden kann. Wenn Sie dagegen das Dialogfeld **Speichern unter** anbieten, kann der Benutzer nur einen einzigen Dateinamen angeben oder auswählen.

Abbildung 9.8 zeigt, daß das Dialogfeld **Speichern unter** fast genauso aussieht wie das Dialogfeld **Datei öffnen**. Mit den **Flags**-Eigenschaften aus Tabelle 9.3 richten Sie dieses Dialogfeld ein wie das Dialogfeld **Datei öffnen**. Dabei können Sie auch die Filter setzen, die Sie im vorigen Abschnitt kennengelernt haben.

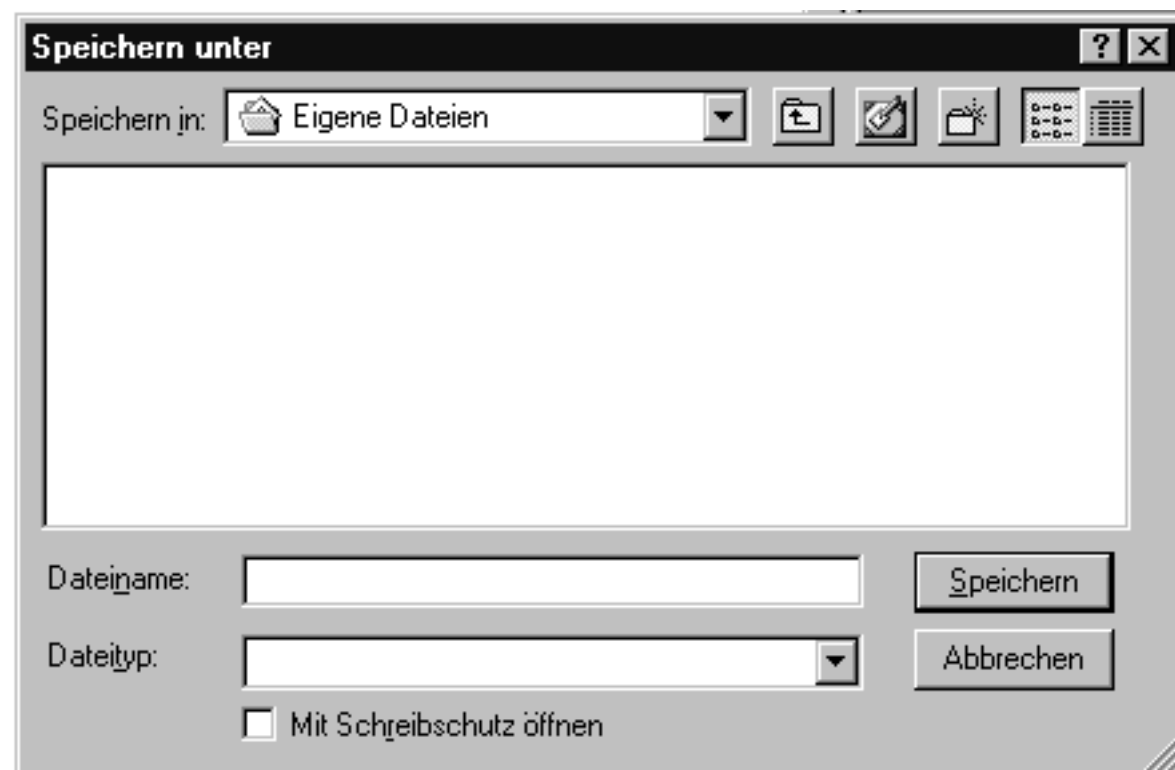


Abbildung 9.8:
Das Standarddialog-Steuerelement zeigt das Dialogfeld **Speichern unter** an.

Das Dialogfeld Drucken

Abbildung 9.9 zeigt das Dialogfeld **Drucken**, das erscheint, wenn Sie die **ShowPrinter** -Methode auf das Standarddialog-Steuerelement anwenden. Ihre Benutzer wählen hier den Druckertyp, die Anzahl der Kopien, den Seitenbereich und verschiedene andere Druckoptionen aus. Die verschiedenen Systeme der Benutzer weisen unterschiedliche Optionen für das Dialogfeld **Drucken** auf. Wenn die Benutzer die gewünschten Werte eingeben, kann Ihre Applikation diese Informationen (aus den Eigenschaften des Standarddialog-Steuerelements) nutzen, um die Druckausgabe richtig zu verarbeiten.



Das Dialogfeld **Drucken**, das auf Ihrem PC angezeigt wird, sieht sehr wahrscheinlich anders aus als das in Abbildung 9.9 gezeigte. Die Werte sind von Ihrem Druckertyp abhängig.

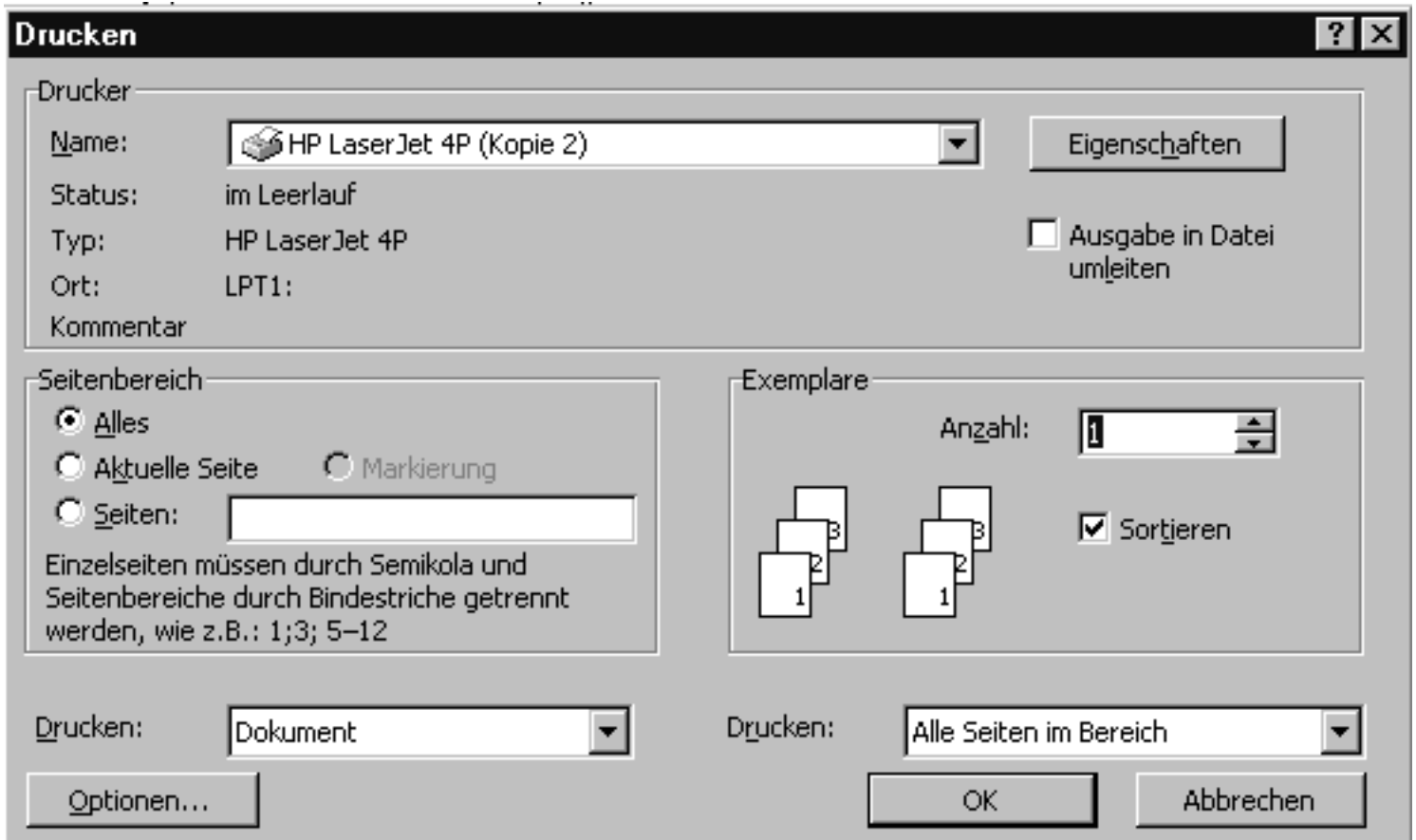


Abbildung 9.9:
Im Dialogfeld Drucken können die Benutzer Druckeroptionen auswählen.

Listing 9.3 zeigt Code, der das Dialogfeld **Drucken** öffnet, nachdem es in einem Menü ausgewählt wurde.

Listing 9.3: Mit Hilfe des Standarddialogs leiten Sie den Ausdruck weiter

```

1: Private mnuFilePrint_Click()
2: Dim intBegin As Integer, intEnd As Integer
3: Dim intNumCopies As Integer, intI As Integer
4: ' Abbrechen ist auf True gesetzt
5:
6: On Error Goto dbErrHandler
7: ' Dialogfeld Drucken anzeigen
8: cbdPrint.ShowPrinter
9: ' Benutzerdefinierte Werte aus dem Dialogfeld übernehmen
10: intBegin = cbdPrint.FromPage
11: intEnd = cbdPrint.ToPage
12: intNumCopies = cbdPrint.Copies
13: '
14: ' Beliebig viele Kopien ausdrucken
15: For intI = 1 To intNumCopies
16: ' Code, der die Ausgabe an den Drucker sendet
17: Next intI
18: Exit Sub
19:
20: dbErrHandler:

```

```
21: ' Der Benutzer hat auf die Abbrechen-Schaltfläche geklickt
22: Exit Sub
23: End Sub
```

Wie Listing 9.3 zeigt, müssen Sie keine Eigenschaften setzen, bevor Sie das Dialogfeld **Drucken** anzeigen (außer möglicherweise `DialogTitle`, wo Sie angeben, was in der Titelleiste des Dialogfelds erscheinen soll), aber Sie können die Rückgabewerte des Dialogfelds auswerten, die in Eigenschaften wie `Copies`, `FromPage` und `ToPage` abgelegt sind, um festzustellen, wie der Benutzer einen Bericht ausdrucken möchte.

Das Dialogfeld Hilfe

In Kapitel 20 erfahren Sie, wie Sie das Dialogfeld **Hilfe**, das Sie mit dem `StandardDialog`-Steuerelement erzeugen, in Ihre Applikationen integrieren.

Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie man `StandardDialog`s einrichtet, um gebräuchliche Aufgaben auszuführen. Wenn Sie beispielsweise ein Dialogfeld zum Öffnen einer Datei anzeigen möchten, sollten Sie das `StandardDialog`feld verwenden, so daß Ihre Benutzer sofort mit dem Windows-konformen Dialog vertraut sind und damit die Lernkurve abgeflacht wird.

Das `StandardDialog`-Steuerelement benötigt einige Eigenschaften, und man muß die richtige Methode darauf anwenden. Manche `StandardDialog`s funktionieren nicht, wenn diese Eigenschaften nicht gesetzt sind. Ihr Code muß sicherstellen, daß Sie die Auswahlen des Benutzers interpretieren können oder die **Abbrechen**-Schaltfläche korrekt verarbeiten, wenn der Benutzer das Dialogfeld schließt.

In Kapitel 10 erfahren Sie, wie Sie die Mausbewegungen des Benutzers überwachen, so daß Ihre Applikation entsprechend mit den Benutzern kommunizieren kann. Darüber hinaus lernen Sie, wie die Listenfeld-bezogenen Steuerelemente programmiert werden, in denen Sie Ihren Benutzern mehrere Auswahlmöglichkeiten anbieten.

Fragen und Antworten

Frage:

Warum unterstützt das `StandardDialog`-Steuerelement keine anderen Dialogfelder, die wir aus verschiedenen Anwendungen kennen, beispielsweise **Ansicht, **Zoom** aus Word und Excel?**

Antwort:

*Das `StandardDialog`-Steuerelement kann nicht alles unterstützen, weil es sonst zu unübersichtlich wäre und zu viele Ressourcen verbrauchen würde. Nichtsdestotrotz werden Ihnen viele Dialogfelder bereitgestellt, die in Windows-Programmen gebräuchlich sind. Sie können in Visual Basic mit Hilfe von Formularen und Steuerelementen zwar selbst Dialogfelder anlegen, aber nicht jedes Dialogfeld ist allgemein genug, um einen Windows-Standard darzustellen. Beispielsweise bieten die meisten Windows-Applikationen die Menüauswahl **Ansicht**, **Zoom** nicht an, auch wenn Word und Excel sie beinhalten.*

Frage:

Welche Steuerelemente kann ich meiner Werkzeugsammlung hinzufügen?

Antwort:

Sie können Ihrer Werkzeugsammlung ActiveX-Steuerelemente hinzufügen. Das bezieht sich auch auf Ihre eigenen ActiveX-Steuerelemente (siehe Kapitel 17) und auf Steuerelemente, die Sie aus anderen Quellen erhalten. Sie finden solche Steuerelemente auf der Web-Site von Microsoft, ebenso wie an anderen Stellen im Internet. Es gibt mehrere gute Computer-Zeitschriften, die ebenfalls Steuerelemente anbieten.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was müssen Sie mit der Werkzeugsammlung tun, damit Sie ein Standarddialog- Steuerelement auf Ihrem Formular plazieren können?
2. Nennen Sie die Dialogfelder, die mit dem Standarddialog-Steuerelement erzeugt werden.
3. Welche Aufgabe hat das Standarddialog-Steuerelement?
4. Warum kann man die Größe des Standarddialog-Steuerelements auf dem Formular nicht verändern?
5. Richtig/Falsch. Das Dialogfeld **Öffnen** öffnet keine Datei.
6. Was bedeutet die **F i l t e r**-Eigenschaft in dateibezogenen Dialogfeldern?
7. Was bewirkt die **F l a g s**-Eigenschaft?
8. Richtig/Falsch. Sie müssen einen **F l a g s**-Wert setzen, damit Visual Basic das Dialogfeld **Schriftart** anzeigt.
9. Richtig/Falsch. Sie müssen einen **F l a g s**-Wert setzen, damit Visual Basic das Dialogfeld **Drucken** anzeigt.
10. Richtig/Falsch. Die **Show**-Methode zeigt ein Standarddialog-Steuerelement an.

Übungen

1. Ändern Sie den Code in Listing 9.2 so ab, daß er die **Abbrechen**-Schaltfläche verarbeitet. Stellen Sie sicher, daß der Code keine Eigenschaften ändert, wenn der Benutzer auf **Abbrechen** klickt.
2. Schreiben Sie eine Prozedur, die das in Abbildung 9.10 gezeigte Dialogfeld erzeugt. Verwenden Sie dazu den in der Abbildung gezeigten Filter *** . t x t**. Fügen Sie Code ein, um die Dialogfeldeinstellungen zu ignorieren, falls der Benutzer auf die Schaltfläche **Abbrechen** klickt.



Abbildung 9.10:
Legen Sie dieses Dialogfeld an.

Woche 2

Tag 10

Mehr Power mit Maus und Steuerelementen

In diesem Kapitel erfahren Sie, wie man Programme schreibt, die auf Mausbewegungen , Klicks und Verschiebeoperationen reagieren. Die Maus ist für die Bedienung von Windows-Programmen unabdingbar, deshalb sollten Sie - abhängig von den Anforderungen Ihres Programms - eine Mausunterstützung bereitstellen, um Ihren Benutzern eine weitere Möglichkeit zu bieten, mit den Steuerelementen auf Ihrem Formular zu arbeiten.

Die Beschreibung der Maus bildet den Übergang zu ganz neuen Steuerelementen, die Sie heute kennenlernen werden, die *Listen-Steuerelemente*. Sie haben bereits in anderen Applikationen mit Listenfeldern gearbeitet, und hier erfahren Sie, wie Sie eigene Listen anlegen und verwalten. Es gibt mehrere Arten von Listen-Steuerelementen, die Sie heute alle kennenlernen werden.

Die Listen-Steuerelemente stellen Ihren Benutzern Listen mit Elementen bereit, aus denen sie auswählen können. Diese Listen sind den *Variablen-Arrays* ganz ähnlich, die Sie ebenfalls heute kennenlernen werden. Nachdem Sie mehr über diese Arrays erfahren haben, können Sie Ihre Programme effizienter machen, wenn große Datenmengen zu verarbeiten sind.

Heute lernen Sie die folgenden Dinge kennen:

- Mausereignisse
- Wie man feststellt, welche Maustaste der Benutzer geklickt hat
- Drag&Drop-Operationen
- Das Timer-Steuerelement
- Listenfelder und Kombinationsfelder
- Elemente in Listen-Steuerelementen initialisieren, hinzufügen und daraus entfernen
- Arrays deklarieren und verwenden
- Steuerelementfelder

Reaktion auf die Maus

Eine der wichtigsten Eigenschaften von Windows-Applikationen ist, daß sie auf die Maus reagieren. Windows sendet Mausereignisse an Ihr Programm, während der Benutzer Ihre Applikation mit der

Maus bedient. Wenn Sie Programme schreiben, müssen Sie Mausereignisse überwachen und gegebenenfalls darauf reagieren. Wenn Ihre Benutzer mit der Maus auf ein Optionsfeld oder auf ein Kontrollkästchen klickt, muß Ihr Programm natürlich nicht auf die Maus reagieren, aber der Klick löst ein Ereignis für diese Steuerelemente aus. Visual Basic überwacht auch Mausereignisse, mit denen ein Element über den Bildschirm verschoben oder Informationen kopiert und eingefügt werden.



Sie sollten Ihre Windows-Programme so schreiben, daß sie sowohl mit der Tastatur als auch mit der Maus bedient werden können. Der Windows- Standard gibt vor, daß alle Programme auch alleine über die Tastatur ansprechbar sein sollen, falls das nötig ist. Damit kann ein Benutzer, der die Tastatur bevorzugt, oder dessen Maus kaputt ist, seine Windows-Applikationen ebenfalls ausführen. Nichtsdestotrotz sind einige Programme nicht für den Einsatz ohne Mausunterstützung geeignet. Beispielsweise ist ein Zeichenprogramm relativ schwierig zu handhaben, wenn man die Maus dafür nicht einsetzt.



Visual Basic beinhaltet keine Maus-Steuerelemente für Ihre Werkzeugsammlung, weil Ihre Applikation nur über Ereignisse auf die Maus reagiert, und nicht über die Steuerelementeigenschaften.

Mausereignisse

Sie haben volle Kontrolle darüber, wie Ihre Applikation auf Mausereignisse reagiert. Ein Mausereignis kann durch eine der folgenden Aktionen ausgelöst werden:

- Mausbewegung
- Klick mit der Taste
- Doppelklick
- Klick mit der rechten Taste
- Drag&Drop-Operation

Einrichtung des Mauszeigers

Wenn der Benutzer die Maus verschiebt, wird der Mauszeiger (manchmal auch als Mauscursor bezeichnet) über den Bildschirm geschoben und zeigt damit die Bewegung an. Häufig ändert eine Applikation das Aussehen des Mauszeigers während einer Drag&Drop-Operation oder beispielsweise, wenn der Benutzer die Maus über ein Objekt auf dem Bildschirm verschiebt, das nicht mit der Maus aktiviert werden kann. Der veränderte Mauszeiger kann etwa eine Sanduhr anzeigen, woran man erkennt, daß eine Verarbeitung stattfindet, beispielsweise eine Sortieroperation.

Ihre Applikation kann bestimmen, wie der Mauszeiger aussehen soll. Tabelle 10.1 listet die möglichen Formen des Mauszeigers auf. Um den Umriß zu ändern, sobald die Maus über ein bestimmtes Steuerelement auf Ihrem Formular geschoben wird, setzen Sie die `MousePointer`-Eigenschaft des betreffenden Steuerelements. Fast jedes Steuerelement, das Sie auf einem Formular anlegen können, beinhaltet die `MousePointer`-Eigenschaft, die einen der in Tabelle 10.1 gezeigten Werte annehmen kann. Diese Werte können zur Laufzeit durch die Zuweisung der Konstantennamen oder zur Entwurfszeit durch Auswahl eines der Werte in der `MousePointer`-Eigenschaft gesetzt werden.

Tabelle 10.1: Sie bestimmen, wie der Mauszeiger aussehen soll

Konstantenname	Beschreibung
VbArrow	Normaler Mauszeiger
VbCrosshair	Fadenkreuz
VbIbeam	I-Form
VbIconPointer	Kleines Quadrat innerhalb eines Quadrats
VbSizePointer	Pfeil mit vier Spitzen, die nach oben, unten, links und rechts zeigen
VbSizeNESW	Doppelpfeil, der nach Nordost und nach Südwest zeigt
VbSizeNS	Doppelpfeil, der nach unten und oben zeigt
VbSizeNWSE	Doppelpfeil, der nach Nordwest und Südost zeigt
VbSizeWE	Doppelpfeil, der nach links und rechts zeigt
VbUpArrow	Pfeil nach oben
VbHourglass	Sanduhr (d.h. Warten)
VbNoDrop	Das internationale Not-Zeichen mit einem roten Schrägstrich durch ein Zeichen, d.h. an der betreffenden Stelle kann nichts abgelegt werden
VbErrorHourGlass	Pfeil mit Sanduhr
vbArrowQuestion	Pfeil mit Fragezeichen
vbSizeAll	Pfeil mit zwei Spitzen, der erscheint, wenn Sie die Größe eines Fensters ändern
vbCustom	Der Umriß, der durch die MouseIcon-Eigenschaft festgelegt wird



Sie können auch einen eigenen Mauszeiger entwerfen. Der Cursor muß in der 16x16-Auflösung dargestellt werden, so wie andere Icons (Icons haben die Dateinamenerweiterung `.ICO`; die meisten Zeichenprogramme unterstützen die Entwicklung von Standard-Icons). Falls Sie statt der vordefinierten Zeiger aus Tabelle 10.1 eigene Icon-Grafikdateien anzeigen möchten, setzen Sie die `MouseIcon`-Eigenschaft auf Ihre Standard-Icondatei und die `MousePointer`-Eigenschaft auf `99 - Benutzerdefiniert`. Der Mauszeiger behält Ihre benutzerdefinierte Form bei, bis Sie ihn explizit wieder ändern. In Bonusprojekt 5 wird genauer demonstriert, wie der Mauszeiger geändert wird.

Wenn der Benutzer die Maus verschiebt und damit klickt

Windows erzeugt Mausbewegungen und Klickereignisse und sendet sie an Ihr Programm. Ihr Programm ignoriert diese Ereignisse, wenn Sie keine Ereignisprozeduren dafür angeben; Sie können jedoch Code für Ereignisse bereitstellen, auf die Sie reagieren möchten. Tabelle 10.2 beschreibt, was bei einem Mausereignis passiert.

Tabelle 10.2: Windows erzeugt diese Mausereignisse, wenn die Maus eingesetzt wird

Ereignis	Beschreibung
Click	Der Benutzer hat auf eine Maustaste geklickt
DblClick	Der Benutzer hat auf eine Maustaste doppelgeklickt
MouseDown	Der Benutzer hält eine Maustaste gedrückt
MouseMove	Der Benutzer hat die Maus verschoben
MouseUp	Der Benutzer hat die Maustaste losgelassen

Alle Mausereignisse sind Steuerelementen zugeordnet. Sie finden fast für jedes Steuerelement Mausereignisse (indem Sie das Drop-down-Listefeld **Objekt** im Codefenster aufklappen) und alle Formulare. Wenn Sie beispielsweise prüfen möchten, ob auf Ihrem Formular frmTest ein Mausklick erfolgt ist, heißt die entsprechende Ereignisprozedur frmTest_Click().



Einige Ereignisse für Maustastenklicks bedingen, daß Sie ein Argument der Ereignisprozedur auswerten, um zu prüfen, welche Taste der Benutzer angeklickt hat. Nur die Ereignisprozeduren MouseDown und MouseUp übergeben Argumente, die angeben, welche Maustaste der Benutzer geklickt hat.

Ist ein Doppelklick ein einzelnes Ereignis oder setzt er sich aus zwei Klick-Ereignissen zusammen? Das hängt davon ab, wie exakt der Benutzer den Doppelklick ausführt. Windows stellt die klickbezogenen Mausereignisse in der folgenden Reihenfolge zur Verfügung:

- MouseDown
- MouseUp
- Click
- DblClick
- MouseUp

Als erstes tritt ein MouseDown-Ereignis auf, wenn der Benutzer die Maustaste klickt; anschließend erfolgen ein MouseUp- und dann ein Click-Ereignis. Wenn der Benutzer mit einer Maustaste doppelklickt, treten auch die Ereignisse DblClick und MouseUp auf. (Windows löst kein MouseDown-Ereignis aus, wenn der Benutzer mit der Maus doppelklickt.)

Für die Ereignisse MouseDown, MouseMove und MouseUp müssen die folgenden vier Argumente übergeben werden:

- `intButton`. Gibt an, welche Taste gedrückt wurde: 1 für die linke Taste, 2 für die rechte Taste und 4 für beide Tasten (oder für die mittlere Taste, falls Sie eine Dreitastenmaus verwenden).
- `intShift`. Gibt den Shift-Status an, indem mittels eines Bitvergleichs gezeigt wird, ob der Benutzer beim Verschieben oder Klicken mit der Maus die Tasten (Alt), (Strg) oder (ª) gedrückt hatte.
- `sngX`. Der horizontale Twip-Wert für die Position, wo der Benutzer die Maus geklickt oder verschoben hat.

- sngY. Der vertikale Twip-Wert für die Position, wo der Benutzer die Maus geklickt oder verschoben hat.

Visual Basic erzeugt etwa alle 10 bis 15 Twips ein Bewegungsereignis, wenn der Benutzer die Maus verschiebt. Das ist ein extrem kleiner Fensterabschnitt. Visual Basic erzeugt jedoch nicht für jedes Twip ein Bewegungsereignis.

Die folgende Anweisung deklariert eine `MouseDown`-Ereignisprozedur, wo Sie sehen, wie die Argumente übergeben werden:

```
Private Sub imgMouse_MouseDown(intButton As Integer, _  
intShift As Integer, sngX As Single, sngY As Single)
```

Nach dem Eintritt in diese Prozedur enthalten `sngX` und `sngY` die Twip-Koordinaten des Mausklicks. `intButton` enthält den Wert 1, 2 oder 4, der angibt, welche Taste gedrückt wurde. Sie brauchen nicht immer zu wissen, welche Taste der Benutzer gedrückt hat, aber wenn Sie auf einen Klick mit der linken Maustaste anders reagieren wollen als auf einen Klick mit der rechten Maustaste, ermitteln Sie mit Hilfe des `MouseDown`-Ereignisses, welche Taste gedrückt wurde. Wenn der Benutzer (^a), (Strg) oder (Alt) gedrückt hat, müssen Sie wissen, welche dieser Tasten in Verbindung mit der Maustaste gedrückt wurde. Dazu führen Sie eine Überprüfung ähnlich der in Listing 10.1 aus.

Listing 10.1: Anhand des Shift-Werts erkennen Sie, welche Taste der Benutzer in Verbindung mit einem Mausereignis gedrückt hat

```
1: Private Sub imgMouse_MouseDown(intButton As Integer, intShift As  
_  
Integer, sngX As Single, sngY As Single)  
2: Dim intShiftState As Integer  
3: intShiftState = intShift And 7 ' Spezielles bitweises Und  
4: Select Case intShiftState  
5: Case 1  
6: ' Code für Shift-Kombinationen  
7: Case 2  
8: ' Code für Strg-Kombinationen  
9: Case 3  
10: ' Code für Alt-Kombinationen  
11: Case 4  
12: ' Code für Shift+Strg-Kombinationen  
13: Case 5  
14: ' Code für Shift+Alt-Kombinationen  
15: Case 6  
16: ' Code für Strg+Alt-Kombinationen  
17: Case 7  
18: ' Code for Shift+Strg+Alt-Kombinationen  
19: End Select  
20: End Sub
```

Der spezielle Und-Vergleich in Zeile 3 wertet ein internes Bit-Flag aus, um festzustellen, welche Taste der Benutzer zusammen mit dem Mausereignis gedrückt hat.

Was Sie tun sollten

Stellen Sie fest, ob ein Mausklick mit einer bestimmten Taste kombiniert wurde, falls Ihre Applikation erlaubt, daß Text selektiert wird, indem die Maus verschoben wird, während die (Strg)-Taste gedrückt ist, oder daß mehrere Elemente ausgewählt werden, indem geklickt wird, während die (Strg)-Taste gedrückt ist. Viele Steuerelemente, wie beispielsweise Listenfelder, die Sie heute noch kennenlernen werden, verarbeiten automatisch »(Strg)+Maus«-Kombinationen, wenn Sie Werte aus der Liste auswählen. Sie müssen sich also keine Gedanken über Tastencodes machen, die innerhalb von Listefeldern in Kombination mit Mausereignissen auftreten.



Bonusprojekt 5 demonstriert eine vollständige Applikation, die zeigt, wie man auf Mausklicks und Mausbewegungen reagiert.

Drag&Drop-Operationen verwalten

Wenn Ihr Benutzer ein Objekt von einem Formular an eine andere Position verschiebt, muß Ihre Applikation darüber Bescheid wissen! *Drag&Drop* ist ein Prozeß, wobei der Benutzer mit der Maus ein Objekt auf dem Bildschirm anklickt, die Maustaste gedrückt hält und das Objekt an eine andere Position auf dem Bildschirm zieht. Überraschenderweise ist die Programmierung von Drag&Drop-Operationen relativ einfach - dank des Windows-Betriebssystems, das entsprechende Informationen erzeugt, wenn Drag&Drop-Ereignisse auftreten.

Visual Basic unterstützt zwei Arten von Drag&Drop-Operationen:

- Automatisches Drag&Drop
- Manuelles Drag&Drop

Die erste Methode ist die einfachere. Dabei lösen Sie das Drag&Drop automatisch über Steuerelement-Eigenschaften aus. Fast jedes Steuerelement in der Werkzeugsammlung besitzt die Eigenschaft **DragMode**. Diese Eigenschaft ermöglicht dem Benutzer, das Steuerelement mit der Maus zu verschieben. Beim Verschieben zeigt Visual Basic den Umriß des Steuerelements an. Ihre Aufgabe ist es, das Steuerelement an die Position zu verschieben, wo der Benutzer die Maus losläßt. Der automatische Modus zeigt zwar den Umriß des Steuerelements an, verschiebt aber das Objekt nicht wirklich.

Das **DragDrop**-Ereignis des Formulars steuert, wo das verschobene Objekt abgelegt wird. Um das Verschieben zu realisieren, setzen Sie die **DragMode**-Eigenschaft des Steuerelements auf **1 - Automatisch**. Das Steuerelement kann dann verschoben werden und zeigt dabei den Umriß an. Die Ereignisprozedur **Form_DragDrop()** übernimmt die zweite Hälfte der Drag&Drop-Operation, indem sie das verschobene Steuerelement an seiner neuen Position ablegt.

Beim Verschieben eines Objekts erscheint normalerweise sein Umriß, Sie können das Icon für diesen Umriß aber mit Hilfe der **DragIcon**-Eigenschaft ändern. Diese Eigenschaft verweist dann auf eine andere Icon-Datei (beispielsweise auf die Icon-Dateien im **\Graphics**-Ordner, falls Sie die Grafik für Visual Basic installiert haben). Wenn ein Benutzer das Steuerelement verschiebt, ersetzt das Icon während des Verschiebens den Mauszeiger. Nachdem der Benutzer das Verschieben abgeschlossen hat, kann eine **Form_DragDrop()**-Prozedur die Aufgabe übernehmen, das Objekt an seine endgültige Position zu verschieben. Das **DragDrop**-Ereignis verschiebt das Steuerelement von der Ausgangsposition an die Endposition. Hier der Code, der diese Verschiebung vornimmt:

```

1: Private Sub frmTitle_DragDrop(Source As Control, X As Single, Y
As _
Single)
2: ' Dieser Code nimmt als Argument das Steuerelement entgegen,
3: ' das der Benutzer verschoben hat
4: Source.Move X, Y ' Das Objekt wird an die Endposition verschoben
5: End Sub

```

Die Move-Methode in Zeile 4 verschiebt das Steuerelement von seiner Ausgangsposition an die Endposition (das ist dort, wo der Benutzer die Maustaste losgelassen hat).



Das **DragOver**-Ereignis tritt auf, wenn der Benutzer ein Steuerelement über ein anderes Steuerelement zieht. Wenn Sie den Mauszeiger ändern möchten, sobald er über Objekte geschoben wird, auf die keine anderen Steuerelemente verschoben werden sollen, ändern Sie den Mauszeiger innerhalb der **DragOver**-Ereignisprozedur für alle Steuerelemente, die keine verschobenen Objekte aufnehmen. **DragOver** nimmt vier Argumente entgegen:

- Das Steuerelement
- Die x-Koordinate des Mauszeigers
- Die y-Koordinate des Mauszeigers
- Den Verschiebestatus, der einen von drei möglichen Werten annehmen kann: 0 (wenn beim Verschieben das Objekt berührt wird), 1 (wenn das Verschieben das Objekt verläßt) und 2 (wenn das Steuerelement durch das Objekt verschoben wird)

Das manuelle Drag&Drop verhält sich ähnlich wie das automatische Drag&Drop, wobei es drei Unterschiede gibt:

- Sie müssen die **DragMode**-Eigenschaft auf 0 - **Manuell** setzen.
- Beim manuellen Drag&Drop kann das Steuerelement auf ein **MouseDown**-Ereignis reagieren, bevor das Verschieben beginnt, so daß die Koordinaten der Ausgangsposition des Steuerelements aufgezeichnet werden können.
- Sie müssen der **MouseDown**-Ereignisprozedur Code hinzufügen, der das Verschieben initiiert.

Die **MouseDown**-Ereignisprozedur kann die spezielle **Drag**-Methode für das Objekt ausführen, wenn Sie den Drag&Drop-Prozeß fortsetzen möchten. Der folgende Code verschiebt das Anzeigefeld, wenn dessen **DragMode**-Eigenschaft gleich 0 - **Manuell** ist:

```

Private Sub imgMouse_MouseDown(Button As Integer, Shift As Integer,
X As _
Single, Y As Single)
' Klicken auf dem Anzeigefeld
txtMouse.Text = "Klick auf dem Anzeigefeld bei " & X & ", " & Y
imgMouse.Drag
End Sub

```

Die **Drag**-Methode setzt das Drag&Drop weiter. Ohne die **Drag**-Methode könnte die Ereignisprozedur **MouseDown**() die Drag&Drop-Operation nicht initiieren. Verwenden Sie manuelle Drag&Drop-Operationen, wenn Sie die Drag&Drop-Beschränkungen einrichten möchten, die vor und während des Drag&Drop-Prozesses gelten sollen.



In Bonusprojekt 7 erzeugen Sie eine Applikation, die Ihnen ermöglicht, durch einen Klick mit der rechten Maustaste ein Pop-up-Menü anzuzeigen.

Listenfelder

In diesem Abschnitt erfahren Sie mehr über die Steuerelemente in Ihrer Werkzeugsammlung. Viele der Steuerelemente, die Sie noch kennenlernen sollen, bedingen eine Programmierung, damit sie funktionieren. Sie können sie nicht einfach auf einem Formular anlegen, wie es beispielsweise bei einer Schaltfläche der Fall war, weil sie initialisiert werden müssen - und das kann nur im Code erfolgen. Ein Listenfeld beispielsweise kann erst zur Laufzeit mit Elementen gefüllt werden. Einige Steuerelemente, wie etwa Bezeichnungsfelder, können zwar initialisiert werden, aber viele der mehrwertigen Steuerelemente können nicht initialisiert werden. Die Steuerelemente, die mehr als einen Wert aufnehmen können (wie etwa Listenfelder), müssen programmiert werden.

Hier eine Auflistung der verschiedenen Listenfeldtypen:

- Einfaches Listenfeld
- Drop-down-Listenfeld
- Drop-down-Kombinationsfeld
- Einfaches Kombinationsfeld

Einfache Listenfelder

Das einfache Listenfeld bietet Ihren Benutzern eine Möglichkeit, ein oder mehr Elemente aus einer Liste auszuwählen. Mit dem Listenfeld-Steuerelement in der Werkzeugsammlung fügen Sie Ihrem Formular ein einfaches Listenfeld hinzu.



Einige der Eigenschaften eines Listenfelds werden im Eigenschaftenfenster initialisiert, beispielsweise die Position, die Größe und die Farbe, aber die Elementliste wird in der Regel nicht im Eigenschaftenfenster vorgegeben.



Der obige Hinweis sagt aus, daß die Elementliste eines Listenfelds *in der Regel* nicht im Eigenschaftenfenster initialisiert wird, weil Ihnen Visual Basic die Möglichkeit gibt, die Liste zur Laufzeit zu initialisieren. Wenn Sie jedoch die `List`-Eigenschaft im Eigenschaftenfenster eines Listenfelds öffnen, wird eine leere Liste angezeigt, wo Sie Werte einfügen können. Listenfelder enthalten jedoch fast immer Werte, die vom Benutzer und anderen Datenquellen stammen. Sie sollten nur kleine Listenfelder, deren Werte sich nicht ändern, direkt aus dem Eigenschaftenfenster initialisieren. Das restliche Kapitel geht von der Standardinitialisierung von Listenfeldern aus: mit Methoden, die die Liste zur Laufzeit anlegen.

Mit der `AddItem`-Methode fügen Sie einer Liste Elemente hinzu (siehe Listing 10.2).

Listing 10.2: Die Initialisierung eines Listenfelds erfolgt am besten in der Ereignisprozedur `Form_Load()`

```

1: Private Sub Form_Load()
2: ' Initialisiert die Listenwerte
3: lstColors.AddItem "Rot"
4: lstColors.AddItem "Blau"
5: lstColors.AddItem "Grün"
6: lstColors.AddItem "Gelb"
7: lstColors.AddItem "Orange"
8: lstColors.AddItem "Weiß"
9: End Sub

```

Die Ausgangswerte eines Listenfelds werden am besten in der Ereignisprozedur `Form_Load()` geladen. Während der Programmausführung können weitere Elemente hinzugefügt werden. Die Ereignisprozedur `Form_Load()` stellt sicher, daß die Ausgangswerte, falls beim Programmstart solche vorhanden sind, geladen werden, bevor der Benutzer das Formular mit dem Listenfeld sieht. Beim Starten Ihrer Applikation lädt das Laufzeitmodul zunächst das Formular (oder mehrere Formulare, falls Sie MDI-Formulare verwenden). Formulare unterstützen das Ereignis `Load`, das beim Laden eines Formulars auftritt. Durch die Formular-Befehle `Load` und `Unload` können Sie genau angeben, wann ein Formular geladen werden soll, und wann es aus dem Speicher entfernt werden soll, um Ressourcen freizugeben.

Nachdem Sie die Elemente geladen haben, die bei der ersten Anzeige des Formulars berücksichtigt werden, können Sie mit der `AddItem`-Methode weitere Elemente in das Listenfeld eintragen. Angenommen, Ihr Listenfeld heißt `lstColors`. Sie schreiben die `AddItem`-Methode für das Listenfeld so, wie im vorigen Code gezeigt. Sie fügen ein Element in die Liste ein, indem Sie die `AddItem`-Methode, gefolgt von dem einzufügenden Element, ausführen. Visual Basic fügt die Elemente in der Reihenfolge in die Liste ein, wie Ihr Code sie einfügt, es sei denn, Sie setzen die `Sorted`-Eigenschaft für das Listenfeld auf `True`, dann sortiert Visual Basic die Listeneinträge in alphabetischer oder numerischer Reihenfolge.



Sie können in der `List`-Eigenschaft des Eigenschaftfensters Ausgangswerte eintragen, aber das macht die Wartung sehr schwierig. Das Einfügen der Ausgangswerte im Code macht die Elementliste besser wartbar.

Die folgende Anweisung fügt der Liste in der oben gezeigten `Form_Load()`-Prozedur den Eintrag »Aquamarin« hinzu:

```
lstColors.AddItem "Aquamarin"
```



Manchmal ist es schwierig, zwischen Befehlen, Deklarationen, Methoden, Eigenschaften und Steuerelementen von Visual Basic zu unterscheiden. Stellen Sie sich eine Methode so vor, daß sie ein Element auffordert, sich selbst einzutragen. Mit anderen Worten, `lstColors.AddItem "Aquamarin"` bedeutet »Füge mir ein Element namens Aquamarin hinzu«. Als Listenfeld weiß `lstColors`, wie man diese Anforderung erfüllt, weil die Entwickler von Visual Basic dem Listenfeld diese Methode mitgegeben haben. Die meisten Steuerelemente haben eigene Methoden. Mehr über die Methoden der einzelnen Steuerelemente entnehmen Sie der Dokumentation.

Angenommen, das Listenfeld `lstColors` wird in der Mitte eines Formulars angezeigt, das die zuvor gezeigte Formularprozedur `Form_Load()` beinhaltet. Das Listenfeld erscheint dann mit den

Farbennamen wie in Abbildung 10.1 gezeigt.



Wenn Sie zur Entwurfszeit nicht einen oder mehrere Listenwerte initialisieren, löschen Sie die `List`-Eigenschaft aus dem Eigenschaftenfenster, nachdem Sie dem Listenfeld einen Namen zugewiesen haben. Visual Basic verwendet den Namen des Listenfelds als erstes Element in der Liste, es sei denn, Sie löschen die Liste.



Abbildung 10.1:
Das Listenfeld enthält die hinzugefügten Farbwerte.

Das in Abbildung 10.1 gezeigte Listenfeld ist ein sogenanntes *einfaches Listenfeld*.



Ein *einfaches Listenfeld* zeigt Elemente einer Liste an. Der Programmierer initialisiert die Liste und der Benutzer kann ihr keine Elemente direkt hinzufügen.

Für das Listenfeld werden vertikale und horizontale Bildlaufleisten angezeigt, falls die Höhe oder die Breite nicht ausreichend sind, um alle Listenelemente anzuzeigen. Wenn das Listenfeld alle Elemente anzeigen kann, die Applikation aber später weitere Elemente einfügt, zeigt Visual Basic die Bildlaufleisten während der Programmausführung an, sobald das erforderlich wird.

Die Benutzer wählen Elemente aus dem Listenfeld aus. Das ist die Aufgabe eines Listenfelds: dem Benutzer die Auswahl eines Elements aus einer Elementeliste zu bieten, so daß er den Wert nicht manuell eingeben muß. Wenn ein Benutzer ein Element in der Liste auswählt, passiert folgendes:

- Das Element verwendet eine andere Farbe, um die Auswahl zu kennzeichnen.
- Visual Basic kopiert die Auswahl in die `Text`-Eigenschaft des Listenfelds.

`lstColors.Text` wird also während der Programmausführung geändert, abhängig davon, welches Element der Benutzer ausgewählt hat.

`Text` kann nur einen einzigen Wert enthalten. Um die `Text`-Eigenschaft des Listenfelds zu verstehen, betrachten Sie, was passiert, wenn Sie der in Abbildung 10.1 gezeigten Applikation ein Textfeld namens `txtColor` hinzufügen würden. Die folgende Prozedur sendet das ausgewählte Listenelement an das Textfeld, sobald der Benutzer ein Element ausgewählt hat (`lstColors`):

```
Private Sub lstColors_Click()  
' Ausgewähltes Element in das Textfeld kopieren.  
txtColor.Text = lstColors.Text  
End Sub
```

Wenn Sie auf einem Formular ein Textfeld anlegen, das das ausgewählte Listenelement enthält, sollten Sie zur Entwurfszeit die `Text`-Eigenschaft für das Textfeld im Eigenschaftenfenster löschen, so daß im Textfeld erst dann ein Eintrag erscheint, wenn der Benutzer ein Element ausgewählt hat. Der Standardwert der `Font`-Eigenschaft für das Textfeld muß fast immer geändert werden, weil die Einstellung zu klein ist. Außerdem müssen Sie das Listefeld vergrößern, wenn Sie mehr als zwei Einträge gleichzeitig sehen möchten. Abbildung 10.2 zeigt, was passiert, wenn ein Listenelement ausgewählt wurde.



Abbildung 10.2:
Das Textfeld enthält jetzt das im Listefeld ausgewählte Element.

Listenelemente werden anhand ihrer Indexwerte voneinander unterschieden. Der Index beginnt mit Null für das erste Element und wird sequentiell weitergezählt. Die `ListIndex`-Eigenschaft des Listenfelds enthält den Wert des aktuell ausgewählten Elements. Sie können also ganz einfach feststellen, welches Element ausgewählt ist. Andere Listefeldmethoden, wie beispielsweise `RemoveItem`, verwenden die `ListIndex`-Eigenschaft, um Elemente aus der Liste zu entfernen.

Um beispielsweise das dritte Element aus der Liste zu entfernen, würden Sie folgendes schreiben:

```
lstColors.RemoveItem 2
```

Der Index wird automatisch entfernt, wenn Elemente aus der Liste entfernt werden. Das vierte Element hat den Indexwert 3 (weil der Index bei 0 beginnt), aber wenn ein Element vor diesem Element entfernt wird, wird das vierte Element zum dritten Element und erhält damit den Indexwert 2. Wenn Sie die **Sorted**-Eigenschaft auf **True** gesetzt haben, sortiert Visual Basic die Indexwerte nach jedem Hinzufügen oder Entfernen neu.



Manchmal ist es sinnvoll, daß der Benutzer mehrere Elemente aus der Liste auswählt. Dazu hält er die (Strg)-Taste gedrückt, während er Elemente im Listenfeld auswählt. Offensichtlich kann die **Value**-Eigenschaft des Listenfelds nicht alle ausgewählten Werte aufnehmen, deshalb müssen andere Maßnahmen ergriffen werden, wenn Sie Mehrfachauswahlen zulassen. Bonusprojekt 5 zeigt, wie Sie ein Listenfeld einrichten, damit es Mehrfachauswahlen erlaubt und erkennt.



Die Indexwerte von Listenfeldern verhalten sich ein bißchen wie Arrays, wie Sie im letzten Abschnitt dieses Kapitels noch erfahren werden.

Die Benutzer können einem Listenfeld nicht direkt Werte hinzufügen. Ein Listenfeld nimmt nur Elemente auf, die im Code mit Hilfe der **AddItem**-Methode hinzugefügt werden. Die Benutzer können auch keine Elemente direkt löschen. Dazu wird die Methode **RemoveItem** aufgerufen. Angenommen, Sie möchten, daß alle Einträge eines Listenfelds entfernt werden. Sie könnten zwar die **RemoveItem**-Methode auf alle Elemente im Listenfeld anwenden, aber eine **For**-Schleife ist in diesem Fall sinnvoller (wir gehen dabei von dem Listenfeld aus dem vorigen Beispiel aus):

```
1: ' Die ersten fünf Elemente aus der Liste entfernen
2: For intI = 0 To 5
3: lstColors.RemoveItem 0
4: Next intI
```

Bei einer **For**-Schleife müssen Sie wissen, wie viele Listenelemente es gibt. Weil Ihr Code die Elemente in die Liste einfügt, sollte es nicht schwierig sein, einen Zähler zu verwalten.

Wenn Sie den Benutzern in Ihrem Programm die Möglichkeit schaffen, Elemente in die Liste einzufügen (beispielsweise mit einem Textfeld zur Dateneingabe, wo eine Schaltfläche die **AddItem**-Methode auslöst), könnte die Liste größer werden, aber mit Hilfe einer Variablen, die herauf- oder heruntergezählt wird, sobald ein Element eingetragen oder gelöscht wird, kann der Zähler aktuell gehalten werden. (Der Benutzer kann in einem einfachen Listenfeld kein direktes Einfügen oder Löschen vornehmen.)

Sie können zwar eine Zählervariable führen, die bei jeder Operation für Ihre Listenelemente aktualisiert wird, aber das ist gar nicht nötig, weil Visual Basic die Anzahl der Listenelemente intern für Sie verwaltet. Die **ListCount**-Eigenschaft enthält die aktuelle Anzahl der Listenelemente. **ListCount** ist immer um eins größer als der höchste **ListIndex**, weil **ListIndex** bei Null beginnt. Immer wenn Sie ein ganzes Listenfeld in einer **For**-Schleife irgendwie verarbeiten möchten, brauchen Sie die Gesamtzahl der Listenelemente, also **ListCount**, wie in der folgenden Schleife gezeigt:

```

1: ' Alle Elemente aus der Liste entfernen
2: intTotal = lstColors.ListCount ' Anzahl speichern
3: For intI = 1 To intTotal
4: lstColors.RemoveItem 0
5: Next intI

```



Die `Clear`-Methode des Listenfeld entfernt alle Einträge, ohne daß dazu eine Schleife benötigt wird. Wenn Sie den gesamten Listeninhalt löschen möchten, verwenden Sie also statt einer `For`-Schleife über den gesamten Listenbereich die Anweisung `lstColors.Clear`.

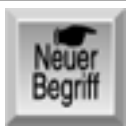
Kombinationsfelder

Das Kombinationsfeld kann in drei Varianten genutzt werden:

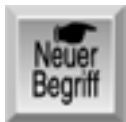
- Als Drop-down-Listen-Kombinationsfeld
- Als einfaches Kombinationsfeld
- Als Drop-down-Kombinationsfeld



Während ein einfaches Listenfelds auf dem Formular die Größe annimmt, mit der Sie es entworfen haben, wird für das *Drop-down-Listen-Kombinationsfeld* immer nur eine einzige Zeile auf dem Formular angezeigt, bis der Benutzer die Liste öffnet, um die Einträge zu sehen.



Das *einfache Kombinationsfeld* sieht aus wie ein einfaches Listenfeld mit angeschlossenem Textfeld. Die Benutzer fügen einem einfachen Kombinationsfeld Werte hinzu, indem Sie diese in das dafür vorgesehene Textfeld eingeben.



Das *Drop-down-Kombinationsfeld* spart Bildschirmplatz, weil es geschlossen bleibt und wie ein einzeliges Listenfeld aussieht, bis der Benutzer es auswählt. Damit wird das Drop-down-Kombinationsfeld geöffnet und zeigt eine Elementliste an. Der Benutzer kann etwas aus dieser Liste auswählen oder einen neuen Wert in das Textfeld eingeben, den Sie der Liste im Code hinzufügen.

Was die Initialisierung und den Zugriff aus dem Code betrifft, verhalten sich alle Kombinationsfelder ähnlich wie das Listenfeld. Der wichtigste Unterschied liegt darin, wie sie auf dem Bildschirm erscheinen und wie Sie es Ihren Benutzern erlauben, Daten auszuwählen oder einzugeben.

Alle drei Kombinationsfeldtypen stammen aus einer einzigen Quelle: dem Kombinationsfeld-Steuerelement in der Werkzeugsammlung. Wenn Sie ein Kombinationsfeld auf einem Formular platzieren, müssen Sie Visual Basic mit Hilfe der `Style`-Eigenschaft mitteilen, welche Form dieses Kombinationsfeld haben soll. Der Standardstil für ein Kombinationsfeld ist 0 - Dropdown-Kombinationsfeld. Das ist das einfachste Kombinationsfeld, mit dem wir hier beginnen werden.



Beachten Sie, daß die `Sorted`-Eigenschaft Listen automatisch für Sie alphabetisch oder numerisch sortiert, auch wenn die Benutzer der Liste neue Werte hinzufügen. Wenn Sie für die `Sorted`-Eigenschaft nicht `True` angegeben haben, bleibt die Liste unsortiert und zeigt die Elemente in der Reihenfolge an, in der sie eingegeben wurden.

Das Drop-down-Listen-Kombinationsfeld verhält sich genau wie ein Listenfeld, außer daß ein Listenfeld in der Regel mehr Platz auf einem Formular einnimmt als ein Drop-down-Listen-Kombinationsfeld. Wenn ein Benutzer auf den Pfeil neben der Drop-down-Liste klickt, wird es geöffnet und zeigt eine Elementliste an.

Abbildung 10.3 zeigt ein Formular mit zwei Drop-down-Listen, die dieselben Werte enthalten. Bevor das rechte Kombinationsfeld aufgeklappt wurde, nahmen beide Steuerelemente denselben Platz auf dem Formular an. Ein Benutzer kann das Kombinationsfeld durch Anklicken öffnen und einen oder mehrere Werte dort auswählen.



Abbildung 10.3:
Ein Drop-down-Listen-Kombinationsfeld nimmt nur einen kleinen Raum auf dem Formular ein, bis ein Benutzer es öffnet.

Drop-down-Listen-Kombinationsfelder werden eingesetzt, wenn eine Auswahlliste bereitgestellt werden soll, gleichzeitig aber nicht viel Platz auf dem Formular zur Verfügung steht. Wie Sie in Abbildung 10.3 sehen, zeigt ein Drop-down-Listenfeld erst dann Text an, wenn es vom Benutzer aufgeklappt wird. Sie sollten Ihre Benutzer also mit Hilfe eines Bezeichnungsfelds oder eines Meldungsfelds darauf hinweisen, welche Werte das Drop-down-Listen-Kombinationsfeld enthält.

Mit der `AddItem`-Methode fügen Sie dem Drop-down-Listen-Kombinationsfeld Einträge hinzu. Der folgende Code fügt sechs Farben in die beiden in Abbildung 10.3 gezeigten Kombinationsfelder ein.

```

Private Sub Form_Load()
' Beide Kombinationsfelder initialisieren
cboColor1.AddItem "Rot"
cboColor1.AddItem "Blau"
cboColor1.AddItem "Grün"
cboColor1.AddItem "Gelb"
cboColor1.AddItem "Orange"
cboColor1.AddItem "Weiß"

cboColor2.AddItem "Rot"
cboColor2.AddItem "Blau"
cboColor2.AddItem "Grün"
cboColor2.AddItem "Gelb"
cboColor2.AddItem "Orange"
cboColor2.AddItem "Weiß"
End Sub

```

Das Drop-down-Listen-Kombinationsfeld unterstützt dieselben Methoden und Eigenschaften wie das Listenfeld.

Das nächste Kombinationsfeld, das einfache Kombinationsfeld, verhält sich ein bißchen wie ein Listenfeld mit einem Textfeld oben in der Liste. Mit anderen Worten, der Benutzer kann einen Listeneintrag auswählen oder einen neuen Wert in die Liste eingeben. Wenn Sie einem Formular ein Kombinationsfeld hinzufügen und dessen **Style**- Eigenschaft auf **1 - Einfaches Kombinationsfeld** setzen, ändert Visual Basic das Steuerelement so, daß Sie seine Größe so anpassen können wie für ein Listenfeld. Wenn Sie ein einfaches Kombinationsfeld nicht groß genug machen, daß es während der Laufzeit alle Einträge anzeigen kann, zeigt es Bildlaufleisten an, so daß die Benutzer alle Einträge sichten können.

Abbildung 10.4 zeigt, wie die Farben in einem einfachen Kombinationsfeld aufgelistet werden. Ihr Code lädt das Feld mit Hilfe der **AddItem**-Methode mit Daten, und die Benutzer können zusätzliche Einträge einfügen. Achten Sie darauf, daß die **Text**-Eigenschaft des einfachen Kombinationsfelds gelöscht werden muß, wenn das Steuerelement auf dem Formular angelegt wird; andernfalls erscheint dort, wo der Benutzer Daten eingeben kann, der Name des Steuerelements. Wenn Sie zur Entwurfszeit eine **Text**-Eigenschaft angeben, verwendet das einfache Kombinationsfeld diese als Vorgabewert, den der Benutzer akzeptieren oder ändern kann.

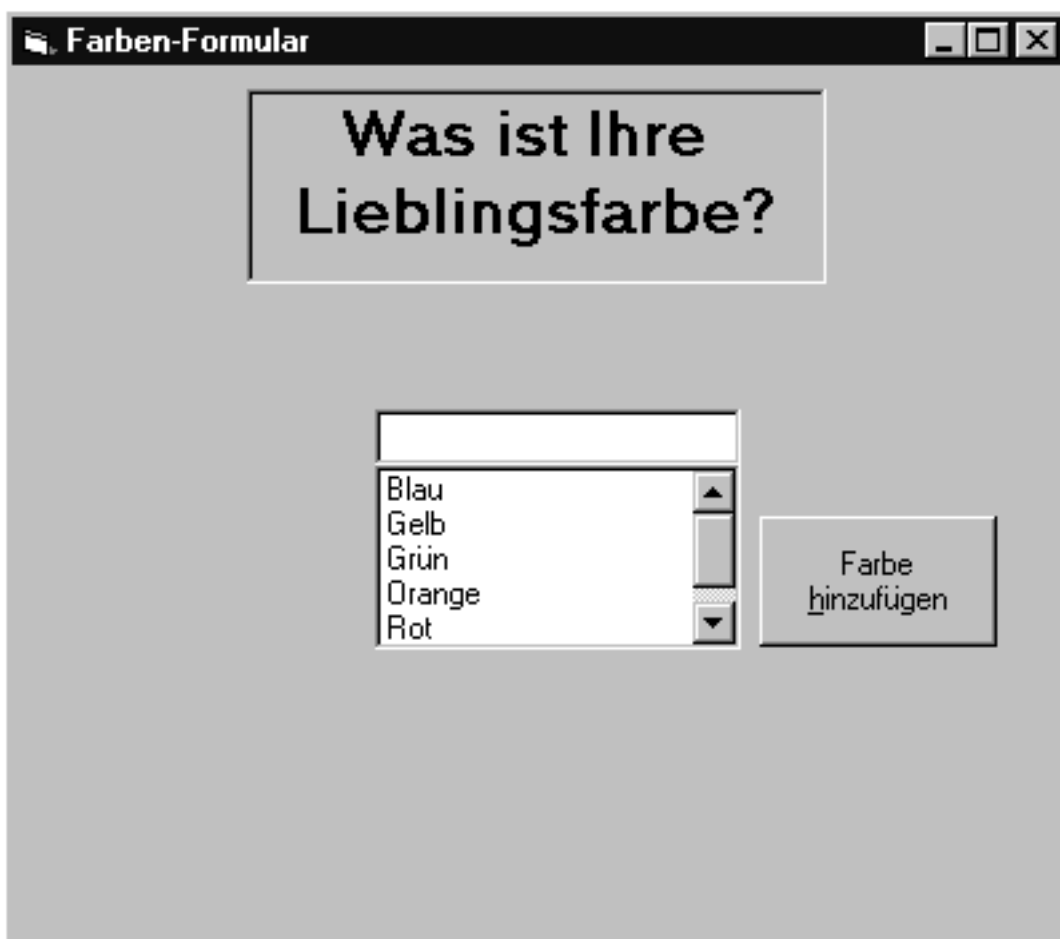


Abbildung 10.4:
Ein Benutzer kann einen Farbwert eingeben oder einen Eintrag aus der Liste wählen.

Die hier im Kombinationsfeld aufgelisteten Farben erscheinen aufgrund der `Sorted`-Eigenschaft alphabetisch sortiert. Um sicherzustellen, daß die Elemente alphabetisch sortiert angezeigt werden, egal in welcher Reihenfolge sie in die Liste eingetragen wurden, setzen Sie die `Sorted`-Eigenschaft auf `True`.

 Hinweis	Wenn Sie auf einem Formular ein einfaches Kombinationsfeld anlegen und seine <code>Style</code>-Eigenschaft auf <code>1 - Einfaches Kombinationsfeld</code> setzen, müssen Sie die Größe des Steuerelements so festlegen, wie es zur Laufzeit erscheinen soll. Anders als eine Drop-down-Liste behält das einfache Kombinationsfeld immer dieselbe Größe bei.

Ein einfaches Kombinationsfeld fügt die Einträge des Benutzers nicht automatisch ein. Sie müssen Code bereitstellen, der die Einträge in die Liste einfügt, falls der Benutzer Werte hinzufügen kann. Die meisten Programmierer verwenden einfache Kombinationsfelder auf ihren Formularen, so daß die Benutzer einen Eintrag auswählen oder einen neuen Wert eingeben können, aber wenn Sie sicherstellen möchten, daß der eingegebene Wert der Kombinationsfeldliste hinzugefügt wird, brauchen Sie in der `LostFocus()`-Ereignisprozedur des einfachen Kombinationsfelds den folgenden Code:

```
Private Sub cboColor_LostFocus()  
cboColor2.AddItem cboColor1.Text  
End Sub
```

Die `LostFocus()`-Ereignisprozedur wird ausgeführt, wenn das Steuerelement den Fokus verliert, also wenn der Benutzer auf ein anderes Steuerelement klickt oder den Fokus mit der (`Y__`)-Taste vom

Kombinationsfeld entfernt. Sobald der Fokus an ein anderes Steuerelement übergeht, wird die `LostFocus()`-Ereignisprozedur ausgeführt und der Code speichert den vom Benutzer eingegebenen Wert (der sich in der `Text`-Eigenschaft befindet) mit der `AddItem`-Methode in der Liste. Abbildung 10.4 zeigt eine Schaltfläche an, weil der Benutzer nach dem Eintrag einer neuen Farbe den Fokus an ein anderes Steuerelement weitergeben muß, so daß das neue Element hinzugefügt wird.

Das beste aller Listen- und Kombinationsfeld-Steuerelemente scheint das Drop-down-Kombinationsfeld zu sein. Es belegt nicht viel Platz auf dem Bildschirm, weil es geschlossen bleibt, bis der Benutzer es anklickt. Wenn es geöffnet wird, zeigt es eine Liste mit Elementen an. Die Benutzer können einen Listeneintrag auswählen oder einen neuen Wert eingeben. Die `Selected`- und `ListCount`-Eigenschaften können so wie andere Listenelement-Eigenschaften auch auf die Drop-down-Kombinationsfelder angewendet werden, so daß Ihr Code immer erkennt, ob der Benutzer einen neuen Wert eingegeben oder einen existierenden Wert ausgewählt hat. Anders als das Drop-down-Listenelement kann das Drop-down-Kombinationsfeld ergänzt werden.

Setzen Sie die `Style`-Eigenschaft auf `0` - `DropDown-Kombinationsfeld` (das ist der Standard), wenn Sie mit Drop-down-Kombinationsfeldern öffnen möchten. Abbildung 10.5 zeigt anhand unseres Beispiels die Farbliste - hier als Drop-down-Kombinationsfeld.

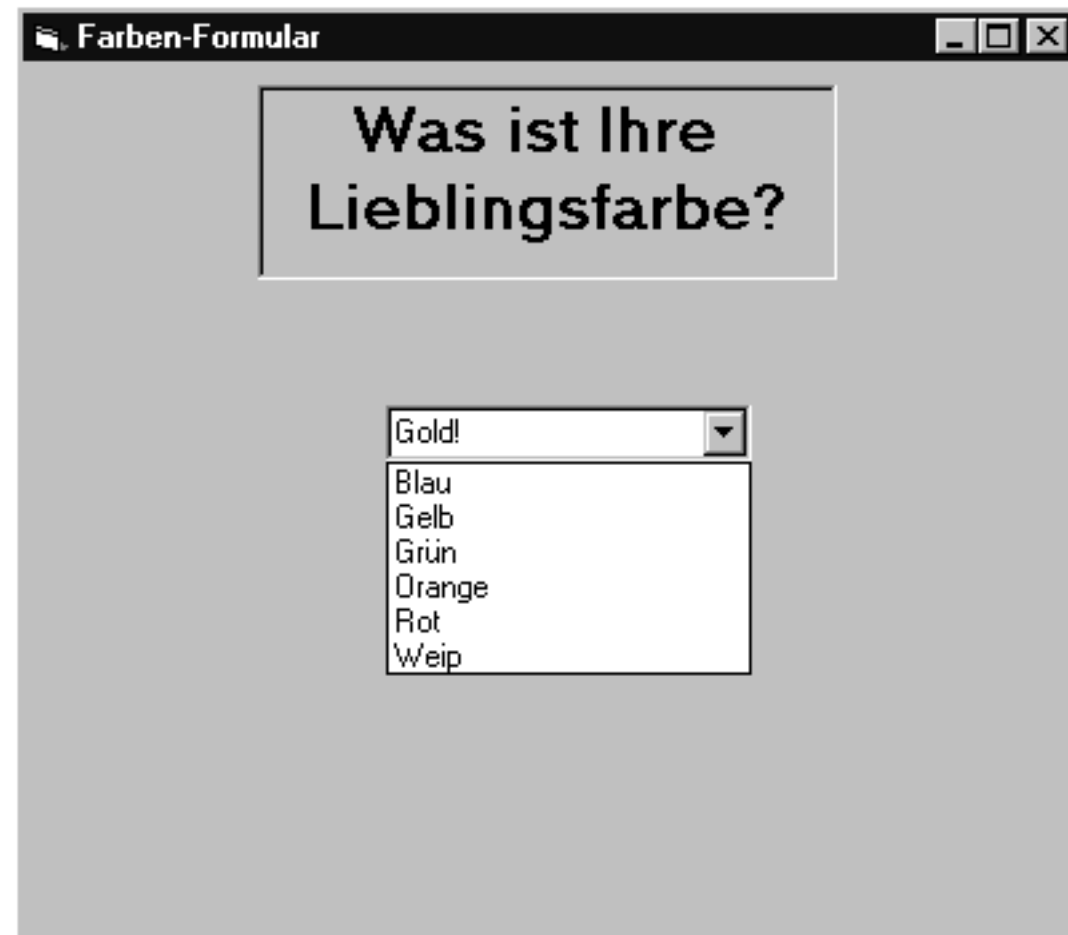


Abbildung 10.5:
Der Benutzer wählt einen Wert aus oder gibt einen neuen Wert ein.

Wie für das einfache Kombinationsfeld löschen Sie auch für das Drop-down-Kombinationsfeld die `Text`-Eigenschaft, wenn Sie das Steuerelement auf dem Formular anlegen, so daß oben im Kombinationsfeld kein Eintrag erscheint. Die Benutzer können neue Werte eingeben (die in die Liste eingefügt werden, wenn der Fokus vom Steuerelement abgezogen wird) oder das Drop-down-Kombinationsfeld öffnen, um die Liste anzuzeigen und einen Eintrag daraus

auszuwählen.

Der Timer

Das Timer-Steuerelement ermöglicht Ihnen, abhängig von der internen Uhr des Computers, Reaktionen zu erzeugen. Sie können Code schreiben, der ausgeführt wird, nachdem eine bestimmte Zeitspanne vergangen ist. Sie können das Timer-Steuerelement aber auch einsetzen, um eine Hintergrundverarbeitung zu realisieren. Ihre Computeruhr löst 18mal in einer Sekunde ein Timer-Ereignis aus. Die Uhr ist lebenswichtig für die Arbeit von CPU, Speicher und Platte, weil der Datenfluß in einer definierten und zeitgerechten Weise erfolgen muß.



Egal wie langsam oder schnell Ihr PC ist, erzeugt er 18mal in einer Sekunde ein Timer-Ereignis.

Visual-Basic-Applikationen können auf die Zeitereignisse der Uhr reagieren. Der Benutzer erzeugt keine Timer-Ereignisse, sehr wohl aber Ihr PC. Windows übergibt diese Ereignisse an jedes ausgeführte Programm. Sie können ein bestimmtes Zeitintervall festlegen, nach dem Windows eine Ereignismeldung an Ihre Applikation schickt. Wie für alle Ereignisse können Sie auch für das Timer-Ereignis Ereignisprozeduren schreiben, die ausgeführt werden, sobald das Timer-Ereignis auftritt. Sie schreiben also Code, der ausgeführt wird, nachdem eine bestimmte Zeit vergangen ist, und Sie sind dabei nicht von der Geschwindigkeit des PC beim Benutzer abhängig, weil die Zeit konstant ist.

Das Timer-Steuerelement nimmt die Timer-Ereignisse entgegen und reagiert entsprechend seiner Eigenschaften darauf. Wenn Sie das Timer-Steuerelement auf einem Formular ablegen, bestimmen Sie das Zeitintervall, das angibt, wie oft das Ereignis auftreten soll. Dieses Zeitintervall ist eine der Eigenschaften zur Steuerung des Timers. Wenn das Zeitintervall abgelaufen ist, löst der Timer die Ereignisprozedur aus, die Sie für die Verarbeitung der Timer-Ereignisse bereitgestellt haben.



Sie können Ihrer Applikation mehrere Timer-Steuerelemente hinzufügen, ebenso wie man gleichzeitig mehrere Instanzen anderer Steuerelemente erzeugen kann. Ihre Applikation könnte etwa jede Minute eine Ereignisprozedur ausführen, jede halbe Stunde und jede Stunde (beispielsweise für ein Zeiterfassungssystem). Dazu legen Sie drei Timer-Steuerelemente auf dem Formular an und bestimmen die drei Zeitintervalle.



Wenn Sie erst einmal mit dem Timer-Steuerelement vertraut sind, werden Ihnen viele Anwendungszwecke dazu einfallen. Sie können die Timer-Ereignisprozeduren beispielsweise verwenden, um eine Hintergrundverarbeitung zu realisieren. Außerdem können Sie damit Grafiken animieren, indem Sie sie für jedes der vorgegebenen Timer-Ereignisse neu zeichnen, beispielsweise jede halbe Sekunde.

Am besten plazieren Sie einen Timer auf Ihrem Formular, indem Sie auf das Timer- Icon in der Werkzeugsammlung doppelklicken und ihn auf Ihrem Formular dann an eine Position schieben, wo er die anderen Objekte nicht stört. Das Timer-Steuerelement kann wie das Standarddialogfeld nicht in der Größe geändert werden, und es wird zur Laufzeit ebenfalls nicht auf dem Formular angezeigt.

Abbildung 10.6 zeigt ein Timer-Steuerelement in der Mitte eines neuen Formulars.

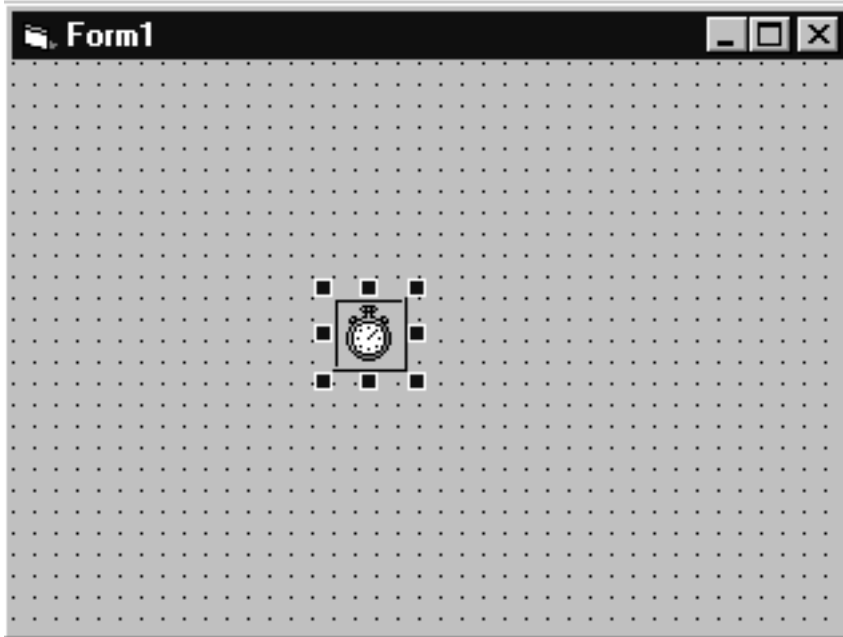


Abbildung 10.6:

Sie können ein Timer-Steuerelement auf dem Formular verschieben, es jedoch nicht vergrößern.

Das Timer-Steuerelement unterstützt nur sehr wenige Eigenschaften. Von den sechs Eigenschaften, die Sie zur Entwurfszeit setzen können, sind fünf völlig eindeutig:

- `Left` und `Top` bestimmen die Position des Timers.
- `Enabled` bestimmt, ob der Timer aktiviert ist.
- `Tag` enthält Anmerkungen, die Sie dem Steuerelement mitgeben möchten.
- `Index` legt die Position des Steuerelements in einem Steuerelementfeld fest.



Wenn Sie die `Enabled`-Eigenschaft eines Timers zur Entwurfszeit auf `False` setzen, reagiert das Timer-Steuerelement erst dann auf Ereignisse, wenn `Enabled` im Code auf `True` gesetzt wird.

Die einzige Eigenschaft, die wichtig und für den Timer wirklich eindeutig ist, ist `Interval`. Die `Interval`-Eigenschaft bestimmt die Frequenz, mit der Ihr Timer-Steuerelement Ereignisse erzeugen soll. Sie geben zur Entwurfszeit oder zur Laufzeit die Anzahl Millisekunden ein, die vergehen sollen, bevor ein Timer auf ein Ereignis reagiert. Wenn Sie beispielsweise die `Interval`-Eigenschaft auf 1000 setzen, tritt alle 1000 Millisekunden ein Ereignis auf, also etwa einmal pro Sekunde.

Das Timer-Steuerelement scheint auf den ersten Blick einige Nachteile zu haben. Die `Interval`-Eigenschaft kann Werte zwischen 0 und 64767 aufnehmen. Sie können also nur Zeitintervalle von etwa 65 Sekunden angeben, nicht mehr. Wenn Sie ein Ereignisintervall setzen möchten, das größer als 65 Sekunden ist, dann ist das nicht so einfach. Sie können in der Timer-Ereignisprozedur des Timer-Ereignisses jedoch Ereignisse ignorieren und zur Applikation zurückkehren, ohne auf das Ereignis zu reagieren, bis eine bestimmte Zeit vergangen ist. Mit anderen Worten, auch wenn die Ereignisprozedur, die Sie für das Timer-Steuerelement geschrieben haben, alle 60 Sekunden ausgelöst wird, können Sie oben in der Ereignisprozedur Code angeben, der zur Applikation zurückkehrt und nicht auf das Ereignis reagiert, bis seit der vorherigen Ausführung der

Ereignisprozedur eine bestimmte Zeit vergangen ist.



Das Timer-Steuerelement ist nicht besonders exakt. Obwohl der Quarz in Ihrer Computeruhr sehr genau ist, geht Genauigkeit verloren, während Windows ein Timer-Ereignis an Ihre Applikation sendet. Auch andere Ereignisse können den Timer verlangsamen, beispielsweise ein Netzwerkzugriff oder eine Modemaktualisierung. Ihr Computer kann nicht zwei Dinge gleichzeitig tun, und ein Timer-Steuerelement innerhalb einer Visual-Basic- Applikation besitzt nicht immer die höchste Priorität. Das Timer-Steuerelement funktioniert, wenn es eine Zeittoleranz von etwa einer Sekunde gibt; es gibt kein Steuerelement in Visual Basic, das eine höhere Genauigkeit bieten würde.

Das Timer-Steuerelement unterstützt nur ein einziges Ereignis: das **Timer**-Ereignis. Wenn Ihr Timer-Steuerelement also `tmrClock` heißt, brauchen Sie nur eine einzige Ereignisprozedur zu schreiben: `tmrClock_Timer()`. Dort legen Sie den Code ab, den Visual Basic ausführen soll, nachdem das vorgegebene Intervall vergangen ist.

Der Übung halber führen Sie die folgenden Schritte aus, um eine einfache Applikation zu erstellen, die das **Timer**-Steuerelement demonstriert:

1. Legen Sie ein neues Projekt an und erzeugen Sie auf dem Formular ein **Timer**- Steuerelement.
2. Setzen Sie die **Interval**-Eigenschaft des Timers auf `1000`, so daß die **Timer**-Ereignisprozedur etwa jede Sekunde ausgeführt wird.
3. Geben Sie dem Timer den Namen `tmrTimer1` und verschieben Sie ihn in die untere linke Ecke des Formularfensters.
4. Legen Sie neben dem ersten Timer noch einen zweiten an, `tmrTimer2`. Setzen Sie seine **Interval**-Eigenschaft auf `500`, so daß seine Ereignisprozedur etwa jede halbe Sekunde ausgeführt wird.
5. Fügen Sie zwei Textfelder ein, `txtTime1` und `txtTime2`. Positionieren Sie die Textfelder wie in Abbildung 10.7 gezeigt.

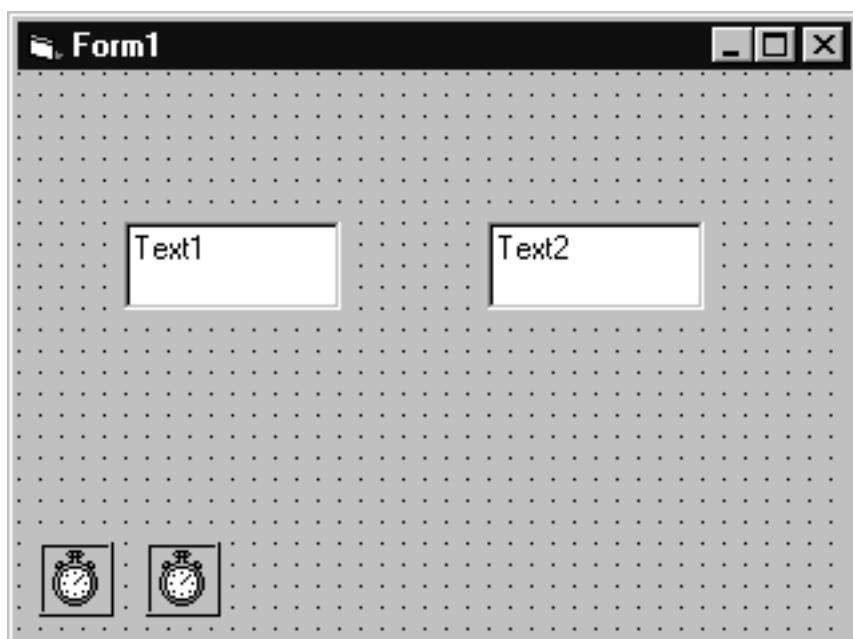


Abbildung 10.7:

Die Timer-basierte Applikation ist fast fertig.

6. Geben Sie für die Text-Eigenschaften der Textfelder 1 ein und setzen Sie die Schriftgröße auf 18. Setzen Sie die Alignment-Eigenschaften auf 2 - Zentriert, so daß der Text zur Laufzeit in der Mitte der Textfelder erscheint. Setzen Sie die With-Eigenschaften der beiden Textfelder auf 1000.

7. Doppelklicken Sie auf den ersten Timer, um in das Codefenster zu wechseln. Geben Sie für die Prozedur `tmrTimer1_Timer()` den folgenden Code ein (Visual Basic schreibt die erste und die letzte Zeile für Sie):

```
Private Sub tmrTimer1_Timer()  
' Eins zum angezeigten Wert addieren  
txtTimer1.Text = txtTimer1.Text + 1  
End Sub
```

8. Fügen Sie in die Prozedur für den zweiten Timer den folgenden Code ein:

```
Private Sub tmrTimer2_Timer()  
' Eins zum angezeigten Wert addieren  
txtTimer2.Text = txtTimer2.Text + 1  
End Sub
```

Dieser Code addiert die Zahl 1 zu den Werten in den Textfeldern, weil Visual Basic die Werte mit dem Datentyp `Variant` ablegt.

9. Fügen Sie Bezeichnungsfelder unterhalb der Textfelder ein. *Langsam* und *Schnell*.

10. Führen Sie die Applikation aus. Ihr erstes Formular sollte aussehen wie in Abbildung 10.8 gezeigt. Hier passieren zwei Dinge: Das erste Textfeld wird jede Sekunde aktualisiert, das zweite jede halbe Sekunde. Abhängig von der Genauigkeit der Timer-Ereignisse sollte das zweite Textfeld zweimal so schnell aktualisiert werden wie das erste.



Abbildung 10.8:

Die beiden Timer werden in unterschiedlichen Zeitintervallen aktualisiert.

Die Arbeit mit Arrays

Das Listenfeld hat die Verwendung von Indexwerten demonstriert. Jedes Element in einem Listenfeld hat einen Index, der beginnend mit Null gezählt wird, also 0, 1, 2 usw. Das Listenfeld hat einen einzigen Namen, kann aber in seiner Liste viele Elemente enthalten. Sie unterscheiden diese Elemente anhand der Indexwerte.

Die indizierten Elemente in einem Listenfeld stellen eine passende Analogie zu dem neuen Konzept dar, das Sie in diesem Abschnitt kennenlernen werden: Arrays. Während ein Listenfeld ein Steuerelement ist, das indizierte Elemente enthält, ist ein Array eine Variablenliste, die indizierte Elemente enthält.



Ein *Array* ist eine Liste von Variablen, die alle denselben Datentyp und denselben Namen haben. Die Programmierer unterscheiden die einzelnen Variablen in dieser Liste anhand ihrer Indizes.



Der *Index* beschreibt die Position eines Elements in einem Array.

Wenn Sie verschiedene Werte ablegen möchten, beispielsweise Bonuswerte für einen Verkäufer, sind einzelne Variablen gut geeignet. Wenn Sie jedoch eine Liste ähnlicher Werte speichern möchten, sollten Sie ein Array verwenden.

Nicht alle Ihre Visual-Basic-Daten werden in Arrays abgelegt, nachdem Sie gelernt haben, wie man sie deklariert und verwendet. Arrays sind für Datenlisten vorgesehen. Sie verwenden weiterhin einzelne Variablen für Schleifenzähler und Benutzereingaben. Wenn Sie mehrere Variationen von Daten haben, die in Ihrer Applikation verarbeitet werden sollen (beispielsweise Tabellenfelder, die in den Speicher eingelesen werden), ist ein Array ein perfektes Behältnis für diese Daten.



Was tun, wenn Sie die Bonusstatistik von 100 Verkäufern verarbeiten und einen durchschnittlichen, einen minimalen und einen maximalen Bonus sowie die Standardabweichung zwischen den Werten berechnen möchten? Eine Möglichkeit, die Bonusbeträge für die 100 Verkäufer zu verwalten, wäre die Deklaration von 100 Variablen, die alle unterschiedliche Namen tragen, beispielsweise `curBonus1`, `curBonus2` usw. Stellen Sie sich vor, wie es aussieht, wenn Sie diese Variablen addieren! Sie müssten für jede Addition und für jeden Vergleich der Bonuswerte alle 100 Variablen in einer einzelnen Anweisung auflisten. Eine solche Situation führt ganz von selbst zu Arrays. Statt der 100 Variablennamen brauchen Sie nur ein einziges Array. Es enthält eine Liste mit 100 Bonuswerten. Das Array erhält einen einzigen Namen. Um die Arraywerte zu durchlaufen, brauchen Sie nicht 100 Variablennamen aufzulisten, sondern führen eine einfache Schleife für die Indexwerte von 1 bis 100 aus. Sehen Sie, wie praktisch Arrays sein können?

Um das Ganze weiter zu verdeutlichen, hier ein Codeausschnitt, der 100 separate Variablen addiert, die verschiedene Namen tragen:

```
curTotal = curBonus1 + curBonus2 + curBonus3 + curBonus4 + ...
```

Und hier der Code, der 100 Elemente aus einem Array addiert:

```
For intCtr = 1 To 100
curTotal = curTotal + curBonus(intCtr)
Next intCtr
```

Sie können die Bonuswerte gegebenenfalls auch rückwärts durchlaufen. Darüber hinaus können Sie beliebig auf die Arraywerte zugreifen. Angenommen, Sie möchten den Durchschnitt für den ersten und den letzten Bonus im Array berechnen. Dazu verwenden Sie die folgende Anweisung:

```
curAvgBonus = (curBonus(1) + curBonus(100)) / 2.0
```

Diesem Code entnehmen Sie die folgenden Informationen über Arrays:

- Arrayindizes stehen in Klammern hinter dem Arraynamen.
- Alle Elemente eines Arrays haben denselben Datentyp.
- Durch Angabe des Index kann ein Zugriff auf beliebige Arraydaten erfolgen.
- Der Index reicht von 1 bis zu der Gesamtzahl der Elemente im Array.

Das letzte Element zeigt, daß hier der Anfangsindex 1 verwendet wird. Visual Basic erlaubt die Verwendung des Index 0, aber die Visual-Basic-Programmierer bevorzugen (anders als die C- und C++-Programmierer) einen Indexstartwert von 1 und ignorieren den Index 0. In Wirklichkeit beginnen alle Indizes mit 0, es sei denn, Sie fügen in den Deklarationsbereich des Moduls die folgende Anweisung ein:

```
Option Base 1
```

Visual Basic bietet noch eine Möglichkeit, den unteren Indexbereich festzulegen, wie Sie im nächsten Abschnitt noch erfahren werden.



Einige Visual-Basic-Programmierer verwenden ein zusätzliches Präfix für Arraynamen, beispielsweise `strar` für String-Arrays oder `intar` für Integer- Arrays.



Visual Basic unterstützt zwei Arraytypen: statische Arrays und dynamische Arrays. Dieses Buch bezieht sich auf statische Arrays, die eine feste Größe haben und zur Laufzeit nicht geändert werden können. (Die Größe dynamischer Arrays kann während der Programmausführung geändert werden.) Statische Arrays sind viel effizienter und werden in der Visual-Basic-Programmierung häufiger verwendet.

In gewisser Hinsicht kann man den Zugriff auf einen Arraywert mit den Postfächern auf dem Postamt vergleichen. Die Adresse aller Postfächer ist dieselbe (sie befinden sich alle innerhalb eines Gebäudes), aber die Post wird in das Fach mit der entsprechenden Nummer gelegt.

Arraydeklaration

Arrays müssen vor der Verwendung deklariert werden. Dazu verwenden Sie die Anweisungen `Public` oder `Dim`, die auch für die Deklaration einzelner Variablen verwendet werden. Welche Anweisung Sie verwenden, hängt davon ab, welchen Gültigkeitsbereich Sie für Ihr Array brauchen, und wo es deklariert werden soll.

Mit `Public` deklarieren Sie ein öffentliches Array, das in der gesamten Applikation verwendet werden kann (in allen Modulen). Die `Public`-Anweisung muß im allgemeinen Deklarationsbereich des Moduls stehen. Wenn Sie im Deklarationsbereich eine `Dim`-Anweisung verwenden, erzeugen Sie ein Array auf Modulebene, das im gesamte Modul genutzt werden kann. Mit `Dim` am Anfang einer Prozedur deklarieren Sie lokale Arrays.

Der einzige Unterschied zwischen der Deklaration von Arrays und der Deklaration einzelner Variablen ist die Einführung der Indizes in den Deklarationsanweisungen. Hier die Formate der beiden Anweisungen:

```
Public arName(intSub) [As dataType][, arName(intSub) [As  
dataType]]...
```

```
Dim arName(intSub) [As dataType][, arName(intSub) [As dataType]]...
```

Die Arraynamen sehen aus wie normale Variablennamen (außer daß gegebenenfalls ein zusätzliches Präfix angegeben wird). Sie können ein Array eines beliebigen Datentyps erzeugen, `dataType` kann also `Integer`, `Single` oder einen der anderen Datentypen haben. Der `intSub`-Teil der Befehle beschreibt die Anzahl der Elemente und wie diese angesprochen werden. In den oben gezeigten Anweisungsformaten kann `intSub` das folgende Format annehmen:

```
[intLow To] intHigh
```



In diesem Kapitel erfahren Sie, wie man eindimensionale Arrays erzeugt, das sind Arrays mit einem einzigen Index. Es ist in Visual Basic auch möglich, mehrdimensionale Arrays anzulegen, sogenannte *Tabellen*.

Anders als andere Programmiersprachen bietet Visual Basic eine Ausnahme zu der Regel, die besagt, daß alle Arrayelemente denselben Datentyp haben müssen. Ein Array kann nur Elemente eines Datentyps enthalten, aber Sie können ein Array auch als `Variant` deklarieren, so daß die Arrayelemente unterschiedliche Datentypen haben dürfen.

Die folgende Anweisung deklariert ein Integer-Array mit fünf Elementen (vorausgesetzt im Deklarationsbereich steht die Anweisung `Option Base 1`):

```
Dim intCounts(5) As Integer
```

Abbildung 10.9 zeigt, wie dieses Array im Speicher aussieht.



Ohne die Anweisung `Option Base 1` würde diese `Dim`-Anweisung für das Array `intCounts` ein Array mit sechs Elementen mit den Indizes 0 bis 5 deklarieren.

Was steht in `intCounts(1)` bis `intCounts(5)`? Sie wissen es nicht, und der Programmierer weiß es auch nicht. Wie alle anderen Variablen muß das Programm die Arrayelemente initialisieren, bevor sie verwendet werden können. Durch die Angabe des Index können Sie Arrayelemente überall dort einsetzen, wo Sie auch andere Variablen einsetzen können, beispielsweise wie folgt:

```
intNumber = intCounts(2) * intFactor / 15
```

```
txtValue.Text = intCounts(4)
```

```
intCounts(5) = 0
```

Die beiden folgenden Anweisungen sind äquivalent:

```
Dim intCounts(5) As Integer
```

```
Dim intCounts(1 To 5)
```

Mit dem To-Zweig legen Sie den Anfangs- und den Endwert für den Index fest. Betrachten Sie die drei folgenden Arraydeklarationen:

```
Public varCustNumber(200 To 999) As Variant
```

```
Public strCustName(200 To 999) As String
```

```
Public curCustBalance(200 To 999) As Currency
```

Der erste Index in jedem der Arrays ist 200, der letzte ist 999. Mit dieser Deklaration würde die Verwendung von `strCustName(4)` einen Fehler erzeugen, weil die Indizes erst mit 200 beginnen.



Der obere Index muß nicht unbedingt die Anzahl der Arrayelemente widerspiegeln, wenn Sie die To-Klausel verwenden. Die drei oben gezeigten Arrays beispielsweise enthalten je 800 Elemente (mit den Indizes von 200 bis 999).

Sie stellen vielleicht fest, daß für Ihre Daten Indexbereiche erforderlich sind, die sich von den Standards unterscheiden (also denen, die bei 0 oder 1 beginnen). Die vorigen Deklarationen könnten beispielsweise genutzt werden, um Kundeninformationen zu speichern, wenn die niedrigste Kundennummer gleich 200 ist. In diesem Fall wäre es sinnvoll, Arrayindizes ab 200 zu verwenden und den ersten Kunden an der entsprechenden Arrayadresse abzulegen, 200.

Der höchste Index, den Sie deklarieren, sollte so viel Platz schaffen, daß Sie genügend Elemente für die Datenliste haben.

Visual Basic beinhaltet eine spezielle, eingebaute Funktion, `Array()`. Diese Funktion ermöglicht die schnelle Deklaration und Initialisierung eines Arrays.



Die `Array()`-Funktion verhält sich irgendwie wie die alten BASIC-Anweisungen `READ` und `DATA`. Es ist ganz einfach, kleine Arrays zu initialisieren, wenn Sie die Datenwerte bereits bei der Programmierung kennen.

Ein `Variant`-Datentyp kann alle anderen Datentypen beinhalten, auch Arrays. Angenommen, Sie möchten die Anzahl der Tage jedes Monats im Array `Days` ablegen (Schaltjahre werden ignoriert). Eine `Variant`-Variable wird wie folgt deklariert:

```
Dim Days As Variant
```

Die Initialisierung des Arrays kann mit Hilfe der `Array()`-Funktion in einem Schritt erfolgen (statt mit einer `For`-Schleife):

```
Days = Array(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
```

Wenn im Deklarationsbereich des Moduls die Anweisung `Option Base 1` steht, ist der erste Index von `Array()` gleich 1, andernfalls 0. Auch Strings und Datumswerte werden ganz einfach deklariert

und initialisiert, indem Sie einem Variant-Wert die einfache Array()-Funktion zuweisen.

Die Verwendung von Arrays

Wie Sie im vorigen Abschnitt gesehen haben, können Sie Arrays für Berechnungen einsetzen, so wie ganz normale Variablen, wie in der folgenden Anweisung gezeigt:

```
curFamilyDues(5) = curFamilyDues(4) * 1.5
```

Um die Daten in einem Array nutzen zu können, brauchen Sie nur den Index des betreffenden Arrayelements.

Am besten lernen Sie, wie man Indizes verwendet, indem Sie Beispiele betrachten. Das folgende Beispiel zeigt, wie Arrayelemente mit Hilfe einer InputBox()-Funktion gefüllt werden. Die meisten Programme erhalten ihre Eingabedateien aus Dateien und Formularen. Weil Arrays sehr große Datenmengen aufnehmen können, ist es nicht sinnvoll, die Daten bei jeder Programmausführung einzugeben. Zuweisungen sind ebenfalls nicht ausreichend, weil sie für große Datenmengen und interaktive Programme nicht geeignet sind.

Das Programm in Listing 10.3 deklariert zwei Arrays für eine Vereinsverwaltung mit 35 Familiennamen und ihre jährlichen Beiträge. Das Programm fordert Eingaben an und druckt die Daten aus.



Wenn Sie dieses Programm ausführen, ändern Sie die Anzahl der Familiennamen am besten von 35 auf 5, damit Sie nicht so viele Eingaben vornehmen müssen.

Listing 10.3: Arrays vereinfachen die Datenablage

```
1: Private Sub association ( )
2: ' Prozedur zur Eingabe von 35 Namen und entsprechender Beiträge
3: Dim strFamilyName(35) As String ' Arrayelemente reservieren
4: Dim curFamilyDues(35) As Currency
5: Dim intSub As Integer
6: Dim intMsg As Integer ' MsgBox() Rückgabe
7:
8: ' Die Daten werden in einer Schleife ermittelt
9: For intSub = 1 To 35
10: strFamilyName(intSub) = InputBox("Nächster Familienname: ")
11: curFamilyDues(intSub) = InputBox("Beiträge?")
12: Next intSub
13:
14: ' Jetzt können Sie diese Daten anzeigen
15: ' Dieses Beispiel verwendet einfach ein paar Meldungsfelder,
16: ' weil Sie momentan noch nicht mehr kennen
17: intSub = 1 ' Ersten Index initialisieren
18: Do
19: intMsg = MsgBox("Familie " & intSub & " ist " &
20: strFamilyName(intSub))
21: intMsg = MsgBox("Der Beitrag beträgt " & curFamilyDues(intSub))
22: intSub = intSub + 1
```

```
32: Loop Until (intSub > 35)
33: End Sub
```

Beachten Sie, daß das Programm alle Eingaben und Ausgaben mit Hilfe einfacher Routinen ausführt. Die Eingaberoutine in den Zeilen 9 bis 22 verwendet eine **For**- Schleife, die Druckroutine in den Zeilen 28 bis 32 eine **Do**-Schleife. Welches Schleifenkonstrukt Sie verwenden, ist nicht wichtig. Hier soll nur demonstriert werden, daß man gar nicht viel Code zu schreiben braucht, um große Datenmengen einzugeben und auszudrucken. Die Arrayindizes und die Schleifenanweisungen ermöglichen den Ausdruck.

Dieses Beispiel demonstriert *parallele Arrays*, d.h. zwei Arrays, die nebeneinander eingesetzt werden. Jedem Element in einem Array ist ein Element im anderen Array zugeordnet. Parallele Arrays verhalten sich im Speicher ähnlich wie verknüpfte Felder in Tabellen.

Dieses Programm ist ein praktisches Beispiel, aber es funktioniert nur, wenn genau 35 Familien aufgenommen werden. Was passiert, wenn der Verein wächst? In diesem Fall müssen Sie das Programm ändern. Die meisten Programme setzen deshalb keine so strikte Obergrenze für die Daten, wie es hier der Fall war. Die meisten Programmierer legen mehr als genug Arrayelemente an, um auch die größten Arrays verwalten zu können. Das Programm ermöglicht den Benutzern, zu steuern, wie viele dieser Elemente wirklich benutzt werden.

Das Programm in Listing 10.4 ist dem in Listing 10.3 sehr ähnlich, außer daß es für jedes Array 500 Elemente deklariert. Damit werden auf alle Fälle genügend Arrayelemente für den Verein reserviert. Die Benutzer geben nur so viele Einträge ein, wie tatsächlich vorhanden sind (1 bis maximal 500). Das Programm ist sehr flexibel. Es erlaubt eine variable Anzahl von Mitgliedern. Die Obergrenze wird erst erreicht, wenn es mehr als 500 Mitglieder gibt.



Deklariieren Sie ausreichend viel Platz für Ihre Bedürfnisse, aber deklarieren Sie nicht zu viel. Für jedes zusätzliche Arrayelement, das Sie reservieren, aber nicht nutzen, wird Speicherplatz vergeudet.

Listing 10.4: Sie können mehr Elemente reservieren, als Sie momentan brauchen

```
1: Private Sub varyNumb ()
2: ' Prozedur zur Aufnahme von Namen und Beiträgen
3: Dim strFamilyName(500) As String ' Arrayelemente reservieren
4: Dim curFamilyDues(500) As Currency
5: Dim intSub As Integer, intNumFam As Integer
6: Dim intMsg As Integer ' MsgBox()-Rückgabe
7: intNumFam = 1
8:
9: ' Die folgende Schleife fragt nach Familiennamen und Beiträgen,
10: ' bis der Benutzer die Eingabetaste drückt, ohne einen Namen
   einge-
11: ' geben zu haben. Wenn ein String mit der Länge Null übergeben
   wird,
12: ' wird die Do-Schleife vorzeitig beendet.
13: Do
14: strFamilyName(intNumFam) = InputBox("Nächster Familienname?")
15: If (strFamilyName(intNumFam) = "") Then Exit Do ' Beenden
```



```

16: curFamilyDues(intNumFam) = InputBox("Beiträge?")
17: intNumFam = intNumFam + 1 ' Indexvariable inkrementieren
18: Loop Until (intNumFam > 500)
19:
20: ' Nachdem die letzte Schleife beendet ist, ist intSub um eins
21: ' größer als die tatsächliche Anzahl der eingegebenen Werte
22:
23: ' Alle Eingabedaten anzeigen
24: For intSub = 1 To intNumFam - 1
25: intMsg = MsgBox("Familie" & intSub & "ist " &
strFamilyName(intSub))
26: intMsg = MsgBox("Ihre Beiträge: " & curFamilyDues(intSub))
27: Next intSub
28: End Sub

```

Die Schleife wird vorzeitig beendet, wenn der Benutzer die Eingabetaste drückt, ohne eine Eingabe vorgenommen zu haben. Sie müssen nicht 500 Elemente eingeben, weil 500 Elemente reserviert sind.

Das Programm in Listing 10.5 zeigt, wie man in beliebiger Reihenfolge auf verschiedene Arrayelemente zugreift. Das Programm fordert die Verkaufsdaten für die letzten 12 Monate an. Anschließend wartet es, bis ein Benutzer den Monat eingegeben hat, den er anzeigen möchte. Die Verkaufszahlen für diesen Monat werden ausgedruckt, ohne daß die anderen Monate berücksichtigt werden. Dies ist der Ansatz eines Suchprogramms, das Daten ermittelt, die in Arrays abgelegt sind. Die Daten werden in einem Array abgelegt und die Benutzer wählen einen bestimmten Abschnitt dieser Daten zur Anzeige aus.

Listing 10.5: Code, der nach Daten suchen soll

```

1: Private Sub salary ()
2: ' Verkaufszahlen für 12 Monate speichern und eine Auswahl
anbieten
3: Dim curSal(1 To 12) As Currency ' Elemente für 12 Verkaufszahlen
4: Dim intSub As Integer ' Schleifenindex
5: Dim intNum As Integer ' Monat, den der Benutzer auswählt
6: Dim intMsg As Integer ' MsgBox()-Rückgabe
7: Dim strAns As String
8:
9: For intSub = 1 To 12
10: curSal(intSub) = InputBox("Verkaufszahlen für Monat" &
Str(intSub) _
& "?", 0.00)
11: Next intSub
12:
13: ' Monat anfordern
14: Do
15: intNum = InputBox("Für welchen Monat (1-12) Verkaufszahlen
anzeigen?")
16: intMsg = MsgBox("Verkaufszahlen für " & Str(intNum) & " : " & _
curSal(intNum))
17: strAns = InputBox("Weiter (Y/N)?")
18: Loop While (strAns = "Y" Or strAns = "y")

```

19: End Sub

Nachdem der Benutzer in den Zeilen 9 bis 11 die 12 Verkaufszahlen eingegeben hat, kann er sie alle einzeln anfordern, indem er einfach nur den betreffenden Monat angibt (den Indexwert).

Das Programm in Listing 10.6 zeigt ein paar der mathematischen Operationen, die Sie für Arrays ausführen können. Sehen Sie sich das Programm an. Es fordert eine Liste mit Temperaturen an, bis der Benutzer den Wert -99 eingibt, woran es erkennt, daß es keine weiteren Temperaturen gibt. Anschließend wird die Durchschnittstemperatur berechnet. Dazu werden alle Temperaturen addiert und durch die Gesamtzahl der eingegebenen Temperaturen dividiert.

Listing 10.6: Der Benutzer kann Ihrem Programm mitteilen, daß keine weiteren Daten eingegeben werden sollen

```
1: Private Sub tempAvg ()
2: ' Eine Temperaturliste abfragen und den Durchschnitt bilden
3: Dim sngTemp(1 To 100) As Single ' Bis zu 100 Eingaben
4: Dim sngTotalTemp As Single ' Gesamtsumme
5: Dim sngAvgTemp As Single
6: Dim intSub As Integer ' Index
7: Dim intMsg As Integer ' MsgBox()-Rückgabe
8:
9: ' Temperatureingabe anfordern
10: For intSub = 1 To 100 ' Obergrenze
11: sngTemp(intSub) = InputBox("Nächste Temperatur (-99 = Ende)?")
12: ' Wenn der Benutzer abbricht, Zähler dekrementieren und Schleife
' verlassen
13: If (sngTemp(intSub) = -99) Then
14: intSub = intSub - 1 ' Korrektur bei vorzeitigem Abbruch
15: Exit For
16: End If
17: sngTotalTemp = sngTotalTemp + sngTemp(intSub) ' Gesamtsumme
18:
19: Next intSub
20: ' Durchschnitt berechnen
21: sngAvgTemp = sngTotalTemp / intSub
22: intMsg = MsgBox("Durchschnittstemperatur: " & sngAvgTemp)
23: End Sub
```

Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie die Maus in Ihre Visual-Basic-Applikationen integrieren. Die Mausereignisse informieren Ihr Programm über Klicks, Doppelklicks und die verwendeten Tasten. Mit Hilfe geeigneter Methoden stellen Sie Drag&Drop-Operationen bereit, die es dem Benutzer ermöglichen, Objekte von einer Position an eine andere zu verschieben.

Außerdem haben Sie den Timer kennengelernt, mit dem Sie in Ihren Applikationen zeitabhängige Verarbeitungen realisieren. Der Timer verwaltet Zeitintervalle. Wenn eine bestimmte Zeit vergangen ist, wird ein Timer-Ereignis ausgelöst.

Es gibt unterschiedliche Varianten von Listen-Steuerelementen. Listenfelder und Kombinationsfelder

werden mit Hilfe der Werkzeugsammlung angelegt. Der Unterschied zwischen beiden Varianten kann verwirrend sein, aber die Beispiele in diesem Kapitel haben alle vier Kombinationen beschrieben, so daß Sie nun entscheiden können, welche für Ihre Bedürfnisse am besten geeignet ist.

Im letzten Abschnitt des Kapitels wurden Arrays vorgestellt, die den Elementen eines Listenfelds von der Struktur her sehr ähnlich sind. Ein Variablen-Array enthält mehrere Werte, auf die Sie über den numerischen Index zugreifen.

In Kapitel 11 erfahren Sie, wie die Formulare für Visual-Basic-Anwendungen verwaltet werden.

Fragen und Antworten

Frage:

Woher weiß ich, welches Listen-Steuerelement ich verwenden soll?

Antwort:

Die Listen-Steuerelemente verhalten sich alle ein bißchen anders. Sie nutzen sie in Form von Listenfeldern und Kombinationsfeldern. Das Listenfeld-Steuerelement legt ein Listenfeld an, in dem der Benutzer Elemente auswählt. Diese Liste behält ihre Höhe und Breite bei, ebenso wie die Position. Der Benutzer kann der Liste keine Elemente hinzufügen. Falls nicht alle Listenelemente angezeigt werden können, werden Bildlaufleisten bereitgestellt.

Das Kombinationsfeld kann durch seine Style-Eigenschaft eine der drei Varianten von Listen-Steuerelementen darstellen. Das Drop-down-Kombinationsfeld bleibt geschlossen, bis der Benutzer es aufklappt. Damit sparen Sie Platz. Das einfache Kombinationsfeld bietet dem Benutzer eine Möglichkeit, neue Werte in die Liste einzutragen. Das Drop-down-Listenfeld kombiniert die platzsparende Funktion des Drop-down-Kombinationsfelds mit der Möglichkeit, Elemente in die Liste einzutragen.

Frage:

Soll ich Listenfelder oder Arrays verwenden, um dem Benutzer meine Wertelisten anzuzeigen?

Antwort:

Wenn Sie einem Benutzer eine Werteliste anzeigen möchten, müssen Sie nicht zwischen Listenfeldern und Arrays wählen. Sie können innerhalb derselben Applikation beides verwenden. Das Array kann die Daten aufnehmen, während Ihre Applikation sie abfragt und verarbeitet, und in einem der Listen-Steuerelemente können Sie die Daten anzeigen.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welche Mausereignisse reagieren auf bestimmte Maustasten? Welche Mausereignisse reagieren auf jede Maustaste?

2. Wie stellen Sie fest, welche Taste geklickt wurde, wenn ein `MouseDown`-Ereignis auftritt?
3. Wie ändern Sie das Icon, das während einer Verschiebeoperation angezeigt wird?
4. Wie richten Sie eine Ereignisprozedur ein, die auf ein Zeitintervall reagiert, das größer als 65 Sekunden (also die Obergrenze für das Timer-Steuerelement) ist?
5. Wie wird ein Listenfeld initialisiert?
6. Wie stellt ein Programm fest, welches Element der Benutzer ausgewählt hat?
7. Nennen Sie zwei Möglichkeiten, alle Elemente aus einem Listenfeld zu entfernen.
8. Warum müssen Sie mindestens ein zusätzliches Steuerelement auf einem Formular anlegen, das ebenfalls den Fokus erhalten kann, wenn Ihre Benutzer in einem Kombinationsfeld neue Werte eingeben können?
9. Wie stellen Sie sicher, daß die Werte in einem Listen-Steuerelement sortiert vorliegen, egal wie viele Elemente hinzugefügt oder entfernt wurden?
10. Wie viele Elemente reserviert die folgende `Dim`-Anweisung?
`Dim varStaff(-18 To 4) As Variant`

Übungen

1. Schreiben Sie ein Programm, das die Mitglieder Ihrer Familie in einem Listenfeld anzeigt. Geben Sie so viele Familienmitglieder an, daß im Listenfeld Bildlaufleisten angezeigt werden. Achten Sie darauf, daß die Liste stets sortiert vorliegt.
2. Führen Sie das Beispielprojekt `ListCmbo` aus, das in Visual Basic enthalten ist. Die Applikation stellt eine Datenbank mit Computerbuchverlagen bereit. Klicken Sie auf die Optionsschaltfläche **Standard-Kombinationsfeld verwenden** und klicken Sie dann, um das Feld **Staat** anzuzeigen. Klicken Sie auf das Optionsfeld **Standard-Listenfeld verwenden** und klicken Sie erneut auf das Feld **Staat**. Sie erkennen schnell, wie sich die beiden Listen unterscheiden.
3. Ändern Sie die in Übung 1 entwickelte Applikation so ab, daß alle drei Kombinationsfeldtypen auf dem Formular angezeigt werden und dieselben Daten enthalten. Wenn sich ein Kombinationsfeld ändert (wenn der Benutzer beispielsweise einen neuen Wert einträgt), soll diese Änderung auch in den beiden anderen Kombinationsfeldern berücksichtigt werden.

Woche 2

Bonusprojekt 4: Mehrfachauswahlen in Listefeldern

Die Applikation in diesem Bonusprojekt demonstriert, wie man Listenfelder so einrichtet, daß der Benutzer mehrere Elemente auswählen kann. Außerdem wird hier Ihr Wissen über Arrays vertieft. In der Applikation werden Mehrfachauswahlen ermöglicht, indem die entsprechenden Eigenschaftswerte gesetzt werden. Außerdem erkennt sie, ob der Benutzer mehrere Werte ausgewählt hat, und ermittelt diese in der Liste, um sie an anderer Stelle weiterzuverarbeiten.

Die Mehrfachauswahl wird häufig benötigt. Beispielsweise könnte ein Verwaltungsprogramm eine Liste verschiedener Schulen anzeigen, und die Benutzer kennzeichnen eine oder mehrere der Schulen, die sie besucht haben. Oder eine Firma bietet einem Kunden mehrere Produkte in einer Liste an und ermöglicht ihm, nicht nur ein einzelnes davon auszuwählen, sondern *mehrere* - was für die Verkaufszahlen jedenfalls besser ist.

Vorbereitung auf Mehrfachauswahlen

Um ein Listen-Steuerelement so einzurichten, daß es Mehrfachauswahlen unterstützt, ändern Sie die `MultiSelect`-Eigenschaft. Wenn Sie diese Eigenschaft auf `1 - Simple` setzen (im Gegensatz zum Standardwert, `0 - Kein`), können die Benutzer mehrere Werte auswählen. Wenn Sie die `MultiSelect`-Eigenschaft auf `2 - Erweitert` setzen, können die Benutzer die (^a)-Taste gedrückt halten, während sie klicken, um so einen ganzen Elementbereich auszuwählen. Außerdem können sie die (Strg)-Taste drücken, während sie klicken, um so einzelne Elemente auszuwählen, die nicht nebeneinander liegen (so wie sie es von den Öffnen-Dialogfeldern in Windows gewohnt sind).

Abbildung BP4.1 zeigt eine Applikation mit mehreren Textfeldern, die den Wert »Ausgewählt« anzeigen, wenn das Listenelement für die entsprechende Stadt selektiert ist, und »Nicht ausgewählt«, wenn das Listenelement nicht ausgewählt ist. Das Listefeld oben auf dem Formular, »Ziel«, hat die `MultiSelect`-Eigenschaft `2 - Erweitert`, um eine Mehrfachauswahl zu ermöglichen.

Ziel

Chicago
Dallas
Seattle
Washington

Chicago
Nicht ausgewählt

Washington
Nicht ausgewählt

Dallas
Nicht ausgewählt

Houston
Nicht ausgewählt

Seattle
Nicht ausgewählt

Dayton
Nicht ausgewählt

Abbildung BP4.1:
Jetzt sind Mehrfachauswahlen in Listenfeldern möglich.

Als erstes legen Sie die Steuerelemente auf dem Formular an. Gehen Sie dazu nach Tabelle BP4.1 vor.

Tabelle BP4.1: Legen Sie diese Steuerelemente an und setzen Sie die entsprechenden Eigenschaften

Steuerelement	Eigenschaftswert
Formular Name	frmDest
Formular Caption	Listenfeld-Demo
Formular Height	6600
Formular Width	7230
Bezeichnungsfeld #1 Name	lblListBoxCap
Bezeichnungsfeld #1 Caption	Ziel
Bezeichnungsfeld #1 Schriftstil	Fett
Bezeichnungsfeld #1 Schriftgröße	24
Bezeichnungsfeld #1 Height	600

Bezeichnungsfeld #1 Left	2190
Bezeichnungsfeld #1 Top	120
Bezeichnungsfeld #1 Width	3060
Bezeichnungsfeld #2 Name	lblChicago
Bezeichnungsfeld #2 BorderStyle	Fest einfach
Bezeichnungsfeld #2 Caption	Chicago
Bezeichnungsfeld #2 Schriftstil	Fett
Bezeichnungsfeld #2 Schriftgröße	18
Bezeichnungsfeld #2 Height	495
Bezeichnungsfeld #2 Left	360
Bezeichnungsfeld #2 Top	1995
Bezeichnungsfeld #2 Width	2655
Bezeichnungsfeld #3 Name	lblWashington
Bezeichnungsfeld #3 Caption	Washington
Bezeichnungsfeld #3 BorderStyle	Fest einfach
Bezeichnungsfeld #3 Schriftstil	Fett
Bezeichnungsfeld #3 Schriftgröße	18
Bezeichnungsfeld #3 Height	495
Bezeichnungsfeld #3 Left	4035
Bezeichnungsfeld #3 Top	1980
Bezeichnungsfeld #3 Width	2655
Bezeichnungsfeld #4 Name	lblDallas
Bezeichnungsfeld #4 Caption	Dallas
Bezeichnungsfeld #4 BorderStyle	Fest einfach
Bezeichnungsfeld #4 Schriftstil	Fett
Bezeichnungsfeld #4 Schriftgröße	18
Bezeichnungsfeld #4 Height	495
Bezeichnungsfeld #4 Left	360
Bezeichnungsfeld #4 Top	3480
Bezeichnungsfeld #4 Width	2655
Bezeichnungsfeld #5 Name	lblHouston
Bezeichnungsfeld #5 Caption	Houston

Bezeichnungsfeld #5 BorderStyle	Fest einfach
Bezeichnungsfeld #5 Schriftstil	Fett
Bezeichnungsfeld #5 Schriftgröße	18
Bezeichnungsfeld #5 Height	495
Bezeichnungsfeld #5 Left	4035
Bezeichnungsfeld #5 Top	3480
Bezeichnungsfeld #5 Width	2655
Bezeichnungsfeld #6 Name	lblSeattle
Bezeichnungsfeld #6 Caption	Seattle
Bezeichnungsfeld #6 BorderStyle	Fest einfach
Bezeichnungsfeld #6 Schriftstil	Fett
Bezeichnungsfeld #6 Schriftgröße	18
Bezeichnungsfeld #6 Height	495
Bezeichnungsfeld #6 Left	360
Bezeichnungsfeld #6 Top	4920
Bezeichnungsfeld #6 Width	2655
Bezeichnungsfeld #7 Name	lblDayton
Bezeichnungsfeld #7 Caption	Dayton
Bezeichnungsfeld #7 BorderStyle	Fest einfach
Bezeichnungsfeld #7 Schriftstil	Fett
Bezeichnungsfeld #7 Schriftgröße	18
Bezeichnungsfeld #7 Height	495
Bezeichnungsfeld #7 Left	4035
Bezeichnungsfeld #7 Top	4920
Bezeichnungsfeld #7 Width	2655
Listenfeld Name	lstFirstList
Listenfeld Height	840
Listenfeld Left	2865
Listenfeld MultiSelect	2-Extended
Listenfeld Top	870
Listenfeld Width	1335
Textfeld #1 Name	txtChicago

Textfeld #1 Schriftgröße	24
Textfeld #1 Schriftstil	Fett
Textfeld #1 Height	495
Textfeld #1 Left	120
Textfeld #1 Text	Nicht ausgewählt
Textfeld #1 Top	2520
Textfeld #1 Width	3105
Textfeld #2 Name	txtWashington
Textfeld #2 Schriftgröße	24
Textfeld #2 Schriftstil	Fett
Textfeld #2 Height	495
Textfeld #2 Left	3840
Textfeld #2 Text	Nicht ausgewählt
Textfeld #2 Top	2520
Textfeld #2 Width	3105
Textfeld #3 Name	txtDallas
Textfeld #3 Schriftgröße	24
Textfeld #3 Schriftstil	Fett
Textfeld #3 Height	495
Textfeld #3 Left	120
Textfeld #3 Text	Nicht ausgewählt
Textfeld #3 Top	3960
Textfeld #3 Width	3105
Textfeld #4 Name	txtHouston
Textfeld #4 Schriftgröße	24
Textfeld #4 Schriftstil	Fett
Textfeld #4 Height	495
Textfeld #4 Left	3840
Textfeld #4 Text	Nicht ausgewählt
Textfeld #4 Top	3960
Textfeld #4 Width	3105
Textfeld #5 Name	txtSeattle

Textfeld #5 Schriftgröße	24
Textfeld #5 Schriftstil	Fett
Textfeld #5 Left	120
Textfeld #5 Height	495
Textfeld #5 Text	Nicht ausgewählt
Textfeld #5 Top	5400
Textfeld #5 Width	3105
Textfeld #6 Name	txtDayton
Textfeld #6 Schriftgröße	24
Textfeld #6 Schriftstil	Fett
Textfeld #6 Height	495
Textfeld #6 Left	3720
Textfeld #6 Text	Nicht ausgewählt
Textfeld #6 Top	5400
Textfeld #6 Width	3105

Der Code

Listing BP4.1 zeigt den Code für dieses Projekt.

Listing BP4.1: Initialisierung des Listenfelds und Test auf Mehrfachauswahl

```

1: Private Sub Form_Load()
2: ' Ausführen beim Laden des Formulars
3: lstFirstList.AddItem "Chicago"
4: lstFirstList.AddItem "Dallas"
5: lstFirstList.AddItem "Seattle"
6: lstFirstList.AddItem "Washington"
7: lstFirstList.AddItem "Houston"
8: lstFirstList.AddItem "Dayton"
9: End Sub
10:
11: Private Sub lstFirstList_Click()
12: ' Ausgewähltes Element in das erste Textfeld kopieren
13: ' Ausgewählten Index in das erste Textfeld kopieren
14: If lstFirstList.Selected(0) Then
15: txtChicago.Text = "Selektiert"
16: Else
17: txtChicago.Text = "Nicht ausgewählt"
18: End If
19:

```

```

20: If lstFirstList.Selected(1) Then
21: txtDallas.Text = "Selektiert"
22: Else
23: txtDallas.Text = "Nicht ausgewählt"
24: End If
25:
26: If lstFirstList.Selected(2) Then
27: txtSeattle.Text = "Selektiert"
28: Else
29: txtSeattle.Text = "Nicht ausgewählt"
30: End If
31:
32: If lstFirstList.Selected(3) Then
33: txtWashington.Text = "Selektiert"
34: Else
35: txtWashington.Text = "Nicht ausgewählt"
36: End If
37:
38: If lstFirstList.Selected(4) Then
39: txtHouston.Text = "Selektiert"
40: Else
41: txtHouston.Text = "Nicht ausgewählt"
42: End If
43:
44: If lstFirstList.Selected(5) Then
45: txtDayton.Text = "Selektiert"
46: Else
47: txtDayton.Text = "Nicht ausgewählt"
48: End If
49:
50: End Sub

```

Analyse

Die Zeilen 3 bis 8 initialisieren das Listenfeld. Weil die `MultiSelect`-Eigenschaft des Listenfelds auf `Erweitert` gesetzt ist, kann der Benutzer zur Laufzeit eine Mehrfachauswahl vornehmen.

Wenn eine Mehrfachauswahl möglich sein soll, muß Visual Basic ein spezielles Array anlegen, das sich wie ein Variablen-Array verhält, aber aus Eigenschaften besteht. Das Eigenschaften-Array heißt für die Listenfelder `Selected`. In diesem Fall umfaßt das Array den Bereich `Selected(0)` bis `Selected(5)`, weil im Listenfeld sechs Einträge vorhanden sind. Das Array enthält nur Boolesche Datentypen.



Wenn der Benutzer ein Element deselektiert, indem er ein ausgewähltes Element noch einmal anklickt, wird der `Selected`-Wert dieses Elements wieder auf `False` gesetzt.

Woche 2

Bonusprojekt 5: Die Übung mit der Maus

Die Applikation in diesem Bonusprojekt demonstriert, wie man Programme schreibt, die auf die Maus reagieren. Durch die Reaktion auf mausbezogene Ereignisse und die Auswertung der Argumente einer Ereignisprozedur kann Ihre Applikation feststellen, ob der Benutzer geklickt, doppelgeklickt, die Maus bewegt oder ein Objekt verschoben hat.

Dieses Bonusprojekt geht etwas anders vor als die vorherigen. Als erstes sehen Sie ein kurzes Beispiel, das beschreibt, wie Sie das für die Maus verwendete Icon ändern. Anschließend lernen Sie eine Applikation kennen, die auf Mausereignisse reagiert.

Änderung des Maus-Icons

Es ist ganz einfach, den Mauszeiger zu ändern, der angezeigt wird, wenn sich die Maus über einem bestimmten Steuerelement befindet. Wir betrachten hier ein extrem einfaches Formular mit nur einer Schaltfläche. Angenommen, Sie möchten den Mauscursor so ändern, daß ein Smiley angezeigt wird, wenn der Benutzer die Maus über die Schaltfläche cmdHappy schiebt. Klicken Sie im Eigenschaftenfenster für die Schaltfläche auf die drei Punkte, um ein Icon aus dem Verzeichnis \Graphicx\Icons\Misc anzuzeigen. Die Datei Face03.ico ist am besten dafür geeignet.

Nachdem Sie das Maus-Icon angezeigt haben, ändern Sie die `MousePointer`-Eigenschaft der Schaltfläche auf `99 - Benutzerdefiniert`. Der benutzerdefinierte Wert weist Visual Basic an, die `MouseIcon`-Eigenschaft auszuwerten, um das Icon nur anzuzeigen, wenn der Benutzer die Maus über die Schaltfläche bewegt. Abbildung BP5.1 zeigt den resultierenden Mauszeiger.

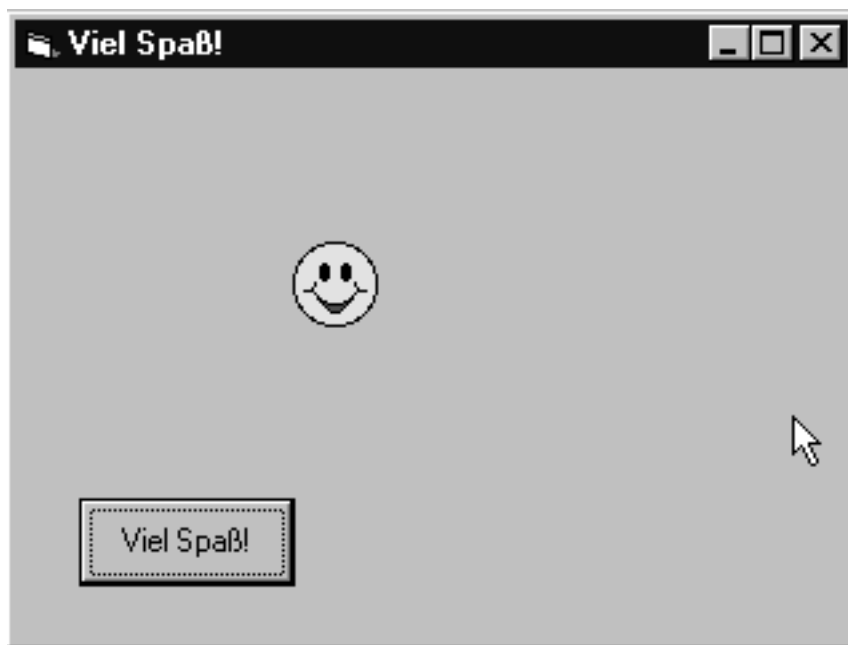


Abbildung BP5.1:
Der Mauszeiger sieht damit recht fröhlich aus.

Wenn Sie eine kleine Applikation mit einer Befehlsschaltfläche in der Mitte des Formulars anlegen, wie in Abbildung BP5.1 gezeigt, sehen Sie, daß Visual Basic es übernimmt, das Icon zu ändern, wenn Sie den Mauszeiger über die Schaltfläche schieben.

Maus-Programmierung

Dieses Projekt beschreibt ein neues Projekt mit einem Anzeigefeld, das ein Bild darstellt, wie Sie es aus Kapitel 2 kennen. Sie setzen den Wert der `Picture`-Eigenschaft für das Anzeigefeld auf das Bullseye-Icon, das Sie im Verzeichnis `\Graphics\Icons\Misc` finden. Das Textfeld `txtMouse`, das Sie dem Formular hinzufügen (siehe Tabelle BP5.1), zeigt den Mausstatus an, während Sie die Maus bewegen und klicken. Dieses Projekt zeigt, wie mausbezogene Ereignisprozeduren Mausaktionen feststellen.

Abbildung BP5.2 zeigt das Formular, das Sie anlegen.



Abbildung BP5.2:
Der Benutzer klickt auf dieses Bild.

Als erstes fügen Sie dem Formular die Steuerelemente hinzu. Gehen Sie dabei vor, wie in Tabelle BP5.1 beschrieben.

Tabelle BP5.1: Diese Steuerelemente legen Sie auf dem Formular an

Steuerelement	Eigenschaftswert
Formular Name	frmMouse
Formular Caption	Maussteuerung
Formular Height	4230
Formular Width	5265
Textfeld Name	txtMouse
Textfeld Alignment	2 - Zentriert
Textfeld Schriftstil	Fett
Textfeld Schriftgröße	14
Textfeld Height	1095
Textfeld Left	840
Textfeld MultiLine	True
Textfeld Text	Verwenden Sie die Maus
Textfeld Top	1320
Textfeld Width	3255
Anzeigefeld Name	imgMouse
Anzeigefeld Height	480
Anzeigefeld Left	2400
Anzeigefeld Picture	\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Misc\Bullseye
Anzeigefeld Top	480

Der Code

Listing BP5.1 zeigt den Code, den Sie für dieses Projekt brauchen.

Listing BP5.1: Test auf Mausereignisse

```

1: Private Sub Form_Click()
2: txtMouse.Text = "Sie haben auf das Formular geklickt"
3: Beep ' Ein Click-Ereignis ist aufgetreten
4: End Sub
5: Private Sub Form_DblClick()
6: txtMouse.Text = "Sie haben auf das Formular doppelgeklickt"
7: End Sub
8:
9: Private Sub Form_MouseDown(intButton As Integer, _
intShift As Integer, sngX As Single, sngY As Single)
10: ' Klick auf dem Formular
11: txtMouse.Text = "Sie haben geklickt an der Position: " & sngX &
", " & sngY
12: End Sub
13: ' Die Argumente wurden in der vorigen Prozedur ignoriert
14:
15: Private Sub Form_MouseMove(intButton As Integer, _
intShift As Integer, sngX As Single, sngY As Single)
16: txtMouse.Text = "Mausbewegung..."
17: End Sub
18: ' Die Argumente wurden in der vorigen Prozedur ignoriert
19:
20: Private Sub imgMouse_Click()
21: txtMouse.Text = "Sie haben auf das Bild geklickt"
22: End Sub
23:
24: Private Sub imgMouse_DblClick()
25: txtMouse.Text = "Sie haben auf das Bild doppelgeklickt"
26: End Sub
27:
28: Private Sub imgMouse_MouseDown(intButton As Integer, intShift As
_
Integer, sngX As Single, sngY As Single)
29: '
30: ' Klick über dem Bild
31: txtMouse.Text = "Klick an der Position: " & sngX & ", " & sngY
32: End Sub
33:
34: Private Sub imgMouse_MouseMove(intButton As Integer, _
intShift As Integer, sngX As Single, sngY As Single)
35: txtMouse.Text = "Sie haben die Maus über das Bild bewegt"
36: End Sub

```

Analyse

Die verschiedenen Ereignisprozeduren überwachen die Mausaktivität. Die Zeilen 1 bis 4 stellen die einfachste Ereignisprozedur dar, die reagiert, wenn Sie irgendwo auf dem Formular klicken. Das Beep wird ausgegeben, weil man das Click-Ereignis nicht sehen kann. Es passiert zu schnell und

ist vorbei, bevor Sie es erkannt haben. Das `MouseDown` -Ereignis, das Sie aus Kapitel 10 kennen, tritt vor einem `Click`-Ereignis auf, wenn beide Ereignisse im Code angegeben sind. Sie sehen also, wie der `MouseDown`-Text erscheint, der anzeigt, wo auf dem Formular Sie geklickt haben. Wenn Sie die Maustaste loslassen, wird der Text für das `Click`-Ereignis kurz angezeigt. Die `Beep`-Anweisung stellt den hörbaren Beweis dar, daß das `Click`-Ereignis irgendwann aufgetreten ist.



Beachten Sie, daß auch ein Doppelklick auf dem Formular ein `MouseDown`- Ereignis und danach ein `Click`-Ereignis erzeugt, bevor er das `DbClick`-Ereignis (das in den Zeilen 9 bis 12 verarbeitet wird) erzeugt. Sie hören den Ton, sobald Sie mit der Maus doppelklicken. Zuerst sehen Sie jedoch die Koordinaten, weil das `MouseDown`-Ereignis anzeigt, wo es aufgetreten ist.



Einer der Nachteile, der die Programmierer stark frustriert, ist, daß bei jedem Mausklick ein `MouseMove`-Ereignis ausgelöst wird.

Die Zeilen 9 bis 12 demonstrieren die Verwendung der Mausargumente im `MouseDown`- Ereignis. Die `Twip`-Koordinatenwerte `sngX` und `sngY` zeigen, wo Sie die Maus geklickt haben. Beachten Sie, daß dieser Code sowohl für einen Linksklick als auch für einen Rechtsklick ausgeführt wird, weil beide das `MouseDown`-Ereignis auslösen. Die Prozedur wird nur ausgeführt, wenn Sie über dem Formular klicken; sobald Sie die Maustaste loslassen, folgt ein `Click`-Ereignis.

Die Zeilen 15 bis 17 verarbeiten die Mausbewegung über dem Formular.

Die Zeilen 20 bis 22 demonstrieren, was passiert, wenn Sie die Maus auf das Bullseye- Icon schieben. Wenn Sie auf das Icon klicken, wird das Textfeld aktualisiert und zeigt an, daß Sie auf das Bild und nicht auf das Formular geklickt haben (das macht die `Click`-Ereignisprozedur des Anzeigefelds). Aufgrund des `MouseDown`-Ereignisses, das dem `Click`- Ereignis vorausgeht, sehen Sie jedoch den schnell aufgeblendeten Text für `Click` nicht; wenn Sie jedoch eine `Beep`-Anweisung in das `Click`-Ereignis eingebaut haben, hören Sie den Ton, sobald der Text für das `MouseDown`-Ereignis im Feld angezeigt wird.

Die Zeilen 24 bis 26 zeigen, wie die `DbClick`-Ereignisprozedur auf Doppelklicks über dem Anzeigefeld reagiert. Die Zeilen 28 bis 32 demonstrieren das `MouseDown`-Ereignis für das Anzeigefeld, das auftritt, bevor das `Click`-Ereignis erfolgt. Die Zeilen 34 bis 36 schließlich verarbeiten die Mausbewegungen über dem Bild.



Sie könnten die Koordinaten für ein `MouseMove`-Ereignis im Textfeld anzeigen, aber diese würden sich zu schnell ändern, als daß Sie sie lesen könnten.

Versuchen Sie, dieses Bonusprojekt nachzuvollziehen, damit Sie verstehen, was Sie bei der Maus-Programmierung erwartet. Außerdem werden Sie dann verstehen, warum das `MouseDown`-Ereignis den Ereignissen `Click` und `DbClick` vorausgeht.

Automatisches Drag&Drop implementieren

Es ist ganz einfach, einer Applikation Drag&Drop hinzuzufügen. Sie ändern die `DragIcon`-Eigenschaft des Anzeigefelds einfach so, daß ein anderes Icon erscheint, wenn Sie die Maus ziehen. Verwenden Sie beispielsweise das Icon `Clock02.ico`. Um ein automatisches Drag&Drop zu implementieren, ändern Sie die `DragMode`-Eigenschaft auf `1 - Automatisch`, so daß sich Visual Basic um die Details beim Verschieben von Objekten kümmert.

Anschließend fügen Sie die folgende `MouseDown`-Ereignisprozedur ein:

```
Private Sub Form_DragDrop(cntSource As Control, sngX As Single, _
sngY As Single)
Source.Move sngX, sngY ' Bild fallenlassen
End Sub
```

Das `Source`-Argument für das `DragDrop`-Ereignis enthält das Steuerelement, das Sie auf dem Formular ablegen möchten. Die Koordinaten `sngX` und `sngY` teilen dem Steuerelement mit, wo es abgelegt wird. Die `Move`-Methode verschiebt das Steuerelement (in diesem Fall das Anzeigefeld) an die Endposition.

Führen Sie die Applikation aus und verschieben Sie das `Bullseye`-Bild an eine andere Position auf dem Formular. Für die meisten Drag&Drop-Operationen verwenden Sie das automatische Drag&Drop, so daß Visual Basic die Details für Sie übernimmt.

Manuelles Drag&Drop implementieren

Für manuelle Drag&Drop-Operationen setzen Sie die `DragMode`-Eigenschaft auf `0 - Manuell` und ändern die `MouseDown`-Prozedur wie folgt:

```
Private Sub imgMouse_MouseDown(intButton As Integer, _
intShift As Integer, sngX As Single, sngY As Single)
' Klick über dem Bild
txtMouse.Text = "Klick über dem Bild bei: " & sngX & ", " & sngY
imgMouse.Drag ' Drag&Drop initiieren
End Sub
```

Die einzige neue Anweisung ist hier `imgMouse.Drag`. Ihr `MouseDown`-Ereignis muß die Drag&Drop-Operation initiieren. Der Vorteil dabei ist, daß Sie gegebenenfalls andere Operationen ausführen können, die für das `MouseDown`-Ereignis erforderlich sind, bevor Sie die Drag&Drop-Operation starten. Ohne den manuellen Drag-Modus unterliegt das Verschieben nicht Ihrer Kontrolle.



Verwenden Sie das manuelle Drag&Drop, wenn Sie neben dem Drag&Drop auch noch andere Operationen für `MouseDown` ausführen wollen.

Die `Drag`-Methode aktiviert das Drag&Drop. Ohne die `Drag`-Methode könnte die `MouseDown()`-Ereignisprozedur die Drag&Drop-Operation nicht initiieren. Verwenden Sie manuelle Drag&Drop-Operationen, wenn Sie kontrollieren möchten, wie das Drag&Drop ausgeführt wird.

Woche 2

Tag 11

Formulare

Dieses Kapitel erklärt genauer, wie man Formulare verwaltet und programmiert, als das bisher möglich war. Sie lernen hier die wichtigsten Eigenschaften, Ereignisse und Methoden von Formularen kennen. Eine der praktischsten Methoden, die Ihnen dabei begegnen wird, ist `Print`, die Text direkt auf das Formular ausgibt, ohne daß dafür ein Steuerelement erforderlich ist. Interessanterweise basiert `Print` direkt auf dem `PRINT`- Befehl der alten BASIC-Sprache - beide arbeiten genau gleich.

Sie lernen nicht nur, wie Sie Ihre Formulare steuern, sondern erhalten auch bessere Einsicht in die Vor- und Nachteile von MDI- und SDI-Applikationen mit mehreren Formularen. Sie erfahren, wie man Symbolleisten und sogenannte *CoolBars* in einem Formularfenster anlegt, um den Benutzern eine zusätzliche Methode an die Hand zu geben, mit einem Programm zu arbeiten.

Heute lernen Sie die folgenden Dinge kennen:

- Eigenschaften, Ereignisse und Methoden von Formularen
- Formular-Auflistungen
- Text mit der `Print`-Methode auf einem Formular ausgeben
- Mehr über MDI- und SDI-Applikationen
- Benutzerdefinierte Formulareigenschaften
- Symbolleiste
- CoolBars
- Warum Band-Objekte für Symbolleisten und Coolbars so wichtig sind

Eigenschaften, Ereignisse und Methoden

Sie haben zwar erst 10 Tage mit der Programmiersprache Visual Basic verbracht, aber dabei so viele Steuerelemente und Sprachkonstrukte kennengelernt, daß Sie auf dem besten Weg zum Visual-Basic-Profi sind! Nichtsdestotrotz wurde eine der wichtigsten Komponenten, das Formular, bisher kaum erklärt. Dieses Kapitel soll sich ganz auf das Formular konzentrieren.

Sie wissen bereits, wie man mit Hilfe von Eigenschaften das Aussehen und das Verhalten von Formularen ändert. Sie wissen, wie man einen Titel für das Formular hinzufügt, seine Größe verändert und steuert, und wie dem Benutzer die Schaltflächen zur Größenänderung bereitgestellt werden sollen, wie beispielsweise die Minimier- und Maximier-Schaltflächen. Darüber hinaus kennen Sie auch schon die Ereignisprozedur `Form_Load()`, mit der Listen-Steuerelemente initialisiert werden. Diese Prozedur ist ausgezeichnet dazu geeignet, alle möglichen Initialisierungen aufzunehmen, die erfolgen sollen, bevor der Benutzer das Formular sieht. (Die Angabe des Formularnamens ist hier nicht erforderlich, weil der Code innerhalb des Formularmoduls steht. Das einzige Formular, auf das sich der Code bezieht, ist das aktuelle Formular, unabhängig davon, welchen Namen es hat.)



In der Ereignisprozedur `Form_Unload()` ist Platz für den Code, der alle Aufräumarbeiten ausführt, nachdem ein Formular vom Bildschirm des Benutzers und aus dem Speicher entfernt wurde. Anders als die Methoden `Form.Hide` und `Form.Show` sind `Form_Load()` und `Form_Unload()` für die Speicherverwaltung zuständig. `Form_Load()` lädt das Formular zur Laufzeit in den Speicher, `Form_Unload()` entfernt es zur Laufzeit aus dem Speicher. Die Methoden `Show` und `Hide` sind nur Aliase für die `True`- und `False`-Werte der `Visible`-Eigenschaft für das Formular.

Was Sie tun sollten

Verwenden Sie die Methoden `Form.Hide` und `Form.Show`, wenn Sie ein Formular verbergen oder anzeigen möchten. `Form_Load()` und `Form_Unload()` sollten nur benutzt werden, wenn Sie ein Formular in den Speicherraum der Applikation laden bzw. daraus entfernen.

Der nächste Abschnitt, »Formular-Auflistungen«, beschreibt eine spezielle Aufgabe des `Form_Unload()`-Ereignisses.

Häufig sehen Sie Code wie den folgenden in einer `Form_Load()`-Prozedur:

```
frmAForm.Left = (Screen.Width - frmAForm.Width) / 2
```

frmAForm.Top = (Screen.Height - frmAForm.Height) / 2

Dieser Code zentriert das Formular innerhalb der Bildschirmkoordinaten. In Kapitel 16 lernen Sie spezielle Objekte kennen, die von Visual Basic unterstützt werden. Eines dieser Objekte ist `Screen`, das den Bildschirm des Benutzers repräsentiert. Das `Screen`-Objekt ändert sich abhängig von der Bildschirm- oder Grafikkartenauflösung des Benutzers. Sie ermitteln die Bildschirmauflösung des Benutzers, indem Sie die `Width`- und `Height`-Eigenschaften des `Screen`-Objekts auswerten.



Sie können die Schreibarbeit reduzieren, indem Sie das Standardobjekt in einer Ereignisprozedur weglassen. Statt den Formularnamen anzugeben, schreiben Sie den Code zum Zentrieren des Formulars etwa wie folgt:

```
Left = (Screen.Width - Width) / 2
Top = (Screen.Height - Height) / 2
```

So lange sich dieser Code innerhalb der Ereignisprozedur für das Formular befindet, beispielsweise in `Form_Load()`, können Sie den Formularnamen weglassen. Durch die Verringerung der Schreibarbeit laufen Sie jedoch gleichzeitig Gefahr, daß Ihr Code mehrdeutig wird. Wenn Sie immer alle Objekte angeben, auch die Standardobjekte, machen Sie Ihren Code deutlicher und einfacher wartbar.



Ein *Standardobjekt* ist das Objekt (ein Formular, Steuerelement oder irgendein anderes Objekt), das Visual Basic im Code verwendet, falls Sie kein bestimmtes Objekt spezifizieren.

Dieser Code zur Zentrierung des Formulars ist sehr interessant, weil Sie daran die Beziehung des Formulars zum Bildschirm, die Eigenschaften des `Screen`-Objekts und die `Load`- und `Unload`-Ereignisse gut erkennen. Nichtsdestotrotz sollten Sie beachten, daß das Eigenschaftfenster für das Formular eine spezielle Eigenschaft enthält, `StartPosition`, die Sie auf einen von vier Werten setzen können, wie in Tabelle 11.1 gezeigt.

Tabelle 11.1: Mit den `StartPosition`-Werten bestimmen Sie die Ausgangsposition eines Formulars.

Eigenschaft	Konstantenname	Wert	Beschreibung
0 - Manuell	<code>vbStartUpManual</code>	0	Es wird keine Ausgangsposition für das Formular vorgegeben
1 - Fenstermitte	<code>vbStartUpOwner</code>	1	Zentriert das Formular im Fenster.
2 - Bildschirmmitte	<code>vbStartUpScreen</code>	2	Zentriert das Formular auf dem Bildschirm.
3 - Windows-Standard	<code>vbStartUpWindowsDefault</code>	3	Ausrichtung an der linken oberen Bildschirmecke.

Die Ereignisse für ein Formular sind für eine Applikation wichtig, die dieses Formular benutzt. Anders als die `Load`- und `Unload`-Ereignisse, die Sie bereits kennen, und die mausbezogenen Formularereignisse, die Sie gestern kennengelernt haben, werden Sie jetzt einige der eher formularbezogenen Ereignisse betrachten. Tabelle 11.2 listet drei dieser Ereignisse auf.

Tabelle 11.2: Diese formularbezogenen Ereignisse können Sie in Ihrer Applikation verwenden

Ereignis	Beschreibung
<code>Activate</code>	Tritt auf, wenn das Formular aktiviert wird, weil der Benutzer auf einen nicht überdeckten Abschnitt des Formulars klickt oder aus einer anderen Applikation in das Formular wechselt.
<code>Deactivate</code>	Tritt auf, wenn eine andere Applikation oder ein anderes Formular aktiviert werden.
<code>Resize</code>	Tritt auf, wenn der Benutzer die Größe des Formulars ändert, oder wenn das Programm die Eigenschaften <code>Height</code> und <code>Width</code> mit Hilfe von Anweisungen ändert.



Mit Hilfe der `Resize`-Eigenschaft ordnen Sie die Steuerelemente auf einem Formular neu an, wenn der Benutzer die Formulargröße geändert hat. Falls Sie die Steuerelemente relativ auf dem Formular plazieren (unter Verwendung der Größeneigenschaften des Formulars, die die Steuerelemente anordnen), bleiben diese zentriert, wenn der Benutzer die Formulargröße ändert.

Die Option **Projekt, Eigenschaften** erzeugt das in Abbildung 11.1 gezeigte Dialogfeld. Dort finden Sie auch das Listenfeld **Startobjekt**, wo Sie eines der Formulare in der Applikation oder eine spezielle Prozedur namens `Main` als Startobjekt auswählen. Einige Applikationen haben kein Startformular. Es könnte sich dabei etwa um ein Hintergrund-Utility handeln, das nie ein Formular anzeigt, oder vielleicht einen Datenwert überprüft, um festzustellen, welches Formular angezeigt werden soll. Oder es könnte ein Kennwort abgefragt werden, um das Startformular anzuzeigen.

Abbildung 11.1:
Im Dialogfeld Projekt, Eigenschaften legen Sie das Startformular oder die Startprozedur fest.

Wenn Sie für ein Projekt Code ausführen möchten, bevor ein Formular geladen wird, sollten Sie die Prozedur `Main()` anlegen (in der allgemeinen Prozedur des Codemoduls, nicht in einem Formularmodul) und im **Eigenschaften**-Dialogfeld im Dropdown-Listenfeld **Startobjekt** die Prozedur `Main()` auswählen. Der Code in `Main()` wird ausgeführt, bevor Formulare geladen werden. Falls `Main()` für kein Formular die `Show`-Methode ausführt, wird kein Formular geladen.



Mehr über die formularbezogenen Methoden erfahren Sie in den Kapiteln 13 und 14.

Formular-Auflistungen

Wie bereits erwähnt, wird ein *Formular* häufig auch als *Objekt* bezeichnet. Visual Basic unterstützt mehrere Objektarten - beispielsweise Steuerelemente, Formulare und Objekte, die sich außerhalb Ihrer Applikation befinden, etwa ein OLE-Objekt. (OLE steht für *Object Linking and Embedding*; mehr darüber erfahren Sie in den Kapiteln 16 und 17.)

Alle Ihre Formularobjekte bilden die **FORMS**-Auflistung, die sich jedesmal ändert, wenn Sie Formulare in Ihrem Projekt anlegen oder daraus entfernen. Die **FORMS**-Auflistung enthält die Namen aller Formulare. Der Name eines Ihrer Formulare in der **FORMS**-Auflistung könnte also `frmAboutBox` sein. Ein vordefiniertes Objekt namens `Form` definiert das aktuell geöffnete Formular. Visual Basic verwendet dieses vordefinierte Formularobjekt, um das Standardobjekt zu verwalten, das Sie nicht spezifizieren müssen, wenn Sie Methoden ausführen, ohne den Namen des aktuellen Formulars anzugeben.



Eine **FORMS**-Auflistung ist die Gruppe aller Formulare, die aktuell in Ihrer Applikation definiert sind.

Zugriff auf die FORMS-Auflistung

Bevor Sie auf die **FORMS**-Auflistung zugreifen, sollten Sie wissen, daß Visual Basic Ihnen ermöglicht, auf alle Formulare innerhalb der **FORMS**-Auflistung zuzugreifen, ohne die Namen der einzelnen geöffneten Formulare anzugeben.

Angenommen, Sie haben drei Formulare geöffnet, `frmAcPay`, `frmAcRec` und `frmAcReview`. Das vordefinierte Objekt `Forms` enthält diese drei Formulare. Jedes Formular hat einen Index, beginnend bei Null. Mit Angabe dieses Index können Sie auf das Formular zugreifen, ohne seinen Namen angeben zu müssen. Dieser Index folgt dem **FORMS**-Objekt in Klammern. Tabelle 11.3 zeigt, wie Sie unter Verwendung des Index auf die drei Formulare zugreifen.



Die **FORMS**-Auflistung verhält sich wie die Werte in einem Listenfeld oder die Elemente eines Arrays.

Tabelle 11.3: Mit Hilfe von Indizes greifen Sie auf die Objekte der Forms-Auflistung zu

Formularname	Index-Schreibweise
fmrAcPay	Forms(0)
frmAcRec	Forms(1)
frmAcReview	Forms(2)



Visual Basic bietet noch eine Notation, ein bestimmtes Formular innerhalb einer Auflistung anzusprechen, wenn Sie den Formularnamen kennen. Hier ein Beispiel:

Forms! [frmAcPay]

Dieses Beispiel greift auf das Formular `frmAcPay` innerhalb der aktuellen Auflistung zu. (Anders als in Anweisungen müssen die eckigen Klammern hier angegeben werden.) Sie können auch Klammern verwenden, um auf ein Formular einer Auflistung zuzugreifen:

Forms! ("frmAcPay")

Die Indizes

Sie können auch auf einzelne Steuerelemente auf Ihren Formularen zugreifen, indem Sie statt eines Namens einen Index angeben. Auf diese Weise durchlaufen Ihre Programme alle Felder eines Formulars, ohne auf die einzelnen Namen zugreifen zu müssen.

Angenommen, Sie haben das Formular `frmStore` mit fünf Steuerelementen: drei Bezeichnungsfeldern (`lblStoreNum`, `lblStoreName` und `lblStoreLoc`) und zwei Listenfeldern (`lstStoreEmps` und `lstStoreMgrs`). Ihre Visual-Basic-Prozeduren greifen über einen Index auf diese Steuerelemente zu, beginnend bei 0, wie in Tabelle 11.4 gezeigt. Beachten Sie, daß Sie den Steuerelementnamen durch ein Ausrufezeichen von der FORMS-Auflistung abtrennen, um ein einzelnes Steuerelement innerhalb der Auflistung anzusprechen.

Tabelle 11.4: Mit Hilfe von Indizes greifen Sie auf die Steuerelemente auf einem Formular zu.

Steuerelement	Indexnotation
lblStoreNum	Forms!frmStore(0)
lblStoreName	Forms!frmStore(1)
lblStoreLoc	Forms!frmStore(2)
lstStoreEmps	Forms!frmStore(3)
lstStoreMgrs	Forms!frmStore(4)



Verwechseln Sie die Indizes für spezifische Formularnamen nicht mit denen in einer FORMS-Auflistung. Wenn dem Index FORMS vorausgeht, bezieht er sich auf ein bestimmtes Formular. Wenn der Formularnamen vor dem Index erscheint, wie in Visual Basic 11.4 gezeigt, verweist der Index auf ein Steuerelement auf diesem Formular.

Die Count-Eigenschaft

Auflistungen haben ebenso wie Steuerelemente Eigenschaften Die Count-Eigenschaft steht Ihnen bei der Arbeit mit Forms-Auflistungen zur Verfügung. Sie vereinfacht Ihnen die Programmierung, so daß Sie nichts darüber wissen müssen, wie viele einzelne Formulare sich in der Auflistung befinden. Mit Hilfe von Count schreiben Sie allgemeine Prozeduren, die für alle aktuell geöffneten Formulare ausgeführt werden. Die Count-Eigenschaft enthält immer einen Integer.



Count enthält die Anzahl der Steuerelemente eines Formulars, wenn Sie diese Eigenschaft auf einen bestimmten Formularnamen anwenden (beispielsweise enthält `frmAcPay.Count` die Anzahl der Steuerelemente auf dem Formular `frmAcPay`). Count berücksichtigt dabei sowohl die verborgenen als auch die sichtbaren Steuerelemente.

Der folgende Code deklariert eine Integervariable, `intC`, und speichert die Anzahl der geöffneten Formulare in `intC`:

```
Dim intC As Integer
```

```
intC = Forms.Count ' Anzahl geöffneter Formulare speichern
```

Wenn Sie die Anzahl der Steuerelemente auf einem Formular brauchen, verwenden Sie ebenfalls die `Count`-Eigenschaft. Der folgende Code deklariert eine Integer-Variable, `intCC`, und speichert die Anzahl der Steuerelemente des Formulars `frmMyForm` in `intCC`:

```
Dim intCC As Integer
intCC = frmMyForm.Count ' Anzahl Steuerelemente auf dem Formular speichern
```



`Count` nach einem Formularnamen bezieht sich auf die Anzahl der Steuerelemente auf diesem Formular. `Count` nach der Forms-Auflistung bezieht sich auf die Anzahl der Formulare in diesem Projekt.

Eine `For`-Schleife ist perfekt dazu geeignet, alle im aktuellen Projekt geöffneten Formulare zu durchlaufen. Achten Sie darauf, daß die Schleife mit dem Anfangswert Null initialisiert werden muß, wie hier gezeigt, weil das erste Formular den Indexwert 0 hat.

Der folgende Code durchläuft alle geöffneten Formulare und verbirgt sie:

```
For intI = 0 To Forms.Count - 1
Forms(intI).Visible = False ' Formular verbergen
Next intI
```

Vielleicht möchten Sie alle Formulare verbergen, wenn Sie eine Systemaufgabe ausführen, für die kein I/O durch den Benutzer erforderlich ist. Nachdem die Arbeit erledigt ist, wiederholen Sie die Schleife und setzen die `Visible`-Eigenschaft aller Formulare wieder auf `True`.

Visual Basic unterstützt eine spezielle `For`-Schleife, `For Each`. Diese Schleife durchläuft eine Auflistung, ohne daß Sie dafür eine Steuervariable angeben müßten. Der folgende Code verbirgt ebenfalls alle geöffneten Formulare:

```
Dim varFrmObj As Variant
For Each varFrmObj In Forms
varFrmObj.Visible = False ' Formular verbergen
Next
```

Die einzige Schleifenvariable, die Sie für `For Each` deklarieren müssen, ist eine `Variant`-Variable, die den Namen der einzelnen Formulare aufnimmt, während die Schleife die Auflistung durchläuft. Auf dieselbe Weise richtet die erste Anweisung der folgenden `For Each`-Schleife eine Schleife ein, die jedes Steuerelement im Formular `frmMyForm` durchläuft, unabhängig davon, wie viele Steuerelemente die Auflistung enthält:

```
For Each varControl In frmMyForm
```

Formulare aus dem Speicher entfernen

Wie bereits im Abschnitt »Eigenschaften, Ereignisse und Methoden« beschrieben, gibt es für Formulare einen speziellen Verwendungszweck für die Funktion `Form_Unload()`. Das `Unload`-Ereignis kann für Aufräumroutinen ganz praktisch sein, beispielsweise um alle Daten auf der Festplatte zu speichern, bevor die Applikation beendet wird; manchmal ist es aber auch unbedingt notwendig, um ein Programm sauber zu beenden.

Angenommen, Ihre Applikation enthält mehrere Formulare. Wenn sie ein Formular verbirgt, kann der Benutzer nicht erkennen, daß es noch geladen ist. Wenn er das Fenster der Hauptanwendung schließt, bleibt dieses eine Formular weiterhin geladen und Windows glaubt, das Programm würde immer noch ausgeführt. Das Programm belegt also immer noch einen Teil Ihres Speichers, auch wenn der Benutzer glaubt, alle Formulare seien geschlossen.

Viele Visual-Basic-Programmierer fügen für das Beenden einer Applikation (beispielsweise in der Menüoption **Datei, Beenden**) immer den folgenden Code ein:

```
For intCtr = (Forms.Count - 1) to 0 Step - 1
Unload Forms(intCtr) ' Verborgene und angezeigte Formulare entfernen
Next intCtr
```

Text auf Formularen

Statt eines Bezeichnungsfelds oder eines Textfelds können Sie auch die `Print`-Methode verwenden, um Text auf einem Formular auszugeben. Einer der Nachteile von `Print` ist, daß Sie zusätzlichen Code brauchen, damit `Print` sich so verhält, wie Sie das wollen. `Print` ist kein Steuerelement, Sie können also nicht einfach zur Entwurfszeit irgendwelche Eigenschaften dafür setzen; statt dessen bestimmt Ihr Programm im Code, wie `Print` sich verhalten soll. Beispielsweise müssen Sie Code schreiben, der bewirkt, daß `Print` seine Ausgaben an einer bestimmten Stelle im Formular vornimmt, falls Sie sicherstellen möchten, daß es keine anderen Steuerelemente überschreibt.



Mit `Print` lernen Sie nicht nur, wie Sie Text an Formulare senden, sondern auch, wie man Text an den Drucker schickt. Wie Sie in Kapitel 13 erfahren werden, ist `Print` die wichtigste Methode, um direkte Ausgaben an den Drucker zu schicken.

Mit `Print` senden Sie Text an ein Objekt. In diesem Kapitel handelt es sich bei diesem Objekt um das Formular für die Applikation. `Print` wird auf Formulare, Bildfelder, das Drucker-Objekt und das Debug-Objekt (ein spezielles Fenster, an das Sie während der Programmausführung Testwerte senden) ausgeführt. Am einfachsten ist es jedoch, mit `Print` Ausgaben auf einem Formular vorzunehmen.

Jetzt sollen Sie die Verwendung von `Print` ein bißchen üben, ohne eine vollständige Applikation zu entwickeln. Öffnen Sie ein neues Projekt und doppelklicken Sie auf das Fenster `Form1`, um das Codefenster zu öffnen. Weil Sie das Projekt gerade erst geöffnet haben, stehen Ihnen noch keine anderen Objekte zur Verfügung als `Form1` und der allgemeine Deklarationsbereich. Alle vordefinierten Prozeduren, die das Formular erkennt, werden in einem Drop-down-Listenfeld angezeigt. Wählen Sie aus der Liste den Eintrag `Click` und geben Sie den in Listing 11.1 gezeigten Code ein (Visual Basic stellt die Hüllzeilen für Sie bereit, so daß Sie nur die Anweisungen dazwischen eingeben müssen).

Listing 11.1: Die `Print`-Anweisung schreibt direkt auf das Formular

```
1: Private Sub Form_Click()  
2: ' Demonstriert, wie man auf ein Formular schreibt  
3: Dim strString As String  
4: strString = "Visual Basic"  
5: ' Den String dreimal ausgeben  
6: Form1.Print strString & " " & strString & " " strString  
7: End Sub
```

Bei der Programmausführung erscheint das Formular, aber es passiert nichts, bevor Sie nicht auf das Formular klicken. Sie haben Ihren Code in die Routine `Form_Click()` eingegeben - es soll also gar nichts passieren, bis Sie nicht auf das Formular geklickt haben. Klicken Sie mehrere Male auf das Formular. Ihr Formular sollte ähnlich wie in Abbildung 11.2 gezeigt aussehen.

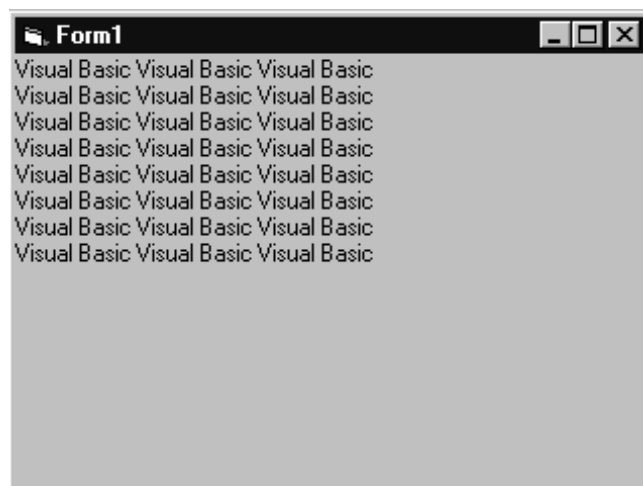


Abbildung 11.2:
Die `Print`-Methode sendet Ausgaben direkt an das Formular.

`Print` ist eine der einfachsten Methoden, mit der Sie Informationen aus Ihrem Programm auf das Formular ausgeben. Dazu geben Sie einfach nur den Formularnamen und die `Print`-Methode an, durch einen Punkt voneinander getrennt. Hier das Format der `Print`-Methode, wenn Sie sie auf ein Formular anwenden:

```
frmFormName.Print ZuDruckendeDaten
```

`frmFormName` ist das Formular, auf dem Sie Ausgaben anzeigen möchten, und `ZuDruckendeDaten` sind die Daten, die ausgegeben werden sollen. Mögliche Ausgaben sind Konstanten (numerische, Strings und Datumswerte), Variablenwerte und Steuerelemente.

Formatieren mit `Print`

Die Ausgaben der `Print`-Methode können mit den Funktionen `Spc()` oder `Tab()` formatiert werden. Beide werden innerhalb der `Print`-Methode ausgeführt, um Abstände in die auszugebenden Daten einzufügen.

Betrachten Sie Listing 11.2. Dort werden `Spc()` und das Semikolon (;) eingesetzt, um zwei Strings in derselben Zeile auszugeben. `Spc(5)` weist die `Print`-Methode an, fünf Leerzeichen zu überspringen, so daß die Textausgabe in der sechsten Spalte beginnt. Wenn Sie eine `Print`-Anweisung mit einem Semikolon abschließen, wird die nächste `Print`-Anweisung dort fortgesetzt, wo die letzte ihre Ausgabe beendet hat, und nicht in der nächsten Zeile, wie es ohne Semikolon der Fall wäre.

Listing 11.2: Ein Carriage Return kann unterdrückt werden, so daß zwei `Print`-Anweisungen ihre Ausgaben in dieselbe Zeile schreiben

```
1: Private Sub Form_Click()  
2: Dim strString As String  
3: strString = "Visual Basic"
```

```

4: Form1.Print "*"; Spc(5); strString; ' Beachten Sie das Semikolon
5: Form1.Print Spc(2); strString
6: End Sub

```

Wenn Sie mehrere Male auf das Formular klicken, um den Code auszuführen, erscheint die folgende Ausgabe:

```

* Visual Basic Visual Basic
* Visual Basic Visual Basic
* Visual Basic Visual Basic

```

Der Code erzwingt, daß die `Print`-Methode fünf Leerzeichen überspringt, bevor der Text `Visual Basic` ausgegeben wird. Nach zwei weiteren Leerzeichen gibt die zweite `Print`-Anweisung ebenfalls `Visual Basic` aus. Wenn Sie zum nächsten Mal klicken, wird die Ereignisprozedur erneut ausgeführt und die Ausgabe wiederholt.

Wenn Sie statt `Spc()` die Funktion `Tab()` verwenden, springt Visual Basic an die Position des in Klammern angegebenen Spaltenarguments und gibt die nächsten Daten dort aus. `Spc()` erzwingt, daß das nächste `Print` eine bestimmte Anzahl von Leerzeichen überspringt, während `Tab()` festlegt, daß die Ausgabe in einer bestimmten Spalte beginnen soll. Listing 11.3 zeigt ein Beispiel.

Listing 11.3: Mit den Funktionen `Tab()` und `Spc()` werden die Abstände festgelegt, die bei der `Print`-Methode verwendet werden

```

1: Private Sub Form_Click()
2: Dim strString As String
3: strString = "Visual Basic"
4:
5: Form1.Print "*"; Tab(5); strString; Tab(20); strString
6: Form1.Print "*"; Spc(5); strString; Spc(20); strString
7: End Sub

```

Das `Tab()` in Zeile 5 sorgt dafür, daß die Ausgabe in einer bestimmten Spalte erfolgt, während das `Spc()` in Zeile 6 festlegt, daß eine bestimmte Anzahl an Leerzeichen übersprungen wird.

```

* Visual Basic Visual Basic
* Visual Basic Visual Basic

```

Mit Hilfe der `Print`-Methode können Sie auch leere Zeilen auf einem Formular erzeugen, ohne dort Daten anzeigen zu müssen. Betrachten Sie den Code in Listing 11.4.

Listing 11.4: Mit `Print` erzeugen Sie leere Zeilen auf dem Formular

```

1: Private Sub Form_Click()
2: Dim strString As String
3: Dim CurLine As Integer
4:
5: CurLine = 1
6: strString = "Visual Basic"
7:
8: ' Zeile ausgeben
9: Form1.Print strString & " steht in Zeile #" & CurLine
10:
11: For CurLine = 2 To 6
12: Form1.Print ' Leere Zeilen ausgeben!
13: Next CurLine
14:
15: ' Jetzt eine Zeile ausgeben
16: Form1.Print strString & " steht in Zeile #" & CurLine
17: End Sub

```

Die Ausgabe zeigt fünf Leerzeilen zwischen den Ausgaben an:

```
Visual Basic steht in Zeile 1
```

```
Visual Basic steht in Zeile 7
```

Die Zeilen 11 bis 13 geben Leerzeilen aus, weil die `Print`-Methode keine Ausgaben dafür spezifiziert.

Positionierung der `Print`-Methode

Häufig möchte man ganz genau festlegen, wo eine Ausgabe auf dem Formular stattfinden soll. Visual Basic unterstützt mehrere formularbezogene Eigenschaften, die Sie für `Print` einsetzen können, um Text auf einem Formular auszugeben. Diese Eigenschaften verwenden die aktuelle Position des Textcursors, den Visual Basic bei der Ausführung der `Print`-Methode entsprechend mitbewegt.

Informationen über den Textcursor sind in den Eigenschaften `CurrentX` und `CurrentY` enthalten. Mit Hilfe dieser Eigenschaften bestimmen Sie exakt, wo die Ausgabe erscheinen soll.

Eine weitere Eigenschaft, `ScaleMode`, bestimmt, wie sich `CurrentX` und `CurrentY` verhalten. Ein Formular kann verschiedene Modi erkennen, abhängig davon, welche `ScaleMode` -Eigenschaft Sie gesetzt haben. Dieser Modus bestimmt die Skalierung für die Ausgabe von Grafik und Text auf einem Formular. Tabelle 11.5 zeigt die anderen `ScaleMode`-Werte, die Sie setzen könnten. Viele davon sind grafikbezogen. In Kapitel 13 werden Sie viele dieser Werte genauer kennenlernen.

Tabelle 11.5: Die `ScaleMode`-Werte bestimmen die Koordinaten der `Print`-Methode

Konstantenname	Wert	Skalierung
<code>vbUser</code>	0	Der Programmierer legt die Skalierung fest.
<code>vbTwips</code>	1	Dies ist der Standardwert für <code>ScaleMode</code> , bis Sie etwas anderes angeben. Gibt eine Skalierung in Twips an.
<code>vbPoints</code>	2	Ein Punkt (1/72 Inch).
<code>vbPixels</code>	3	Der kleinste Punkt auf dem Objekt (für einen Bildschirm bestimmt dessen Punktgröße die Pixelgröße).
<code>vbCharacters</code>	4	Die Größe eines Zeichens.
<code>vbInches</code>	5	Ein Inch.
<code>vbMillimeters</code>	6	Ein Millimeter.
<code>vbCentimeters</code>	7	Ein Zentimeter.
<code>vbHimetric</code>	8	Statt Pixel verwenden einige Programmierer das <i>Himetric</i> -Maß, das geräteunabhängig ist, so daß Windows die Koordinaten in die zur Laufzeit höchstmögliche Auflösung umsetzt.
<code>vbContainerPosition</code>	9	Der <code>ScaleMode</code> -Wert des Containers für das Steuerelement (das ist ein Objekt, das das aktuelle Objekt enthält) legt den <code>ScaleMode</code> -Wert des aktuellen Objekts fest, wenn er als Positionswert verwendet wird.
<code>vbContainerSize</code>	10	Der <code>ScaleMode</code> -Wert des Containers, in dem sich das Steuerelement befindet, legt den <code>ScaleMode</code> -Wert des aktuellen Objekts fest, wenn er als Größenwert verwendet wird.

Die gebräuchlichste `ScaleMode`-Eigenschaft für Text ist der Zeichenmodus. Das bedeutet, wenn `CurrentX` und `CurrentY` auf 5 gesetzt sind, beginnt die nächste `Print`-Methode ihre Ausgabe in Spalte 5, Zeile 5. Die Startposition der `ScaleMode`-Eigenschaft ist die obere linke Formularecke (mit den Koordinaten 0,0). Die in Listing 11.5 gezeigte `Click`-Ereignisprozedur verwendet die Eigenschaften `ScaleMode`, `CurrentX` und `CurrentY` .

Listing 11.5: Die Ausgabe wird mit den Eigenschaften `CurrentX` und `CurrentY` gesteuert

```
1: Private Sub Form_Click()  
2: ' Für Zeichen einrichten  
3: Form1.ScaleMode = vbCharacters ' Zeichen (4)  
4:  
5: Form1.CurrentX = 20 ' 20 Zeichen weiter  
6: Form1.CurrentY = 6 ' 6 Zeilen nach unten  
7: Form1.Print "Diagonal nach unten"  
8:  
9: Form1.CurrentX = 0 ' Zurück nach links  
10: Form1.CurrentY = 0 ' Nach oben  
11: Form1.Print "Oben links"  
12: End Sub
```

Zeile 3 setzt den `ScaleMode`-Wert auf den Konstantenwert `vbCharacters`, der die Zeichen-Skalierung festlegt. Betrachten Sie die Ausgabe in Abbildung 11.3. Die Ausgabe der zweiten `Print`-Methode erscheint weiter oben auf dem Formular als die Ausgabe der ersten `Print`-Methode. Das wurde durch die Koordinatenpositionierung realisiert.

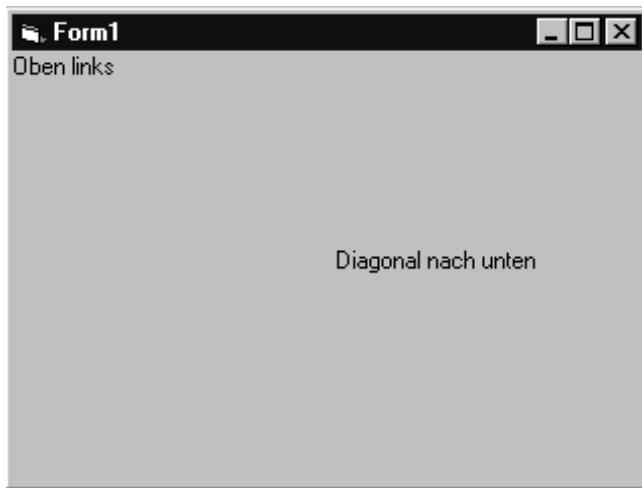


Abbildung 11.3:
Mit **CurrentX** und **CurrentY** positionieren Sie den Textcursor der **Print-Methode**.

Neue Eigenschaften für Formulare

Es ist möglich, benutzerdefinierte Eigenschaften für ein Formular zu entwickeln. Sie brauchen zwar nicht für jede Ihrer Applikationen benutzerdefinierte Formulareigenschaften, aber manchmal werden Sie feststellen, daß Sie an Formularen immer wieder dieselben Anpassungen vornehmen, die irgendwann zu eigenen Formulareigenschaften führen.

Angenommen, Sie möchten unten im Formular immer einen Titel anzeigen. Wenn Sie ein Formular mit einem Bezeichnungsfeld erzeugen, müssen Sie nicht nur das Bezeichnungsfeld verwalten, sondern dieses auch ändern, wenn sich der Titel ändern soll. Würde es sich bei dem Titel um eine Formulareigenschaft handeln, könnten Sie auf das Bezeichnungsfeld verzichten, weil der Titel immer mit der entsprechenden Formulareigenschaft gesetzt wird.

Am einfachsten lernen Sie, neue Formulareigenschaften anzulegen, indem Sie ein einfaches Beispiel betrachten. Legen Sie ein neues Projekt an. Verwenden Sie dazu die in Tabelle 11.6 aufgelisteten Steuerelemente und Eigenschaften. Diese Applikation demonstriert, wie die **ScaleMode**-Werte die Koordinaten der **Print-Methode** festlegen.

Tabelle 11.6: Die Koordinaten der **Print-Methode werden mit Hilfe von **ScaleMode**-Werten geändert**

Eigenschaft	Wert
Formular Name	frmTitle
Formular Caption	Titelleiste unten
Formular Height	3720
Formular StartUpPosition	2 - Bildschirmmitte
Formular Width	3975
Befehlsschaltfläche Name	cmdTitle
Befehlsschaltfläche Caption	&Titel anzeigen
Befehlsschaltfläche Height	495
Befehlsschaltfläche Left	1320
Befehlsschaltfläche Top	1200
Befehlsschaltfläche Width	1215

Wenn der Benutzer auf die Schaltfläche klickt, erscheint unten auf dem Formular ein Titel. Dazu wird die **Click**-Ereignisprozedur der Schaltfläche ausgeführt:

```
1: Private Sub cmdTitle_Click()  
2: frmTitle.BottomTitle = "Eine neue Eigenschaft"  
3: End Sub
```

Betrachten Sie Zeile 2 genauer. Irgend etwas scheint hier falsch zu sein: Es gibt keine Eigenschaft namens **BottomTitle**. Nach den nächsten paar Schritten wird es sie geben! Gehen Sie wie folgt vor, um die neue Eigenschaft **BottomTitle** in die Eigenschaftenliste Ihres Formulars aufzunehmen:

1. Klicken Sie im Projektfenster auf die Schaltfläche **Code anzeigen**, um das Codefenster anzuzeigen.

2. Geben Sie in den Deklarationsbereich vor der Ereignisprozedur `cmdTitle_Click()` die folgende Zeile ein:

```
Dim strTitle As String
```

Damit wissen Sie, daß `strTitle` eine für das gesamte Projekt öffentliche Variable ist. Öffentliche Variablen sind verpönt, warum deklarieren wir also `strTitle` hier als öffentlich? Sie wissen, daß alle Steuerelemente für die gesamte Applikation öffentlich sind. Es gibt kein privates Steuerelement und keine private Eigenschaft. Die Variable `strTitle` nimmt den Wert einer neuen Formulareigenschaft auf, die Sie im nächsten Schritt anlegen werden. Die `BottomTitle`-Eigenschaft braucht stets Zugriff auf ihre Platzhaltervariable, `strTitle`, also muß diese als öffentliche Variable deklariert werden.

3. Jetzt fügen Sie Ihrem Projekt eine neue Prozedurform hinzu: eine `Property Get`-Prozedur. Es gibt für alle von Ihnen definierten Eigenschaften eine `Property Get`-Prozedur. Visual Basic führt die `Property Get`-Prozedur automatisch aus, wenn das Programm auf die entsprechende neue Eigenschaft zugreift. Mit anderen Worten, sobald Sie eine `Property Get`-Prozedur für die `BottomTitle`-Eigenschaft erstellt haben, führt Visual Basic diese automatisch aus, wenn die Applikation den Wert von `BottomTitle` abfragt, um den aktuellen Wert der Eigenschaft zurückzugeben.



Eine `Property Get`-Prozedur gibt den Wert einer definierten Eigenschaft zurück.

Geben Sie den Code aus Listing 11.6 in das Codefenster ein.

Listing 11.6: Die `Property Get`-Prozedur gibt den Wert der Eigenschaft zurück

```
1: Public Property Get BottomTitle()  
2: ' Diese Applikation gibt den Wert der  
3: ' BottomTitle-Eigenschaft zurück, der  
4: ' eigentlich in der Variablen  
5: ' public strTitle verwaltet wird  
6: BottomTitle = strTitle  
7: End Property
```

Beachten Sie, daß Formulare die Eigenschaft `BottomTitle` nicht als Standard besitzen; Sie definieren diese Eigenschaft. Der Wert der `BottomTitle`-Eigenschaft ist in der öffentlichen Variablen `strTitle` abgelegt. `Property Get` sorgt jedoch dafür, daß die restliche Applikation glaubt, es handle sich bei `BottomTitle` wirklich um eine Eigenschaft. Tatsächlich aber stammt der Eigenschaftswert aus einer öffentlichen Variablen.

4. Jetzt brauchen Sie noch eine `Property Let`-Prozedur für die neue `BottomTitle`-Eigenschaft. Die `Property Let`-Prozedur ändert den Wert der `BottomTitle`-Eigenschaft. In Wirklichkeit ändert `Property Let` jedoch den Wert der öffentlichen Variablen, die als Platzhalter für die `BottomTitle`-Eigenschaft dient.



Eine `Property Let`-Prozedur weist der von Ihnen definierten Eigenschaft einen Wert zu.

Geben Sie den in Listing 11.7 gezeigten Code in das Codefenster ein.

Listing 11.7: Die `Property Let`-Prozedur setzt den Wert der Eigenschaft

```
1: Public Property Let BottomTitle(strTitleEntered)  
2: ' Diese Prozedur weist der Variable strTitle  
3: ' den Eigenschaftswert zu, den das Programm für  
4: ' BottomTitle verwendet  
5: '  
6: ' Das übergebene Argument stellt den Wert dar, den  
7: ' das Programm für BottomTitle ablegt  
8: strTitle = strTitleEntered  
9: '  
10: ' Die nächste Zeile soll unten im Formular ausgegeben werden  
11: frmTitle.CurrentY = (frmTitle.Height - 600)  
12: '  
13: ' Falls das Formular zu klein ist, um die Zeichen auszugeben,  
14: ' soll einfach nichts gemacht werden  
15: If frmTitle.CurrentY < 600 Then  
16: Exit Property  
17: Else  
18: ' Eigenschaftswert auf dem Formular ausgeben  
19: Print strTitle  
20: End If
```

21: End Property

Zeile 8 weist der öffentlichen Variablen den Wert zu, der der Prozedur übergeben wurde. Egal, wo die Applikation der BottomTitle-Eigenschaft einen Wert zuweist, wird diese Prozedur ausgeführt, wobei dieser Wert als Argument übergeben wird.

Die Zeilen 11 und 15 zeigen den Titel an. Zeile 11 stellt sicher, daß der Titel 600 Twips oberhalb der unteren Fensterkante angezeigt wird, egal wie der Benutzer die Formulargröße geändert hat. Falls der Benutzer das Fenster zu klein gemacht hat, so daß nicht einmal mehr eine Textzeile angezeigt werden kann, verläßt Zeile 16 die Prozedur, ohne daß irgend etwas gemacht wurde. So lange der Titel auf das Formular paßt, gibt Zeile 19 den Wert an der Position CurrentY aus. (CurrentX ist 0, die linke Fensterkante. Der Code ändert CurrentX nicht.)



Nachdem Sie die Property Get- und Property Let-Prozeduren für die BottomTitle -Eigenschaft eingegeben haben, wird diese Teil der Eigenschaftenliste für das Formular. Das Eigenschaftenfenster zeigt diese neue Eigenschaft zwar nicht an, aber alle anderen Eigenschaftslisten werden aktualisiert. Abbildung 11.4 beispielsweise zeigt, was passiert, wenn Sie einer der Formular-Eigenschaften einen Wert zuweisen möchten. Das Popup-Menü mit den Eigenschaften listet die BottomTitle-Eigenschaft mit auf.

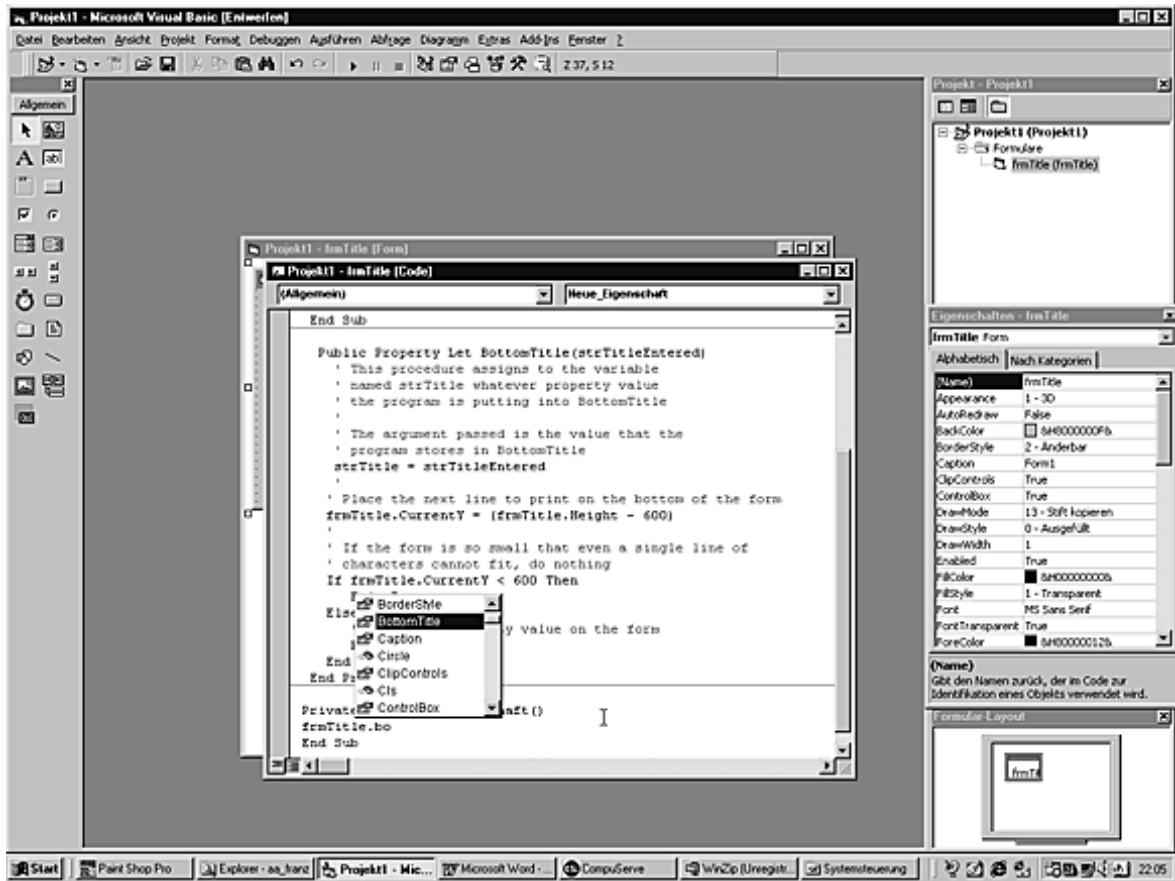


Abbildung 11.4:
Der Eigenschaftswert für BottomTitle erscheint in der Eigenschaftenliste, die bei der Codeeingabe angezeigt wird.

5. Wenn Sie die Applikation ausführen und auf die Schaltfläche klicken, tritt das Click-Ereignis für die Schaltfläche auf und die zuvor eingegebene Click-Ereignisprozedur wird ausgeführt. Die Click-Ereignisprozedur weist der Formulareigenschaft BottomTitle den Wert »Eine neue Eigenschaft« zu. Die Property-Prozeduren übernehmen den Rest.

Mehrere Formulare in einem Projekt

Bisher enthielten die meisten Ihrer Applikationen nur ein einziges Formular. Es wurde noch nicht viel über Applikation mit mehreren Formularen gesagt, weil das für einfache Applikationen nicht erforderlich war. Wenn Sie ein zusätzliches Formular brauchen, klicken Sie mit der rechten Maustaste auf das Projektfenster und fügen eines hinzu.

Jetzt werden Sie lernen, wie Sie Ihrer Applikation zusätzliche Formulare hinzufügen. Diese Formular sind spezielle Datenformulare. Deshalb sollten Sie wissen, wie man SDI- (Single Document Interface) und MDI-Applikationen (Multiple Document Interface) unterscheidet und programmiert.

Bei der Verwendung mehrerer Formulare ist es in der Regel erforderlich, zwei oder mehr Formulare zu verwalten. Obwohl Sie jedem Formular einen sprechenden Namen zuweisen, sollte Ihr Programmieraufwand nicht ins Unermeßliche wachsen. Der Formularename bestimmt, welche Steuerelemente Sie jeweils bearbeiten. Außerdem können Sie die Formulare während der Ausführung nach Bedarf anzeigen oder verbergen (und auch die Benutzer können zwischen den Formularen wechseln).



Auch wenn zwei oder mehrere Formulare auf dem Bildschirm des Benutzers angezeigt werden, kann nur jeweils eines davon aktiv sein. Die Benutzer aktivieren ein nicht aktives Formular, indem sie darauf klicken. Auch Ihr Applikationscode kann ein Formular aktivieren. Die Methode `frmForm.Show` aktiviert das Formularfenster mit dem Namen `frmForm`. Dieses Formular überdeckt alle anderen Fenster.

Die MDI-Entwicklung erlaubt, sehr komplex aussehende Applikationen zu erstellen. Fast alle größeren Softwarepakete sind MDI-Applikationen. In diesem Buch werden Sie leistungsfähigere Programme kennenlernen, die Dateien und zusätzliche Steuerelemente verwenden. Dort werden Sie erkennen, wie wichtig MDI-Applikationen sind.

Ein Formular kann auch im **Projekt**-Menü mit dem Eintrag **Formular hinzufügen** eingefügt werden. Visual Basic zeigt das Dialogfeld **Formular hinzufügen** an, das Sie in Abbildung 11.5 sehen. Dort wählen Sie einen bestimmten Formulartyp aus. Auf der Registerkarte **Vorhanden** wählen Sie ein bereits existierendes Formular aus.

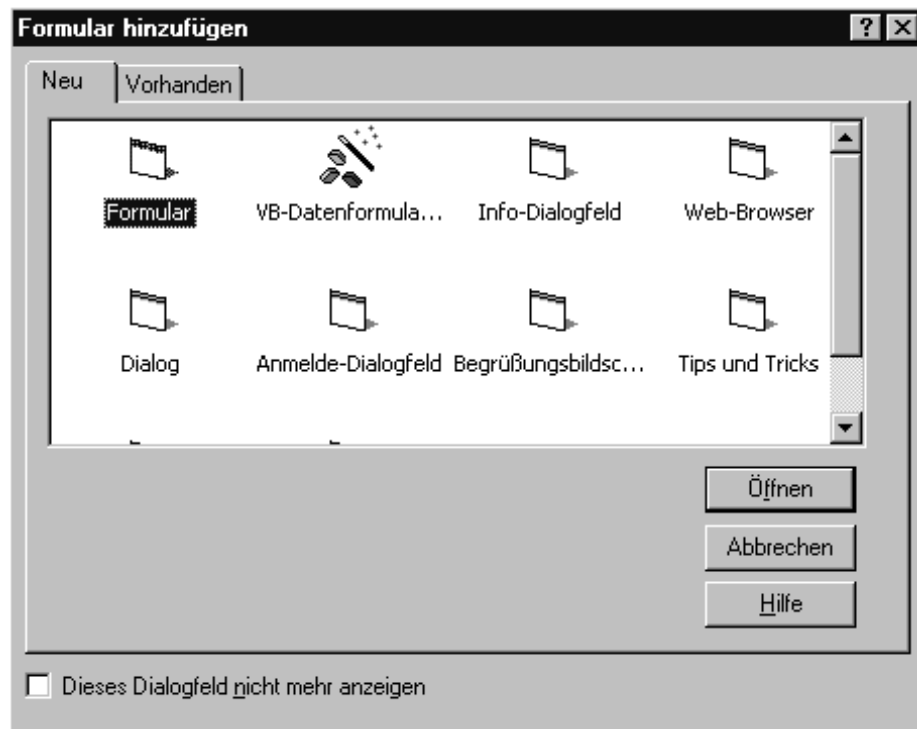


Abbildung 11.5:
Wählen Sie aus, welchen Formulartyp Sie hinzufügen möchten.

Um ein Standardformular hinzuzufügen, doppelklicken Sie auf **Formular**. Visual Basic öffnet ein neues Formular. Es weist den hinzugefügten Formularen sequentiell die Namen **Form2**, **Form3** usw. zu. Sie sollten den Formularen jedoch sofort sprechende Namen zuweisen.



Kapitel 15 erklärt, wie die anderen Formulartypen verwendet werden, die im Dialogfeld **Formular hinzufügen** angeboten werden.

SDI und MDI

Visual Basic unterstützt die folgenden Oberflächenstile:

- **SDI.** Eine SDI-Applikation enthält nur ein einzelnes Datenfenster. Beispielsweise können Sie in Windows Wordpad immer nur ein einzelnes Dokument gleichzeitig öffnen. Eine SDI-Applikation enthält in der Regel kein **Fenster**-Menü in der Menüleiste, weil es nicht möglich ist, zwischen Datenfenstern zu wechseln. Wenn der Benutzer eine neue Datendatei öffnet, ersetzen diese Daten die aktuell im geöffneten Fenster angezeigten Daten.
- **MDI.** Eine MDI-Applikation enthält mehrere Datenfenster (manchmal auch als *Dokumentfenster* bezeichnet). Microsoft Word beispielsweise erlaubt, beliebig viele Fenster zu öffnen. Das MDI von Word stellt sicher, daß jedes Dokument in einem eigenen Fenster angezeigt wird (siehe Abbildung 11.6). Sie wechseln zwischen den Fenstern, indem Sie das Zielfenster mit der Maus anklicken oder ein Fenster im **Fenster**-Menü auswählen. Wenn die Benutzer zwischen den Dokumentfenstern wechseln (das sind aus der Perspektive des Entwicklers Formulare), wird das ausgewählte Fenster zum aktiven Formular, das den Fokus besitzt.

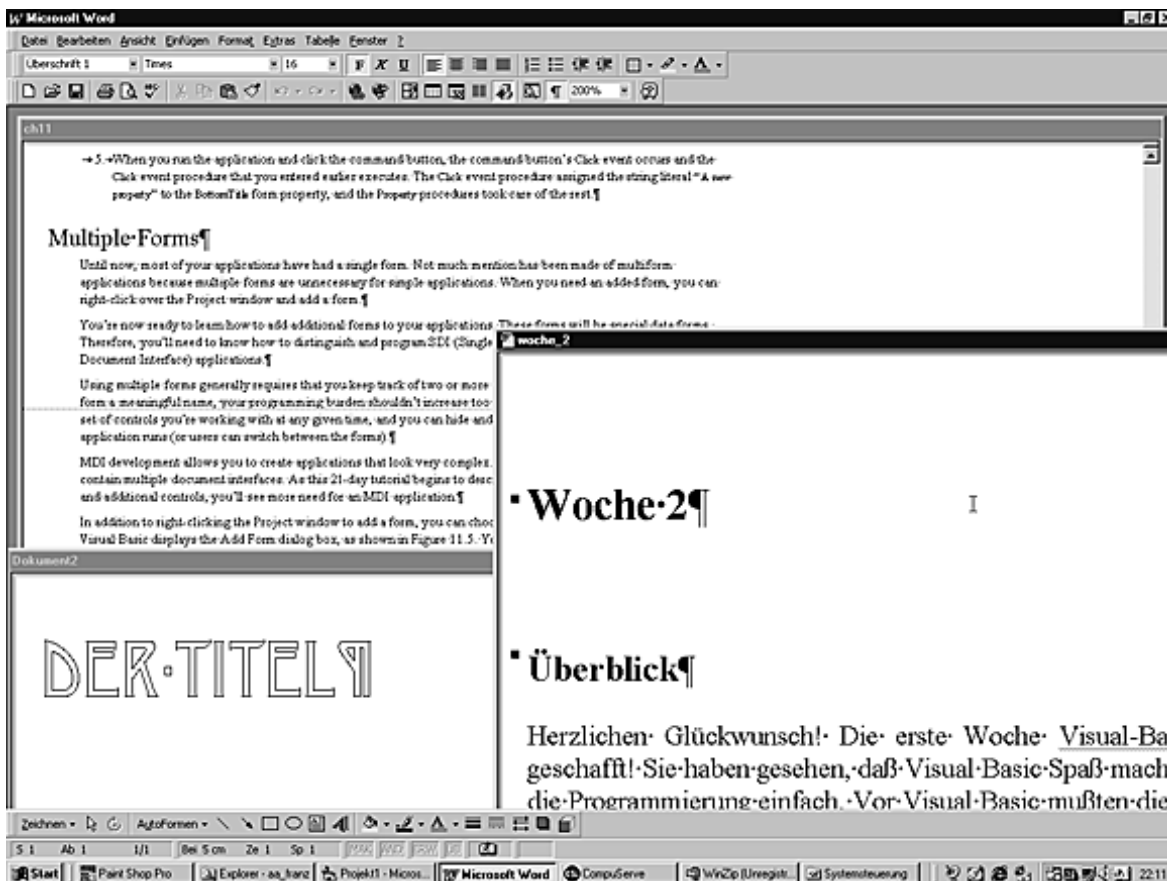


Abbildung 11.6:

MDI-Prozeduren ermöglichen die Verwendung mehrerer Datenfenster.

- **Explorer-Stil.** Dieser Stil wird im Hilfesystem von Visual Basic und auf Bildschirmen des Windows-Explorers verwendet. Eine Applikation, die den Explorer-Stil unterstützt, zeigt zwei Fenster an: eines links und eines rechts. Das linke Fenster enthält eine hierarchische Darstellung der im rechten Fenster detailliert angezeigten Daten. Verwenden Sie den Explorer-Stil, wenn Sie mit einer Applikation arbeiten, die Datendateien und Grafiken verwaltet. Diese Applikationen könnten beispielsweise die Steuerelemente Baumansicht (TreeView) und Listenansicht (ListView) enthalten, die Ihnen helfen, sich in den Fenstern zu bewegen. (Diese Steuerelemente sind normalerweise nicht in der Werkzeugsammlung enthalten, Sie können sie aber über die Menüoption **Projekt, Komponenten** einfügen.)



Eine SDI-Applikation kann ebenfalls mehrere Formulare beinhalten. MDI bedeutet einfach nur, daß Ihre Applikation ein oder mehrere Kindformulare enthalten kann, die Datenmengen enthalten, die sich von anderen Datenmengen innerhalb der Applikation unterscheiden. Eine MDI-Applikation verwendet ein *Steuerformular* (auch als *Elternformular*, *übergeordnetes Formular* oder *primäres Formular* bezeichnet), das die anderen Formulare aufnimmt, und die anderen Formulare können nicht außerhalb der Grenzen des Steuerformulars erscheinen. Wenn Sie eine SDI-Applikation anlegen - was in der Regel der Fall ist -, können Ihre Applikationen zwar mehrere Formulare enthalten, aber keines dieser Formulare wird als unter- oder übergeordnetes Formular betrachtet. Falls Ihre Applikation nur jeweils eine Datenmenge verarbeitet (beispielsweise eine Kundendatei oder eine Angestelltendatei), oder wenn sie überhaupt nicht mit Daten arbeitet, sollten Sie eine SDI-Applikation anlegen.

MDI-Terminologie

Bei der Entwicklung einer Applikation, die mehrere Datendateien verarbeitet, sollten Sie die MDI-Terminologie kennen. Das primäre Formular, das als Container für die anderen Formulare dient, wird häufig als *Elternformular* oder *Elternfenster* bezeichnet. Das Elternformular verhält sich wie ein Behälter für ein oder mehrere Kindfenster (ebenfalls Formulare). In Microsoft Word beispielsweise stellt das Elternfenster den Word-Hintergrund mit Menüleiste, Statusleiste und Symbolleiste dar. Sobald Sie Word-Dokumente öffnen, erscheinen sie als Kindfenster innerhalb des Elternfensters. Die Kindfenster können nicht außerhalb der Grenzen des Elternfensters erscheinen.

Das Elternfenster, das die Grenzen für die Kindfenster darstellt, unterstützt nur die folgenden Steuerelementtypen:

- Steuerelemente, die die **Align**-Eigenschaft unterstützen.
- Steuerelemente ohne sichtbare Oberfläche (mehr darüber erfahren Sie in Kapitel 15).



Wenn die Benutzer Kindfenster minimieren, erscheinen diese Fenster minimiert unten im Elternfenster, nicht in der Taskleiste. Kindfenster sind an ihre Elternfenster gebunden und können nicht außerhalb erscheinen. Das Elternfenster definiert die absoluten Grenzen für die Prozedur.

Der wichtigste Unterschied, der ein Kindfenster-Formular von einem normalen Formular unterscheidet, ist seine `MDIChild`-Eigenschaft, die auf `True` gesetzt ist. Eine MDI-Prozedur kann auch Nicht-Kindformulare enthalten. Beispielsweise könnte sie ein **Info**-Dialogfeld haben (das die Benutzer erreichen, indem sie im **Hilfe**-Menü den Eintrag **Info** auswählen); dieses Dialogfeld ist kein MDI-Kind, weil das Dialogfeldfenster keine Programmdatei enthält.

Wenn Sie eine MDI-Prozedur erstellen möchten, wählen Sie im **Projekt**-Menü den Eintrag **MDI-Formular hinzufügen**. Anschließend fügen Sie die Kindfenster-Formulare mit der `MDIChild`-Eigenschaft `True` ein.

Der Anwendungs-Assistent

Statt eine MDI-Applikation von Grund auf neu zu schaffen, können Sie auch den Anwendungs-Assistenten dafür verwenden. Dieser Assistent macht es viel einfacher, MDI-Applikationen zu erstellen, als die Formular von Hand einzurichten und ihre `MDIChild`-Eigenschaften zu setzen.

Im zweiten Dialogfeld des Anwendungs-Assistenten von Visual Basic wählen Sie aus, welchen Oberflächentyp Sie verwenden möchten (siehe Abbildung 11.7).

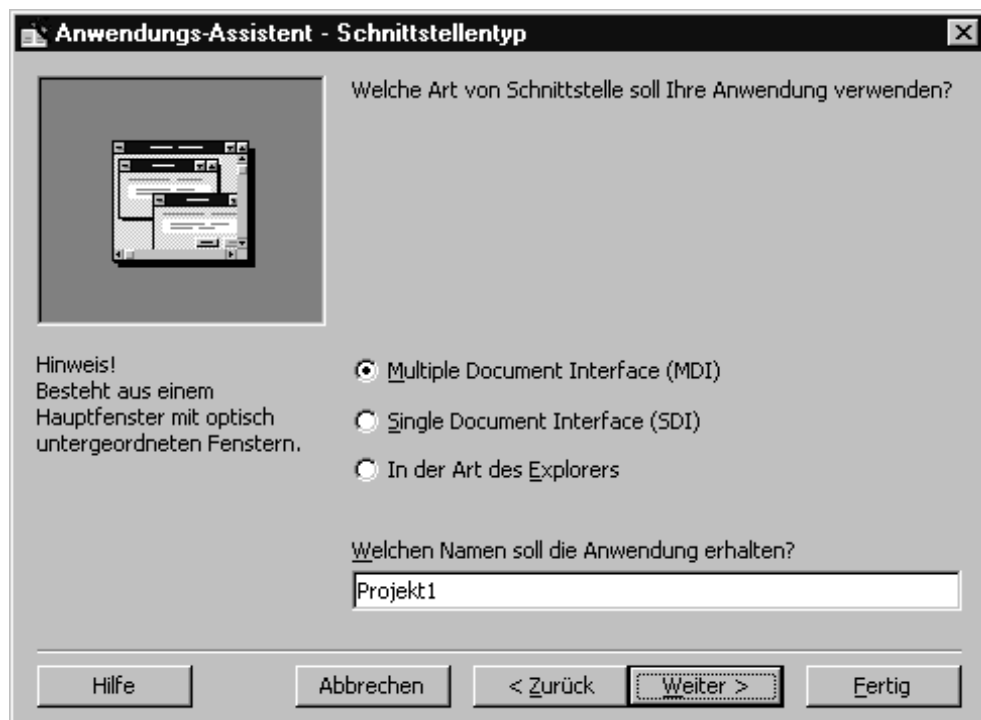


Abbildung 11.7:
Im Anwendungs-Assistent legen Sie fest, welchen Stil Ihre Applikation verwenden soll.

Der Anwendungs-Assistent erzeugt ein Projekt, in dem Sie über den **Neu**-Befehl im **Datei**-Menü mehrere Kindfenster anlegen können. Listing 11.8 wird ausgeführt, wenn Sie im **Datei**-Menü der erzeugten MDI-Applikation den Eintrag **Neu** auswählen (nämlich in der Ereignisprozedur `mnuFileNew.Click`).

Listing 11.8: Eine Formulardokument-Variable zeigt auf das Kindfenster

```
1: Private Sub LoadNewDoc()  
2: Static lDocumentCount As Long  
3: Dim frmD As frmDocument  
4:  
5: lDocumentCount = lDocumentCount + 1  
6: Set frmD = New frmDocument  
7: frmD.Caption = "Dokument " & lDocumentCount  
8: frmD.Show  
9: End Sub
```

Der Code in Listing 11.8 ist etwas trickreich, aber er enthält nichts, was Sie mit Ihrem bisherigen Wissen nicht schon verstehen würden. Die statische Variable in Zeile 2, `lDocumentCount`, ist lokal für die Prozedur, verläßt aber nie ihren Gültigkeitsbereich. Wenn die Prozedur zum ersten Mal ausgeführt wird, ist `lDocumentCount` gleich 0 (alle statischen Variablen sind 0, wenn sie zum ersten Mal in den Gültigkeitsbereich eintreten). Wenn die Prozedur den Wert von `lDocumentCount` ändert (was passiert, wenn die Prozedur in Zeile 5 den Wert 1 addiert), merkt sich Visual Basic den neuen Wert. Keine andere Prozedur kann auf `lDocumentCount` zugreifen, aber ihr Wert bleibt innerhalb dieser Prozedur erhalten. Er verschwindet nicht, wie es der Fall wäre, wenn die Variable lokal und *automatisch* wäre (im Gegensatz zu statisch; alle Variablen, die Sie bisher lokal definiert haben, waren automatisch).



Eine *statische Variable* behält ihren Wert bei, auch wenn die Prozedur, in der sie definiert ist, beendet wird. Wenn also eine Variable zum Prozedurende den Wert 7 enthält, wird dieser Wert beim nächsten Aufruf der Prozedur wieder bereitgestellt, so als ob der Prozeduraufruf in der Schleife einer anderen Prozedur erfolgen würde.

Zeile 3 führt eine interessante Umkehrung der Deklarationsanweisung ein, die Sie noch nicht kennen. Statt eine Variable zu deklarieren, deklariert diese Anweisung ein Formular! Die Applikation enthält beim Start ein Kindformular namens `frmDocument`, das zur Entwurfszeit spezifiziert wurde. Die `Dim`-Anweisung in Zeile 3 deklariert die neue Variable `frmD`, die dasselbe Objekt aufnimmt wie `frmDocument`; mit anderen Worten, statt auf einen Integer- oder String-Datentyp zu verweisen, bezieht sich `frmD` auf ein Dokument, das dieselben Eigenschaften hat wie `frmDocument`. `frmD` ist ein Beispiel für eine Steuerelementvariable.

Nachdem Sie in Zeile 5 die statische Variable aktualisiert haben, die alle neu erzeugten Dokumente beobachtet, erzeugt die `Set`-Anweisung ein neues Dokument und setzt `frmD` als Verweis auf dieses Dokument. `frmD` ist praktisch das neue Dokument. Die nächste Anweisung setzt den Titel des neuen Dokuments auf `Dokument`, gefolgt von der Dokumentnummer. Die letzte Anweisung zeigt das neue Dokument mit Hilfe der `Show`-Methode im Elternfenster an.



Die `Set`-Anweisung verhält sich fast wie eine Zuweisung, außer daß sie einer Steuerelementvariablen ein Steuerelement zuweist. In Listing 11.8 deklariert Zeile 3 kein neues Formular! Statt dessen wird hier eine Formularvariable deklariert. Die Variable ist leer, bis Sie ihr mit der `Set`-Anweisung ein Formular zuordnen, wie in Zeile 6 gezeigt.

Egal wie oft ein Benutzer den Eintrag **Neu** im **Datei**-Menü auswählt, wird die Prozedur `LoadNewDOC ( )` ausgeführt und ein neues Kindfenster wird erzeugt. Wenn Sie den Anwendungs-Assistenten und die resultierende MDI-Applikation ausführen, macht der Code für den **Beenden**-Befehl des **Datei**-Menüs gar nichts - der Anwendungs-Assistent stellt keinen solchen Code zur Verfügung. Sie müssen den Code selbst schreiben, der das aktive Kindfenster aus dem Speicher entfernt, wenn der Benutzer den Menüeintrag **Beenden** im **Datei**-Menü auswählt.



Kindformulare sind für Dialogfelder geeignet, die Sie Ihren Applikationen hinzufügen. Das nächste Kapitel erklärt, wie man Standarddialogfelder erstellt, wie beispielsweise **Datei drucken**, aber Sie können auch eigene Dialogfelder entwickeln, indem Sie ein Kindformular mit den entsprechenden Steuerelementen anzeigen. Setzen Sie für die Standardschaltfläche die `Default`-Eigenschaft und initialisieren Sie die Textfelder mit einer Textvorgabe, um Ihren Benutzern die Eingabe zu vereinfachen. Wenn Sie die `Enabled`-Eigenschaft der Schaltfläche auf `False` setzen, wird sie grau dargestellt und steht nicht zur Verfügung, abhängig davon, was der Benutzer im Dialogfeld gerade macht.

Wenn der Benutzer das Dialogfeld sehen soll, zeigen Sie es mit der `Show`-Methode an:

```
frmDialog.Show vbModal, Me
```

`vbModal` erzeugt ein modales Dialogfeld, auf das der Benutzer mit **OK** oder **Abbrechen** reagieren muß, bevor er in der Applikation weiterarbeiten kann. Die zweite Option der `Show`-Methode ist der Name des Elternformulars, Sie können aber auch `Me` angeben, wenn das Elternformular das Standardformular der Applikation ist.

Symbolleisten

Symbolleisten ermöglichen es Ihren Benutzern, gebräuchliche Befehle und Menüoptionen durch einen Mausklick aufzurufen. Um Ihrer Applikation eine Symbolleiste hinzuzufügen, fügen Sie Ihrer Werkzeugsammlung das Symbolleisten-Steuerelement hinzu. Nachdem Sie die Symbolleiste angelegt haben, brauchen Sie nur noch Code für die `Click`-Ereignisse der einzelnen Schaltflächen auf der Symbolleiste zu schreiben, so wie Sie das von den Menüoptionen her kennen.



Wenn Sie MDI-Applikationen schreiben, können Sie dem Elternformular oder jeder Formularkombination Symbolleisten hinzufügen.

Das Symbolleisten-Steuerelement

Symbolleisten und Coolbars, die Sie im nächsten Abschnitt kennenlernen werden, sind sich in ihrer Funktionalität ähnlich, ebenso in der Weise, wie sie einer Applikation hinzugefügt werden. Die Verwendung von Symbolleisten ist gebräuchlicher (man findet sie heute in fast jeder Windows-Applikation), aber die Coolbar ist etwas einfacher, weil dafür weniger Eigenschaftswerte gesetzt werden müssen. Dieser Abschnitt erklärt, wie man einer Applikation eine Symbolleiste hinzufügt. Im nächsten Abschnitt lernen Sie, wie Sie Coolbars hinzufügen. Nachdem Sie die etwas einfacheren Coolbars kennengelernt haben, beherrschen Sie auch den Umgang mit Symbolleisten (Toolbars).



Wie Sie im nächsten Abschnitt erfahren werden, unterscheiden sich Coolbars von Symbolleisten, weil sie eine Möglichkeit bieten, die Steuerelemente aus dem Bildschirm heraus und wieder auf den Bildschirm zu schieben.

Damit Sie eine Symbolleiste einfügen können, brauchen Sie das Symbolleisten-Steuerelement in Ihrer Werkzeugsammlung. Wählen Sie **Projekt, Komponenten**, um das Dialogfeld **Komponenten** anzuzeigen. Wählen Sie den Eintrag *Microsoft Windows Common Controls 6.0* aus und klicken Sie auf OK. Ihre Werkzeugsammlung zeigt jetzt mehrere neue Steuerelemente an. Schieben Sie Ihren Mauscursor über diese Werkzeuge, um zu erfahren, worum es sich dabei handelt.

Die folgenden Schritte beschreiben die allgemeine Vorgehensweise zum Einfügen einer Symbolleiste in Ihre Applikation:

1. Nachdem Sie Ihrer Werkzeugsammlung das Symbolleisten-Steuerelement (ToolBar) hinzugefügt haben, doppelklicken Sie darauf, um oben in Ihrem Formularfenster eine Symbolleiste einzufügen. (Die Rasterpunkte müssen aktiviert sein, damit Sie den Umriß der Symbolleiste sehen.) Die Symbolleiste erstreckt sich immer über das gesamte Formular, egal wie breit dieses ist. Um die Breiteneinstellungen brauchen Sie sich nicht zu kümmern.
2. Falls Ihre Symbolleiste auf den Schaltflächen Icons anzeigen soll, brauchen Sie eine Abbildungsliste (ImageList) auf Ihrem Formular. Die Abbildungsliste ist eines der Steuerelemente, die Ihrer Werkzeugsammlung hinzugefügt werden, wenn Sie *Microsoft Windows Custom Controls 6.0* aktivieren. Die Abbildungsliste nimmt die Bilder auf, die Sie auf der Symbolleiste anlegen. Sie können die Abbildungsliste an eine beliebige Position schieben, weil es egal ist, wo sie sich befindet. Die Abbildungsliste wird auf dem Formular zwar nie angezeigt, aber die dort enthaltenen Bilder werden auf den entsprechenden Schaltflächen der Symbolleiste dargestellt.



Die Standardposition für eine Symbolleiste ist der obere Formularrand. Sie können die Symbolleiste jedoch auch unten im Formular anzeigen, indem Sie die `Align`-Eigenschaft auf `vbAlignBottom` setzen.

3. Klicken Sie im Eigenschaftfenster der Abbildungsliste auf den Eintrag (**Benutzerdefiniert**), um das Dialogfeld **Eigenschaftenseiten** anzuzeigen (siehe Abbildung 11.8).

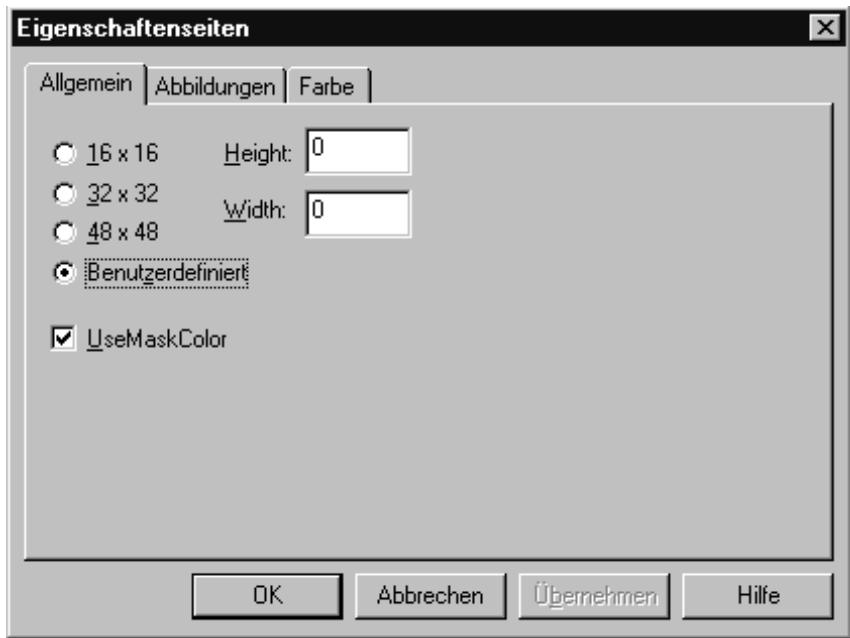


Abbildung 11.8:
Das Dialogfeld Eigenschaftenseiten vereinfacht die Eingabe der Eigenschaftswerte für Abbildungslisten.

4. Klicken Sie auf die Registerkarte **Abbildungen**.
5. Klicken Sie auf die Schaltfläche **Bild einfügen** und wählen Sie ein Icon aus, das hinzugefügt werden soll.
6. Während Sie Icons einfügen, setzt Visual Basic entsprechende Indexwerte. Der Indexwert ist der Wert, über den Sie einer Schaltfläche der Symbolleiste ein Icon zuordnen.
7. Schließen Sie das Eigenschaftfenster für die Abbildungsliste.
8. Klicken Sie im Eigenschaftfenster auf den Eintrag (**Benutzerdefiniert**), um das Dialogfeld **Eigenschaftenseiten** für die Symbolleiste anzuzeigen (siehe Abbildung 11.9). Dieses Dialogfeld vereinfacht die Eingabe der Eigenschaftswerte für die Symbolleiste.

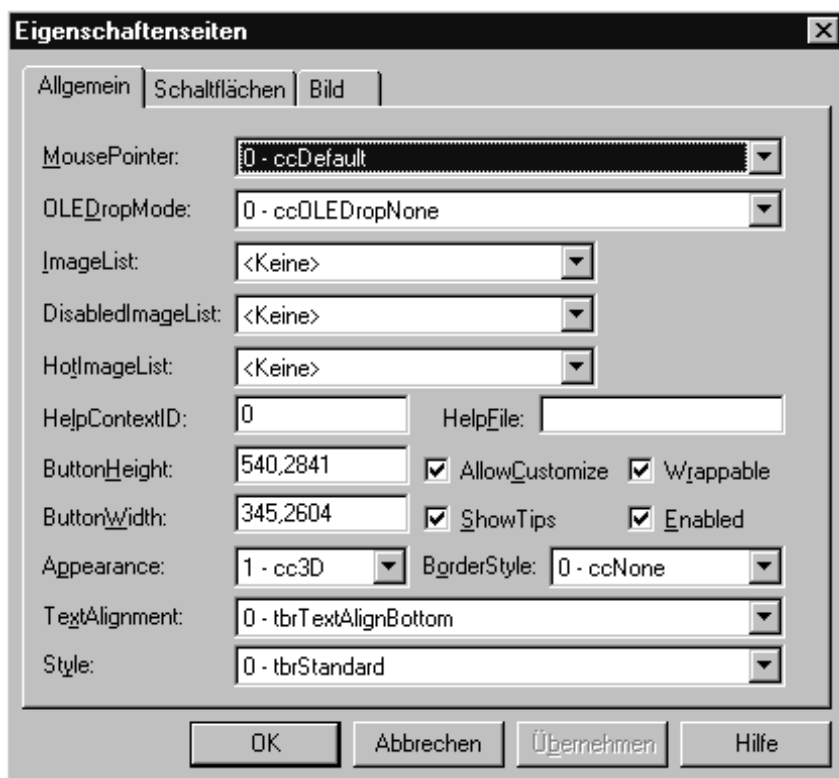


Abbildung 11.9:

Auch für das Symbolleisten-Steuererelement gibt es ein Eigenschaftenseiten-Dialogfeld.

9. Ändern Sie die Eigenschaftswerte entsprechend ab und klicken Sie auf die Schaltfläche **Übernehmen**, um der Symbolleiste im Formularfenster die Eigenschaften zuzuordnen. Stellen Sie sicher, daß Sie in der Option **ImageList** die Bilder für die verschiedenen Schaltflächen der Symbolleiste zugeordnet haben. Aktivieren Sie auf der Registerkarte **Schaltflächen** die Option **ToolTipText**, falls Sie kurze Pop-up-Beschreibungen für Ihre Benutzer bereitstellen möchten.

10. Klicken Sie auf **OK**, um das **Eigenschaften**-Dialogfeld zu schließen.

11. Fügen Sie für alle Schaltflächen der Symbolleiste **Click**-Ereignisprozeduren hinzu.



Setzen Sie die **Wrappable**-Eigenschaft auf **True**, um zu ermöglichen, daß die Symbolleiste über mehrere Zeilen angezeigt wird, falls sie mehr Icons enthält, als in eine Zeile passen.

Wenn Sie Ihren Benutzern etwas Gutes tun wollen, ermöglichen Sie ihnen, die Symbolleiste anzupassen. Setzen Sie die **AllowCustomize**-Eigenschaft auf **True**, so daß der Benutzer zur Laufzeit auf die Symbolleiste doppelklicken kann, um das in Abbildung 11.10 gezeigte Dialogfeld aufzurufen.



Ziehen Sie hier, um die Größe der Cool Bar zu ändern

Klicken Sie hier, um weitere Einträge anzuzeigen

Abbildung 11.10:

Die Symbolleiste kann vom Benutzer zur Laufzeit angepaßt werden.

Coolbars hinzufügen

Beginnend mit Visual Basic Version 6 können Sie Ihren Applikationen Coolbars hinzufügen. Abbildung 11.11 zeigt eine Coolbar aus dem Internet Explorer. Der Benutzer kann diese Symbolleiste mit Hilfe der Griffe links oder rechts zur Seite schieben. Coolbars ermöglichen Ihnen, sehr breite Symbolleisten anzulegen. Ihr Benutzer verschiebt sie dann nach links oder rechts, um die betreffenden Optionen auf dem Bildschirm zu sehen.



Um der Werkzeugsammlung das Coolbar-Steuererelement hinzuzufügen, wählen Sie im Dialogfeld **Projekt, Komponenten** den Eintrag *Microsoft Windows Custom Controls-3 6.0* aus.

Eine Coolbar wird ähnlich einer Symbolleiste angelegt. Zuerst fügen sie Bilder ein und verbinden sie dann über den Key-Wert mit der Coolbar. Coolbars sind viel einfacher als Symbolleisten, weil Sie weniger Eigenschaftswerte dafür setzen müssen.

Um eine Coolbar einzufügen, gehen Sie wie folgt vor:

1. Fügen Sie Ihrer Werkzeugsammlung das Abbildungsliste-Steuerelement hinzu. Dazu wählen Sie **Projekt, Komponenten** und markieren die Option *Microsoft Windows Common Controls 6.0*. Außerdem brauchen Sie das Coolbar-Steuerelement. Markieren Sie im Dialogfeld Komponenten also auch *Microsoft Windows Common Controls-3 6.0*, bevor Sie die Eingabetaste drücken. Die Abbildungsliste und die Coolbar werden jetzt in Ihrer Werkzeugsammlung angezeigt.
2. Doppelklicken Sie auf die Abbildungsliste, um sie Ihrer Applikation hinzuzufügen. Schieben Sie sie an eine Position, wo sie nicht stört. Dieses Steuerelement erscheint zur Laufzeit nicht im Anwendungsfenster. Es dient nur dazu, die Bilder aufzunehmen, die auf den Schaltflächen der Coolbar angezeigt werden.
3. Doppelklicken Sie auf die (Benutzerdefiniert)-Eigenschaft für die Abbildungsliste, um das Dialogfeld **Eigenschaftenseiten** anzuzeigen.
4. Klicken Sie auf die Registerkarte **Abbildungen**.
5. Klicken Sie auf die Schaltfläche **Bild einfügen**. Das Dialogfeld **Bild auswählen** erscheint. Dort wählen Sie eine Grafikdatei aus, die das erste Bild in der Abbildungsliste darstellen soll.
6. Wählen Sie das Bild aus, beispielsweise eine Bitmap aus dem Graphics-Ordner von Visual Basic. Doppelklicken Sie auf einen Dateinamen. Das Dialogfeld wird geschlossen und das Bild erscheint als erstes Icon in der Abbildungsliste (siehe Abbildung 11.12).



Abbildung 11.11:
Das erst Bild in der Abbildungsliste.

7. Fügen Sie der Coolbar weitere Abbildungen hinzu.
8. Doppelklicken Sie auf das Coolbar-Steuerelement, um Ihrem Formular zwei Coolbars (sogenannte *Bänder*) hinzuzufügen.
9. Doppelklicken Sie auf die (Benutzerdefiniert)-Eigenschaft, um das Dialogfeld **Eigenschaftenseiten** für die Coolbar anzuzeigen, wie in Abbildung 11.13 gezeigt.

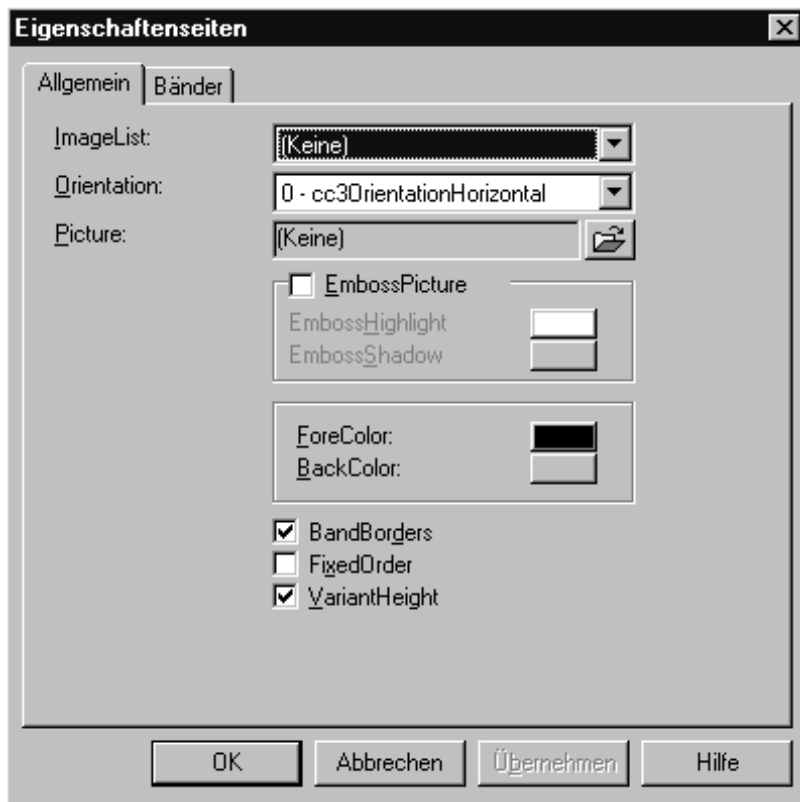


Abbildung 11.12:
Es ist ganz einfach, die Eigenschaften für die Coolbar im Dialogfeld Eigenschaftenseiten zu setzen.

10. Selektieren Sie im Listenfeld **ImageList** die Bildliste. Tabelle 11.7 beschreibt die Optionen, die Sie auf der Registerkarte **Allgemein** setzen können.

Tabelle 11.7: Die Registerkarte Allgemein beschreibt das allgemeine Erscheinungsbild der Coolbar.

Option	Beschreibung
Orientation	Plaziert die Coolbar horizontal (Standard) oder vertikal auf dem Formular. Die meisten Coolbars werden horizontal angelegt.
Picture	Legt fest, welches Bild auf allen Schaltflächen der Coolbar erscheinen soll. Normalerweise zeigt man mit Hilfe der Abbildungsliste auf jeder Schaltfläche ein anderes Bild an, statt mit Picture lauter gleiche Bilder zu erzeugen.
ForeColor	Die Vordergrundfarbe der Coolbar.
BackColor	Die Hintergrundfarbe der Coolbar.
BandBorders	Bestimmt, ob zwischen den verschiedenen Band-Objekten Trennlinien angezeigt werden sollen.
FixedOrder	Bestimmt, ob der Benutzer die Reihenfolge der Band-Objekte auf der Coolbar ändern kann.
VariantHeight	Bestimmt, ob die Band-Objekte der Coolbar alle die Höhe der größten Schaltfläche im Steuerelement haben sollen.



Die **Picture**-Optionen bietet Ihnen Zugriff auf spezielle Farbmodi und Darstellungsformen für die Schaltflächen auf der Coolbar, beispielsweise, ob sie gedrückt angezeigt werden, wenn der Benutzer sie anklickt.

11. Klicken Sie auf die Registerkarte **Bänder**, um die einzelnen Bilder für die Coolbar hinzuzufügen.
12. Sie fügen Bänder hinzu oder entfernen sie, indem Sie auf die Schaltflächen **Band einfügen** oder **Band entfernen** klicken. Tabelle 11.7 beschreibt weitere Eigenschaften, die Sie anpassen können. Wahrscheinlich wollen Sie die Bilder aus Ihrer Abbildungsliste hinzufügen.
13. Sie brauchen die **Image**-Eigenschaft, um den Schaltflächen der Coolbar die Bilder aus der Abbildungsliste hinzuzufügen. Jeder **Image**-Wert gehört zu einem Bild in der Abbildungsliste. Wenn Ihre Abbildungsliste also 10 Bilder enthält, die Sie auf der Coolbar anzeigen möchten, geben Sie im Dialogfeld **Eigenschaftenseiten** 10 **Image**-Werte an.



Sie sehen die Bilder auf der Coolbar nicht sofort, nachdem Sie das Dialogfeld **Eigenschaftenseiten** geschlossen haben. Die Bilder erscheinen erst zur Laufzeit.

Zusammenfassung

Dieses Kapitel hat Formulare und Formulareigenschaften, Ereignisse und Methoden genauer erklärt. Das Formular ist der Hintergrund Ihrer Applikation, ist also ein wichtiger Teil der Programmierung. Mehrere Formulare können auf unterschiedliche Weise angesprochen werden; wie Sie heute erfahren haben, ist die **Forms**-Auflistung eine der einfachsten Möglichkeiten, die Formulare einer Applikation zu durchlaufen.

Sie können nicht nur mit den vorgegebenen Formulareigenschaften arbeiten, sondern Ihrem Formular auch benutzerdefinierte Eigenschaften hinzufügen. Nachdem Sie eine Eigenschaft hinzugefügt haben, können Sie diese setzen und lesen, so als ob es sich um eine der Originaleigenschaften handelte.

Die Programmierung von MDI-Fenstern kann mühsam sein, aber Sie bieten Ihren Benutzern damit die Möglichkeit, unterschiedliche Daten in unterschiedlichen Fenstern zu bearbeiten. Mit Hilfe des Anwendungs-Assistenten erzeugen Sie das Gerüst für die MDI-Applikation und ergänzen diese um die Details.

Die Symbolleiste und die Coolbar fügen zwei weitere Dimensionen für Ihre Benutzer ein. Eine Symbolleiste bietet einfachen Zugriff auf die gebräuchlichsten Befehle. Bevor Sie Icons auf den Schaltflächen einer Symbolleiste plazieren können, müssen Sie diese in einem Abbildungsliste-Steuererelement anlegen. Coolbars sind Symbolleisten, die der Benutzer nach links und rechts verschieben kann. Sie sollten die Coolbars und Symbolleisten als Menüoptionen anbieten, so daß es Ihren Benutzern überlassen bleibt, was sie auf ihren Formularen anzeigen möchten.

Im nächsten Kapitel erfahren Sie, wie man die Platte als externe Speicherquelle für die Daten einer Visual-Basic-Applikation benutzt.

Fragen und Antworten

Frage:

Wozu brauche ich das Abbildungsliste-Steuererelement, wenn ich einer Symbolleiste oder einer Coolbar Icons hinzufügen möchte?

Antwort:

*Microsoft hat die Symbolleisten und die Coolbars so entwickelt, daß sie mit dem Abbildungsliste-Steuererelement zusammenarbeiten. So einfach ist das. Microsoft hätte Ihnen auch ermöglichen können, die Icons direkt in der Eigenschaftenliste der Symbolleisten oder Coolbars anzulegen, aber man hat sich dort entschlossen die Abbildungsliste dazwischenzuschalten. Und Microsoft hat Ihnen damit einen Gefallen getan. Sie können mehrere Abbildungslisten auf einem Formular anlegen und sie als Schablonen für Ihre Symbolleisten und Coolbars verwenden. Sie können beispielsweise eine der Abbildungslisten für die aktiven Icons der Symbolleiste verwenden, und eine andere für deaktivierte Icons. Sie brauchen dazu nur den Namen der Abbildungsliste in der **ImageList**-Eigenschaft der Symbolleiste oder Coolbar zu ändern, und Visual Basic verwendet die andere Abbildungsliste als Quelle für die Abbildungen.*

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Wie hilft Ihnen das **Resize**-Ereignis, sicherzustellen, daß die Steuererelemente Ihres Formulars zentriert sind?
2. Welche Aufgabe hat die **ScaleMode**-Eigenschaft?
3. Welchen Wert hat der erste Index bei vordefinierten Objekt-Auflistungen?
4. Richtig/Falsch. Eine SDI-Applikation kann nicht mehrere Formulare unterstützen.
5. Was ist der Unterschied zwischen einer SDI- und einer MDI-Applikation?
6. Wo erscheinen die Symbolleisten normalerweise im Formular?
7. Welches Steuererelement nimmt die Icons für die Symbolleiste auf?
8. Warum ist es egal, wo Sie die Abbildungsliste auf dem Formular anlegen.
9. Was ist der Unterschied zwischen **Spc ()** und **Tab ()**?

10. Wie gibt Print eine leere Zeile aus?

Übungen

- 1. Laden Sie die folgenden Steuerelemente in Ihre Werkzeugsammlung: Symbolleiste, Coolbar und Abbildungsliste.
- 2. Welche Ausgaben erzeugen die beiden folgenden Anweisungen?

```
Form1.Print "Zeile 1"  
Form1.Print "Zeile 2"
```

- 3. Erstellen Sie mit dem Anwendungs-Assistenten eine Applikation, die den Explorer-Stil verwendet. Führen Sie die Applikation aus, um das Ergebnis zu sehen.
- 4. Schreiben Sie Code, der die Zahlen 1 bis 100 auf einem Formular ausgibt. Trennen Sie die Zahlen jeweils durch ein Leerzeichen. Verwenden Sie keine Steuerelemente für die Ausgabe. Lösen Sie die Ausgabe in der Click-Ereignisprozedur des Formulars aus.
- 5. Schreiben Sie eine Prozedur, die die Gesamtzahl aller Steuerelemente in allen Formularen der aktuellen Applikation ermittelt und ausgibt.

Woche 2

Tag 12

Mit Dateien arbeiten

Dieses Kapitel erklärt die Grundlagen des *Datei-I/O* (Input/Output, Eingaben/Ausgaben). Professionelle Visual-Basic-Programmierer verwenden unterschiedliche Arten von Datei-I/O. Was Sie heute lernen, wird Ihnen helfen, alle anderen dateibezogenen Techniken zu verstehen. Nachdem Sie dieses Kapitel gelesen haben, werden Sie in der Lage sein, mit den flexibleren Daten- und Datenbank-Steuerelementen zu arbeiten.

Es gibt drei Dateitypen: *sequentielle Dateien*, *Dateien mit wahlfreiem Zugriff* und *Binärdateien*. Der sequentielle Dateizugriff stellt die einfachste Methode dar, weist aber gleichzeitig einige Nachteile auf. Sequentielle Dateien sind zwar einfach anzulegen und zu lesen, aber sie können langsam und mühsam sein. Der wahlfreie Zugriff ist viel schneller und praktischer, aber Programme, die Dateien mit wahlfreiem Zugriff einsetzen, sind häufig komplizierter als Programme mit Dateien, die einen sequentiellen Zugriff verwenden. Binärdateien sind spezielle, kompaktere Dateien mit wahlfreiem Zugriff.

Dateien mit wahlfreiem Zugriff können beliebige Daten aufnehmen. Nachdem Sie gelernt haben, wie man benutzerdefinierte Datentypen deklariert, verwenden Sie den Befehl `Put #`, um in einer einzigen Anweisung einen ganzen Informationssatz in eine Datendatei zu schreiben (oder daraus zu lesen).

Heute lernen Sie die folgenden Dinge:

- Dateitypen
- Den Unterschied zwischen sequentiellen Dateien und Dateien mit wahlfreiem Zugriff
- Dateinummern
- Dateien öffnen
- Freie Dateinummern
- Die Befehle `Print #`, `Write #`, `Read #`, `Get #` und `Put #`
- Dateibezogene Steuerelemente

Die Arbeit mit Dateien

Viele aktuelle Programme verwenden Dateien. Und wenn ein Programm keine Dateien verarbeitet, dann ist es selbst in einer Datei abgelegt, die ausgeführt wird, wenn der Benutzer das Programm aufruft. Egal ob ein Programm Daten in einer Datenbank ablegt, oder ob es einfach nur Informationen

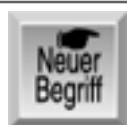
für seinen eigenen Gebrauch speichert (beispielsweise das Farbschema für den Benutzer oder die Fensterposition) - die meisten Programme benutzen Dateien.

Es gibt verschiedene Visual-Basic-Befehle, die für alle Formen der Dateiein- und -ausgabe gebräuchlich sind. Diese Befehle öffnen Dateien, spezifizieren Dateimodi, schließen Dateien und prüfen, welche freien Dateinummern es gibt. Dieser Abschnitt erklärt die Grundlagen zur Dateiverarbeitung. Am besten fangen Sie mit der ersten Anweisung jedes dateibezogenen Projekts an: **Open**.

Die Anweisung Open

Sequentielle Dateien und Dateien mit wahlfreiem Zugriff haben einige gemeinsame Eigenschaften. Für beide Dateitypen verwenden Sie den Befehl **Open**. Welcher Dateizugriff mit **Open** erfolgt, wird durch die der Anweisung übergebenen Argumente bestimmt. **Open** reserviert einen *Datei-Handle*, auch als *Kanal* bezeichnet, mit dessen Hilfe eine Datei gelesen und in sie geschrieben wird.

Die **Open**-Anweisung ordnet dem Handle eine Nummer zu. Wenn dieser Datei-Handle geöffnet ist, kann der Datenstrom fließen. Der Modus, in dem Sie die Dateinummer öffnen - Lesen, Schreiben oder beides - bestimmt, wie der Datenstrom zwischen der Datei und dem Computer fließt.



Ein *Datei-Handle* oder *Kanal* ist ein eindeutiger Pfad zu einer Datei, dem in der **Open**-Anweisung eine Nummer zugeordnet wird. Nachdem die **Open**-Anweisung dem Datei-Handle eine Nummer zugeordnet hat, verwendet das restliche Programm diese Nummer für den Zugriff auf die Datei. Der Code braucht hinter der **Open**-Anweisung nie den betreffenden Dateinamen anzugeben.

Hier das Format der **Open**-Anweisung:

```
Open "strDateinameName" [For Mode] [ZugriffsEinschränkung]
[LockType] As
[#]intDateiNum [Len = intDatensatzLänge]
```



Wenn Sie ein Programm öffnen und einer Datendatei eine Nummer zuordnen, verwendet das Programm diese Dateinummer für den weiteren Dateizugriff.

Alle **Open**-Anweisungen müssen als Argumente den Dateinamen und die Dateinummer beinhalten. Die anderen Argumente sind optional. Ein einfacher Aufruf von **Open** mit den zwingend erforderlichen Parametern könnte wie folgt aussehen:

```
Open "aFile.txt" As #1
```

Diese Anweisung öffnet die Datei `aFile.txt` mit der Dateinummer 1. Um auf diese Datei zuzugreifen, um Ein- oder Ausgaben vorzunehmen, geben Sie die Dateinummer an. Diese Datei wurde für den wahlfreien Zugriff geöffnet, weil der Standard für das **For**-Argument gleich **Random** ist. Alle Befehle in diesem Kapitel arbeiten mit Textdateien. Dabei werden die Dateinamenerweiterungen `.txt` oder `.dat` verwendet, zwei gebräuchliche Erweiterungen für Textdateien.

Die Dateimodi

Beachten Sie, daß der **Mode**-Wert in der **Open**-Anweisung kein Datentyppräfix verwendet. **Mode** muß ein spezielles Schlüsselwort sein, das den Zugriffsmodus für die Datei angibt. Tabelle 12.1 erklärt die Werte, die Sie für das **Mode**-Argument angeben können.

Tabelle 12.1: Verwenden Sie einen dieser Mode-Werte für die Open-Anweisung

Modus	Beschreibung
Append	Öffnet eine Datei für sequentielle Ausgaben, beginnend am Dateiende, falls die Datei bereits existiert. Wenn die Datei noch nicht existiert, legt Visual Basic sie an. Append überschreibt keine existierenden Dateidaten.
Binary	Öffnet eine Datei mit binärem Datenzugriff. Im Binary -Modus können Sie auf Byteebene auf die Datei zugreifen, d.h. Sie können einzelne Bytes in die Datei schreiben und daraus lesen.
Input	Öffnet eine Datei mit sequentiellen Eingaben, beginnend am Dateianfang. Die Daten werden in der Reihenfolge gelesen, wie sie in die Datei geschrieben wurden.
Output	Öffnet eine Datei mit sequentiellen Ausgaben, beginnend am Dateianfang. Wenn die Datei noch nicht existiert, legt Visual Basic sie an; andernfalls überschreibt Visual Basic die Datei.
Random	Öffnet eine Datei mit wahlfreiem Lese- und Schreibzugriff. Dieser Modus erlaubt, daß Daten aus beliebigen Datensätzen gelesen und dorthin geschrieben werden.

For **Mode** ist für die **Open**-Anweisung nicht erforderlich. Wenn Sie keinen Modus angeben, setzt Visual Basic den Modus **Random** voraus und fügt **For Random** für Sie ein. Die folgenden Anweisungen zeigen, wie die verschiedenen Modi zum Öffnen von Dateien verwendet werden:

```
Open "filInpdt.txt" For Input As #1
```

```
Open "Append.txt" For Append As #1
```

```
Open "Output.txt" For Output As #1
```

```
Open "Random.txt" For Random As #1
```

Die letzte Anweisung ist äquivalent zur folgenden Anweisung:

```
Open "Random.txt" As #1
```



Stellen Sie mit der Anweisung **On Error Goto** eine Logik zur Fehlerverarbeitung sicher, wie Sie sie in Kapitel 9 kennengelernt haben. Immer wenn Sie eine Datei öffnen oder darauf zugreifen, könnte ein Fehler auftreten. Eine sinnvolle Fehlerverarbeitung hilft Ihrer Applikation, ein Problem zu lösen, statt Ihren Benutzern seltsame Laufzeitfehler zu präsentieren.

Zugriffsbeschränkungen

Das optionale Argument `AccessRestriction` in der `Open`-Anweisung ermöglichen Ihnen, den Zugriff auf `Read`, `Write` oder `Read Write` zu beschränken. Diese Zugriffsbeschränkungen werden häufig eingesetzt, wenn man Programme schreibt, die auf einem Netzwerk eingesetzt werden.

Die Bereitstellung eines Lesezugriffs mit `Read` erlaubt den Benutzern, den Dateiinhalt anzusehen, sie können ihn jedoch nicht verändern. Der `Write`-Zugriff erlaubt den Benutzern, die Datei zu verändern, und mit `Read Write` dürfen sie beides (Lesen und Verändern).

Dateien sperren

Mit dem `LockType`-Argument spezifizieren Sie, welche Operationen für die Datei von anderen Prozessen ausgeführt werden dürfen. Dieser Parameter ist sehr praktisch, wenn Sie Netzwerkanwendungen entwickeln. Sie können den Zugriff auf eine Datei sperren, so daß nur jeweils ein Benutzer gleichzeitig Zugriff darauf hat. Damit verhindern Sie, daß zwei Benutzer gleichzeitig Änderungen an der Datei vornehmen (was unabdingbar irgendwann zu verlorenen Daten führen würde).

Die gültigen Optionen für `LockType` sind `Shared`, `Lock Read`, `Lock Write` und `Lock Read Write`. `Shared` erlaubt allen Benutzern, gleichzeitig auf die Datei zuzugreifen. `Lock Read` sperrt die Datei so, daß nur die Person, die die Datei zum Lesen geöffnet hat, darauf zugreifen kann. `Lock Write` sperrt die Datei so, daß nur die Person, die die Datei zum Schreiben geöffnet hat, darauf zugreifen kann. `Lock Read Write` sperrt die Datei für alle Benutzer, außer für denjenigen, der die Datei zum Lese- und Schreibzugriff geöffnet hat.

Verwaltung der Datensatzlänge

Die Länge, die in `Len = intDatensatzLänge` angegeben ist, wird für Dateien mit wahlfreiem Zugriff verwendet, um die Größe der Datensätze festzulegen, die der Datei übergeben werden. Diese Größe ist erforderlich, wenn ein Zugriff auf die Datensätze in der Datei erfolgt. Der erste Datensatz in einer Datei beginnt an der Position 1. Alle folgenden Datensätze werden in Inkrementsschritten von 1 abgelegt. Die eigentliche Position, die ein Datensatz in der Datei einnimmt, ist $N \times \text{intDatensatzLänge}$, wobei N die Datensatznummer ist.



Ein *Datensatz* ist eine logische Zeile in einer Datei, die einen kompletten Datensatz enthält. Wenn Ihre Datei beispielsweise Inventardaten enthält, stellt ein Datensatz die Inventarinformation für einen Artikel dar, beispielsweise Beschreibung, Preis und Menge.

Der Zugriff auf Datensätze erfolgt ähnlich dem Zugriff auf Arrays. Während das erste Element eines Arrays in `Array(0)` abgelegt ist, befindet sich das erste Element einer Datei in Datensatz 1. Um die Indexkoordination zu gewährleisten, verwenden Sie in Ihrem Deklarationsbereich die Anweisung `Option Base 1`, oder Sie definieren Ihre Arrays so, daß sie mit dem Element 1 beginnen, und ignorieren den Index 0.

Freie Dateinummern

Visual Basic ermöglicht Ihnen, mehrere Dateien gleichzeitig zu öffnen, so lange Sie jeder Datei eine andere Dateinummer zuordnen. Sie müssen dazu die nächste verfügbare Nummer kennen, insbesondere wenn Sie Dateien in einer Funktion öffnen, die nicht wissen kann, ob bereits andere Funktionen Dateien geöffnet haben. Visual Basic stellt dazu die Funktion `FreeFile()` bereit, mit der die nächste freie Dateinummer ermittelt wird. Mit Hilfe dieser Funktion können Sie garantieren, daß die Dateinummer noch nicht von einer anderen geöffneten Datei verwendet wird. Hier das Format von `FreeFile()`:

```
FreeFile[(intBereichsNummer)]
```

Der optionale Parameter `intBereichsNummer` ermöglicht Ihnen die Angabe, in welchem Bereich die zurückgegebene Dateinummer liegen soll: 1-255 oder 256 -511. Der Standardbereich ist 1-255. Fast alle Visual-Basic-Programmierer verwenden den Standardwert, weil nur wenige Programme mehr als 256 Dateien gleichzeitig verwenden. Ohne den Parameter `intBereichsNummer` brauchen Sie die Klammern für die Funktion nicht anzugeben.

Die folgenden Zeilen ermitteln mit `FreeFile()` eine Dateinummer und öffnen damit eine Datei:

```
intFileNumber = FreeFile  
Open "AccPay.Dat" For Output As intFileNumber
```

Verwenden Sie `FreeFile()`, wenn Sie sicherstellen möchten, daß eine Dateinummer noch nicht genutzt wird. Das ist bei kleinen Applikationen, die nur ein paar Dateien benutzen, nicht erforderlich. Aber auch in kleinen Applikationen kann `FreeFile()` dafür sorgen, daß Sie nicht versehentlich dieselbe Dateinummer für mehrere Dateien gleichzeitig verwenden.



Vermeiden Sie in der `Open`-Anweisung die Abkürzung für `FreeFile`:

```
Open "strDateiname" For Output As FreeFile()
```

Das funktioniert zwar, aber Sie haben dann keine Möglichkeit, die Dateinummer für spätere Dateioperationen zu ermitteln.

Die Close-Anweisung

Sie müssen alle Dateien schließen, die Sie mit der `Open`-Anweisung geöffnet haben. Die Anweisung zum Schließen einer Datei ist `Close`. Diese Anweisung gibt die Dateinummer als einzigen Parameter an. Hier das vollständige Format für `Close`:

```
Close # indDateiNummer [, intDateiNummer2] [, ...intDateiNummerX]
```

Innerhalb einer `Close`-Anweisung können Sie beliebig viele Dateien schließen. Wenn Sie keine Dateinummer angeben, werden alle geöffneten Dateien geschlossen. Das kann sehr praktisch sein, um eine Applikation zu beenden.

Der Code in Listing 12.1 öffnet zwei sequentielle Dateien - eine zum Lesen und eine zum Schreiben. Dazu werden die beiden nächsten verfügbaren Dateinummern verwendet. Anschließend werden beide Dateien geschlossen.

Listing 12.1: `FreeFile()` fordert eine Dateinummer von Visual Basic an

```
1: Dim intReadFile As Integer, intWriteFile As Integer
```

```

2: ' Eingabedateil
3: intReadFile = FreeFile ' Erste Dateinummer ermitteln
4: Open "AccPay.Dat" For Input As intReadFile
5: ' Ausgabedatei
6: intWriteFile = FreeFile ' Nächste Dateinummer ermitteln
7: Open "AccPayOut.Dat" For Output As intWriteFile
8: '
9: ' Code zum Senden des Inhalts der Eingabedatei
10: ' an die Ausgabedatei
11: ' (wie das geht, erfahren Sie später im Kapitel)
12: Close intReadFile
13: Close intWriteFile

```

In diesem Beispiel brauchen Sie nie eine Dateinummer, weil `FreeFile()` in den Zeilen 3 und 6 die verfügbaren Dateinummern zur Verfügung stellt, und der Code diese Werte als Integer-Namen speichert.



Wenn Sie nicht alle geöffneten Dateien schließen, laufen Sie Gefahr, daß diese beschädigt werden - auch wenn das mit der heutigen Hardware kaum noch wahrscheinlich ist. Ganz allgemein kann man sagen, daß der Inhalt einer Datei gefährdet ist, wenn ein Stromausfall auftritt, so lange sie geöffnet ist. Halten Sie also eine Datei nicht länger geöffnet, als es unbedingt sein muß. Wenn Sie eine Datei nicht schließen, schließt das System sie beim Beenden der Applikation.

In einer einzelnen `Close`-Anweisung können beliebig viele Dateien geschlossen werden. Diese einfache Zeile schließt alle geöffneten Dateien:

`Close`

Die folgenden Zeilen schließen nur zwei Dateien, die möglicherweise geöffnet sind:

`Close 3`

`Close 6`

Möglicherweise möchten Sie bestimmte Dateien während der Programmausführung schließen, nachdem Sie sie nicht mehr brauchen, andere Dateien jedoch weiterhin geöffnet bleiben sollen.

Dateien mit sequentiellm Zugriff

Nachdem Sie gelernt haben, wie man Dateien öffnet, schließt und den Zugriffsmodus darauf festlegt, betrachten wir einige Beispiele für Eingaben und Ausgaben mit sequentiellen Dateien. Sie werden eine Variante von `Print` kennenlernen, das Sie gestern genutzt haben, um Text an ein Formular zu senden. `Print` kann also auch Text in eine Datei ausgeben.

Beim *sequentiellen Dateizugriff* greifen Sie sequentiell auf die Datei zu. Wenn Sie eine solche Datei anlegen, muß Ihre Applikation sequentiell daraus lesen und dorthin schreiben, d.h. vom Dateianfang hin zum Dateiende. Diese Einschränkung ist die größte Schwäche sequentieller Dateien.



Für eine *sequentielle Datei* erfolgt der Lese- und Schreibzugriff vom Dateianfang zum Dateiende.

Um die Datei nutzen zu können, müssen Sie die gesamte Datei vom Anfang bis zum Ende verarbeiten. Selbst wenn Sie nur ein Byte in einer 1000-Byte-Datei aktualisieren möchten, müssen Sie die anderen 999 Byte ebenfalls verarbeiten.

Der sequentielle Dateizugriff ist sehr praktisch, wenn Sie eine Textdatei verarbeiten, beispielsweise, um bestimmte Einstellungen vorzunehmen, oder wenn Sie nur sehr wenig Daten speichern, wo die Zugriffsgeschwindigkeit kein Problem darstellt. Dieser Abschnitt betrachtet Visual-Basic-Funktionen, die sequentielle Dateien verarbeiten.

Die Anweisung `Print #` für sequentielle Ausgaben

Um in einem Programm Dateien verwenden zu können, müssen Sie diese öffnen. Nachdem Sie die Dateien geöffnet haben, legen Sie Informationen darin ab. Ein allgemeiner Befehl dafür ist die Anweisung `Print #`. `Print #` schreibt nur in Dateien mit sequentiellem Zugriff. Hier das Format von `Print #`:

```
Print #intDateiNummer, [OutputList]
```

`intDateiNummer` ist die Dateinummer der geöffneten Datei, in die Sie schreiben möchten. `OutputList` kann wie folgt aussehen:

```
[Spc(intN1) | Tab((intN2))] [Ausdruck] [charPos]
```



`Spc()` und `Tab()` arbeiten in der Anweisung `Print #` so, wie Sie es von der `Print`-Anweisung her gewohnt sind, die Sie in Kapitel 11 kennengelernt haben.

Sie können `Spc()` oder `Tab()` verwenden, jedoch nicht beides gleichzeitig. Tabelle 12.2 erklärt die Komponenten von `OutputList`.

Tabelle 12.2: Der Inhalt der Anweisung `Print #` beschreibt die Ausgabe der Methode

Komponente	Beschreibung
<code>Spc(intN1)</code>	Fügt Leerzeichen in die Ausgabe ein, wobei <code>intN1</code> die Anzahl der Leerzeichen angibt.
<code>Tab(intN2)</code>	Positioniert den Einfügepunkt als absolute Spaltennummer, wobei <code>intN2</code> diese Spaltennummer darstellt. <code>Tab</code> ohne Argument positioniert den Einfügepunkt am Anfang der nächsten Ausgabezone (eine <i>Ausgabezone</i> umfaßt je 14 Leerzeichen).
<code>Ausdruck</code>	Ein numerischer oder String-Ausdruck, der die Daten enthält, die Sie an die Datei senden möchten.
<code>charPos</code>	Gibt den Einfügepunkt für das nächste Zeichen an. Ein Semikolon legt fest, daß das nächste Zeichen unmittelbar nach dem letzten ausgegebenen Zeichen erscheint.



In `charPos` kann `Tab()` verwendet werden; es übernimmt hier dieselben Aufgaben, die auch am Anfang einer `Print #-`Anweisung ausgeführt werden. Wenn Sie `charPos` weglassen, erscheint das nächste Zeichen in der nächsten Zeile der Datei.

Die Prozedur in Listing 12.2 öffnet die Datei `Print.txt`, schreibt die Zahlen 1 bis 6 in die Datei und schließt die Datei dann wieder.

Listing 12.2: Mit `Print #` in eine sequentielle Datei schreiben

```
1: Private Sub cmdFile_Click()  
2: Dim intCtr As Integer ' Schleifenzähler  
3: Dim intFNum As Integer ' Dateinummer  
4: Dim intMsg As Integer ' Für MsgBox()  
5: intFNum = FreeFile  
6: ' Pfad gegebenenfalls ändern  
7: Open "C:\Print.txt" For Output As #intFNum  
8:  
9: ' Diese Prozedur beschreiben  
10: intMsg = MsgBox("Datei Print.txt geöffnet")  
11:  
12: For intCtr = 1 To 6  
13: Print # intFNum, intCtr ' Schleifenzähler ausgeben  
14: intMsg = MsgBox("Schreibe " & intCtr & " in Print.txt")  
15: Next intCtr  
16:  
17: Close # intFNum  
18:  
19: intMsg = MsgBox("Datei Print.txt wurde geschlossen")  
20: End Sub
```

Wenn Sie diese Prozedur ausführen, sehen Sie mehrere Meldungsfelder zur Fortschrittsanzeige. Die Prozedur teilt Ihnen in Zeile 10 mit, daß sie die Datei geöffnet hat. In Zeile 13 erfahren Sie, daß sie in die Datei schreibt, und Zeile 14 zeigt an, was geschrieben wurde. In Zeile 17 schließt die Prozedur die Datei und teilt Ihnen in Zeile 19 mit, daß die Datei geschlossen wurde.

Um sicherzustellen, daß die Prozedur funktioniert hat, öffnen Sie den Editor und betrachten die Datei `Print.txt`. Sie sollten hier die Ziffern 1 bis 5 sehen:

```
1  
2  
3  
4  
5
```

Listing 12.2 demonstriert eine einfache `Print #-`Anweisung (Zeile 13). Es gibt keine Anweisung, die die Ausgabe positioniert, die Prozedur gibt also jede Ziffer in einer neuen Zeile aus, wie vom Standard vorgegeben.

Es ist nicht besonders sinnvoll, eine Datei anzulegen und in sie zu schreiben, wenn man die Informationen nicht mehr daraus lesen kann. Der folgende Abschnitt zeigt, wie man Dateien aus einer Datei liest.

Mit Input # eine Datei lesen

Nachdem Sie Daten in eine Datei geschrieben haben, möchten Sie diese irgendwann wieder lesen. Bei sequentiellen Dateien verwenden Sie die Anweisung **Input #**, um Daten sequentiell zu lesen. Sie müssen die Daten in genau der Reihenfolge und in genau dem Format lesen, wie Sie sie geschrieben haben. Hier das Format von **Input #**:

```
Input # intDateiNummer, Variable1[, Variable2][, ...VariableN]
```

Für **Input #** brauchen Sie eine Dateinummer und Variablen, die Daten aufnehmen, die Sie lesen. Die Anweisungen **Input #** und **Print #** sollten dasselbe Format verwenden. Wenn Sie die Daten mit Trennzeichen geschrieben haben, sollten Sie dieselben Trennzeichen für **Input #** verwenden.



Wenn Sie mehrere Variablen in eine Zeile schreiben und sie mit **Input #** zuverlässig wieder lesen möchten, sollten Sie anstatt des **Print #** ein **Write #** verwenden oder manuell Kommata-Trennzeichen einfügen. **Input #** liest in einer numerischen Variablen bis zum ersten Leerzeichen, Komma oder EOL-Zeichen. In einem String liest es bis zum ersten Komma oder EOL-Zeichen, es sei denn, der String enthält Anführungszeichen.

Die folgende Anweisung liest fünf Variablen aus einer geöffneten sequentiellen Datei. Die Variablen befinden sich alle in derselben Zeile:

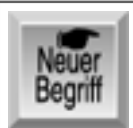
```
Input #intDateinummer V1, V2, V3, V4, V5
```

Die Anweisung **Print #**, mit der die Datei angelegt wurde, muß mit dem Format der Anweisung **Input #** übereinstimmen und dieselben Variablentypen verwenden. Andernfalls kann **Input #** die Daten nicht lesen.

Input # ist relativ einfach, weil es die entgegengesetzte Aufgabe von **Print #** ausführt. Wie der nächste Abschnitt erklärt, gibt **Write #** Daten häufig in einem allgemeineren Format als **Print #** aus, so daß Sie sich weniger darum kümmern müssen, ob die Anweisung **Input #** dem ursprünglich ausgegebenen Code entspricht.

Die Anweisung Write # gibt auch Ausgaben in Dateien aus

Es gibt noch einen Befehl, mit dem Sie Informationen in eine sequentielle Datei schreiben: **Write #**. **Write #** und **Print #** unterscheiden sich nur unmerklich. Alle Daten, die **Write #** in eine Datei schreibt, werden durch Kommata voneinander getrennt. Außerdem schließt **Write #** automatisch alle Stringdaten in Anführungszeichen ein (die Anführungszeichen erscheinen auch in der Datei); es schließt alle Datendaten in Pfundzeichen ein, schreibt Boolesche Daten als **#TRUE#** oder **#FALSE#** und sendet Null-Daten und Fehlercodes als **#NULL#** bzw. **#Error fehlercode#** an die Datei. **fehlercode#** ist eine Fehlernummer, die den Fehler erklärt, der aufgetreten ist. Eine vollständige Liste der Fehlercodes und ihre Bedeutung finden Sie in der Online-Hilfe von Visual Basic.



In einer *Datei mit Komma-Trennzeichen* werden die Daten durch Kommas voneinander abgetrennt.

Die Abtrennung von Daten durch Kommas ist insbesondere für Dateien erforderlich, die von

bestimmten Mail- und Tabellenkalkulationsprogrammen gelesen werden. Die Kommas machen das Lesen weniger fehleranfällig, weil nachfolgende **Input-#**-Anweisungen nicht genau mit der **Write-#**-Anweisung übereinstimmen müssen.



Um Daten einfacher lesen zu können, sollten Sie sie immer mit **Write #** anstelle von **Print #** schreiben.

Hier das Format von **Write #**:

Write # *intDateiNummer*, [*OutputList*]

OutputList ist die Liste mit einer oder mehreren Variablen, die Sie aus der für die Dateinummer geöffneten Datei lesen möchten.

Wenn Sie Werte in einer Zeile hintereinander ausdrucken möchten, geben Sie ein Semikolon hinter der Variablen **intCtr** aus, wie in Listing 12.3 gezeigt.

Listing 12.3: Mit Hilfe des Semikolons schreiben Sie mehrere Werte in eine einzige Zeile

```
1: Private Sub cmdFile_Click()  
2: Dim intCtr As Integer ' Schleifenzähler  
3: Dim intFNum As Integer ' Dateinummer  
4: Dim intMsg As Integer ' MsgBox()-Rückgabe  
5: intFNum = FreeFile  
6: ' Ändern Sie gegebenenfalls den Pfad  
7: Open "C:\Print.txt" For Output As #intFNum  
8:  
9: ' Diese Prozedur beschreiben  
10: intMsg = MsgBox("Datei Print.txt wurde geöffnet")  
11:  
12: For intCtr = 1 To 6  
13: Print # intFNum, intCtr; ' Beachten Sie das Semikolon!!  
14: intMsg = MsgBox("Schreiben von " & intCtr & " in Print.txt")  
15: Next intCtr  
16:  
17: Close # intFNum  
18:  
19: intMsg = MsgBox("Datei Print.txt wurde geschlossen")  
20: End Sub
```

Bei der Ausführung dieser Prozedur enthält die angelegte Datei die folgenden Daten:

1 2 3 4 5 6

Beachten Sie die Leerzeichen zwischen den Zeichen, die in eine Zeile ausgegeben wurden. **Print #** fügt die Leerzeichen ein, weil vor allen positiven Zahlen das imaginäre Pluszeichen ausgegeben wird.

Sie sollten ein bißchen mit den Parametern für **Print #** experimentieren, um zu sehen, welche Ergebnisse sie für Ihre Dateien erzeugen.

Nachdem Sie Daten in eine Datei geschrieben haben, erfolgt das Lesen häufig in einer anderen Prozedur oder sogar in einer anderen Applikation. Die Prozeduren in Listing 12.4 zeigen Beispiele

dafür, wie Sie Dateien schreiben und die Daten dann in Variablen zurücklesen.

Listing 12.4: Sie können innerhalb derselben Prozedur in eine Datei schreiben und daraus lesen

```
1: Private Sub cmdFileOut_Click ()
2: ' Die sequentielle Datei anlegen
3: Dim intCtr As Integer ' Schleifenzähler
4: Dim intFNum As Integer ' Dateinummer
5: intFNum = FreeFile
6:
7: Open "Print.txt" For Output As #intFNum
8:
9: For intCtr = 1 To 5
10: Print # intFNum, intCtr; ' Schleifenzähler schreiben
11: Next intCtr
12:
13: Close # intFNum
14: End Sub
15:
16: Private Sub cmdFileIn_Click ()
17: ' Sequentielle Datei lesen
18: Dim intCtr As Integer ' Schleifenzähler
19: Dim intVal As Integer ' Werte lesen
20: Dim intFNum As Integer ' Dateinummer
21: Dim intMsg As Integer ' MsgBox()
22: intFNum = FreeFile
23: Open "Print.txt" For Input As #intFNum
24:
25: For intCtr = 1 To 6
26: Input # intFNum, intVal
27: ' Ergebnisse anzeigen
28: intMsg = MsgBox("Geladen wurde " & intVal & " aus Print.txt")
29: Next intCtr
30:
31: Close # intFNum
32: intMsg = MsgBox("Die Datei Print.txt wurde geschlossen.")
33: End Sub
```

Nachdem die Ausführung der ersten Prozedur in Listing 12.4 die Daten in die Datei geschrieben hat, kann die Prozedur `cmdFileIn_Click()` sie aus der Datei lesen.

Jetzt betrachten Sie die Prozedur in Listing 12.5, die eine Datei namens `Write.txt` anlegt.

Listing 12.5: Mit Write Ausgaben in sequentielle Dateien vornehmen

```
1: Private Sub cmdFile_Click ()
2: Dim intCtr As Integer ' Schleifenzähler
3: Dim intFNum As Integer ' Dateinummer
4: intFNum = FreeFile
5:
6: Open "c:\Write.txt" For Output As #intFNum
7:
```

```

8: For intCtr = 1 To 5
9: Write # intFNum, intCtr; ' Schleifenzähler schreiben
10: Next intCtr
11:
12: Close # intFNum
13: End Sub

```

Nach Ausführung dieser Prozedur prüfen Sie mit dem Editor, welche Datei erzeugt wurde. Sie werden den Unterschied zwischen den Anweisungen `Print #` und `Write #` sofort erkennen. Hier der Inhalt von `Write.txt`:

1, 2, 3, 4, 5,

Wenn Sie hinter den geschriebenen Daten kein Semikolon angeben, werden diese in jeweils separaten Zeilen abgelegt, und die Daten werden nicht durch Kommata getrennt, weil das bei einem einzigen Wert pro Zeile nicht nötig ist. (In diesem Fall verhalten sich `Write #` und `Print #` identisch.)



Wenn Sie häufig sequentielle Dateien einsetzen, werden Sie schnell lernen, Ihren Code zu verbessern. Beispielsweise könnten Sie als ersten Datenwert in der Datei die Anzahl der enthaltenen Werte ablegen. Auf diese Weise wissen alle folgenden Programme, die die Daten lesen, wie viele Werte es gibt, und können eine entsprechende Schleife verwenden.



Sie haben in diesem Kapitel schon viel gelernt, aber Sie werden überrascht sein: Visual-Basic-Programmierer verwenden die sequentielle Dateiverarbeitung höchst selten! Sie vergeuden mit den hier beschriebenen Themen jedoch nicht Ihre Zeit, weil Sie für alle gebräuchlichen Dateizugriffsmethoden und -Steuerelemente die Grundlagen brauchen, die Sie für die sequentielle Dateiverarbeitung kennengelernt haben.

Dateien mit wahlfreiem Zugriff

Dateien mit wahlfreiem Zugriff können in beliebiger Reihenfolge gelesen und geschrieben werden. Beispielsweise können Sie Ihre Kundendatensätze in eine solche Datei schreiben und später daraus beliebige Datensätze lesen. Wäre die Kundendatei sequentiell, müßten Sie jeden Datensatz in der Datei lesen, der dem gesuchten Datensatz vorausgeht.



Eine *Datei mit wahlfreiem Zugriff* kann in beliebiger Reihenfolge gelesen oder geschrieben werden, ohne daß Sie alle anderen Daten in der Datei lesen müssen.

Wie den sequentiellen Zugriff verwenden die Programmierer heute auch den wahlfreien Zugriff nicht mehr in seiner eigentlichen Form, weil es viele Datenzugriffs-Steuerelemente und komplexe Prozeduren zur Dateiverarbeitung gibt. Nichtsdestotrotz basieren die meisten datenbankbezogenen Dateizugriffe auf den Konzepten, die Sie hier kennenlernen werden.

Dateien mit wahlfreiem Zugriff stellen eine gute Gelegenheit dar, neue Programmtechniken vorzustellen, die sogenannten *benutzerdefinierten Datentypen*. Dateien mit wahlfreiem Zugriff lesen und schreiben häufig Datensätze. Mit Hilfe der benutzerdefinierten Datentypen von Visual Basic definieren Sie Datenelemente, die genau so aussehen wie die Datensätze, die geschrieben und gelesen

werden sollen.



Ein *benutzerdefinierter Datentyp* wird von Ihnen selbst definiert und anstelle eines der eingebauten Datentypen (wie z.B. **Integer** und **String**) verwendet.

Wahlfreier Zugriff

Ein Großteil der Arbeit mit Dateien mit wahlfreiem Zugriff entspricht der sequentiellen Dateiverarbeitung. Beispielsweise verhalten sich die Anweisungen **Open** und **Close** für beide Dateitypen gleich. Der einzige Unterschied ist der Zugriffsmodus.



Wenn Sie Visual Basic nicht mitteilen, in welchem Modus es eine Datei öffnen soll, verwendet es **For Random**. Angenommen, Sie schreiben folgendes:

```
Open "Random.txt" As #1
```

Visual Basic ändert diese Zeile in:

```
Open "Random.txt" For Random As #1
```

Die folgende Anweisung öffnet eine Datei für wahlfreien Zugriff:

```
Open "Random.txt" For Random As #1
```

Sie können eine Datei als Datei mit wahlfreiem Zugriff öffnen und sie dann für sequentiellen Zugriff verwenden. Sie opfern bei der Verarbeitung die Vorteile einer Datei mit wahlfreiem Zugriff, aber das ist teilweise gewollt, insbesondere wenn Sie die Datensätze in einer vordefinierten Reihenfolge eingegeben haben und jetzt einen Bericht drucken oder die Dateidaten in genau dieser Reihenfolge anzeigen möchten.

Betrachten Sie ein Beispiel für den Unterschied zwischen sequentiellem und wahlfreiem Zugriff. Angenommen, Sie legen eine Datei an, die 10 Zeilen mit Inventardaten enthält. Um im sequentiellen Modus die sechste Zeile zu lesen, müssen Sie die ersten fünf Einträge lesen, um zum sechsten zu gelangen, und danach müssen Sie die letzten vier Einträge lesen. Im wahlfreien Modus greifen Sie direkt auf den sechsten Datensatz zu, lesen die Daten und schließen die Datei.

Dasselbe gilt für das Schreiben in eine Datei. Wir gehen wieder von derselben 10zeiligen Datei aus. Sie möchten den achten Datensatz ändern. Mit sequentiellem Zugriff müssen Sie alle 10 Datensätze in der Datei lesen, den achten ändern und alle 10 Datensätze wieder zurück in die Datei schreiben. Im wahlfreien Modus greifen Sie direkt auf den achten Datensatz zu und ändern diesen ab.

Bei einer Datei mit 10 Datensätzen stellt die Verwendung einer Datei mit wahlfreiem Zugriff keine großen Vorteile dar. Bei einer Datei mit 10000 Datensätzen sparen Sie durch den wahlfreien Zugriff jedoch sehr viel Zeit und verringern außerdem den System-Overhead.

Get und Put

Für Dateien mit wahlfreiem Zugriff werden zwei Anweisungen verwendet: **Put #** und **Get #**. Diese Anweisungen sind **Print #** und **Input #** für den sequentiellen Zugriff sehr ähnlich. Der größte Unterschied zwischen diesen Anweisungen ist, daß **Print #** und **Input #** hintereinander alle Datenabschnitte betrachten und sich so durch die Datei arbeiten. Es gibt keine Möglichkeit, eine

Positionierung auf einen bestimmten Datensatz vorzunehmen, um die Verarbeitung auf genau diesen zu begrenzen.

Das Format von `Put #` und `Get #` unterscheidet sich von `Print #` und `Input #`:

```
Put [#]intFileNum, [intRecNum,] Variable
```

```
Get [#]intFileNum, [intRecNum,] Variable
```

Wie Sie sehen, verwenden diese Anweisungen eine Datensatznummer. Durch die Angabe der Datensatznummer ist es möglich, ganz bestimmte Daten zu lesen oder zu aktualisieren.

Datensatznummern beginnen bei 1. Die Variable, die Sie lesen oder schreiben, kann einen beliebigen Datentyp haben - es darf sich sogar um ein Array oder eine benutzerdefinierte Variable handeln. Die Freiheit, beliebige Variablentypen als Einheit zu verwenden, ist die leistungsfähigste Eigenschaft von Dateien mit wahlfreiem Zugriff.

Die Beispiele im nächsten Abschnitt beinhalten Prozeduren, die bestimmte Datensätze in Dateien mit wahlfreiem Zugriff lesen und schreiben.

Benutzerdefinierte Datentypen

Sie haben in diesem Buch bereits Variablen und Arrays kennengelernt. Jetzt werden Sie erfahren, wie Sie aus vorhandenen Datentypen eigene Datentypen erzeugen. Benutzerdefinierte Datentypen werden manchmal auch als *Strukturen* oder *Records* bezeichnet.

Für ein Programm zur Verwaltung eines Adreßbuchs könnten Sie für jedes der darin verwendeten Felder eine einzelne Variable anlegen, beispielsweise `strFName` für den Vornamen, `strLName` für den Nachnamen usw. Dieser Ansatz würde funktionieren, aber so eine Programmierung ist müßig, wenn Sie sehr viele Kontakte zu verwalten haben. Viel einfacher wäre es, einen benutzerdefinierten Datentyp zu verwenden, der dieselben Informationen enthält und diese als eine Einheit behandelt.

Ein benutzerdefinierter Datentyp gruppiert existierende Datentypen, um daraus einen neuen Datentyp anzulegen. Diese Gruppierung wird auch als *zusammengesetzte Deklaration* bezeichnet.



Ein benutzerdefinierter Datentyp setzt sich aus bereits existierenden Datentypen zusammen (dabei handelt es sich um eingebaute Datentypen von Visual Basic oder um andere benutzerdefinierte Datentypen).

Eigene Datentypen werden mit der `Type`-Anweisung angelegt. Sie hat das folgende Format:

```
[Private | Public] Type TypeName
VarName1[(ArrayGröße)] As ExistierenderTyp [* StringLänge]
VarName2[(ArrayGröße)] As ExistierenderTyp [* StringLänge]
:
:
End Type
```

Beachten Sie, daß der Name des benutzerdefinierten Datentyps dem Schlüsselwort `Type` folgt. Als Name kann ein Wort verwendet werden, das kein reserviertes Wort, kein Schlüsselwort und kein deklarerter Variablenname ist. Wenn Sie beispielsweise den Variablennamen `Client` bereits deklariert haben, können Sie keinen benutzerdefinierten Datentyp namens `Client` anlegen.

Sie deklarieren alle benutzerdefinierten Typen auf Modulebene; es ist nicht erlaubt, sie innerhalb von Prozeduren zu deklarieren. Sie können einen Typ in einem Formularmodul deklarieren, dort aber als **Private**. Der Datentyp ist dann privat für den Code des Formularmoduls.

Betrachten Sie den Code in Listing 12.6. Dort erfahren Sie mehr über die **Type**-Anweisung.

Listing 12.6: Eigene Datentypen werden mit der Type-Anweisung deklariert

```
1: ' Modul-Seite des Projekts
2: Type UserType
3: strFName As String
4: strLName As String
5: End Type
6: Public Names As UserType
```

Dieser Code erzeugt, beginnend in Zeile 2, den benutzerdefinierten Datentyp **UserType**. Der neue Datentyp enthält zwei Strings, **strFName** und **strLName**. Zeile 6 erzeugt die Variable **Names** mit dem Typ **UserType**.



UserType ist keine Variable, sondern ein von Ihnen definierter Typ. **Names** ist der Variablenname, und **strFName** und **strLName** sind *Elemente* (oder *Felder*) der Variablen. Damit haben Sie der Programmiersprache Visual Basic für die Dauer der Programmausführung einen neuen Datentyp hinzugefügt. Neben **Integer** und **Boolean** können Sie damit auch Variablen des Datentyps **UserType** deklarieren.



Um auf die einzelnen Felder eines Datentyps zuzugreifen, geben Sie den Variablennamen, einen Punkt und dann den Feldnamen an.

Die folgenden Anweisungen initialisieren die oben deklarierte Variable und arbeiten mit ihr:

```
Names.strFName = "John"
Names.strLName = "Doe"
lblFName.Caption = "First Name: " & Names.strFName
lblLName.Caption = "Last Name: " & Names.strLName
```

Mit der Option *** StringLänge** hinter dem Feldtyp **As String** begrenzen Sie die Größe der String-Variablen, die in einer Struktur verwendet werden. Der String fester Länge setzt die absolute Länge des Strings auf **StringLänge**. Das ist normalerweise erforderlich, wenn Sie Ihre Strukturen wahlfrei in eine Datei schreiben oder daraus lesen. Sie brauchen eine fixe String-Länge, um sicherzustellen, daß jedes Wort, das in die Datei geschrieben wird, dieselbe Größe hat, so daß Sie problemlos darauf zugreifen können.

Listing 12.7 zeigt, wie die String-Größen aus Listing 12.6 konstant gemacht werden.

Listing 12.7: Feste String-Größen bestimmen die Länge der geschriebenen Strings

```
1: ' Modul-Seite des Projekts
2: Type UserType2
3: strFName As String * 8
4: strLName As String * 20
5: End Type
```

6: Public Names As UserType2

Die Strings fester Länge begrenzen die String-Länge auf ein unveränderliches Maximum. Falls die String-Daten nicht den gesamten Speicher belegen, den Sie für den String reserviert haben, füllt Visual Basic die restliche Länge mit Leerzeichen auf, um sicherzustellen, daß alle Variablen vom Typ UserType2 mit derselben Datensatzlänge geschrieben werden, unabhängig davon, welchen Inhalt sie haben.

Die Prozedur in Listing 12.8 demonstriert, wie man mit Dateien mit wahlfreiem Zugriff arbeitet.

Listing 12.8: Sie können in einen beliebigen Datensatz schreiben

```
1: Private Sub cmdCreate_Click()  
2: ' Diese Prozedur erzeugt die Datei  
3: Dim intFile As Integer ' Freie Dateinummer  
4: Dim intCtr As Integer ' Schleifenzähler  
5:  
6: intFile = FreeFile  
7: Open "c:\Random.Txt" For Random As #intFile Len = 5  
8:  
9: ' Datensätze durchlaufen und in die Datei schreiben  
10: For intCtr = 1 To 5  
11: Put # intFile, intCtr, intCtr  
12: Next intCtr  
13:  
14: Close intFile  
15: End Sub  
16:  
17: Private Sub cmdChange_Click()  
18: ' Diese Prozedur ändert den dritten Datensatz  
19: Dim intFile As Integer ' Freie Dateinummer  
20:  
21: intFile = FreeFile  
22: Open "c:\Random.Txt" For Random As #intFile Len = 5  
23:  
24: ' Einen neuen dritten Datensatz schreiben  
25: Put #intFile, 3, 9 ' Datensatz 3, Wert: 9  
26: Close # intFile  
27: End Sub  
28:  
29: Private Sub cmdDisplay_Click()  
30: ' Diese Prozedur zeigt die Datei an  
31: Dim intFile As Integer ' Freie Dateinummer  
32: Dim intVal As Integer ' Wert lesen  
33: Dim intCtr As Integer ' Schleifenzähler  
34: Dim intMsg As Integer ' Für MsgBox()  
35: intFile = FreeFile  
36: Open "c:\Random.Txt" For Random As #intFile Len = 5  
37:  
38: intMsg = MsgBox("Datei Random.Txt wurde geöffnet...")
```



```

39:
40: ' Datensätze durchlaufen und Datei schreiben
41: For intCtr = 1 To 5
42: Get # intFile, intCtr, intVal
43: intMsg = MsgBox(intVal & " aus Random.Txt gelesen")
44: Next intCtr
45: Close # intFile
46:
47: intMsg = MsgBox("Datei Random.Txt wurde geschlossen")
48: End Sub

```

Eingebettete benutzerdefinierte Typen

Sie wissen jetzt, wie man einen benutzerdefinierten Datentyp anlegt, aber was tun Sie, wenn Sie einen benutzerdefinierten Datentyp innerhalb eines anderen benutzerdefinierten Datentyps verwenden möchten? Eines der Felder soll also statt eines in Visual Basic eingebauten Typs einen benutzerdefinierten Datentyp haben. Dazu ist es erforderlich, daß der einzubettende benutzerdefinierte Datentyp *vor* dem benutzerdefinierten Datentyp deklariert wird, in den er eingebettet werden soll.

Listing 12.9 zeigt ein Beispiel für einen benutzerdefinierten Datentyp, `Address`, der als Feld in einen anderen benutzerdefinierten Datentyp eingebettet wird.

Listing 12.9: Benutzerdefinierte Datentypen können innerhalb anderer benutzerdefinierter Datentypen verwendet werden

```

1: ' Eingabe im Deklarationsbereich des Codemoduls
2: Type Address
3: strStreet As String
4: strCity As String
5: strZip As String
6: End Type
7:
8: Type UserType3
9: strFName As String * 10
10: strLName As String * 25
11: typAddr As Address ' Ein weiterer Datentyp
12: End Type
13:
14: Public Names As UserType3 ' Deklaration einer
    Applikationsvariablen

```

Listing 12.10 zeigt Code, der diese Felder initialisiert, und demonstriert, wie Sie in die Felder innerhalb von Feldern gelangen.

Listing 12.10: Nachdem Sie einen öffentlichen Datentyp definiert haben, kann jedes Modul ihn für Variablendeklarationen nutzen

```

1: Names.strFName = "Jonathan"
2: Names.strLName = "Doe"
3:
4: Names.typAddr.strStreet = "AnyStreet"
5: Names.typAddr.strCity = "AnyTown"

```

```

6: Names.typAddr.strZip "12345-9876"
7:
8: ' Mit den Daten arbeiten
9: lblFName.Caption = "Vorname: " & Names.strFName
10: lblLName.Caption = "Nachname: " & Names.strLName
11: lblAddr.Caption = "Straße: " & Names.strAddr.strStreet
12: lblCty.Caption = "Stadt: " & Names.strAddr.strCity
13: lblZip.Caption = "PLZ: " & Names.strAddr.strZip

```

Datei-Steuerelemente

In Kapitel 9 haben Sie erfahren, wie man mit Hilfe des Standarddialogs die Dialogfelder **Datei öffnen** und **Datei speichern** erzeugt. In diesen Dialogfeldern kann der Benutzer Dateien auswählen. Sie erlauben ihm, Ordner, Laufwerke und sogar vernetzte Laufwerke nach Daten zu durchsuchen. Beide Dialogfelder folgen dem Windows- Standard.

Bei der Arbeit mit Dateien, insbesondere mit sequentiellen Dateien und Dateien mit wahlfreiem Zugriff, die Sie in diesem Kapitel kennengelernt haben, brauchen Sie manchmal Zugriff auf ein Verzeichnis (das seit Windows 95 als *Ordner* bezeichnet wird), ein Laufwerk oder einen Dateinamen, ohne ein vollständiges Dialogfeld wie **Datei speichern** oder **Datei öffnen** anzuzeigen.

Visual Basic stellt drei spezielle Listenfelder zur Verfügung, die Ihnen helfen sollen, Verzeichnisse, Laufwerke und Dateien zu verwalten:

- **Verzeichnis-Listenfeld** - Hier wählt der Benutzer ein Verzeichnis aus.
- **Laufwerks-Listenfeld** - Hier wählt der Benutzer ein Laufwerk aus.
- **Datei-Listenfeld** - Hier wählt der Benutzer einen Dateinamen aus.

Abbildung 12.1 zeigt ein Formularfenster mit den drei speziellen Listenfeldern.



Sie fragen sich vielleicht, warum Visual Basic diese dateibezogenen Steuerelemente zur Verfügung stellt, nachdem Sie bereits das Standarddialog- Steuerelement kennen, wo Sie alle diese Steuerelemente in einem Dialogfeld haben, ohne sie einzeln auf einem Formular anlegen zu müssen. Diese Listen-Steuerelemente erlauben Ihnen, ganz spezielle Listen auf einem Formular anzulegen, wo Sie möglicherweise nur bestimmte Informationen über eine Datei brauchen. Beispielsweise könnte es sein, daß Sie Daten auf die Festplatte eines Benutzers schreiben möchten. Ihre Applikation legt dazu den Dateinamen und das Verzeichnis fest, aber möglicherweise möchten Sie den Benutzer fragen, auf welches Laufwerk die Information geschrieben werden soll.



Diese Listen arbeiten nicht in Kombination miteinander, es sei denn, Sie programmieren sie entsprechend. Wenn Sie beispielsweise drei Steuerelemente auf einem Formular anlegen und die Applikation ausführen, bewirkt der Wechsel des Laufwerks nicht gleichzeitig einen Wechsel des Verzeichnisses oder Dateinamens, die in den beiden anderen Listen angezeigt werden. Um die drei Steuerelemente synchron zu halten, müssen Sie eine Ereignisprozedur dafür schreiben.



Abbildung 12.1:
Für die Arbeit mit Dateien stehen Ihnen diese drei speziellen Listenfelder zur Verfügung.

Das Laufwerks-Listenfeld

Mit Hilfe des Laufwerks-Listenfelds wählt der Benutzer ein Laufwerk aus. Dieses Steuerelement ist intelligent genug, den Host-Computer zu durchsuchen und dabei festzustellen, welche Laufwerke - lokal und remote, Disketten, Festplatten und CD-ROMs - es auf dem System des Benutzers gibt. Die Auswahl wird grafisch angezeigt, wenn der Benutzer das Laufwerks-Listenfeld öffnet (siehe Abbildung 12.2).



Das Laufwerks-Listenfeld zeigt zuerst das Laufwerk an, in dem der Benutzer die Applikation gestartet hat, Sie können dieses Standardlaufwerk aber auch überschreiben, indem Sie im Code auf ein anderes Laufwerk verweisen.

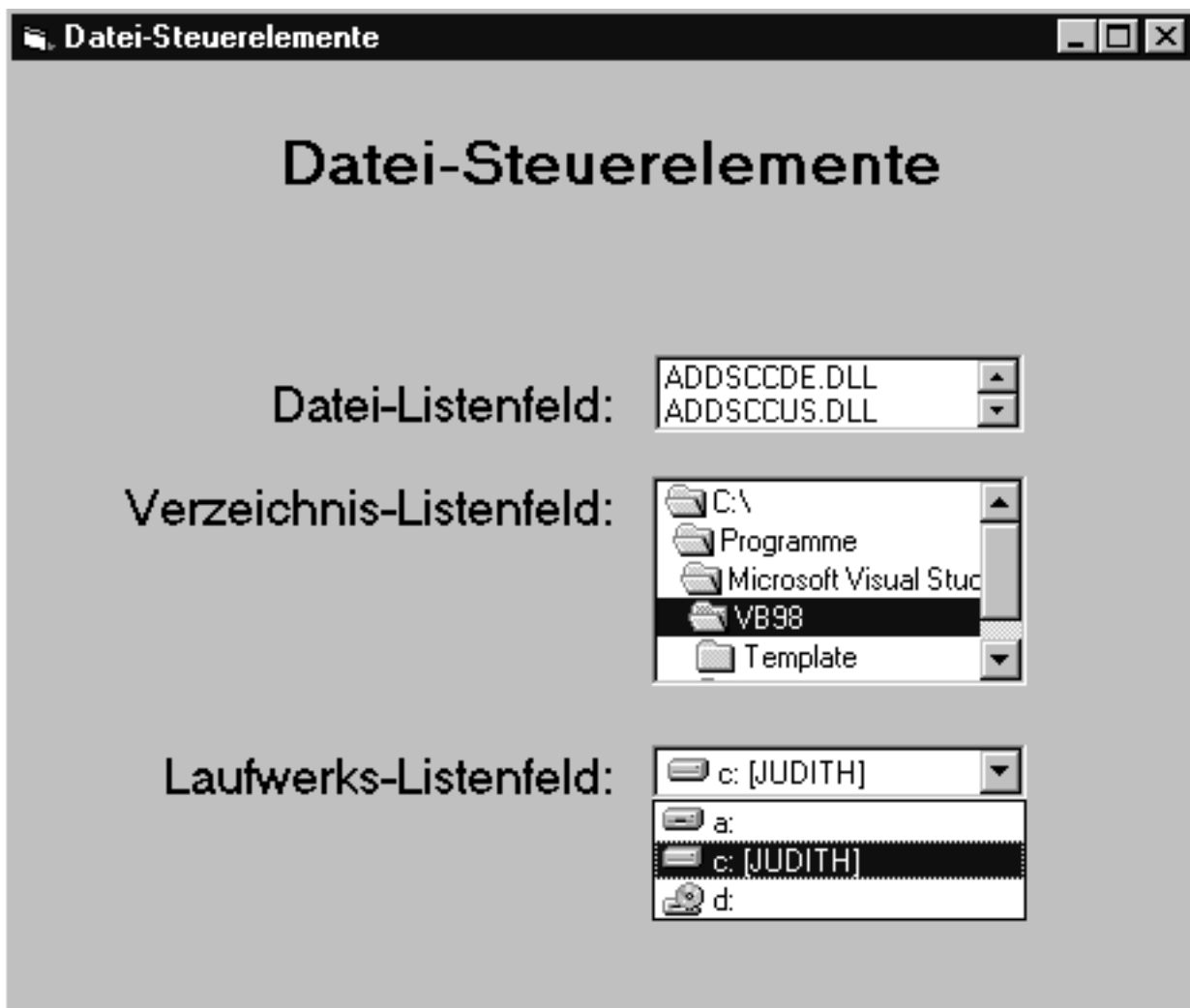


Abbildung 12.2:
Im Laufwerks-Listenfeld wählen die Benutzer ein Laufwerk aus.

Das Verzeichnis-Listenfeld

Im Verzeichnis-Listenfeld wählt der Benutzer einen Verzeichnisordner aus. Dieses Steuerelement ist intelligent genug, um den Host-Computer zu durchsuchen und dabei festzustellen, welche Verzeichnisse es auf diesem System gibt. Das Verzeichnis-Listenfeld zeigt diese Auswahlen im Standardformat von Windows grafisch an.

Sie wissen, daß das Verzeichnis-Listenfeld nicht erkennt, welches Laufwerk im Laufwerks-Listenfeld ausgewählt wurde. Dazu müssen Sie das Laufwerk mit dem Verzeichnis verknüpfen, wie am Ende dieses Kapitels gezeigt wird.



Das Verzeichnis-Listenfeld zeigt zunächst das Verzeichnis an, von dem aus der Benutzer die Applikation gestartet hat, aber Sie können dieses Standardverzeichnis überschreiben, indem Sie im Code auf ein anderes Verzeichnis verweisen.

Das Datei-Listenfeld

Im Datei-Listenfeld wählt der Benutzer eine Datei aus. Dieses Steuerelement ist intelligent genug, den Host-Computer zu durchsuchen und dabei festzustellen, welche Dateien im Dateisystem vorhanden sind. Das Datei-Listenfeld zeigt diese Auswahl im Standardformat von Windows grafisch an.

Wie das Verzeichnis-Listenfeld kann auch das Datei-Listenfeld nicht feststellen, welches Laufwerk (oder Verzeichnis) im Laufwerks- oder Verzeichnis-Listenfeld ausgewählt wurde. Sie müssen die Steuerelemente verknüpfen, wie Sie später in diesem Kapitel noch lernen.



Das Datei-Listenfeld zeigt zuerst die Dateien aus dem Verzeichnis an, in dem der Benutzer die Prozedur gestartet hat. Dieses Standardverzeichnis kann überschrieben werden, indem der Code auf ein anderes Verzeichnis verweist, bevor das Datei-Listenfeld mit dem Ordner verknüpft wird.

Dateibezogene Befehle

Visual Basic unterstützt mehrere Befehle für Laufwerke und Verzeichnisse, die auf Dateilisten-Steuerelemente angewendet werden. Diese Befehle sind in Tabelle 12.3 aufgelistet.

Tabelle 12.3: Die Programmiersprache Visual Basic enthält Befehle für Laufwerke und Verzeichnisse

Befehl	Beschreibung
ChDrive <i>strDrive</i>	Setzt das Standardlaufwerk auf das im String-Ausdruck angegebene Laufwerk.
ChDir <i>strDirectory</i>	Setzt das Standardverzeichnis auf das im String-Ausdruck angegebene Verzeichnis. Falls Sie im String kein Laufwerk angeben, wählt Visual Basic das Verzeichnis auf dem aktuellen Laufwerk aus.
Kill <i>strFileSpc</i>	Löscht die im String-Ausdruck durch Wildcards spezifizierten Dateien.
MkDir <i>strDirectory</i>	Legt das im String-Ausdruck angegebene Verzeichnis an.
RmDir <i>strDirectory</i>	Löscht das im String-Ausdruck angegebene Verzeichnis.



RmDir erzeugt einen Fehler, wenn Sie versuchen, ein Verzeichnis zu entfernen, das noch Dateien enthält.

Neben den in Tabelle 12.3 gezeigten Anweisungen unterstützt Visual Basic auch die Funktion `Dir()`, die prüft, ob es Dateien gibt, und `CurDir()`, die den Namen des aktuellen Verzeichnisses zurückgibt.

Angenommen, Sie möchten, daß das Laufwerks-Listenfeld und das Verzeichnis-Listenfeld auf das Verzeichnis `C:\MyFiles` verweisen. Geben Sie in der Prozedur `Form_Load()` den folgenden Code ein:

```
ChDrive "C:"  
ChDir "\MyFiles"
```

Das Laufwerks-Listenfeld, das Verzeichnis-Listenfeld und das Datei-Listenfeld zeigen jetzt auf das Verzeichnis `C:\MyFiles`, wenn sie auf dem Formular angezeigt werden, und nicht mehr auf das aktuelle Verzeichnis der Applikation.

Die `Dir()`-Funktion bedarf einiger Erklärungen. Angenommen, Sie möchten wissen, ob die Datei `SALES98.DAT` im Hauptverzeichnis Ihres Laufwerks D existiert. Dazu führen Sie den folgenden Code aus:

```
If (Dir("c:\SALES98.DAT")) = "SALES98.DAT" Then
intMsg = MsgBox ("Die Datei existiert")
Else
intMsg = MsgBox ("Die Datei existiert nicht")
End If
```

Die `Dir()`-Funktion gibt den Dateinamen zurück, den Sie als Argument übergeben. Der Dateiname wird nur zurückgegeben, wenn sich diese Datei in dem angegebenen Verzeichnisargument befindet. Falls `Dir()` den Dateinamen nicht zurückgibt, existiert die Datei nicht auf dem Laufwerk.

Sie können `Dir()` einen Dateinamen übergeben, der Wildcards enthält:

```
Dir ("c:\Sales*.DAT")
```

Zusammenfassung

Dieses Kapitel hat die Arbeit mit Dateien vorgestellt. Mit diesen Grundlagen sind Sie auf die komplexeren Steuerelemente und Befehle beispielsweise für die Arbeit mit Datenbanken vorbereitet.

Heute haben Sie gelernt, sequentielle Dateien und Dateien mit wahlfreiem Zugriff zu lesen und zu schreiben. Diese Dateien sind praktisch für die Aufnahme von Textwerten. Die Programmierung solcher Dateien ist mit Hilfe der Visual-Basic-Befehle relativ einfach.

Die Dateisteuerelemente in Ihrer Werkzeugsammlung bieten Zugriff auf das Laufwerk, das Verzeichnis und die Dateien auf dem System Ihres Benutzers. Um die Dialogfelder **Datei öffnen** und **Datei speichern** bereitzustellen, verwenden Sie am besten das Standarddialog-Steuerelement. Für spezielle Informationen über ein Laufwerk, ein Verzeichnis oder eine Datei verwenden Sie jedoch am besten die Dateisteuerelemente.

Im nächsten Kapitel erfahren Sie, wie Sie etwas auf ein anderes Gerät schreiben - den Drucker.

Fragen und Antworten

Frage:

Warum werden die benutzerdefinierten Datentypen als »benutzerdefiniert« bezeichnet, obwohl der Benutzer eigentlich nichts mit ihrer Definition zu tun hat?

Antwort:

Der Begriff beinhaltet zwar das Wort »Benutzer«, aber der Datentyp wird durch den Programmierer definiert und genutzt. In gewisser Weise ist der Programmierer der Benutzer des neuen Datentyps, weil er ihn in seiner Applikation verwendet.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt

haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Wie viele geöffnete Dateien können Sie mit einer `Close`-Anweisung schließen?
2. Welche Funktion gibt die nächste freie Dateinummer zurück?
3. Was passiert, wenn Sie eine Datei für sequentielle Ausgaben öffnen, diese Datei aber schon existiert?
4. Was passiert, wenn Sie eine Datei öffnen, um sequentiell Daten einzufügen, diese Datei aber schon existiert?
5. Welchen Dateityp öffnet die folgende Anweisung?
`Open "TestFile.dat" For Append As #1`
6. Warum brauchen `Open`-Anweisungen für Dateien mit wahlfreiem Zugriff Informationen über die Länge der Daten?
7. Warum sollten Sie für einen benutzerdefinierten Datentyp die absolute String-Länge definieren, wenn Sie diese Strings in eine Datei mit wahlfreiem Zugriff schreiben und daraus lesen möchten?
8. Welche Visual-Basic-Anweisung definiert einen neuen Datentyp?
9. Richtig/Falsch. Der folgende Code deklariert eine neue benutzerdefinierte Variable namens `CustRec`:

```
Type CustRec
strFName As String * 10
strLName As String * 15
curBalance As Currency
blnDiscount As Boolean
End Type
```
10. Was ist der Unterschied zwischen einer `Dir ( )`-Funktion mit Argument und einer `Dir ( )`-Funktion ohne Argument?

Übungen

1. **Fehlersuche:** Frannie hat ein Problem. Sie erhält einen Fehler, wenn sie eine Applikation mit der folgenden Anweisung ausführt. Können Sie ihr die wahrscheinliche Ursache für diesen Fehler erklären? (Dabei soll vorausgesetzt sein, daß das Verzeichnis `Bill` existiert und sich im Hauptverzeichnis `C:` befindet.)
`Rmdir "C:\Bills"`
2. Schreiben Sie eine Prozedur, die eine sequentielle Datei anlegt, die die folgenden Informationen enthält: Name, Alter und Lieblingsfarbe. Legen Sie fünf Datensätze in dieser Datei an (die jeweils einen Namen, ein Alter und eine Lieblingsfarbe enthalten). Geben Sie diese Informationen in drei `For`-Schleifen ein. Tip: Initialisieren Sie drei Arrays, eines für jeden Wert, den Sie schreiben.
3. Legen Sie ein Dialogfeld an, das das Dialogfeld **Datei öffnen** nachbildet. Verwenden Sie nur die

Steuerelemente Laufwerks-Listenfeld, Verzeichnis-Listenfeld und Datei-Listenfeld und die Schaltflächen **OK** und **Abbrechen**. Schreiben Sie Code, so daß in einer Applikation mit diesen Dialogfeldern die Verzeichnis- und Dateilisten geändert werden, wenn der Benutzer ein anderes Laufwerk oder ein anderes Verzeichnis auswählt. (Für Ihre eigenen Applikationen werden Sie für eine solche Aufgabe das Standarddialog-Steuerelement verwenden, aber diese Übung soll demonstrieren, in welcher Beziehung die Datei-Steuerelemente zueinander stehen.)

❖Kapitel Inhalt Index ❖ Top Kapitel❖

Woche 2

Bonusprojekt 6: Dateien lesen und schreiben

Die Applikation in diesem Bonusprojekt demonstriert, wie sequentielle Dateien gelesen werden. Sie verwendet das Standarddialog-Steurelement, um die Namen der zu öffnenden Dateien vom Benutzer zu erfahren. Dieses Bonusprojekt baut auf den Prozeduren zur Arbeit mit Dateien auf, die Sie in Kapitel 12 kennengelernt haben. Das hier entwickelte Programm wird für die Anzeige von Dateien eingesetzt. Der Benutzer wählt eine Datei aus, die in einem Listenfeld erscheint. Das Programm führt die folgenden Aufgaben aus:

- Es ermöglicht dem Benutzer, im Dialogfeld **Öffnen** eine Datei auszuwählen.
- Es ermöglicht dem Benutzer, im Dialogfeld **Farbe** die Hintergrundfarbe der Datei zu ändern.
- Es ermöglicht dem Benutzer, das Formular zu vergrößern oder zu verkleinern. Nachdem das Formular eine neue Größe erhalten hat, werden die Steuerelemente entsprechend angepaßt.
- Es führt im Code zum Öffnen der Datei eine Fehlerverarbeitung durch, so daß der Benutzer auf die **Abbrechen**-Schaltfläche klicken kann, um die aktuelle Datei beizubehalten, und ohne eine andere Datei auszuwählen.
- Es ermöglicht dem Benutzer, Batch- oder Textdateien anzuzeigen (mit den Dateinamenerweiterungen .BAT bzw. .TXT), die allerdings maximal 4096 Byte groß sein dürfen.

Abbildung BP6.1 zeigt die Dateianzeige, nachdem eine Datei ausgewählt wurde.

Datei-Anzeige

Datei

Dateifarbe ändern

ausschliesslich unter der Voraussetzung, dass die nachfolgenden Lizenzbedingungen erfüllt sind und anerkannt werden:

Installation:

Um einem DATENVERLUST vorzubeugen, weisen wir ausdrücklich darauf hin, dass vor der Installation von neuer Software grundsätzlich eine Sicherung Ihrer Daten erfolgen sollte. Dies gilt ebenfalls für die Installation von cFos. Falls Sie Ihre Daten jetzt noch nicht gesichert haben, raten wir, die Installation abubrechen, Ihre Daten danach zu sichern und dann erst die Installation neu zu starten. Darüberhin empfehlen wir in regelmässigen Abständen Datensicherungen durchzuführen.

Garantie / Gewährleistung / Haftung

Absolut keine. Zur Klarstellung:

Abbildung BP6.1:
Der Benutzer kann in dem Listenfeld beliebige Text- oder Batch-Dateien anzeigen.

Zuerst legen Sie die grafischen Objekte auf dem Formular an. Anschließend schreiben Sie den Code, der die Aktionen des Benutzers unterstützt.

Die Oberfläche

Legen Sie auf Ihrem Formular die in Tabelle BP6.1 beschriebenen Steuerelemente an.

 Hinweis	Rücken Sie alle Menüoptionen im Menü-Editor ein, außer der ersten, Datei, die in der Menüleiste des Benutzers erscheinen soll. Durch die Einrückung aller nachfolgenden Menüoptionen veranlassen Sie, daß diese Einträge in der Drop-down-Liste für Datei erscheinen.

Tabelle BP6.1: Legen Sie diese Steuerelemente auf dem Formular an

Steuerelementeigenschaft	Eigenschaftswert
Formular Name	frmFile

Formular Caption	File Viewer
Formular Height	4620
Formular Width	6570
Listenfeld Name	lstFile
Listenfeld Height	2205
Listenfeld Left	720
Listenfeld Top	1320
Listenfeld Width	4815
Befehlsschaltfläche Name	cmdColor
Befehlsschaltfläche Caption	Datei&farbe ändern
Befehlsschaltfläche Height	495
Befehlsschaltfläche Left	2760
Befehlsschaltfläche Top	480
Befehlsschaltfläche Width	1215
Menüeintrag #1 Caption	&Datei
Menüeintrag #1 Name	mnuFile
Menüeintrag #2 Caption	&Öffnen
Menüeintrag #2 Name	mnuFileOpen
Menüeintrag #3 Caption	-
Menüeintrag #3 Name	mnuFileSep1
Menüeintrag #4 Caption	&Beenden
Menüeintrag #4 Name	mnuFileExit
Standarddialogfeld Name	comFile
Standarddialogfeld DialogTitle	Datei öffnen
Standarddialogfeld InitDir	c:\
Standarddialogfeld Filter	Text (*.txt) *.txt Batch (*.bat) *.bat
Standarddialogfeld CancelError	True
Standarddialogfeld MaxFileSize	4096

Eine der einfachsten Methoden, das Standarddialogfeld einzurichten, ist, den Eintrag (Benutzerdefiniert) im Eigenschaftfenster anzuklicken und das dafür angezeigte Dialogfeld zu initialisieren, wie in Abbildung BP6.2 gezeigt. Die Verwendung des Dialogfelds ist einfacher als die Eingabe von Werten im Eigenschaftfenster.

Abbildung BP6.2:

Im Dialogfeld für die benutzerdefinierten Eigenschaften geben Sie die Eigenschaften für das Standarddialogfeld ein.



Sie brauchen keine Eigenschaften für die Farben zu setzen. Das Dialogfeld für die Farbauswahl muß nicht mit Werten initialisiert werden, bevor Ihr Programm es anzeigt. Die Farbauswahl des Benutzers ändert keine der dateibezogenen Eigenschaften, die Sie für das Standarddialogfeld-Steuerelement setzen.

Codeeingabe

Listing BP6.1 zeigt den Ereigniscode, den Sie für dieses Projekt eingeben müssen.



Statt den Code zeilenweise einzugeben, können Sie sich bei der Eingabe auch vom Editor unterstützen lassen. Wenn Sie **Extras, Prozedur hinzufügen** wählen, öffnet Visual Basic das Dialogfeld, das Sie in Abbildung BP6.3 sehen. Geben Sie einfach den Namen der Prozedur, ihren Typ und ihren Gültigkeitsbereich ein. Visual Basic legt automatisch die erste und die letzte Zeile für die Prozedur an, die Sie dann nicht mehr schreiben müssen.

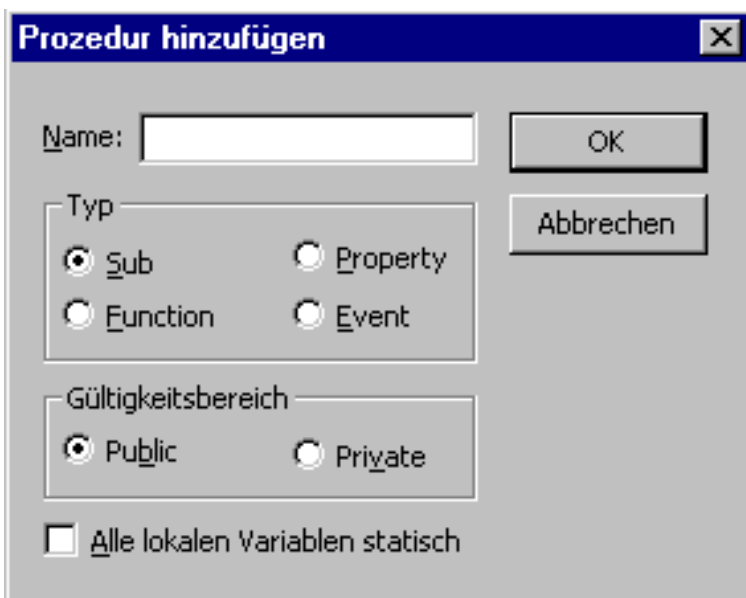


Abbildung BP6.3:
Visual Basic schreibt die erste und die letzte Zeile einer Prozedur für Sie.

Listing BP6.1: Eine Datei in ein Listenfeld laden

```

1: Private Sub cmdColor_Click()
2: ' Im Standarddialogfeld ändert der
3: ' Benutzer die Hintergrundfarbe des
4: ' Listenfelds.
5: comFile.ShowColor
6: lstFile.BackColor = comFile.Color
7: End Sub
8:
9: Private Sub Form_Resize()
10: Dim intMsg As Integer ' For MsgBox()
11: ' Ändert die Größe des Listenfelds, wenn der Benutzer
12: ' dem Formular eine andere Größe gibt
13: '
14: ' Diese Ereignisprozedur wird ausgeführt, wenn das
15: ' Formular zum ersten Mal geladen wird
16: '
17: ' Sicherstellen, daß das Formular nicht zu klein ist, so
18: ' daß das Listenfeld angezeigt werden kann
19: If (frmFile.Width < 400) Or (frmFile.Height < 3500) Then
20: ' Listenfeld verbergen
21: ' und den Benutzer warnen
22: lstFile.Visible = False
23: intMsg = MsgBox("Das Formular ist zu klein" , vbCritical)
24: Else
25: ' Listenfeld aktivieren, wenn es zuvor
26: ' deaktiviert wurde
27: lstFile.Visible = True
28: ' Listenfeld auf dem Formular ausrichten
29: ' Schaltfläche auf dem Formular ausrichten
30: lstFile.Width = frmFile.Width - 1440

```

```

31: lstFile.Height = frmFile.Height - 2500
32: cmdColor.Left = (frmFile.Width / 2) - 500
33: End If
34: End Sub
35:
36: Private Sub mnuFileExit_Click()
37: ' Option zum Beenden des Programms
38: End
39: End Sub
40:
41: Private Sub mnuFileOpen_Click()
42: Dim strFileLine As String
43: ' Falls der Benutzer auf Abbrechen klickt
44: On Error GoTo comErrorHandler
45: '
46: ' Dialogfeld Datei öffnen anzeigen
47: comFile.ShowOpen
48: ' Code fortsetzen, wenn der Benutzer auf OK klickt
49: ' oder zur Fehlerbehandlung gehen, wenn er auf Abbrechen klickt
50: '
51: ' Die vom Benutzer ausgewählte Datei öffnen
52: Open comFile.FileName For Input As #1
53: ' Platz für die neue Datei schaffen
54: lstFile.Clear
55: '
56: ' Eine Zeile aus der Datei lesen
57: Line Input #1, strFileLine
58: lstFile.AddItem strFileLine
59: '
60: ' Weiterlesen und in die Liste eintragen, bis
61: ' das Dateiende erreicht ist
62: Do Until (EOF(1))
63: Line Input #1, strFileLine
64: lstFile.AddItem strFileLine
65: Loop
66: ' Geöffnete Dateien schließen
67: Close
68: comErrorHandler:
69: ' Nichts tun, wenn der Benutzer auf Abbrechen geklickt hat
70: End Sub

```

Analyse

Die erste Ereignisprozedur in den Zeilen 1 bis 7 zeigt das Standarddialogfeld unter Verwendung der Methode ShowColor an. Wenn der Benutzer eine Farbe auswählt, setzt Zeile 6 den Hintergrund des Listenfelds auf diese Farbe. Die Datei, die der Benutzer ausgewählt hat, wird auf dieser neuen Hintergrundfarbe angezeigt. (Auch wenn keine Datei angezeigt wird, erhält das Listenfeld diese neue Farbe.) Das Listenfeld behält die Farbe während der gesamten Programmsitzung, es sei denn, der

Benutzer wählt erneut eine andere Farbe aus.

Zeile 9 beginnt die zweitlängste Ereignisprozedur des Programms. Wenn der Benutzer dem Formular eine andere Größe gibt, wird automatisch die Ereignisprozedur `Form_Resize()` ausgeführt. (Auch beim Laden des Formulars wird die Prozedur automatisch ausgeführt.)

Diese Prozedur stellt sicher, daß die Schaltfläche und das Listefeld in der Mitte des Formulars angezeigt werden, unabhängig davon, wie groß dieses ist. Wenn der Benutzer das Formular zu klein macht, so daß das Listefeld nicht mehr angezeigt werden kann, stellt Zeile 19 sicher, daß die folgenden Ereignisse stattfinden:

- Das Listefeld wird nicht angezeigt (andernfalls würde Visual Basic einen Fehler zurückgeben, weil es in so einem kleinen Formular nicht Platz hat).
- Der Benutzer wird gewarnt, daß das Listefeld zu klein ist.

Vorausgesetzt, bei der Größenänderung wurde das Listefeld nicht entfernt, zeigt Zeile 27 es an. Seine Breite und Höhe werden entsprechend der `Width`- und `Height`-Werte des Formulars angepaßt. Darüber hinaus wird die Schaltfläche abhängig von der `Width`-Eigenschaft des Formulars an einer anderen Stelle angezeigt.

Mit Zeile 41 beginnt die längste Prozedur des Programms. Diese Prozedur öffnet die ausgewählte Datei und liest sie. Wenn der Benutzer **Datei, Öffnen** auswählt, zeigt das Standarddialog-Steuerelement das Dialogfeld **Öffnen** an, wie in Abbildung BP6.4 gezeigt. Die Ereignisprozedur führt in Zeile 47 die Methode `ShowOpen` aus, um das Dialogfeld **Öffnen** zu erzeugen.

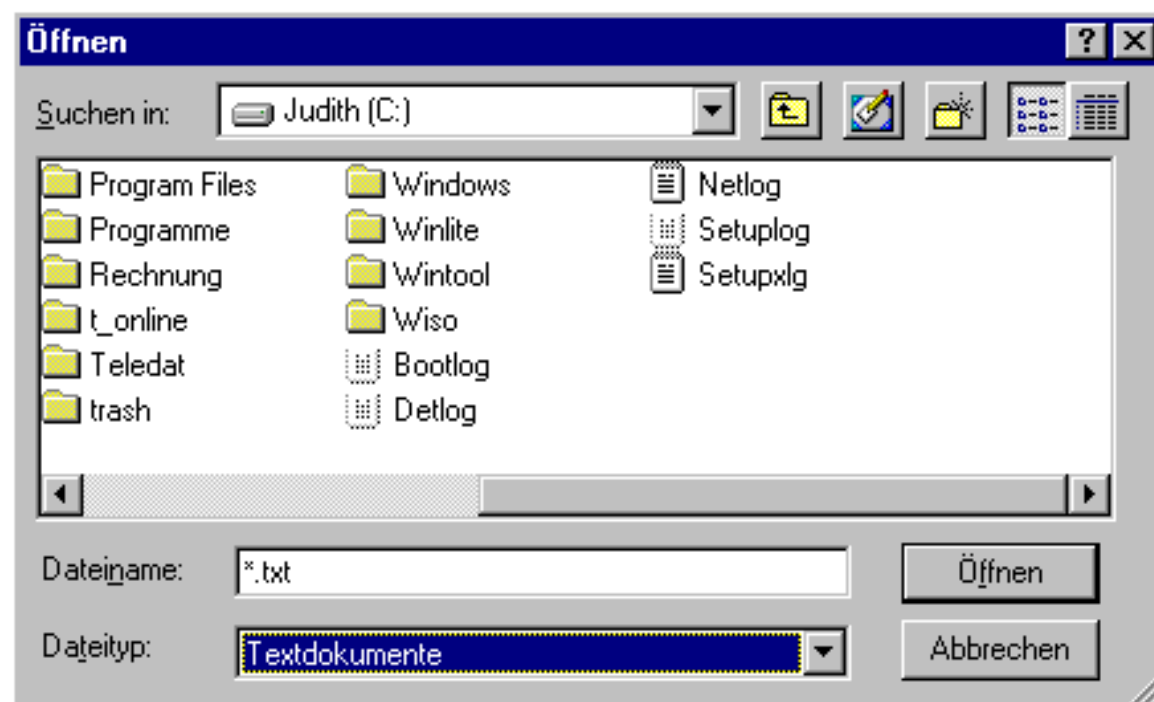


Abbildung BP6.4:
Der Benutzer wählt im Dialogfeld Öffnen eine Datei aus.

	Dem Benutzer werden abhängig vom Inhalt seiner Festplatte andere Ordner und Dateien im Dialogfeld Öffnen angezeigt.

Dem Benutzer wird das Dialogfeld **Öffnen** angezeigt, nachdem er in der Menüleiste den Eintrag **Datei,**

Öffnen ausgewählt hat. Er kann das Drop-down-Listenfeld **Dateityp** öffnen, um Dateien mit der Erweiterung .TXT oder .BAT auszuwählen (es werden immer alle Ordner angezeigt, unabhängig davon, welchen Dateityp der Benutzer ausgewählt hat). Der Benutzer kann die Ordner nach beliebigen Dateien auf dem PC oder im Netzwerk durchsuchen.

Zeile 44 richtet den Code für die Fehlerbehandlung ein, für den Fall, daß der Benutzer im Dialogfeld **Öffnen** auf die **Abbrechen**-Schaltfläche klickt. Die Fehlerbehandlung veranlaßt, daß die Ausführung an das Ende der Ereignisprozedur springt. Alle Elemente auf dem Formular und im Listenfeld bleiben unverändert.

Wenn das Programm Zeile 52 erreicht, hat der Benutzer eine Datei ausgewählt. Zeile 52 öffnet diese Datei zum sequentiellen Lesen.



Diese Datei wird nur für Eingaben, nicht für Ausgaben geöffnet. Das Programm kann den Dateiinhalt also nicht überschreiben.

Zeile 54 löscht das Listenfeld, so daß der Inhalt der ausgewählten Datei dort angezeigt werden kann. Die Zeilen 57 und 58 lesen die erste Zeile der Datei und tragen sie mit der AddItem-Methode in das Listenfeld ein. Zeile 57 stellt Ihnen einen neuen Befehl vor: `Line Input #`. Sie kennen bereits `Input #` und `Get #`, und werden mit `Line Input #` keine Probleme haben. Hier sein Format: `Line Input #intDateiNummer, strZeilenVar`

`Line Input` liest eine vollständige Zeile (bzw. einen Datensatz) aus der Datei in die Stringvariable `strZeilenVar`. Würde man statt dessen `Input #` verwenden, würde möglicherweise keine vollständige Zeile gelesen, wenn es dort Kommata oder Leerzeichen gibt. `Input #` beendet das Lesen, wenn es auf ein Komma oder ein Leerzeichen trifft, auch wenn diese mitten in der Zeile stehen.

Nachdem die erste Zeile gelesen und in das Listenfeld geschrieben wurde, beginnt in Zeile 62 die Schleife, die die restliche Datei liest. Eine interne Funktion, `EOF ( )`, prüft, ob das Dateiende erreicht ist. Die Datei kann aus einer oder mehreren Zeilen bestehen, und das Programm weiß nicht im voraus, um wie viele Zeilen es sich handelt. `EOF ( )` gibt `True` zurück, wenn das Dateiende erreicht ist. Der Rückgabewert `True` beendet die Schleife und das Programm wird in Zeile 66 fortgesetzt.



Warum stand in Zeile 62 nicht einfach folgendes:

```
Do Until (EOF(1) = True)
```

Zeile 62 hätte den Rückgabewert von `EOF ( )` auf `True` überprüfen können, aber es gab keinen Grund dafür. Das Programm würde sogar weniger effizient dadurch. Die `EOF ( )`-Funktion gibt `True` oder `False` zurück, abhängig davon, ob das Dateiende erreicht ist. Wie Sie aus den früheren Kapiteln wissen, werden Funktionen (sowohl Ihre eigenen als auch die eingebauten) zu ihren Rückgabewerten. Der Aufruf von `EOF ( )` wird also zu `True` oder `False`, und für die `Do Until`-Bedingung ist keine weitere Prüfung erforderlich.

Zeile 67 schließt die Datei und das Programm wird fortgesetzt.

Woche 2

Tag 13

Drucken

In diesem Kapitel erfahren Sie, wie man Ausgaben an den Drucker schickt. Sie kennen die Entwicklungsumgebung von Visual Basic, haben schon viel über die Programmiersprache gelernt und sind in der Lage, funktionierende Applikationen zu entwickeln - deshalb sollten Sie jetzt endlich wissen, wie Visual Basic druckt. Es gibt Situationen, in denen eine Applikation einen Ausdruck anlegen muß. Bis jetzt hatten Sie keine Möglichkeit, Informationen an den Drucker zu senden.

Seien Sie gewarnt - die Druckerschnittstelle von Visual Basic ist nicht so einfach wie die anderen Komponenten, die Sie bisher kennengelernt haben. Das Ausdrucken von Daten unter Visual Basic ist nicht trivial. Dieses Kapitel erklärt, wie Sie die verschiedenen Druckerschnittstellen bedienen.

Heute lernen Sie die folgenden Dinge:

- Drucker-Objekte
- Druckereinstellungen
- Die Anweisung `Is TypeOf`
- Methoden für den Drucker
- Druckermethoden zur Nachbildung von Formularmethoden
- Die richtigen Drucktechniken
- Benachrichtigungen vor dem Drucken
- Formulare drucken

Druckerobjekte

Angenommen, Sie möchten den Code in Ihrem Codefenster ausdrucken. Dazu wählen Sie **Datei, Drucken**. An einige PCs sind mehrere Drucker angeschlossen. Ihre Applikationen haben Zugriff auf alle Drucker im System. Auch interne Faxgeräte agieren häufig als Drucker. Sie können also von Ihrer Applikation aus auf jeden Drucker und auf jedes Fax drucken, indem Sie den entsprechenden Drucker zum Standarddrucker für Visual Basic machen. Nachdem Sie den Standarddrucker festgelegt haben, gibt Visual Basic alle Ausgaben an diesen Drucker aus und ignoriert den Standarddrucker des Systems, bis die Applikation beendet ist, oder bis Sie einen anderen Standarddrucker ausgewählt haben.



Um einen anderen Windows-Standarddrucker für das System auszuwählen, gehen Sie in das **Drucker**-Fenster. Dazu klicken Sie auf die **Start**-Schaltfläche und wählen **Einstellungen, Drucker**.

Die Auswahl eines Druckes für Ihre Codelistings ist relativ einfach. Hier wollen wir beschreiben, wie Sie gedruckte Ausgaben aus Ihrer Applikation erzeugen. Bevor wir ins Detail gehen, sollten Sie jedoch ein paar spezielle Objekte kennenlernen - die **Printers** -Auflistung.



Die **Printers**-Auflistung ist eine Liste aller Drucker, die an den PC angeschlossen sind, auf dem Ihre Applikation ausgeführt wird. Dabei werden auch interne Modems berücksichtigt, die zum Versenden von Faxen eingerichtet sind. Die Auflistung beinhaltet nicht unbedingt alle Drucker, die an den PC angeschlossen sind, auf dem Sie die Applikation entwickeln, sondern vielmehr die Drucker, die an den Laufzeit-PC angeschlossen sind.

Die **Printers**-Auflistung ist die Liste aller Drucker im System, auf dem Ihre Applikation ausgeführt wird. Diese Auflistung ist offensichtlich für alle Systeme unterschiedlich. Ein Benutzer könnte Ihre Applikation ausführen, danach einen oder mehrere Systemdrucker hinzufügen oder entfernen und Ihre Applikation dann wieder ausführen, so daß sich die **Printers**-Auflistung schon zwischen den beiden Programmläufen geändert hat. Dieses Kapitel erklärt, wie der Zugriff auf Drucker der aktuellen Auflistung erfolgt.



Die **Printers**-Auflistung enthält dieselbe Druckerliste wie das Systemdialogfeld **Drucken** in der Drop-down-Liste **Name**.

Zugriff auf die **Printers**-Auflistung

Auch auf die **Printers**-Auflistung erfolgt der Zugriff aus dem Code heraus über Indexwerte. Der erste Drucker (der Systemstandarddrucker) hat den Indexwert 0, der zweite Drucker hat den Indexwert 1 usw. Wenn Sie die Drucker in einer **For**-Schleife durchlaufen möchten, ermitteln Sie die Anzahl der Drucker im System mit **Printers.Count - 1**. Alternativ können Sie auch die Anweisung **For Each** verwenden, um die Drucker zu durchlaufen, ohne deren Anzahl zu benötigen (wie Sie im Beispiel in Listing 13.1 noch sehen werden).

Mit der Anweisung **Set Printer** setzen Sie den Standarddrucker für Visual Basic auf einen der Drucker im System. Die folgende Anweisung setzt den Visual-Basic-Standarddrucker auf den zweiten Drucker im System:

```
Set Printer = Printers(1) ' Standarddrucker ändern
```

Natürlich ist es nicht immer einfach, zur Laufzeit einen geeigneten Drucker auszuwählen. Wie wissen Sie überhaupt, nach welchem Drucker Sie suchen? Sie können nur nach bestimmten Eigenschaften suchen. Wenn Sie beispielsweise auf einem Drucker drucken möchten, der eine bestimmte Seitengröße unterstützt, durchlaufen Sie alle Drucker im System und suchen nach dieser Seitengröße.

Druckereigenschaften ermitteln

Tabelle 13.1 listet viele wichtige Druckereigenschaften auf, die Sie abfragen können, um festzustellen, ob der Drucker Ihre Bedürfnisse unterstützt. Den meisten dieser Eigenschaften sind Konstantennamen zugeordnet. Statt also für eine Seitengröße die Eigenschaftswerte von 1 oder 2 zu suchen, könnten Sie dafür auch einen Konstantennamen angeben, beispielsweise `vbPRPSLetter` und `vbPRPSLetterSmall`. Bei der Programmwartung sind diese Werte viel besser zu verstehen (auch wenn sie mehr Schreibarbeit erforderlich machen).



In der Online-Hilfe finden Sie für alle Eigenschaften eine Liste mit den unterstützten Konstantennamen.

Tabelle 13.1: Sie können ermitteln, ob die Drucker Ihrer Benutzer die folgenden Eigenschaften unterstützen

Eigenschaft	Beschreibung
<code>ColorMode</code>	Gibt an, ob ein Drucker farbig oder nur schwarzweiß drucken kann.
<code>Copies</code>	Spezifiziert die Anzahl der Kopien, die der Benutzer braucht. (Dieser Wert wird vom Benutzer zur Laufzeit im Dialogfeld Drucken eingegeben, das Ihre Applikation anzeigt, oder durch die Einstellung, die Ihr Code für diese Eigenschaft vorgenommen hat.)
<code>CurrentX</code> , <code>CurrentY</code>	Gibt die X- und Y-Koordinaten für das nächste Zeichen (oder die nächste Grafik) zurück bzw. setzt sie.
<code>DeviceName</code>	Enthält den Druckernamen, z.B. Canon BubbleJet IIC.
<code>DriverName</code>	Enthält den Druckertreiber. (Mehrere Drucker derselben Firma verwenden möglicherweise denselben Druckertreiber.)
<code>Duplex</code>	Gibt an, ob der Drucker beidseitig druckt.
<code>Height</code>	Gibt die Höhe der Druckseite für den ausgewählten Drucker zurück (in <code>ScaleMode</code> -Einheiten).
<code>Orientation</code>	Gibt Hoch- oder Querformat zurück bzw. setzt diese Werte.
<code>Page</code>	Gibt die aktuelle Seitennummer zurück.
<code>PaperBin</code>	Gibt den Papierschacht an oder setzt den Wert. (Beachten Sie, daß nicht alle Drucker mehrere Schächte unterstützen.)
<code>PaperSize</code>	Gibt die aktuell verwendete Papiergröße zurück oder setzt sie.
<code>Port</code>	Gibt den Namen des Druckeranschlusses zurück.
<code>PrintQuality</code>	Gibt die Druckerauflösung zurück oder setzt den Wert.
<code>TrackDefault</code>	Für den Wert <code>False</code> behält <code>TrackDefault</code> die aktuellen Druckereigenschaften bei, wenn Sie den Standarddrucker wechseln; für <code>True</code> werden die Druckereigenschaften zur Laufzeit geändert, wenn ein anderer Standarddrucker ausgewählt wird.

Width	Gibt die Breite der Druckseite für den ausgewählten Drucker zurück (in ScaleMode-Einheiten).
Zoom	Gibt den Skalierungssatz für Druckausgaben zurück oder setzt ihn. Wenn Sie die Zoom-Eigenschaft beispielsweise auf 75 setzen, erscheinen alle weiteren Ausgaben mit 75 Prozent ihrer Originalgröße auf der Seite. (Nicht alle Drucker unterstützen die Zoom-Eigenschaft.)

Bei der Ausführung Ihrer Prozedur stimmen die Druckereigenschaften für das `Printer` -Objekt mit denen des Windows-Standarddruckers im System überein. Wenn Sie einen neuen Standarddrucker für Visual Basic setzen, ändern sich die Eigenschaften entsprechend. Zur Laufzeit können Sie viele dieser Eigenschaften ändern, wie in Tabelle 13.1 beschrieben.

	Viele der Eigenschaften aus Tabelle 13.1 werden Sie für die speziellen Druckermethoden im nächsten Abschnitt benutzen.

Der Code in Listing 13.1 demonstriert, wie Sie die aktuellen Drucker im System durchlaufen.

Listing 13.1: Sie können alle Drucker des Benutzers abfragen

```
1: Dim prnPrntr As Printer
2: For Each prnPrntr In Printers ' Durchläuft alle Drucker
3: frmMyForm.Print prnPrntr.DeviceName
4: Next
```

Der Code gibt die Namen der einzelnen Drucker auf dem aktuellen Formular aus.

In der ersten Zeile wird eine Variable mit dem Datentyp `Printer` deklariert. Wenn Sie mehr über Visual Basic wissen, werden Sie erkennen, daß Sie Variablen fast jeden Datentyps deklarieren können, auch `Printer` oder `Form`. Die Variable `prnPrntr` ermöglicht Ihnen, die Drucker auf dem System zu durchlaufen. Eine äquivalente `For`-Anweisung würde so aussehen:

```
For prnPrntr = 1 to (Printers.Count - 1)
```

	<code>Printer</code> und <code>Form</code> sind Visual-Basic-Objekte. Mehr darüber erfahren Sie in Kapitel 16.

Es kommt nur ganz selten vor, daß Sie die Namen aller Drucker auf einem Formular ausgeben müssen. Nichtsdestotrotz zeigt die Schleife in diesem Beispiel viele Grundlagen, die Sie für die weitere Arbeit mit Druckern benötigen.

Listing 13.2 durchläuft alle Drucker im System und sucht nach einem Farbdrucker, um eine farbige Tabelle auszugeben.

Listing 13.2: Auf dem System des Benutzers wird ein Farbdrucker gesucht

```
1: Dim prnPrntr As Printer
2: For Each prnPrntr In Printers
3: If prnPrntr.ColorMode = vbPRCMColor Then
4: ' Farbdrucker zum Systemstandarddrucker machen
```

```
5: Set Printer = prnPrntr
6: Exit For ' Nicht mehr weitersuchen
7: End If
8: Next ' Ggf. alle durchlaufen
```

Ausgabesteuerung

Das `Printer`-Objekt wird bereitgestellt, damit Sie Daten an den Standarddrucker senden können, ohne sich um spezielle Druckertypen oder Anschlüsse kümmern zu müssen. Sie wenden Methoden auf das `Printer`-Objekt an, um Ausgaben an den Drucker weiterzuleiten. Die Programmierung mit dem `Printer`-Objekt kann mühsam sein, aber Sie werden allgemeine AusgabeprozEDUREN entwickeln, die Sie in allen Ihren Programmen einsetzen können.



Bevor Sie Ausgaben an das `Printer`-Objekt senden, setzen Sie den Standarddrucker mit der Anweisung `Set Printer`, wenn das `Printer`-Objekt auf einen anderen als den Standarddrucker des Systems verweisen soll.

Nachdem Sie einen Standarddrucker ausgewählt haben, leiten Sie mit Hilfe des `Printer`-Objekts Text und Grafik auf den Drucker des Benutzers um. Dieser Abschnitt erklärt, wie Sie das `Printer`-Objekt steuern und Text auf den Drucker ausgeben. In Kapitel 14 lernen Sie Grafikbefehle und Methoden kennen, die Sie auch auf das `Printer`-Objekt anwenden können.

Mit dem `Printer`-Objekt bauen Sie Ihre Ausgaben auf. Das bedeutet, Sie senden Ausgaben an das `Printer`-Objekt, aber es wird noch nichts ausgedruckt. Nachdem Sie die Ausgabe fertig haben und diese an den Drucker senden möchten, führen Sie die Methoden `NewPage` oder `EndDOC` aus, um das Drucken zu beginnen. (Das Drucken wird auch begonnen, wenn Ihre Applikation beendet wird, und Sie die `EndDOC`-Methode nicht ausgeführt haben.)

Drucken in das Printer-Objekt

Eine der einfachsten Methoden, Ausgaben an das `Printer`-Objekt zu senden, ist `Print`. Die folgenden Zeilen senden eine Meldung an den Drucker:

```
Printer.Print "Dieser Bericht enthält Verkaufszahlen von "
Printer.Print dteStart; " bis "; dteFinish; "."
Printer.Print "Weitere Informationen unter der Nummer 319."
```



Windows übernimmt für Sie die Details, die für das Drucken erforderlich sind. Wenn der Benutzer kein Papier im Drucker hat oder den Drucker ausgeschaltet hat, zeigt Windows eine Fehlermeldung an, wie etwa die in Abbildung 13.1 gezeigte. Nachdem der Benutzer das Problem korrigiert hat, klickt er auf **Wiederholen**, um den Ausdruck fortzusetzen, oder auf **Abbrechen**, um die noch nicht gedruckten Ausgaben zu verwerfen.

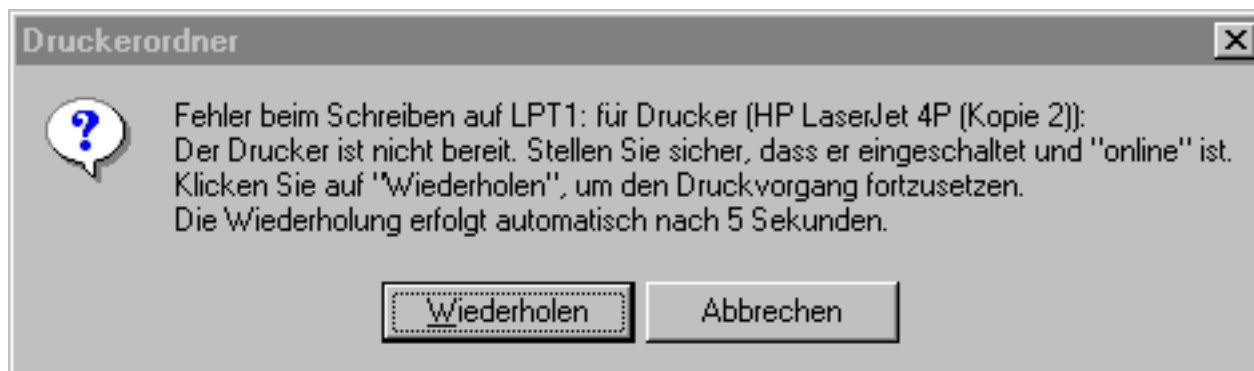


Abbildung 13.1:
Windows ermöglicht dem Benutzer, Fehler zu beheben, die beim Drucken auftreten können.

Natürlich können Sie Konstanten, Variablen und Steuerzeichen an den Drucker senden. Alles, was Sie mit `Print` auf das Formular ausgeben können, können Sie auch auf das `Printer`-Objekt ausgeben, wie die folgenden Anweisungen zeigen:

```
sngTaxRate = .34
strTitle = "Mein Lieblingsmord"
Printer.Print strTitle & " ist der Name des Buchs"
Printer.Print "Die Steuer beträgt" & sngTaxRate
```

Wenn Sie leere Zeilen ausdrucken möchten, geben Sie eine `Print`-Methode ohne Argumente an:

```
Printer.Print ' Druckt eine leere Zeile
```



Tip

Machen Sie Ihre Benutzer darauf aufmerksam, wenn das Drucken beginnt, um sie nicht zu überraschen. Der letzte Abschnitt dieses Kapitels beschreibt, wie Sie dazu am besten vorgehen.

In Kapitel 1 haben Sie mit der `Print`-Methode Ausgaben an ein Formular geschickt. Wie Sie hier jedoch sehen, ist `Print` eine allgemeine Methode, die Ausgaben an alle gültigen Objekte schickt, die Text entgegennehmen.

Mit der Methode `NewPage` gehen Sie beim Ausdruck auf den Anfang der nächsten Seite:

```
Printer.NewPage ' An den Anfang der nächsten Seite springen
```

Ausgaben skalieren

Beim Drucken können Sie die Ausgaben skalieren, um Ränder auf der Seite zu schaffen, die alle zum Drucken verwendeten Methoden respektieren. Nachdem Sie die `ScaleMode` -Eigenschaft auf die gewünschte Einheit für Ihr Programm gesetzt haben, können Sie auch andere Eigenschaften für die Skalierung einsetzen, um festzulegen, wie Ihre Ausgabe auf der Seite erscheint. Tabelle 13.2 listet die Eigenschaften für die Skalierung auf.

Tabelle 13.2: Skalierungseigenschaften bestimmen die Maßeinheiten

Eigenschaft	Beschreibung
<code>ScaleLeft</code>	Definiert die äußerste linke X-Koordinate des Druckbereichs. Der <code>ScaleLeft</code> -Wert von 10 verschiebt den nachfolgenden linken Rand um 10 <code>ScaleMode</code> -Einheiten.

ScaleMode	Bestimmt die Maßeinheiten für die Skalierung. In der Regel werden die ScaleMode -Werte vbPoints (Wert 2), vbCharacters (die Standardzeichenbreite für den Drucker), vbInches oder vbCentimeters für die Textausgabe verwendet.
ScaleHeight	Ändert das vertikale Koordinatensystem des Printer -Objekts.
ScaleTop	Definiert die oberste Y-Koordinate des Druckbereichs. Ein ScaleTop -Wert von 5 verschiebt den nachfolgenden oberen Rand um fünf ScaleMode -Einheiten.
ScaleWidth	Ändert das horizontale Koordinatensystem des Printer -Objekts.

Um einen oberen Rand von fünf Zeichen und einen linken Rand von acht Zeichen einzustellen, führen Sie die folgenden Methoden aus:

```
Printer.ScaleMode = vbCharacters ' Skalierung auf Zeichen setzen
Printer.ScaleTop = 5
Printer.ScaleLeft = 8
```

Alle nachfolgenden Druckmethoden respektieren diese Grenzen.

Die CurrentX- und CurrentY-Positionen

Wenn Sie das Koordinatensystem nicht von der Seitenkoordinate 0,0 mit den Eigenschaften **ScaleHeight** und **ScaleWidth** auf ein anderes X,Y-System gesetzt haben, beginnen die **CurrentX**- und **CurrentY**-Werte des **Printer**-Objekts bei 0,0. (Die Koordinaten verwenden immer die in **ScaleMode** gesetzte Einheit.) Diese Werte können auf andere X- und Y-Koordinatenwerte gesetzt werden, falls Sie die nächste Ausgabe an eine bestimmte Position auf der Seite vornehmen möchten.



Die Eigenschaften **CurrentX** und **CurrentY** respektieren stets die Ränder, die Sie mit **ScaleLeft** und **ScaleTop** gesetzt haben. Das Koordinatenpaar **CurrentX** , **CurrentY** bezieht sich also auf das erste Zeichen in der oberen linken Ecke der aktuellen Seite, wobei die von Ihnen definierten Ränder berücksichtigt werden.

Um eine Meldung 15 Zeilen unterhalb und 25 Zeichen rechts von der Startposition auszugeben, verwenden Sie den folgenden Code:

```
Printer.ScaleMode = vbCharacters
Printer.CurrentY = 14 ' Weil der Startwert gleich 0 ist
Printer.CurrentX = 24
Printer.Print "Achtung, Achtung, Gefahr ist im Verzug!"
```

Das **Printer**-Objekt von Visual Basic unterstützt mehrere Methoden zur Steuerung des Druckprozesses. Mit der **NewPage**-Methode wird der Ausdruck oben auf der nächsten Seite fortgesetzt:

```
Printer.NewPage ' Oben auf der nächsten Seite weiterdrucken
```

Bei der Vorbereitung auf den Ausdruck kann dieser mit **KillDoc** jederzeit abgebrochen werden:

```
Printer.KillDoc ' Die Ausgaben nicht an den Drucker weitergeben
```


KillDoc entfernt alle Ausgaben aus dem Printer-Objekt. Wenn der Benutzer das gedruckte Dokument später doch braucht, muß der Ausdruck wiederholt werden.



Ausdrucke, die bereits auf dem Drucker sind, können mit KillDoc nicht abgebrochen werden. Außerdem ist es nicht möglich, mit KillDoc PrintForm- Aufträge zu löschen.

Microsoft empfiehlt, eine allgemeine Druckroutine innerhalb eines Standardmoduls anzulegen, das Sie von allen Applikationen aus aufrufen können - vorausgesetzt, Sie nehmen dieses Standardmodul in diese Applikationen auf. Der Code in Listing 13.3 nimmt zwei Argumente mit dem Datentyp Object entgegen. Weil die Routine den Datentyp Object verarbeitet, können Sie ihr ein Form- oder ein Printer-Objekt übergeben.

Listing 13.3: Mit dieser Prozedur geben Sie spezielle Steuerelemente aus einem Formular aus.

```
1: Sub PrintAnywhere (Src As Object, Dest As Object)
2: Dest.PaintPicture Src.Picture, Dest.Width / 2, Dest.Height / 2
3: If TypeOf Dest Is Printer Then
4: Printer.EndDoc
5: End If
6: End Sub
```

Angenommen, Ihre Applikation enthält ein Formular mit einem Bildfeld oder einem Anzeigefeld, das Sie auch auf dem Drucker ausgeben möchten. Vielleicht möchten Sie aber auch ein leeres Formular anzeigen, in das die Benutzer Daten eingeben und diese an den Drucker senden.

Diese Routine übernimmt in Zeile 1 die Argumente für Quelle und Ziel. Die Quelle ist immer das Formular, auf dem Sie Ausgaben vornehmen möchten. Das Ziel kann Printer sein. Sie können die Prozedur überall dort aufrufen, wo es sinnvoll ist, das Formular zu drucken, beispielsweise wie folgt:

```
Call PrintAnywhere (frmUserForm, Printer) ' Formular ausdrucken
```

Diese Routine verwendet die PaintPicture-Methode, um ein Formular auszugeben. PaintPicture zeichnet ein Formular in das Objekt, auf die Sie sie anwenden. Für die PaintPicture-Methode sind drei Werte erforderlich: das Formular, in das gezeichnet werden soll, die Breite und die Höhe des Ziels. Dieser Code gibt einfach ein Formular aus, das halb so groß ist wie der Zielbereich. Um sicherzustellen, daß das Formular nach der Ausführung dieser Methode ausgedruckt wird, gibt die If-Anweisung das Formular unmittelbar mit der EndDoc-Methode aus, wenn das Ziel das Printer-Objekt und nicht ein anderes Formular ist. (Sie könnten als Ziel auch ein anderes Formular angeben.)

Die Anweisung If TypeOf demonstriert ein If, das Sie bisher noch nie gesehen haben. Der Befehl If TypeOf . . . Is ermöglicht Ihnen, zu überprüfen, ob ein Objekt einen bestimmten Datentyp hat.



Die Anweisung If TypeOf . . . Is macht viel mehr, als nur zu prüfen, welchen Datentyp ein Objekt hat, wie Sie in Kapitel 16 noch erfahren werden.

Formatieren mit Font

Tabelle 13.3 listet einige schriftbezogene `Printer`-Objekteigenschaften auf, mit denen Sie bestimmte Schrifteigenschaften setzen, bevor Sie Text an den Drucker senden.

Tabelle 13.3: Bevor Sie Text an den Drucker senden, können Sie schriftbezogene Eigenschaften setzen

Eigenschaft	Beschreibung
<code>Font</code>	Gibt eine Schrift zurück, mit der Sie die Schriftattribute setzen.
<code>FontBold</code>	Enthält <code>True</code> oder <code>False</code> , um anzugeben, ob nachfolgende Ausgaben fett dargestellt werden sollen.
<code>FontCount</code>	Gibt die Anzahl der vom Drucker unterstützten Schriften zurück.
<code>FontItalic</code>	Enthält <code>True</code> oder <code>False</code> , um anzugeben, ob nachfolgende Ausgaben kursiv dargestellt werden sollen.
<code>FontName</code>	Enthält den Namen der für die Ausgabe verwendeten Schrift.
<code>Fonts</code>	Enthält eine Liste mit den Namen aller auf dem System installierten Schriften. Der Zugriff auf diese Liste erfolgt wie auf ein Steuerelementfeld, <code>Fonts(0)</code> und <code>Fonts(FontCount - 1)</code> stellen also den ersten bzw. letzten Index dar.
<code>FontSize</code>	Bestimmt die Punktgröße der aktuell verwendeten Schrift.
<code>FontStrikeThru</code>	Enthält <code>True</code> oder <code>False</code> , um anzugeben, ob nachfolgende Ausgaben durchgestrichen dargestellt werden sollen.
<code>FontTransparent</code>	Enthält <code>True</code> oder <code>False</code> , um anzugeben, ob nachfolgende Ausgaben transparent dargestellt werden sollen.
<code>FontUnderline</code>	Enthält <code>True</code> oder <code>False</code> , um anzugeben, ob nachfolgende Ausgaben unterstrichen dargestellt werden sollen.

Mit Hilfe der Schriftattribute fügen Sie Ihren Druckausgaben spezielle Effekte hinzu, um die Schriftinformation für den Text zu ändern. Der folgende Codeabschnitt erzeugt die Wörter *Visual Basic* in großen Buchstaben auf dem Drucker:

```
Printer.FontItalic = True
Printer.FontBold = True
Printer.FontSize = 72 ' 1-Zoll-Buchstaben
Printer.Print "Visual Basic"
```

Die `Printer`-Methoden können nicht nur auf den Drucker, sondern auch auf Formulare angewendet werden. Betrachten Sie den Code in Listing 13.4. Versuchen Sie herauszufinden, was auf dem Formular angezeigt wird. (Wenn Sie diese `Print`-Methoden auf das `Printer`-Objekt anwenden, erfolgt diese Ausgabe auf den Drucker.)

Listing 13.4: Print-Methoden können sowohl auf den Drucker als auch auf Formulare angewendet werden

```
1: Private Sub cmdPrint_Click()
```

```

2: ' Erzeugt mit der Print-Methode eine interessante
3: ' Ausgabe auf dem Formular
4: Dim intCtr As Integer
5: Dim intCurX As Integer
6: Dim intCurY As Integer
7: '
8: ' Schrift einrichten
9: frmPrint.FontItalic = True
10: frmPrint.FontBold = True
11: frmPrint.FontSize = 36 ' 1-Zoll-Buchstaben
12: '
13: ' Einheit auf Twips setzen
14: frmPrint.ScaleMode = vbTwips
15: '
16: ' Aktuelle X- und Y-Position bei jedem
17: ' Schleifendurchgang speichern
18: For intCtr = 1 To 10
19: intCurX = frmPrint.CurrentX
20: intCurY = frmPrint.CurrentY
21: ' Schwarz oder weiß
22: If (intCtr Mod 2) = 1 Then ' Gerader Schleifenzähler
23: frmPrint.ForeColor = vbWhite
24: Else
25: frmPrint.ForeColor = vbBlack
26: End If
27: ' Text in der großen Schrift ausgeben
28: frmPrint.Print "Visual Basic"
29: '
30: ' Druckposition neu festlegen
31: frmPrint.CurrentX = intCurX + 350
32: frmPrint.CurrentY = intCurY + 300
33: Next intCtr
34: End Sub
35:
36: Private Sub cmdExit_Click()
37: End
38: End Sub

```

In den Zeilen 9 bis 11 wird eine große, kursive Schrift für das Formular eingerichtet. Dazu werden die verschiedenen Schriftmethoden eingesetzt. Zeile 14 setzt die Maßeinheit auf Twips, so daß alle folgenden `CurrentX`- und `CurrentY`-Eigenschaften in Twips und nicht in Zeichen angegeben werden. Zeile 18 startet eine Schleife, die den String *Visual Basic* 10mal auf dem Formular ausgibt. Die Darstellung ist eher unüblich.



Wenn Sie dieses Beispiel für Druckerausgaben umschreiben, setzen Sie die Farbenkonstante in Zeile 23 auf einen anderen Wert als `vbWhite`. Wenn Sie einen Farbdrucker haben, können Sie eine Farbe aussuchen, die gedruckt werden kann, beispielsweise `vbBlue` oder `vbRed`.

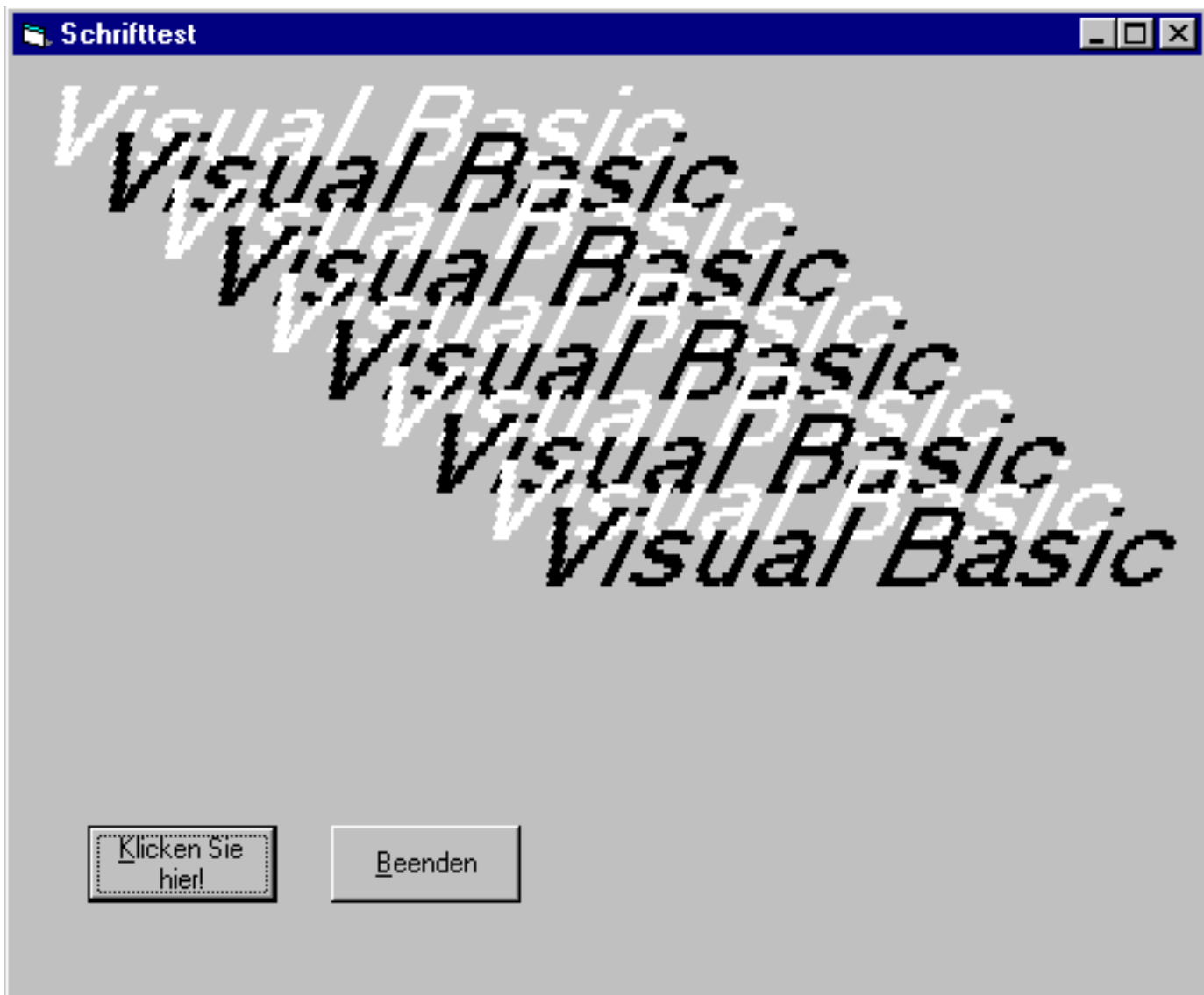


Abbildung 13.2:
Mit den Print-Methoden bestimmen Sie, wie die Ausgabe erfolgt.

Der Ausdruck von Formularen

Eine der einfachsten Methoden, Ausgaben an einen Drucker zu senden, ist der Ausdruck eines Formulars. Visual Basic stellt dazu die Methode `PrintForm` bereit, die auf ein Formular in Ihrem Projekt angewendet wird. Um Ausgaben auf den Drucker vorzunehmen, schreiben Sie einfach mit der `Print`-Methode etwas auf das Formular und drucken dieses dann aus.

Dieser Abschnitt erklärt die Methode `PrintForm`. Ähnlich der `Print`-Methode wenden Sie `PrintForm` auf die Formulare Ihres Projekts an. Bei der Ausführung einer `PrintForm` -Methode beginnt Visual Basic unmittelbar, das Formular auszudrucken. Das Formular sollte also für die Ausgabe fertig sein, bevor Sie `PrintForm` ausführen. Dieser Abschnitt erklärt, wie Sie mit `PrintForm` die besten Resultate erzielen. Sie werden feststellen, daß `PrintForm` nicht besonders gut dafür geeignet ist, unter Visual Basic zu drucken, aber für die Ausgabe eines Formulars auf dem Drucker ist sie ausreichend.



Der vielleicht größte Vorteil von `PrintForm` und den anderen Druckfunktionen von Visual Basic ist, daß die `Printer`-Objekte von Windows verwendet werden. Sie brauchen sich also nicht um spezielle Druckanweisungen zu kümmern, die für einen bestimmten Druckertyp oder ein bestimmtes Modell spezifisch sind.

Hier das Format der `PrintForm`-Methode:

```
[frmFormName.]PrintForm
```

Beachten Sie, daß die Angabe von `frmFormName` optional ist. Wenn Sie keinen Formularnamen angeben, wendet Visual Basic die Methode `PrintForm` auf das aktuelle Formular an.

Um das Formular `frmAccPayable` auszudrucken, verwenden Sie den folgenden Befehl in der Ereignisprozedur oder Modulprozedur, die den Ausdruck anfordert:

```
frmAccPayable.PrintForm ' Das Formular frmAccPayable ausdrucken
```

Falls `frmAccPayable` das aktuelle Formular ist (das Formular mit dem Fokus, dessen Titelleiste hervorgehoben dargestellt ist), können Sie den Formularnamen auch weglassen:

```
PrintForm ' Das Formular frmAccPayable ausdrucken
```

Auch `Me` bezieht sich auf das aktuelle Formular, Sie können also auch die folgende Anweisung verwenden, um es auszudrucken:

```
Me.PrintForm ' Das Formular frmAccPayable ausdrucken
```



In Kapitel 11 wurde die `Print`-Methode erklärt. `Print` sendet Ausgaben direkt an das Formular. Sie senden mit `Print` Ausgaben an ein Formular und drucken dieses dann mit `PrintForm` aus. Beachten Sie jedoch, daß alle Steuerelemente, die auf dem Formular angezeigt werden, auch auf dem Ausdruck erscheinen.

PrintForm-Warnungen

Die eigentliche Stärke von `PrintForm` liegt in seiner Einfachheit. `PrintForm` stellt die praktischste - und einfachste - Druckerausgabe in Visual Basic dar. Leider gibt es auch einige Nachteile, die Sie kennen sollten.

Unabhängig von der Druckerauflösung druckt `PrintForm` das Formular immer in der aktuellen Bildschirmauflösung aus. Diese Auflösung ist in der Regel nicht höher als 96 DPI (*DPI* steht für Dots Per Inch, also Punkte pro Zoll). Druckerauflösungen betragen häufig bis zu 600 DPI, das Formular sieht auf dem Papier also nicht so gut aus wie auf dem Bildschirm (für die Bildschirmauflösung sind 96 DPI völlig ausreichend).

Bevor Sie ein Formular ausgeben, das Steuerelemente und andere grafische Elemente enthält, sollten Sie die `AutoRedraw`-Eigenschaft auf `True` setzen. Standardmäßig ist die `AutoRedraw`-Eigenschaft für ein Formular `False`, d.h. die `Print`-Methode druckt direkt über die grafischen Steuerelemente. Wenn Sie `AutoRedraw` auf `True` setzen, bleibt die Grafik im Vordergrund, während `Print` das dahinterliegende Bild druckt und die Grafik nicht überschreibt. Nutzen Sie die `False`-Eigenschaft von `AutoRedraw`, um Hintergrundgrafiken zu erzeugen. Sie können später dann Text darauf ausgeben, müssen aber `AutoRedraw` auf `True` setzen, bevor Sie das Formular ausdrucken, so daß die Ausgabe auf dem Drucker korrekt erfolgt.

Beachten Sie, daß `PrintForm` die Objekte, die zur Entwurfszeit auf dem Formular angelegt wurden (ebenso wie die Laufzeitwerte für Steuerelemente, wie beispielsweise Bezeichnungsfelder und Textfelder), nur druckt, wenn `AutoRedraw` gleich `False` ist. Falls Sie einem Formular zur Laufzeit Grafiken und Bilder hinzufügen und das Formular mit `PrintForm` ausdrucken möchten, sollten Sie also die `AutoRedraw`-Eigenschaft des Formulars auf `True` setzen, bevor Sie diese Elemente hinzufügen. Andernfalls erscheinen die Ergänzungen, die Sie zur Laufzeit vorgenommen haben, nicht auf dem Ausdruck.



Das Drucken ist für Windows-Programmierer zum Teil eine größere Herausforderung. Testen Sie Ihre Applikation mit so vielen Druckern wie möglich, um sicherzustellen, daß Sie die gewünschten Ergebnisse erzielen. Sie können nicht garantieren, daß Ihre Ausgabe auf jedem Drucker gleich gut aussieht. Sie sollten jedoch einige allgemeine Druckertypen ausprobieren, wenn Sie Ihre Applikationen einem großen Benutzerkreis zur Verfügung stellen. Ihre Applikation ist natürlich davon abhängig, ob die Druckertreiber auf den Systemen Ihrer Benutzer korrekt installiert und eingerichtet sind, und außerdem davon, ob Ihre Benutzer einen geeigneten Drucker verwenden.

Ihre Applikation kann nur zum Teil dafür sorgen, daß der Ausdruck möglichst gut erscheint, weil die Druckerschnittstelle von Windows einen Großteil der Aufgaben übernimmt. Visual Basic versucht Ihnen zu helfen, indem es diesen Puffer zwischen Ihrer Applikation und den Drucker stellt - und Sie müssen auf alle Fälle weniger Code schreiben als die MS-DOS-Programmierer, die jeden möglichen Drucker berücksichtigen mußten (eine unmögliche Aufgabe, weil häufig neue Druckermodelle auf den Markt kamen, nachdem der Code fertiggestellt, aber bevor die Applikation freigegeben wurde).

Listing 13.5 zeigt, wie Sie einen Text auf ein leeres Formular ausgeben und ihn dann an den Drucker senden.

Listing 13.5: Sie können eine Meldung zuerst an das Formular und dann an den Drucker senden

```
1: Dim blnAutoRedraw As Boolean ' Der Wert von AutoRedraw
2: '
3: frmBlank.Print "Abteilungsbericht"
4: frmBlank.Print ' Leere Zeile
5: frmBlank.Print "Abteilung"; Tab(20); "Position"
6: frmBlank.Print "-----"; Tab(20); "-----"
7: frmBlank.Print "Nord"; Tab(20); "Werkzeuge"
8: frmBlank.Print "Süd"; Tab(20); "Pressen"
9: frmBlank.Print "Ost"; Tab(20); "Farben"
10: frmBlank.Print "West"; Tab(20); "Schleifmaschinen"
11: '
12: 'AutoRedraw-Eigenschaft des Formulars speichern
13: '
14: blnAutoRedraw = frmBlank.AutoRedraw
15: '
16: ' Formular ausdrucken
17: '
18: frmBlank.AutoRedraw = True
19: frmBlank.PrintForm
```

```

20: '
21: ' AutoRedraw-Eigenschaft wiederherstellen
22: '
23: frmBlank.AutoRedraw = blnAutoRedraw

```

Dieser Code demonstriert, wie man die `AutoRedraw`-Eigenschaft speichert, bevor man die `PrintForm`-Methode auslöst. In diesem Fall sind Sie vermutlich auf der sicheren Seite, wenn Sie die `AutoRedraw`-Eigenschaft zur Entwurfszeit auf `True` setzen (vorausgesetzt, Sie senden in der Applikation keine Grafiken an das Formular), können Sie die Eigenschaft speichern, bevor Sie die Ausgabe auf ein Formular vornehmen.

Prüfen Sie bei jedem Ausdruck mögliche Fehlerbedingungen. Vielleicht ist der Drucker des Benutzers nicht angeschaltet, nicht an den Computer angeschlossen oder es liegt kein Papier ein. Verwenden Sie dazu den Befehl `On Error Goto`, wie in Listing 13.6 gezeigt.

Listing 13.6: Der Befehl `On Error Goto` fängt Fehler auf, die beim Drucken auftreten

```

1: Private Sub cmdPrintForm_Click ()
2: Dim intBtnClicked As Integer
3: On Error Goto ErrHandler ' Fehler-Handler einrichten.
4: frmAccPayable.PrintForm ' Formular ausdrucken
5: Exit Sub
6: ErrHandler:
7: intBtnClicked = MsgBox("Druckerproblem", vbExclamation, _
"Print Error")
8: End Sub

```



Wenn Sie in einer Visual-Basic-Applikation ein Formular ausdrucken möchten, das der Benutzer ausfüllen soll, sollten Sie das Formular anlegen und darauf die `PrintForm`-Methode anwenden.



Möglicherweise möchten Sie vor dem Ausdruck eines Formulars die Titelleiste, das Icon für das Steuermenü und die Fensterschaltflächen entfernen. Verbergen Sie das Formular vorübergehend und zeigen Sie ein anderes an, während Ihr Code diese Dinge entfernt, indem er die entsprechenden Eigenschaftswerte für die Anzeige auf `False` setzt.

Erschrecken Sie Ihre Benutzer nicht!

Ihre Applikationen sollten nicht zu drucken anfangen, bevor der Benutzer nicht zugestimmt hat. Der Benutzer muß sich normalerweise auf den Ausdruck vorbereiten, weil er möglicherweise noch Papier einlegen oder den Drucker anschalten will. Stellen Sie also ein Dialogfeld zur Verfügung, mit dem er der Applikation mitteilt, wann der Drucker bereit ist. Andernfalls erscheint eine unangenehme Fehlermeldung und der Benutzer mag Ihr Programm nicht mehr und mißtraut ihm fortan.

Listing 13.7 zeigt eine Ereignisprozedur, die Sie für Ihre eigenen Erfordernisse anpassen können. Die Prozedur erzeugt das in Abbildung 13.3 gezeigte Meldungsfeld. Das Meldungsfeld ist ganz einfach, bietet Ihren Benutzern aber die Zeit, den Drucker vorzubereiten, bevor Ihre Applikation irgendwelche Ausgaben erzeugt.

Listing 13.7: Zeigen Sie Ihren Benutzern ein Meldungsfeld an, bevor Sie einen Ausdruck beginnen

```
1: Public Function PrReady() As Boolean
2: ' Sicherstellen, daß der Benutzer für den Ausdruck bereit ist
3: Dim intIsReady As Integer
4: '
5: ' Der Benutzer reagiert auf das folgende Meldungsfeld,
6: ' sobald er für den Ausdruck bereit ist
7: intIsReady = MsgBox("Drucker vorbereiten", vbOKCancel, "Drucken")
8: '
9: If (intIsReady = vbCancel) Then
10: PrReady = False
11: Else
12: PrReady = True
13: End If
14: End Function
```

Beachten Sie, daß die Funktionsdeklaration den Datentyp `Boolean` zurückgibt. Sie können den Funktionsaufruf dieser Prozedur also überall dort vornehmen, wo ein gültiger `Boolean` stehen könnte. Wenn der Benutzer auf die **OK**-Schaltfläche des Meldungsfeldes klickt, gibt Zeile 12 `True` für die Funktion zurück, d.h. der Benutzer ist bereit für den Ausdruck. Wenn der Benutzer dagegen auf die **Abbrechen**-Schaltfläche klickt, gibt Zeile 10 den Wert `False` zurück und der Ausdruck erfolgt nicht.



Abbildung 13.3:
Der Benutzer reagiert auf dieses Meldungsfeld, sobald er für den Ausdruck bereit ist.

Listing 13.8 zeigt die Ereignisprozedur für eine Schaltfläche, mit der die Funktion `PrReady()` aufgerufen werden könnte.

Listing 13.8: Vor dem Ausdruck wird der Wert von `PrReady()` ausgewertet

```
1: Private Sub cmdPrint_Click()
2: ' Drucken, wenn der Benutzer bereit ist, andernfalls
3: ' einfach nichts tun
4: If PrReady() Then
5: ' Call ReportPrint
6: End If
7: End Sub
```


Zusammenfassung

Dieses Kapitel hat gezeigt, wie man Ausgaben auf dem Drucker vornimmt. Ausgaben auf den Drucker können mühsam sein, weil es dafür kein Steuerelement gibt - eine Ausnahme in Visual Basic. Sie müssen also mehrere `Printer`-Methoden anwenden, damit der Ausdruck so aussieht, wie Sie sich das vorstellen. Aber genau dadurch entsteht auch eine gesteigerte Leistungsfähigkeit, weil Sie genau festlegen können, wie der Ausdruck aussehen soll.

Die hier beschriebenen `Printer`-Methoden können auch auf `Form`-Objekte angewendet werden. Sie können die Methoden verwenden, um Text in unterschiedlichen Schriftstilen an das Formular oder an den Drucker zu senden (oder an beide, falls die Ausgabe auf beiden Geräten erscheinen soll). Der Ausdruck von Formularen wird jedoch am besten mit der `PrintForm`-Methode erledigt.

Im nächsten Kapitel lernen Sie die Grafik- und Multimediafunktionen von Visual Basic kennen.

Fragen und Antworten

Frage:

Was kann ich tun, wenn meine Applikation und eine andere Applikation gleichzeitig Ausgaben auf den Drucker vornehmen?

Antwort:

Glücklicherweise nimmt Ihnen Windows die Aufgabe ab, den Druckauftrag in eine Warteschlange zu stellen. Ihr PC kann nicht zwei Dinge gleichzeitig erledigen, und auch wenn es den Anschein hat, daß zwei Applikationen gleichzeitig Ausgaben an den Drucker senden, ist eine der beiden die erste. Windows stellt alle Druckaufträge in Warteschlangen, so daß sie in der Reihenfolge abgearbeitet werden, wie die Applikationen sie erstellen.

Frage:

Sollte ich nicht einfach alle Ausgaben an das Formular schicken und dann das Formular auf den Drucker ausgeben?

Antwort:

Ein Formular wird mit der Bildschirmauflösung und nicht mit der Druckerauflösung ausgegeben. Sie erhalten eine bessere Ausgabe, wenn Sie direkt auf den Drucker drucken. Darüber hinaus möchten Sie vielleicht nicht, daß alle gedruckten Ausgaben Ihrer Applikation auf einem Formular erscheinen. Was passiert beispielsweise, wenn Ihre Applikation Scheckbeträge auf Schecks drucken soll? Sie möchten sicher nicht, daß alle Schecks zuerst auf ein Formular auf dem Bildschirm und dieses dann an den Drucker geschickt wird.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Wie stellt Ihre Applikation fest, wie viele Drucker auf einem System installiert sind?
2. Richtig/Falsch. Die folgende Deklaration deklariert zwei Variablen:
`Dim intI As Integer, prnP As Printer`
3. Welche Eigenschaft legt die Skalierungseinheit für die Eigenschaften des `Printer`- Objekts fest?
4. Wie erzwingt man, daß der Ausdruck auf einer neuen Seite erscheint?
5. Welches `If` prüft, ob ein Objekt einen bestimmten Datentyp hat?
6. Richtig/Falsch. Es ist möglich, Prozeduren nicht nur Variablen, sondern auch Objekte zu übergeben.
7. Richtig/Falsch. `KillDoc` bricht alle Druckausgaben ab, auch `Printer.Print`-Befehle und `PrintForm`-Methoden für Formulare.
8. Mit welcher Abkürzung verweisen Sie auf das aktuelle Formular?
9. Mit welcher Auflösung druckt eine `PrintForm`-Methode?
10. Welchen Wert sollten Sie der `AutoRedraw`-Eigenschaft zuweisen, bevor Sie Formulare ausdrucken?

Übungen

1. Schreiben Sie eine Codezeile, die Ihren Namen, beginnend in Spalte 32, auf dem Drucker ausgibt.
2. **Fehlersuche:** Patty, die für die Druckerprogrammierung zuständig ist, hat Probleme, ihre Berichte richtig auf den Seiten auszugeben. Sie hat vor Jahren gelernt, in BASIC zu programmieren, schreibt aber erst seit kurzem Windows-Applikationen unter Visual Basic. Zuvor nahm Patty völlig zu Recht an, daß eine normale Seite genau 66 Zeilen Text umfaßt - in einer textbasierten Programmierungsumgebung. Patty inkrementiert also ihre Integer-Zählvariable, wenn sie eine Textzeile oder eine leere Zeile ausgibt. Wenn der Zähler 66 erreicht hat, weiß sie, daß sie auf die nächste Seite wechseln muß. Jetzt arbeitet Patty unter Windows und ihre Logik funktioniert nicht mehr. Warum?
3. Ändern Sie Listing 13.2, das nach einem Farbdrucker sucht, so ab, daß es eine Boolesche Variable auf `True` setzt, falls ein Drucker gefunden wurde. Bei der Ausführung wird der Standarddrucker verwendet, falls kein Farbdrucker vorhanden ist. Die neue Boolesche Variable informiert den nachfolgenden Code, ob die Schleife einen Farbdrucker gefunden hat. Der Code soll eine Funktion sein, die einen Booleschen Wert zurückgibt.

Woche 2

Tag 14

Grafik und Multimedia

Im heutigen Kapitel lernen Sie, wie Sie unter Visual Basic Linien, Kreise und Bilder auf Ihrem Formular zeichnen. Außerdem werden Sie den Unterschied zwischen Anzeigefeldern und Bildfeldern kennenlernen. Nachdem Sie erfahren haben, wie man Grafiken einfügt, gehen Sie weiter zu Sound- und Videoclips, die Ihre Applikationen noch interessanter machen werden.

Heute unterstützen alle Computer Multimedia. Multimedia erscheint in Programmen aller Art - seien es Geschäfts-, Lern- oder Anwenderprogramme. Es gibt mehrere Anbieter, die zusätzliche Multimedia-Werkzeuge für Visual Basic entwickelt haben. Wenn Sie sehr viele Multimedia-Applikationen schreiben, sollten Sie diese Werkzeuge ausprobieren. Für den durchschnittlichen VB-Programmierer ist jedoch das Angebot der Multimedia-Werkzeuge, das Visual Basic bereitstellt, völlig ausreichend.

Heute lernen Sie die folgenden Dinge kennen:

- Anzeigefelder und Bildfelder
- Warum das Bildfeld flexibler als das Anzeigefeld ist
- Zeichenmethoden
- Punkte, Linien, Rechtecke, Kreise und Ellipsen
- Das Multimedia-Steuerelement
- Den Status des Multimedia-Geräts ermitteln

Bildfelder und Anzeigefelder

Das Bildfeld und das Anzeigefeld übernehmen beide grundsätzlich dieselbe Aufgabe. Sie ermöglichen, Bilder aus grafischen Dateien auf einem Formular anzuzeigen. Die beiden Steuerelemente weisen jedoch Unterschiede auf:

- Das Bildfeld bietet mehr Flexibilität, weil es zusätzliche Methoden und Eigenschaften unterstützt.
- Das Anzeigefeld ist effizienter und insbesondere für Applikationen auf langsameren PCs besser geeignet.



Weil die Computer heute immer schneller werden, kann man kaum noch erkennen, daß das Bildfeld weniger effizient als das Anzeigefeld ist. Wenn Sie also nicht gerade Applikationen für ältere PCs entwickeln (die man in manchen Firmen und vor allem in Schulen immer noch findet), sollten Sie das Bildfeld verwenden, um seine Flexibilität zu nutzen.

Sowohl das Bildfeld als auch das Anzeigefeld unterstützen die folgenden Grafikformate :

- Bitmaps mit der Dateinamenerweiterung .BMP
- Cursor mit der Dateinamenerweiterung .CUR
- GIF-Dateien (Graphics Interchange Format) mit der Dateinamenerweiterung .GIF
- Icons mit der Dateinamenerweiterung .ICO
- JPEG-Dateien mit der Dateinamenerweiterung .JPEG oder .JPG
- Metadateien mit der Dateinamenerweiterung .WMF oder .EMF (für Enhanced Meta Files)
- RLE-Dateien (Run-Length Encoding) mit der Dateinamenerweiterung .RLE

Der in Visual Basic enthaltene Graphics-Ordner stellt mehrere Dateien dieser Formate bereit.

Die wichtigste Eigenschaft sowohl des Bildfelds als auch des Anzeigefelds ist **Picture**, die die anzuzeigende Grafik enthält. Zur Entwurfszeit doppelklicken Sie im Eigenschaftenfenster auf die **Picture**-Eigenschaft, um das Dialogfeld **Datei öffnen** anzuzeigen und dort eine Grafikdatei mit einer der unterstützten Dateinamenerweiterungen auszuwählen. Wenn Sie zur Laufzeit ein Bild anzeigen möchten, rufen Sie die interne Funktion **LoadPicture()** auf, um der **Picture**-Eigenschaft des Steuerelements eine Grafikdatei zuzuweisen.

Die folgende Zuweisung ordnet der **Picture**-Eigenschaft eines Bildfelds eine Grafik zu:

```
picPortrait.Picture = LoadPicture("c:\MyPhotos\Charlie.wmf")
```

Beachten Sie, daß Sie der **Picture**-Eigenschaft nicht direkt einen Pfad und eine Datei zuweisen können. **LoadPicture()** ist die wichtigste Funktion für Bildfelder und Anzeigefelder. Hier ihr vollständiges Format:

```
LoadPicture([GrafikDateiName] [,varGröße] [,varFarbTiefe], [varX,  
varY])
```

Beachten Sie, daß der Grafikdateiname, also das erste Argument von **LoadPicture()**, optional ist. Bei einem Aufruf von **LoadPicture()** ohne Angabe eines Dateinamens entfernt Visual Basic das Bild aus dem Steuerelement.

Tabelle 14.1 listet die Konstantennamen auf, die Sie für das Argument **varGröße** angeben können. Dieses Argument legt die Bildgröße für Icons und Cursor fest. Das Argument **varGröße** ist sehr wichtig, weil die Benutzer häufig die Anzeigeeigenschaften aus der Systemsteuerung benutzen, um die Größe der Cursor und Icons auf ihrem System festzulegen. Es ist möglich, auf diese Systemwerte zuzugreifen.

Tabelle 14.1: Geben Sie eine dieser varGröße-Konstanten an, um die Bildgröße in LoadPicture() zu steuern, falls Sie ein Icon oder einen Cursor laden.

Konstantenname	Wert	Beschreibung

vbLPsmall	0	Kleine Systemicons, deren Größe durch Ihre Grafikauflösung bestimmt wird.
vbLPLarge	1	Große Systemicons, deren Größe durch Ihre Grafikauflösung bestimmt wird.
vbLPsmallShell	2	Wird im Anzeige -Applet der Systemsteuerung festgelegt. Klicken Sie auf die Registerkarte Darstellung . Dort sehen Sie die Größe der Bilder, die Sie mit dem Wert von varGröße ändern können.
vbLPLargeShell	3	Wird im Anzeige -Applet der Systemsteuerung festgelegt. Klicken Sie auf die Registerkarte Darstellung , um die Icon-Größe zu sehen, um die Bildgröße zu ermitteln, die Sie mit diesem varGröße -Wert ändern können.
vbLPCustom	4	Die Größe wird durch die Argumente varX und varY festgelegt.

Tabelle 14.2 listet die Werte auf, die Sie für das optionale Argument **varFarbTiefe** angeben können, wenn Sie Icons und Cursor laden.

Tabelle 14.2: Mit diesen **varFarbTiefe-Konstanten steuern Sie die Farbtiefe, die **LoadPicture()** zum Laden einer Icon- oder Cursordatei verwendet.**

Konstantenname	Wert	Beschreibung
vbLPDefault	0	Beste Übereinstimmung
vbLPMonochrome	1	2 Farben
vbLPVGAColor	2	16 Farben
vbLPColor	3	256 Farben

Die Werte **varX** und **varY** müssen dann angegeben werden, wenn Sie für die Größe die Werte **vbLPsmallShell** oder **vbLPLargeShell** verwenden.

Wenn Sie Bild- und Anzeigefelder auf einem Formular anlegen, reagieren diese etwas unterschiedlich auf dieselben Größen. Sie müssen die **Stretch**-Eigenschaft eines Anzeigefelds auf **True** setzen, bevor Sie die Eigenschaften **Width** und **Height** spezifizieren. Falls Sie das nicht tun, werden die Breite und die Höhe automatisch auf die Größe der Bitmap gebracht, die Sie in das Steuerelement geladen haben, und **Width** und **Height** werden automatisch gesetzt. Wenn Sie ein Bildfeld auf einem Formular anlegen, wird die Bildgröße automatisch angepaßt, so daß das Bild genau in Ihr Steuerelement paßt. Das Bildfeld ändert also immer die Größe der Bilder, so daß sie den Größeneigenschaften des Steuerelements entsprechen, während das Anzeigefeld seine Größeneigenschaften ändert, es sei denn, Sie setzen seine **Stretch**-Eigenschaft auf **True**.



Sie können die Funktion `LoadPicture()` nicht nur auf Bildfelder und Anzeigefelder anwenden, sondern auch auf Formulare! Sie können also eine Grafik für den Hintergrund Ihres Formulars spezifizieren. Die folgende Anweisung ordnet dem Formularhintergrund ein Scheckformular zu:

```
frmCheck.Picture = LoadPicture("apptbook.wmf")
```

Abhängig von der Position Ihrer Grafikdatei müssen Sie den Pfad eingeben. Dieser Pfad kann sehr lang werden, beispielsweise so:

```
frmCheck.Picture = LoadPicture("d:\program files\Microsoft Visual  
Studio\Common\Graphics\metafile\business\apptbook.wmf")
```

Abbildung 14.1 zeigt das resultierende Formular mit einigen Schaltflächen und Bezeichnungsfeldern. Das Bild stellt den Formularhintergrund dar. (Wenn Sie zur Entwurfszeit eine Grafikdatei auf dem Hintergrund des Formulars anzeigen möchten, verwenden Sie die `Picture`-Eigenschaft im Eigenschaftenfenster.)

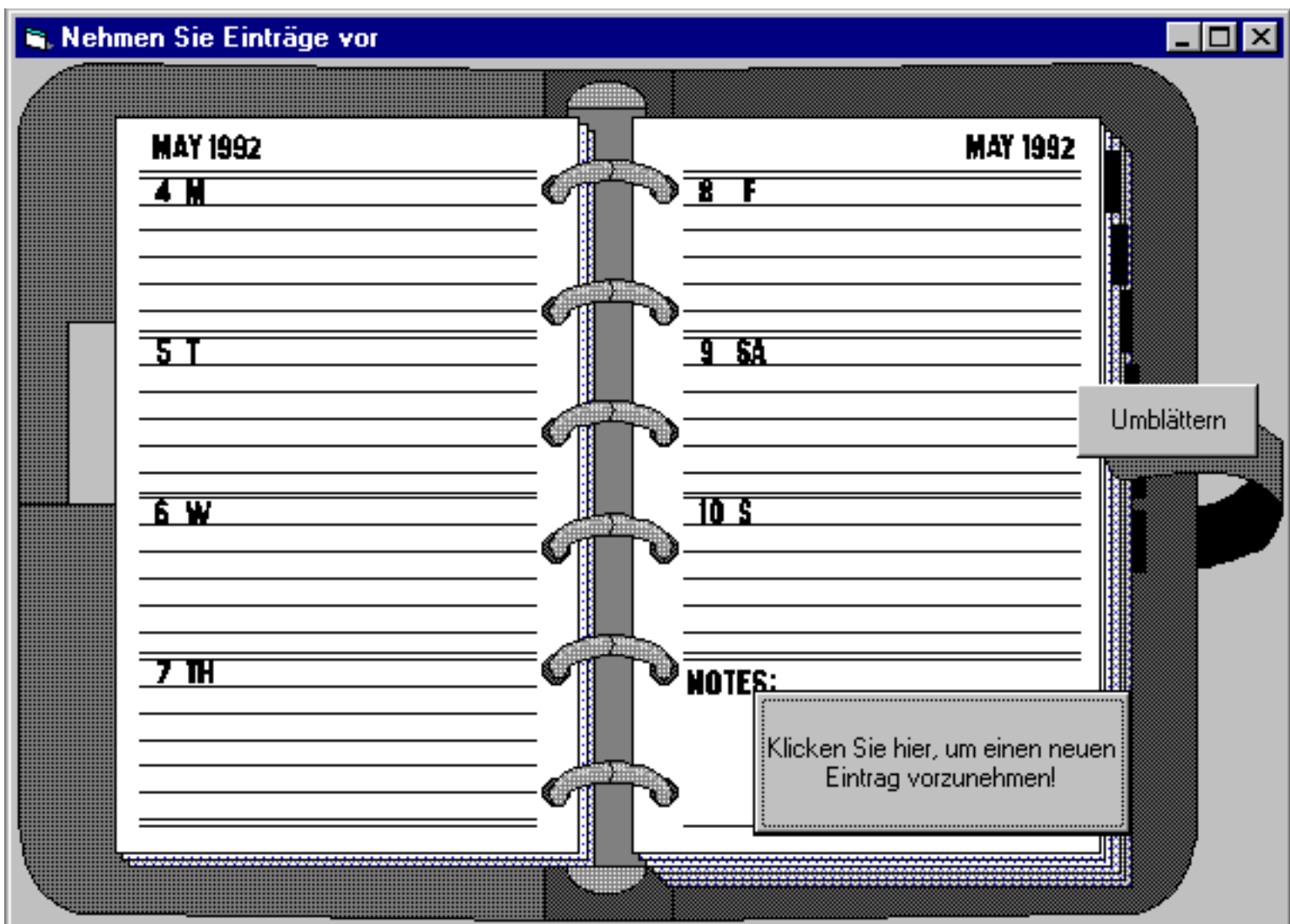


Abbildung 14.1:
Als Formularhintergrund wird eine Grafikdatei angezeigt.

Steuerelemente zum Zeichnen

Die Werkzeugsammlung enthält zwei Zeichenwerkzeuge:

- **Linien-Steuerelement** - Zeichnet gerade Linien zwischen zwei von Ihnen vorgegebenen Punkten.
- **Figur-Steuerelement** - Zeichnet abhängig von den Eigenschaftswerten einen von mehreren Umrissen.

Linien zeichnen

Es gibt zahlreiche Verwendungszwecke für das Linien-Steuerelement, auch wenn Sie in Ihren Applikationen nicht mit Multimedia-Grafik arbeiten. Das Linien-Steuerelement ist gut dazu geeignet, Informationen auf dem Formular hervorzuheben, indem man sie einfach unterstreicht. Abbildung 14.2 zeigt eine Applikation, die das Linien-Steuerelement in einem Eröffnungsbildschirm einsetzt, das fünf Sekunden lang erscheint, nachdem der Benutzer die Applikation aufgerufen hat.

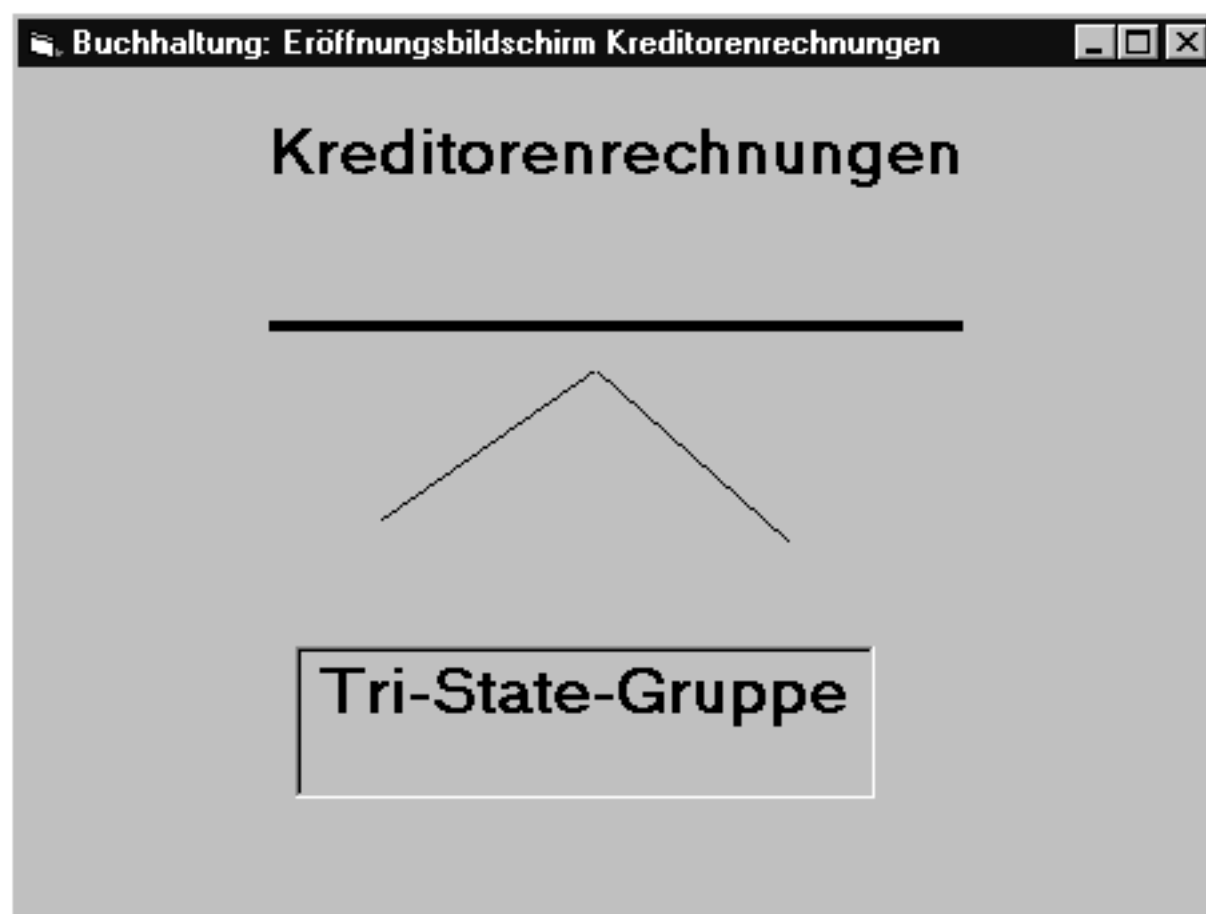


Abbildung 14.2:
Linien markieren wichtige Aussagen auf einem Formular.

Wenn Sie auf das Linien-Steuerelement doppelklicken, legt Visual Basic eine Linie in der Formularmitte an, die an beiden Enden Griffe zur Größenänderung besitzt. Sie benutzen diese Griffe, um die Linie zu verlängern oder zu verkürzen oder um die Richtung der Linie zu ändern. Während Sie die Größenänderung vornehmen, aktualisiert Visual Basic die Eigenschaftswerte, die die Linie definieren.

Die folgenden Eigenschaften sollten Sie verstanden haben, bevor Sie Linien auf Ihrem Formular einsetzen:

- **BorderColor** bestimmt die Linienfarbe.
- **BorderStyle** bestimmt das Linienformat, wie in den Werten in Tabelle 14.3 und in Abbildung 14.3 gezeigt.
- **BorderWidth** legt die Linienbreite fest (in Punkten; ein Punkt ist 1/72 Inch).
- **X1, Y1, X2, Y2** legen die Start- und Endkoordinaten der Linie fest. Ein Punkt auf Ihrem Formular wird durch zwei Werte definiert, und eine Linie wird durch zwei Punkte definiert (die Linie verbindet die beiden Punkte).

Tabelle 14.3: Die **BorderStyle-Eigenschaft bestimmt, wie die Linien auf dem Formular erscheinen**

Konstantenname	Beschreibung
0 - Transparent	Die Hintergrundfarbe des Formulars wird durch die Linie angezeigt.
1 - Ausgefüllt	Ausgefüllte Linie.
2 - Strich	Gestrichelte Linie.
3 - Punkt	Punktierte Linie.
4 - Strich-Punkt	Jedem Strich folgt ein Punkt.
5 - Strich-Punkt-Punkt	Jedem Strich folgen zwei Punkte.
6 - Innen ausgefüllt	Die Linie wird innen ausgefüllt dargestellt.

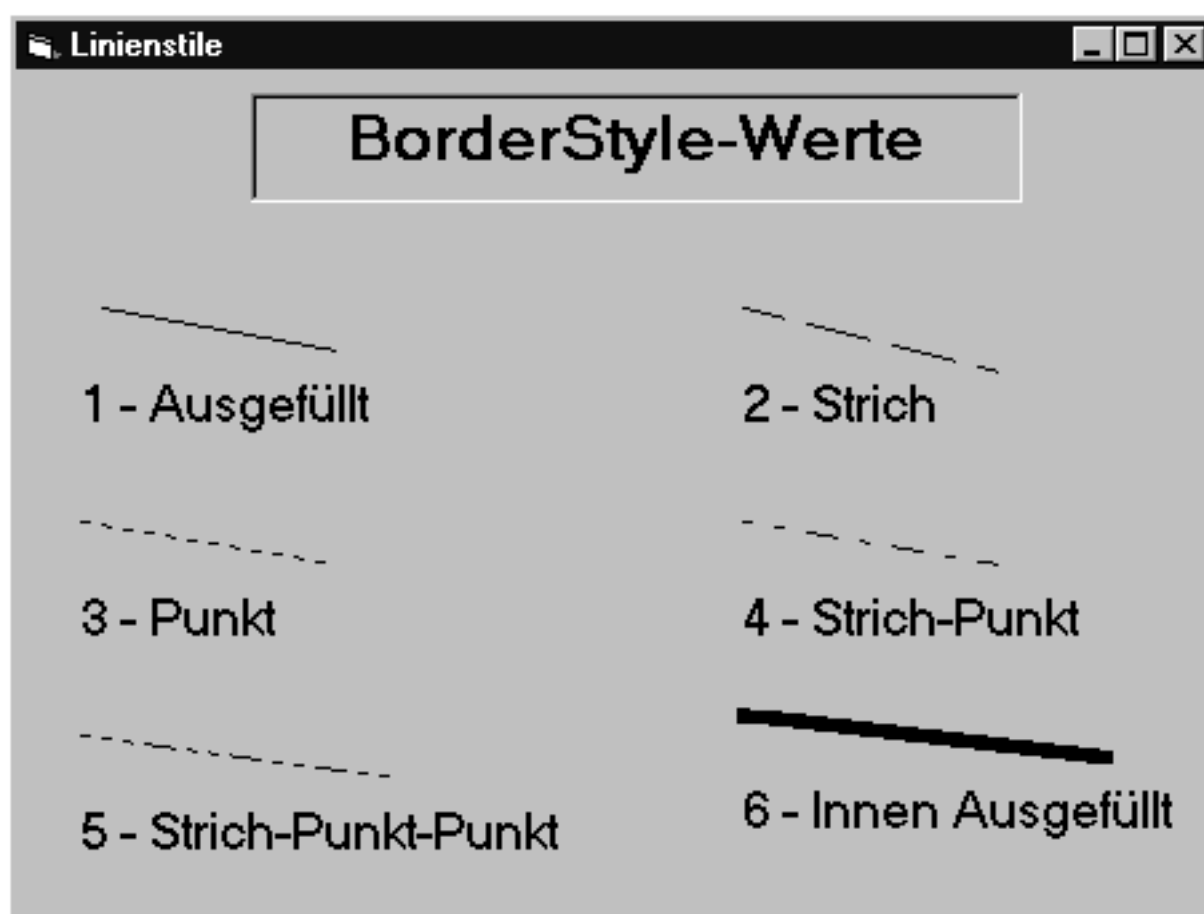


Abbildung 14.3:
Die **BorderStyle**-Eigenschaft legt fest, wie Visual Basic eine Linie zeichnet.

Umrisse zeichnen

Während das Linien-Steuerelement nur Linien zeichnet, kann das Figur-Steuerelement verschiedene Umrisse zeichnen. Die **Shape**-Eigenschaft legt fest, welcher Umriß erzeugt wird, wie in der folgenden Liste beschrieben:

- `0` - `Rectangle` zeichnet ein Rechteck.
- `1` - `Square` zeichnet ein Quadrat.
- `2` - `Oval` zeichnet eine Ellipse.
- `3` - `Circle` zeichnet einen Kreis.
- `4` - `RoundedRectangle` zeichnet ein Rechteck mit abgerundeten Ecken.
- `5` - `RoundedSquare` zeichnet ein Quadrat mit abgerundeten Ecken.

Neben den Werten für die **Shape**-Eigenschaft gibt es noch viele andere wichtige Eigenschaftswerte, die das Erscheinungsbild des Umrisses auf dem Bildschirm beeinflussen (siehe Tabelle 14.4).

Tabelle 14.4: Diese Eigenschaften des Figur-Steuerelements bestimmen, wie die gezeichnete Form aussieht

Eigenschaft	Beschreibung
<code>BackStyle</code>	Wenn <code>BackStyle</code> auf <code>True</code> gesetzt ist, wird der Hintergrund durch die Figur angezeigt, so als ob diese durchsichtig wäre.
<code>BorderColor</code>	Die Rahmenfarbe für die Figur.
<code>BorderStyle</code>	Nimmt einen der Werte aus Tabelle 14.3 an, und stellt die Umrißlinie der Figur entsprechend dar.
<code>BorderWidth</code>	Die Breite der Umrißlinie, angegeben in Twips.
<code>FillColor</code>	Die Farbe für das Füllmuster (angegeben durch die Eigenschaft <code>FillStyle</code>).
<code>FillStyle</code>	Legt das Füllmuster für die Figur fest. Abbildung 14.4 zeigt Beispiele für die acht verschiedenen Füllmuster.
<code>Height</code>	Höchster Punkt der Figur.
<code>Width</code>	Breitester Durchmesser der Figur.

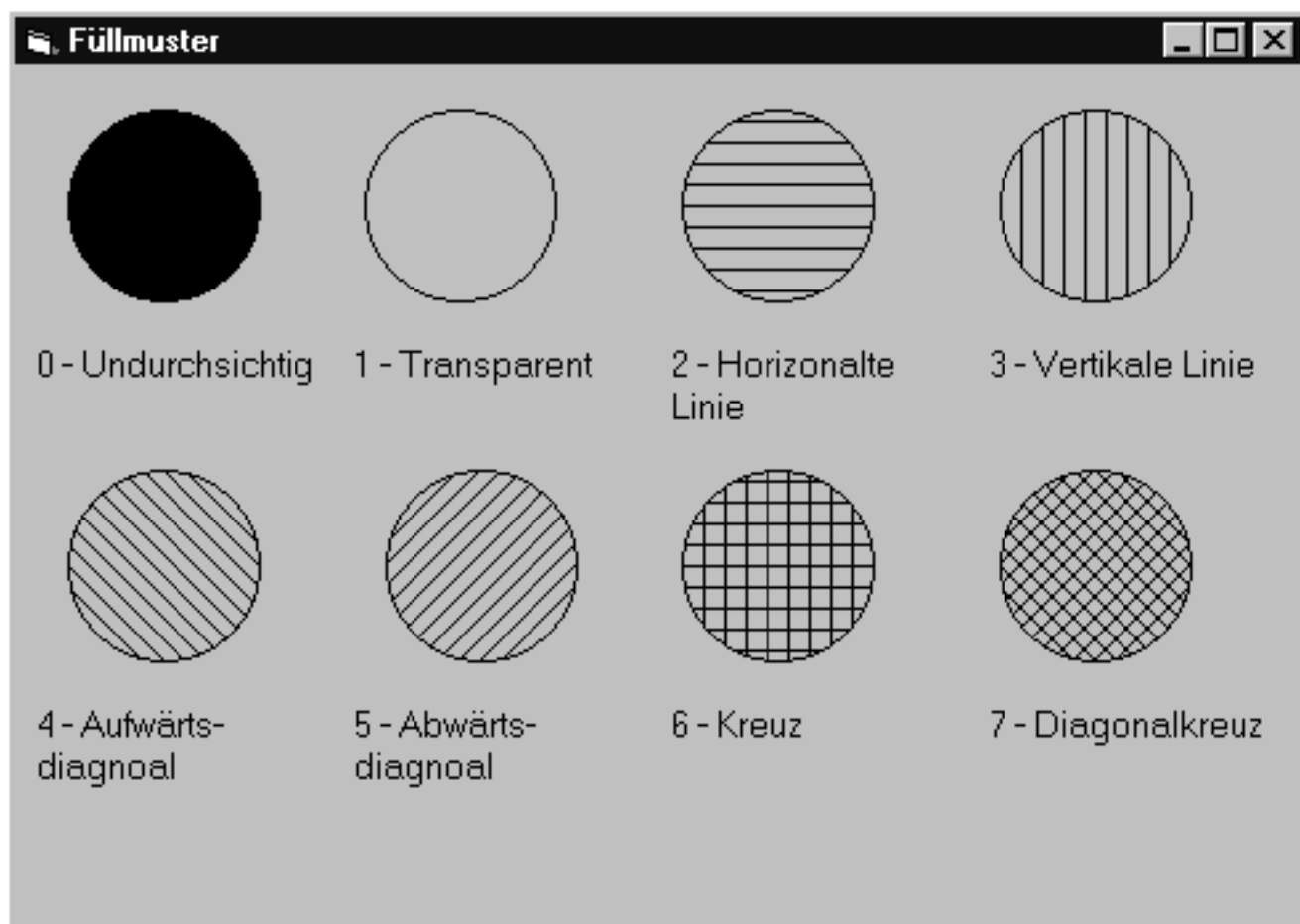


Abbildung 14.4:

Diese acht FillStyle-Muster legen fest, wie Visual Basic das Innere eines Umrisses darstellt.

Jetzt entwickeln Sie eine kleine Applikation, die die Eigenschaften eines Figur-Steuerlements ändert. Abbildung 14.5 zeigt das Ergebnis. Durch Auswahl aus einem der beiden Listenfelder ändern Sie den Umriß und das Füllmuster oben im Formular. Der Umriß wird geändert, sobald Sie einen neuen Wert aus einem der Listenfelder ausgewählt haben.

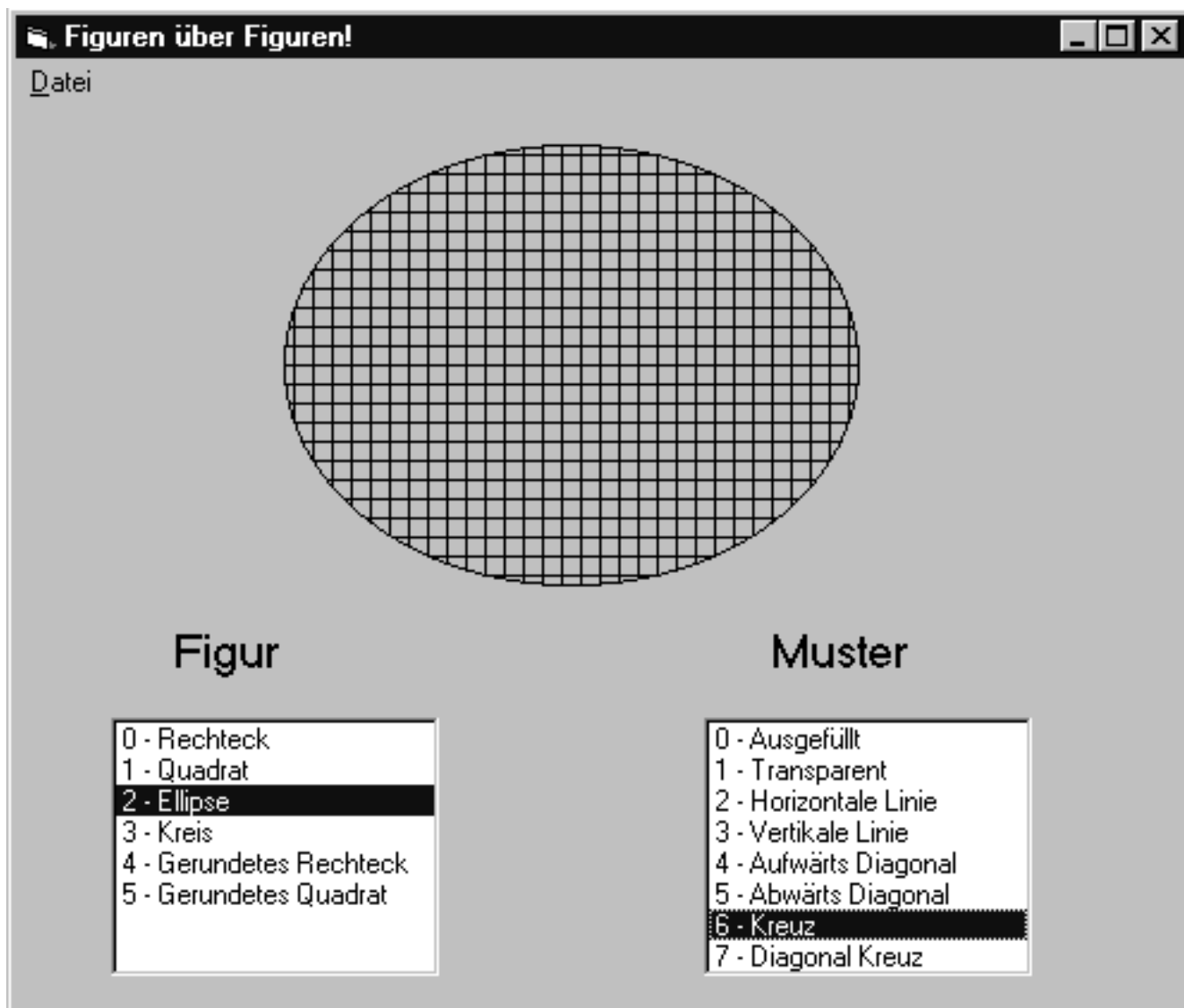


Abbildung 14.5:
Wählen Sie eine Figur und ein Muster aus und beobachten Sie das Ergebnis.

Gehen Sie wie folgt vor, um diese Beispielanwendung zu erzeugen:

1. Legen Sie eine neue Applikation an.
2. Plazieren Sie die in Tabelle 14.5 beschriebenen Steuerelemente auf dem Formular.
3. Fügen Sie den Code aus Listing 14.1 ein, um die beiden Listenfelder zur Laufzeit zu initialisieren.
4. Bei der Ausführung der Applikation wählen Sie in den beiden Listenfeldern eine beliebige Kombination aus Umriß und Füllmuster aus.

Tabelle 14.5: Geben Sie den Steuerelementen auf dem Formular die folgenden Eigenschaften

Steuerelement	Beschreibung
Formular Name	frmShape
Formular Height	7005
Formular Left	105
Formular Top	105

Formular Width	7965
Menüoption #1 Name	mnuFile
Menüoption #1 Caption	&Datei
Menüoption #2 Name	mnuFileExit (um eine Ebene eingerückt)
Menüoption #2 Caption	&Beenden
Figur Name	shpSample
Figur Height	2025
Figur Left	1710
Figur Top	720
Figur Width	4560
Bezeichnungsfeld #1 Name	lblShape
Bezeichnungsfeld #1 Caption	Shape
Bezeichnungsfeld #1 Height	420
Bezeichnungsfeld #1 Left	2160
Bezeichnungsfeld #1 Top	3390
Bezeichnungsfeld #1 Width	1215
Bezeichnungsfeld #2 Name	lblPattern
Bezeichnungsfeld #2 Caption	Pattern
Bezeichnungsfeld #2 Height	420
Bezeichnungsfeld #2 Left	5040
Bezeichnungsfeld #2 Top	3360
Bezeichnungsfeld #2 Width	1215
Listenfeld #1 Name	lstShape
Listenfeld #1 Height	1425
Listenfeld #1 Left	1560
Listenfeld #1 Top	3960
Listenfeld #1 Width	2010
Listenfeld #2 Name	lstPattern
Listenfeld #2 Height	1425
Listenfeld #2 Left	4680
Listenfeld #2 Top	3960
Listenfeld #2 Width	2010

Listing 14.1: Code für die Initialisierung der beiden Listenfelder und die Reaktion auf die vom Benutzer getroffene Auswahl

```
1: Private Sub Form_Load()  
2: ' Dropdown-Listenfeld Figur initialisieren  
3: lstShape.AddItem "0 - Rechteck"  
4: lstShape.AddItem "1 - Quadrat"  
5: lstShape.AddItem "2 - Oval"  
6: lstShape.AddItem "3 - Kreis"  
7: lstShape.AddItem "4 - Gerundetes Rechteck"  
8: lstShape.AddItem "5 - Gerundetes Quadrat"  
9:  
10: ' Dropdown-Listenfeld Füllmuster initialisieren  
11: lstPattern.AddItem "0 - Ausgefüllt"  
12: lstPattern.AddItem "1 - Transparent"  
13: lstPattern.AddItem "2 - Horizontale Linie"  
14: lstPattern.AddItem "3 - Vertikale Linie"  
15: lstPattern.AddItem "4 - Aufwärts Diagonal"  
16: lstPattern.AddItem "5 - Abwärts Diagonal"  
17: lstPattern.AddItem "6 - Kreuz"  
18: lstPattern.AddItem "7 - Diagonal Kreuz"  
19:  
20: ' Ersten Listenwert als Vorgabe bestimmen  
21: lstShape.ListIndex = 0  
22: lstPattern.ListIndex = 0  
23:  
24: End Sub  
25:  
26: Private Sub lstPattern_Click()  
27: ' Muster entsprechend der Auswahl ändern  
28: shpSample.FillStyle = lstPattern.ListIndex  
29: End Sub  
30:  
31: Private Sub lstShape_Click()  
32: ' Figur gemäß der Auswahl ändern  
33: shpSample.Shape = lstShape.ListIndex  
34: End Sub  
35:  
36: Private Sub mnuFileExit_Click()  
37: End  
38: End Sub
```



Weder das Linien- noch das Figur-Steuerelement unterstützen eine Ereignisverarbeitung.

Zeichenmethoden

Visual Basic beschränkt Ihre Grafik nicht auf Linien- und Figur-Steuerelemente. Es ist ohne weiteres möglich, die einzelnen Pixel einer Grafik zu ändern und ein Bild aus vielen Punkten zusammenzusetzen. Die Methode **PSet**, die Sie auf Formularen anwenden, ermöglicht Ihnen, Linien, Rechtecke und Kreise zu zeichnen, ohne dazu Steuerelemente einzusetzen. Wenn Sie zur Laufzeit Linien und Kreise zeichnen möchten, sind die Steuerelemente etwas kompliziert und ermöglichen keine exakten Zeichnungen, wie das mit den Zeichenmethoden der Fall ist.



Sie können die Zeichenmethoden auf Bildfelder anwenden, die Sie auf dem Formular ablegen, aber auch auf das Formular selbst.

Die Methode **PSet** zeigt einzelne Pixel auf einem Formular an bzw. verbirgt sie. Hier das Format von **PSet**:

```
frmName.PSet [Step] (intX, intY) [farbe]
```

Sie können die Skalierung durch die Eigenschaften **ScaleX** und **ScaleY** ändern, aber der Zeilen- und Spaltenschnittpunkt 0,0 (im Format der **PSet**-Methode (**intX**, **intY**)), bezieht sich auf die linke obere Ecke im Formular. Die folgende Anweisung aktiviert das Pixel, das sich 100 Pixel rechts und 200 Pixel unter der linken Formularecke befindet:

```
frmDraw.PSet (100, 200) ' Ein Pixel aktivieren
```

Die Farbe des Pixels ist immer der **ForeColor**-Wert des Formulars (oder des Bildfelds). Sie können die Farbe ändern, indem Sie einen anderen hexadezimalen Farbwert spezifizieren oder eine der folgenden Farbkonstantennamen verwenden: **vbBlack**, **vbRed**, **vbGreen**, **vbYellow**, **vbBlue**, **vbMagenta**, **vbCyan** oder **vbWhite**.

Die folgende Anweisung deaktiviert das Pixel an der Position (100, 200), indem sie die Farbe auf die aktuelle Hintergrundfarbe des Formulars setzt:

```
frmDraw.PSet (100, 200) frmDraw.BackColor ' Deaktiviert ein Pixel
```

Wenn Sie nach dieser **PSet**-Methode die folgende ausführen, erscheint ein weiteres Pixel an der Position (300, 350):

```
frmDraw.PSet (300, 350) ' Aktiviert ein Pixel
```

Die **Step**-Option der Methode **PSet** ändert die Position folgender Pixelplatzierungen. Wenn Sie **PSet** zum ersten Mal aufrufen, macht die **Step**-Option die **intX**- und **intY**-Pixelwerte der nächsten **PSet**-Methode relativ. Wenn Sie also der obigen **PSet**-Methode die **Step**-Option hinzugefügt hätten, wie es in der folgenden Anweisung passiert, würde das Pixel 300 Pixel rechts und 350 Pixel unterhalb des zuvor gezeichneten Pixels erscheinen:

```
frmDraw.PSet Step (300, 350) ' Relative Pixelposition
```

Mit **PSet** können Linien in Schleifen gezeichnet werden:

```
For intX = 1 to 100  
frmDraw.PSet (intX, 250)  
Next intX
```



Verlassen Sie niemals die Pixelgrenzen des Formulars, auf dem Sie zeichnen, sonst tritt ein Laufzeitfehler auf.

Statt Linien pixelweise zu zeichnen, setzen Sie eine andere Methode ein, `Line`, die die Linien effizienter zeichnet. Hier das Format der `Line`-Methode:

```
frmName.Line [Step] (intX1, intY1) - [Step] (intX2, intY2), [Color]  
[B][F]
```

Die beiden Pixelkoordinatenpaare legen die Anfangs- und Endpunkte der Linie fest. Die `Step`-Option, die Sie vor jedem Koordinatenpaar angeben können, wandelt diese in relative Koordinaten in bezug auf den Endpunkt der zuletzt gezeichneten Linie um.

Die folgende Methode zeichnet eine Linie von Pixel 100,100 bis Pixel 150,150.

```
frmDraw.Line (100, 100) - (150, 150)
```

Wie bei der `PSet`-Methode ermöglicht der optionale `farbe`-Wert die Angabe eines hexadezimalen Farbwerts oder eines Farbkonstantennamens. Wenn Sie keine Farbe angeben, verwendet Visual Basic die Vordergrundfarbe des Formulars, um die Linie zu zeichnen.

Um ein Rechteck zu zeichnen, verwenden Sie für die `Line`-Methode die `B`-Option:

```
frmDraw.Line (100, 100) - (150, 150), , B
```

Das zusätzliche Komma muß auch angegeben werden, wenn Sie keinen Farbwert spezifizieren, damit Visual Basic erkennt, daß das `B` die Option zum Zeichnen eines Rechtecks ist. Die beiden Koordinatenpaare geben jetzt die obere linke und die untere rechte Ecke für das Rechteck an.



Die Koordinatenpaare für Linien und Rechtecke geben entgegengesetzte Endpunkte für die Linie bzw. diagonal entgegengesetzte Eckpunkte für das Rechteck an. Linien können in beide Richtungen geneigt werden; das erste Koordinatenpaar kann einen Punkt enthalten, der unterhalb oder rechts vom zweiten Koordinatenpaar liegt.

Das Innere des Rechtecks wird in der Farbe ausgefüllt, die auch die Umrißlinie hat, falls Sie die `F`-Option spezifizieren:

```
frmForm.Line (35, 40) - (150, 175), vbGreen, BF ' Ein grünes  
Rechteck
```

Diese Anweisung zeichnet ein ausgefülltes Rechteck, wobei für die Umrißlinie und das Füllmuster dieselbe Farbe verwendet wird. Wenn Sie ein Rechteck außerhalb der Formulargrenzen zeichnen, schneidet Visual Basic das Rechteck an der Formulargrenze ab. Auch wenn Sie das Formular vergrößern, nachdem Sie ein Rechteck gezeichnet haben, das nicht auf das Formular paßte, wird nie das gesamte Rechteck angezeigt, es sei denn, Sie zeichnen es noch einmal.

Listing 14.2 zeigt die Ereignisprozedur für eine Schaltfläche, die mehrere Rechtecke von der linken oberen Ecke zur rechten unteren Ecke des Formulars zeichnet. Die Ausführung dieser Prozedur erzeugt die in Abbildung 14.6 gezeigten Linien.

Listing 14.2: Mit Hilfe mehrere Line-Methoden zeichnen Sie Muster.

```
1: Private Sub cmdBoxes_Click()
```

```
2: Dim intStartX As Integer
3: Dim intStartY As Integer
4: Dim intLastX As Integer
5: Dim intLastY As Integer
6:
7:
8: intStartX = 0
9: intStartY = 0
10: intLastX = 1000
11: intLastY = 800
12:
13: For intCtr = 1 To 20
14: frmBoxes.Line (intStartX, intStartY)-(intLastX, intLastY), , B
15:
16: ' Vorbereitung für die nächsten Rechtecke
17: intStartX = intStartX + 400
18: intStartY = intStartY + 400
19: intLastX = intLastX + 400
20: intLastY = intLastY + 400
21: Next intCtr
22:
23: End Sub
```

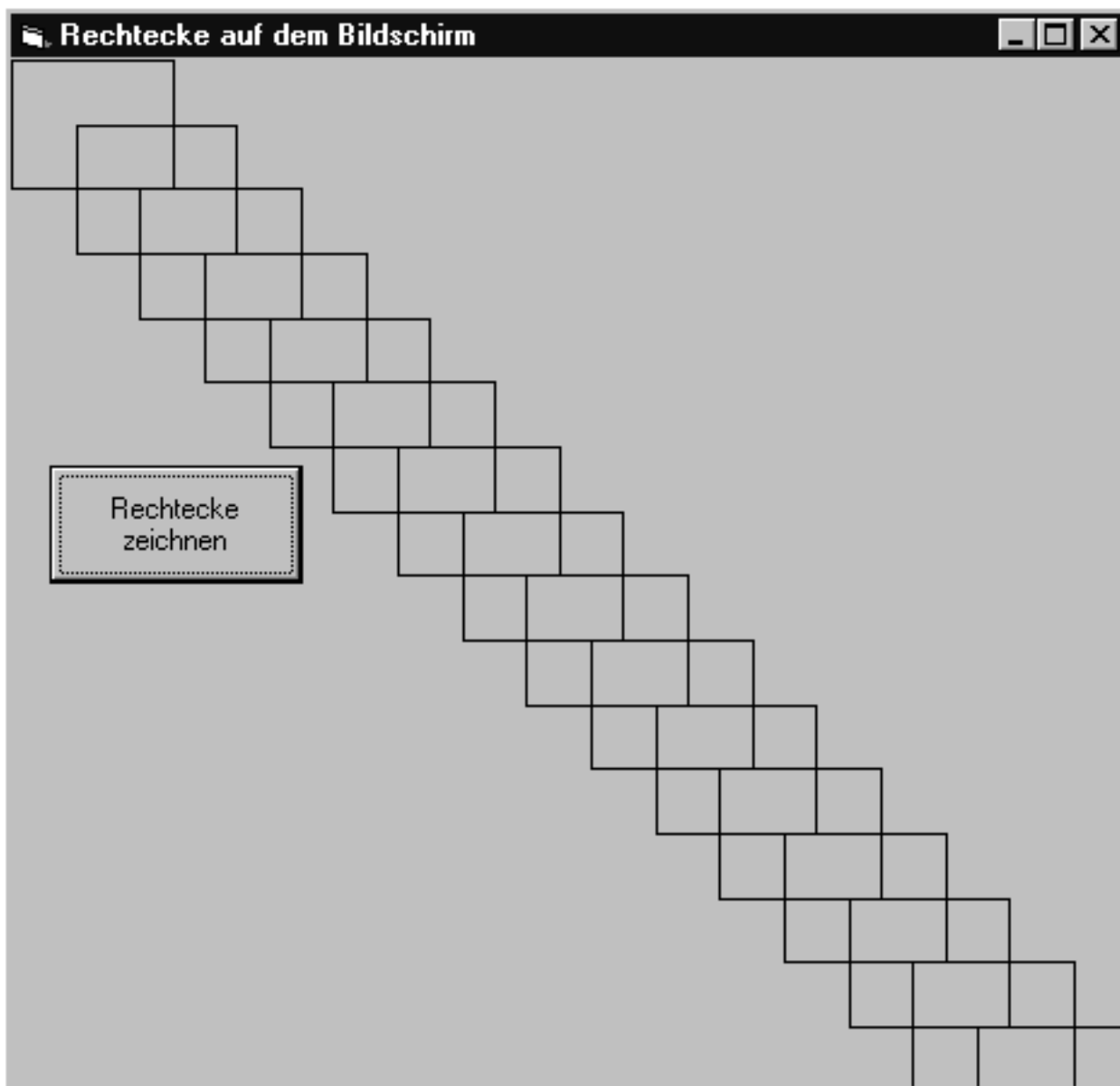



Abbildung 14.6:
Mit der `Line`-Methode werden Rechtecke auf ein Formular gezeichnet.

Die Zeichenmethoden erzeugen nicht nur gerade Linien, sondern können auch für Kreise und Ellipsen eingesetzt werden. Hier das Format für die `Circle`-Methode, mit der Sie Kreise und Ellipsen zeichnen:

```
frmDraw.Circle [Step] (intX, intY) sngRadius, [Color], , , ,  
sngAspect
```



Eine *Ellipse* ist ein in die Länge gezogener Kreis mit ovalem Umriß.



Die Kommas sind Platzhalter für komplexe `Circle`-Argumente, die in diesem Buch nicht beschrieben werden sollen. Wenn Sie das `sngAspect`-Argument verwenden möchten, müssen Sie diese Kommas setzen.

Ein Kreis hat keine Endpunkte; die Koordinaten `intX` und `intY` spezifizieren den Kreismittelpunkt. `sngRadius` enthält den Radius, angegeben in Pixel (es sei denn, Sie haben die

ScaleMode-Eigenschaft für das Formular geändert). Das optionale Schlüsselwort **Step** gibt an, ob das Koordinatenpaar für den Mittelpunkt relativ zu einem zuvor gezeichneten Objekt ist.



Der *Radius* eines Kreises ist der Abstand von einem Punkt auf der Kreislinie vom Kreismittelpunkt.

Die folgende Anweisung zeichnet einen Kreis, dessen Mittelpunkt sich 300 Pixel vom linken Rand und 200 Pixel vom oberen Rand des Formulars befindet, und der einen Radius von 100 Pixel hat:

```
frmDraw.Circle (300, 200), 100
```

Die **Circle**-Methode verwendet die Vordergrundfarbe des Formulars, es sei denn, Sie spezifizieren einen anderen **farbe**-Wert.

Um eine Ellipse zu zeichnen, geben Sie deren Form mit dem Argument **sngAspect** an, um ihr Aspektverhältnis zu bestimmen. Der Aspektwert streckt den Kreis entlang der horizontalen X-Koordinatenachse (für einen **sngAspect**-Wert kleiner 1) oder entlang der vertikalen Y-Koordinatenachse (für einen **sngAspect**-Wert größer 1). Der **sngAspect** -Wert 1 bewirkt, daß ein perfekter Kreis gezeichnet wird.



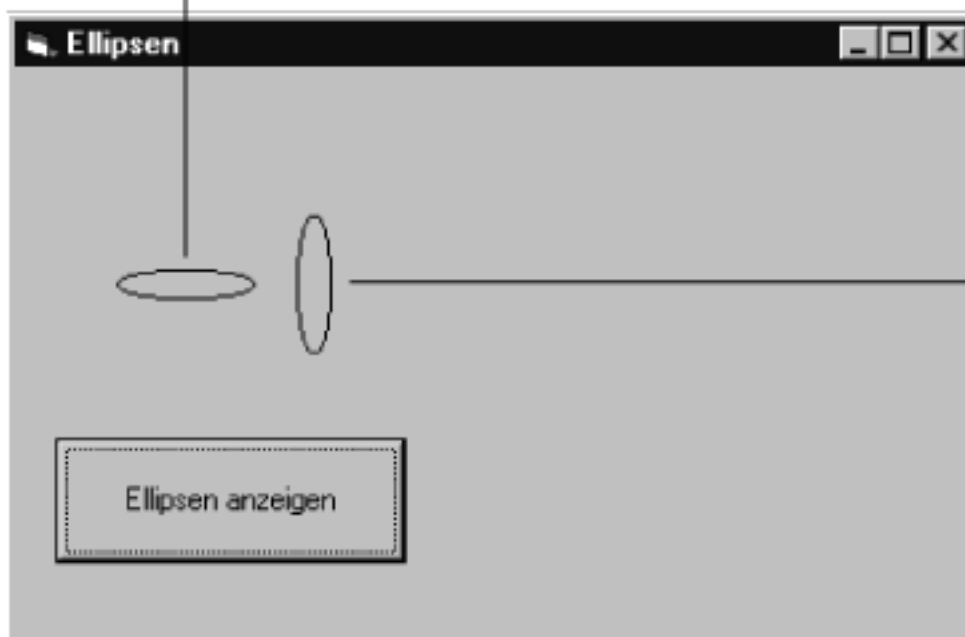
Das *Aspektverhältnis* legt fest, wie ausladend der ovale Umriß der Ellipse sein soll. Er verhält sich wie ein Breiten- und Höhenmaß durch den Mittelpunkt der Ellipse. Das Verhältnis dient außerdem als Multiplikator des Radius in alle Richtungen. Ein Aspektverhältnis von 4 beispielsweise bedeutet, daß die Ellipse viermal länger als hoch ist. Ein Aspektverhältnis von 4/10/ 2 (oder 0.2) bedeutet, daß die Ellipse horizontal fünfmal so lang wie hoch ist.

Die folgende Anweisung zeichnet die in Abbildung 14.7 gezeigten Ellipsen:

```
frmDraw.Circle (1000, 1250), 400, , , , (4 / 10 / 2)
```

```
frmDraw.Circle (1750, 1250), 400, , , , 4
```

Aspektverhältnis 0.2



Aspektverhältnis 4

Abbildung 14.7:
Das Aspektverhältnis bestimmt die Form einer Ellipse.

Das Multimedia-Steuerelement

Das Multimedia-Steuerelement ist extrem leistungsfähig - und dennoch äußerst einfach zu bedienen. Sie brauchen nur sehr wenig Code zu schreiben, um es einsetzen zu können. Mit Hilfe des Multimedia-Steuerelements betten Sie Objekte ein, die die folgenden einfachen Multimedia-Geräte repräsentieren:

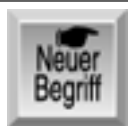
- CD-Player (CDAudio)
- Digitales Audiogerät (DAT)
- Digitale Videodateien (DigitalVideo)
- Overlay (Over lay)
- Scanner (Scanner)
- Video-Player und -Recorder (Vcr)
- Videodisc-Player (Videodisc)
- Andere Geräte, die nicht spezifiziert sind und von Drittherstellern unterstützt werden (Other)



Die Werte in Klammern werden in der **DeviceType**-Eigenschaft des Multimedia-Steuerelements verwendet, die im nächsten Abschnitt beschrieben wird.

Sie können außerdem Objekte einbetten, die die folgenden zusammengesetzten Multimedia-Geräte darstellen:

- Der Audiodatei-Player und -Recorder spielt Wellenformdateien ab und zeichnet sie auf (Dateien mit der Dateinamenerweiterung .WAV).
- Der MIDI-Sequencer spielt MIDI-Dateien (Musical Instrument Digital Interface) ab (Dateien mit der Dateinamenerweiterung .MID).
- Der Videodatei-Player und -Recorder spielt AVI-Dateien (Audio Visual Interleave) ab und zeichnet sie auf (Dateien mit der Dateinamenerweiterung .AVI).



Ein *einfaches Multimedia-Gerät* braucht keine Datei für das Steuerelement. Beispielsweise ist einer Audio-CD, die Sie in Ihr CD-ROM-Laufwerk einlegen, keine Datei zugeordnet; die CD enthält Musik, und man braucht keine Datei dafür. Ein *zusammengesetztes Multimedia-Gerät* braucht eine zusätzliche Datei für die Daten, wie beispielsweise ein Audiodatei-Player, der eine WAV-Datei braucht, um zu wissen, welche Töne er abspielen soll.

Natürlich muß der PC, auf dem Sie Ihre Multimedia-Applikationen ausführen, die Standard-Multimediageräte unterstützen, beispielsweise Sound, Grafik und Video. Außerdem braucht er ein CD-ROM-Laufwerk (oder ein kompatibles Gerät, beispielsweise ein DVD-Laufwerk).

Anwendung des Steuerelements

Wie das Standarddialog-Steuerelement muß auch das Multimedia-Steuerelement in die Werkzeugsammlung eingefügt werden, weil es nicht standardmäßig dort enthalten ist. Dazu drücken Sie (Strg)+(T), um das Dialogfeld **Komponenten** anzuzeigen. Dort wählen Sie das Steuerelement *Microsoft Multimedia Control 6.0* aus. Nachdem Sie **OK** geklickt haben, fügt Visual Basic das Steuerelement als letztes Werkzeug in Ihrer Werkzeugsammlung ein.

Wenn Sie das Multimedia-Steuerelement auf einem Formular anlegen, erscheinen vertraute Schaltflächen. Sie sehen ähnlich aus wie die Schaltfläche auf Ihrem Videorecorder oder Kassettenrecorder (siehe Abbildung 14.8).

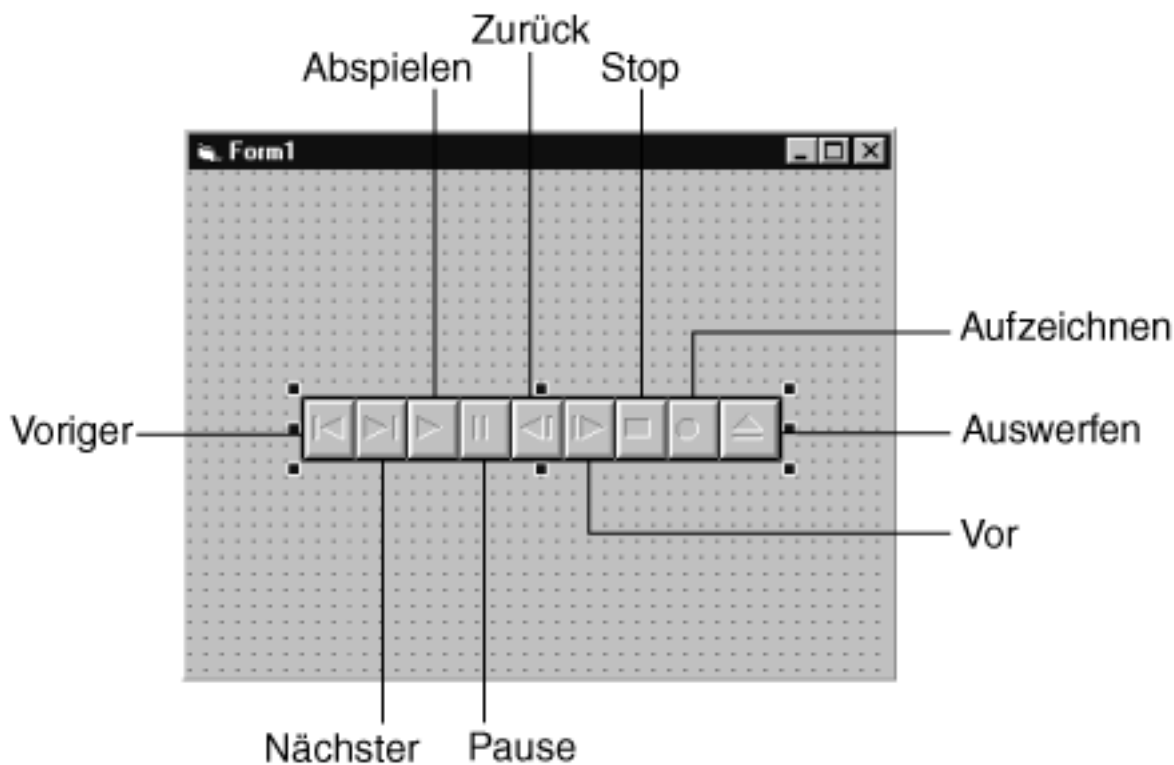


Abbildung 14.8:
Das Multimedia-Steuerelement erzeugt diese Schaltflächen für ein Multimedia-Gerät.

Das Multimedia-Steuerelement ist ein intelligentes Steuerelement, das erkennt, welche Funktionen das Gerät unterstützt, dem Sie es zuordnen. Die Schaltfläche **Abspielen (Play)** wird nicht aktiviert, wenn Sie die CD aus dem CD-ROM-Laufwerk entfernt haben. Natürlich haben Sie über zahlreiche Eigenschaftseinstellungen die komplette Kontrolle darüber, welche Schaltflächen auf dem Steuerelement angezeigt werden. Statt jedoch eine deaktivierte **Aufzeichnen**-Schaltfläche im Multimedia-Steuerelement anzuzeigen, wenn Sie Audio-CDs abspielen (die nicht überschrieben werden können), sollten Sie diese Schaltfläche besser ganz verbergen.



Das Multimedia-Steuerelement stellt Ihrer Applikation Informationen über das Gerät und seine aktuellen Einstellungen bereit. Ihre Applikation kann beispielsweise die Spurnummern einer Audio-CD anzeigen, während diese im CD-ROM-Laufwerk abgespielt wird.

Das Multimedia-Steuerelement enthält eine Liste von Geräten mit vorselektierten Schaltflächen, so daß Sie keine einzelnen Schaltflächen auswählen müssen, die auf einem Steuerelement erscheinen

sollen. Es unterstützt die Eigenschaft **DeviceType**, die, wenn Sie sie mit dem gewünschten Gerät initialisieren (beispielsweise **CDAudio**, d.h. Sie möchten mit dem Multimedia-Steuerelement eine Audio-CD abspielen), automatisch die erforderlichen Schaltflächen anzeigt. In der **DeviceType**-Eigenschaft können alle Geräte angegeben werden, die in der Liste im vorigen Abschnitt aufgezählt wurden. Ihre Aufgabe ist es, der Applikation mitzuteilen, daß das Multimedia-Steuerelement ein CD-Audioplayer ist; das Multimedia-Steuerelement übernimmt es, die richtigen Schaltflächen anzuzeigen.

Wie viele der komplexeren Steuerelemente, beispielsweise das Standarddialog-Steuerelement, enthält auch das Multimedia-Steuerelement den Eintrag (**Benutzerdefiniert**) im Eigenschaftenfenster, den Sie auswählen, um ein Dialogfeld anzuzeigen, das die Auswahl von Eigenschaften vereinfacht. Wenn Sie beispielsweise auf die Registerkarte **Steuerelemente** klicken, erscheint das in Abbildung 14.9 gezeigte Dialogfeld. Hier können Sie exakt bestimmen, welche Schaltflächen auf dem Steuerelement angezeigt werden sollen. Außerdem können Sie festlegen, ob eine Schaltfläche auf dem Steuerelement aktiviert oder deaktiviert angezeigt werden soll.



Das Multimedia-Steuerelement zeigt automatisch alle Schaltflächen an, deaktiviert sie aber, bis Sie eine oder mehrere aktivieren, oder bis Sie einen **DeviceType**-Wert auswählen, der angibt, welche Schaltflächen im Multimedia-Steuerelement sichtbar und aktiviert sein sollen.

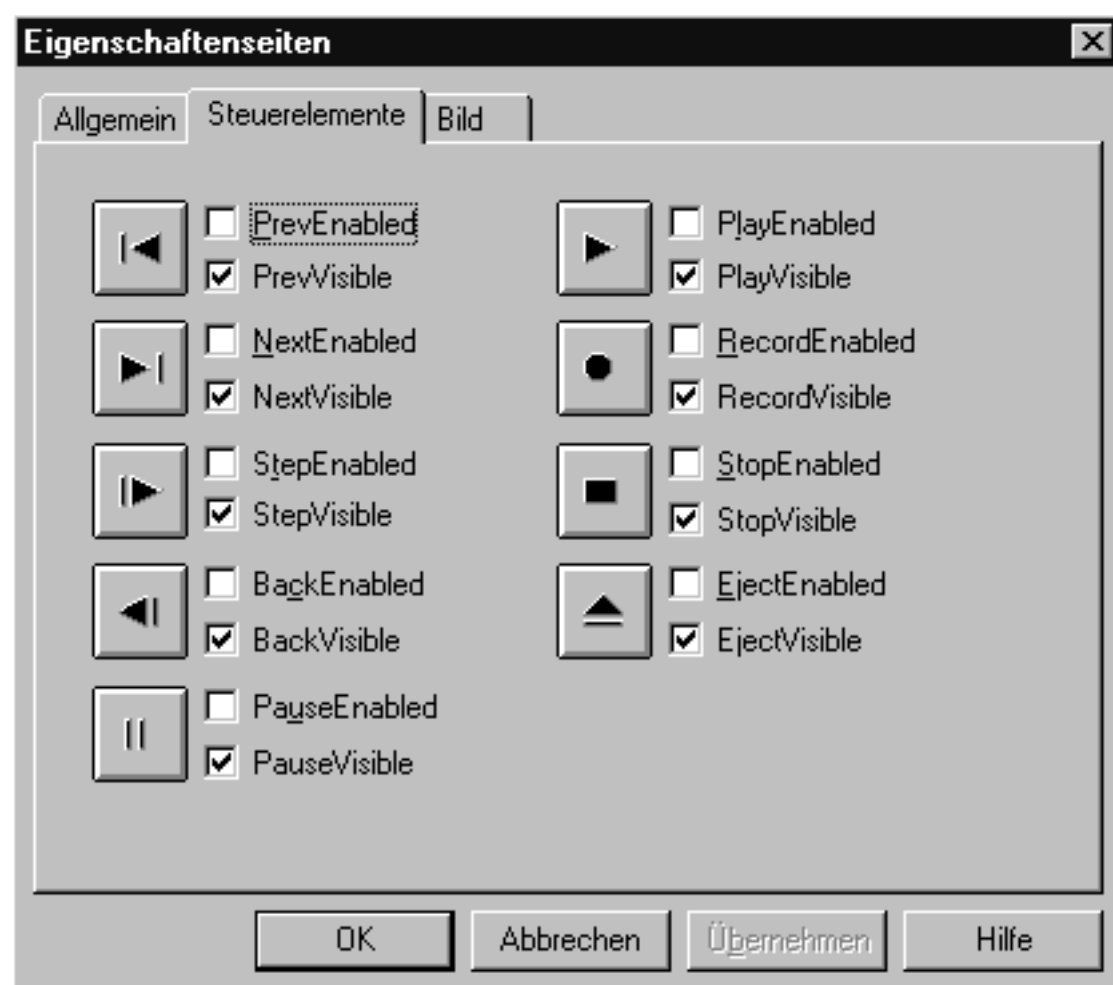


Abbildung 14.9:
Der Eintrag (**Benutzerdefiniert**) im Eigenschaftenfenster macht die Auswahl von Eigenschaften, wie beispielsweise, welche Schaltflächen sichtbar sein sollen, ganz einfach.

Ein Audio-CD-Player

Um eine Audio-CD von einer Visual-Basic-Applikation aus abzuspielen, legen Sie das Multimedia-Steuerelement auf einem Formular an und setzen die **DeviceType**-Eigenschaft auf **CDAudio**, um eine Schaltflächenkombination bereitzustellen, die für das Abspielen von CDs geeignet ist. Damit haben Sie fast alles erledigt, was für einen einfachen CD-Player erforderlich ist. Wenn Sie jedoch möchten, daß die abgespielten Spuren angezeigt werden, sollten Sie ein entsprechendes Bezeichnungsfeld einfügen.

Das Multimedia-Steuerelement aktualisiert seine Spurinformationen mit jedem Auftreten des **StatusUpdate**-Ereignisses. Das Multimedia-Steuerelement aktualisiert sein **StatusUpdate**-Ereignis, wenn die Spur gewechselt wird, ebenso wenn der Benutzer das Abspielen der CD beginnt oder beendet. Sie aktualisieren die Spurbezeichnung also in einer Ereignisprozedur, die Sie für das **StatusUpdate**-Ereignis anlegen.

Die Befehlssprache des Multimedia-Steuerelements

Das Multimedia-Steuerelement unterstützt eine eigene kleine Befehlssprache (obwohl diese nicht annähernd so robust ist wie Visual Basic). Es nimmt in seiner **Command**-Eigenschaft Befehle entgegen, die steuern, welches Gerät das aktuell aktive Gerät ist. Tabelle 14.6 beschreibt die Befehle und ihre Bedeutung.

Tabelle 14.6: Die Befehlssprache für die Command-Eigenschaft des Multimedia-Steuerelements

Befehl	Beschreibung
Back	Springt eine Spur zurück.
Close	Schließt das Gerät.
Eject	Wirft die CD aus dem CD-ROM-Laufwerk aus.
Next	Springt an den Anfang der nächsten Spur (oder an den Anfang der letzten Spur, wenn dies die aktuelle Spur ist).
Open	Öffnet das Gerät.
Pause	Unterbricht das Gerät.
Play	Abspielen.
Prev	Geht zum Anfang der aktuellen Spur zurück. Wenn dieser Befehl innerhalb von drei Sekunden seit dem letzten Prev -Befehl benutzt wird, springt er zum Anfang der vorherigen Spur zurück (oder zum Anfang der ersten Spur, wenn dies die aktuelle Spur ist).
Record	Initialisiert die Aufzeichnung.
Save	Speichert die geöffnete Gerätedatei.
Seek	Springt eine Spur rückwärts oder vorwärts (die Programmierer verwenden normalerweise Next oder Prev , um die mehrdeutige Richtungsangabe dieses Befehls zu umgehen).
Stop	Hält das Gerät an.
Step	Durchläuft die Spuren des Geräts in Vorwärtsrichtung.

Während der Ausführung Ihrer Applikation reagiert das Multimedia-Steuerelement entsprechend, sobald sich der Wert der Command-Eigenschaft ändert.



Ihre Benutzer brauchen nicht unbedingt die Steuerung über das Multimedia- Gerät zu erhalten. Sie können alle Schaltflächen des Steuerelements verbergen und das Gerät ausschließlich über die Command-Eigenschaft in Ihrem Code steuern.

Wir bauen einen CD-Player zusammen

Nachdem Sie das Multimedia-Steuerelement kennengelernt haben, können Sie sofort eine Applikation entwickeln. Starten Sie ein neues Projekt und weisen Sie ihm die in Tabelle 14.7 beschriebenen Eigenschaften zu.

Tabelle 14.7: Die Eigenschaften und Werte für das Formular mit dem Multimedia-Steuerelement

Steuerelement	Beschreibung
Formular Name	frmCD
Formular Caption	CD Player
Formular Height	3600
Formular Width	4800
Bezeichnungsfeld #1 Name	lblCD
Bezeichnungsfeld #1 Alignment	2 - Zentriert
Bezeichnungsfeld #1 BorderStyle	1 - Fest einfach
Bezeichnungsfeld #1 Caption	CD Player
Bezeichnungsfeld #1 Schriftstil	Fett Unterstrichen
Bezeichnungsfeld #1 Schriftgröße	18
Bezeichnungsfeld #1 Height	495
Bezeichnungsfeld #1 Left	1320
Bezeichnungsfeld #1 Top	480
Bezeichnungsfeld #1 Width	1935
Bezeichnungsfeld #2 Name	lblTrack
Bezeichnungsfeld #2 Alignment	1 - Rechts
Bezeichnungsfeld #2 Caption	Spur :
Bezeichnungsfeld #2 Schriftstil	Fett
Bezeichnungsfeld #2 Schriftgröße	12
Bezeichnungsfeld #2 Height	255

Bezeichnungsfeld #2 Left	1200
Bezeichnungsfeld #2 Top	2280
Bezeichnungsfeld #2 Width	1215
Bezeichnungsfeld #3 Name	lblTrackNum
Bezeichnungsfeld #3 Caption	(blank)
Bezeichnungsfeld #3 Schriftstil	Fett
Bezeichnungsfeld #3 Schriftgröße	12
Bezeichnungsfeld #3 Height	375
Bezeichnungsfeld #3 Left	2520
Bezeichnungsfeld #3 Top	2280
Bezeichnungsfeld #3 Width	615
Multimedia-Steuerelement Name	mmcCD
Multimedia-Steuerelement DeviceType	CDAudio

Nachdem Sie diese Steuerelemente angelegt haben, geben Sie den Code aus Listing 14.5 ein.

Listing 14.3: Dieser Code verwaltet den CD-Player

```

1: Private Sub Form_Load()
2: ' CD öffnen
3: mmcCD.Command = "Open"
4: End Sub
5:
6: Private Sub Form_Unload(Cancel As Integer)
7: ' Multimedia-Steuerelement entfernen
8: mmcCD.Command = "Close"
9: End Sub
10:
11: Private Sub mmcCD_StatusUpdate()
12: ' Spurnummer im Bezeichnungsfeld aktualisieren
13: lblTrackNum.Caption = mmcCD.Track
14: End Sub

```

Die Applikation öffnet das Multimedia-Steuerelement zum Abspielen der CD, wenn das Formular geladen wird. Zeile 8 entfernt den CD-Player aus dem Speicher, unmittelbar bevor die Applikation beendet wird (wie Dateien sollten Sie auch geöffnete Multimedia-Geräte schließen). Zeile 13 aktualisiert die Spuranzeige, wenn sich der CD- Status ändert. Der Wert in der `UpdateInterval`-Eigenschaft des Multimedia-Steuerelements spezifiziert das Intervall zwischen den Statusaktualisierungen (der Standardwert ist 1000).

Abbildung 14.10 zeigt den CD-Player bei der Ausführung. Die Applikation ist einfach, aber sie funktioniert! Sie können die Applikation so ergänzen, daß sie durch die Menü-Option **Datei, Beenden** verlassen werden kann. Außerdem könnten Sie eine Fehlerverarbeitung einfügen, wie im nächsten Abschnitt beschrieben wird.



Offensichtlich werden einige Schaltflächen in dieser Applikation nicht benötigt und sind deshalb deaktiviert. Sie können diese Schaltflächen auch völlig verbergen, um das Multimedia-Steuerelement übersichtlicher zu machen.



Sie haben es hier nicht nur geschafft, das Multimedia-Steuerelement als CD-Player einzusetzen, sondern können es damit auch für alle anderen Multimedia-Geräte anwenden. In den folgenden Abschnitten werden Sie sehen, daß das Multimedia-Steuerelement fast überall gleich bedient wird, unabhängig davon, welches Multimedia-Gerät es darstellen soll.

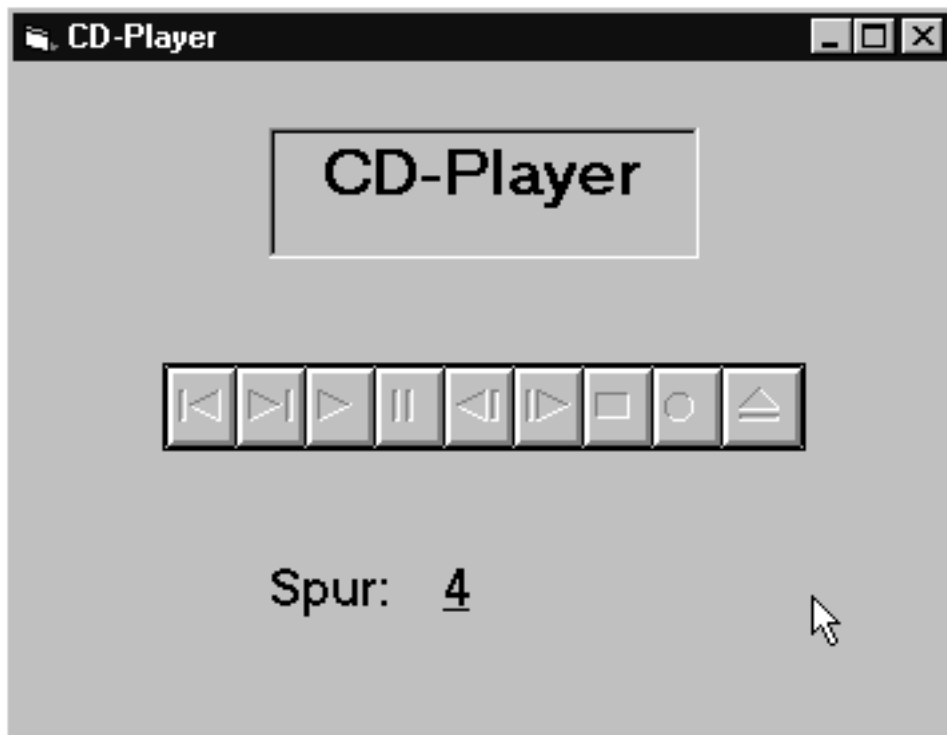


Abbildung 14.10:
Dieser CD-Player ist ganz einfach zu entwickeln.

Meldungen und Fehlerverarbeitung

Während das Multimedia-Steuerelement seine Aufgabe ausführt, werden verschiedene Bedingungen für Meldungen und die Fehlerverarbeitung aktualisiert, so daß Sie ein fehlerfreies Abspielen sicherstellen können.

Die `Notify`-Eigenschaft erzeugt ein `Done`-Ereignis, wenn der Befehl `Next` für das Multimedia-Steuerelement erfolgreich ausgeführt wurde. Wenn Sie also nur dann einen Befehl für das Multimedia-Steuerelement absetzen möchten, nachdem der vorherige Befehl verarbeitet wurde, setzen Sie die `Notify`-Eigenschaft auf `True`. Die `Wait`-Eigenschaft stellt, wenn sie auf `True` gesetzt ist, sicher, daß die Steuerung erst dann an Ihre Applikation zurückgegeben wird, wenn der vorhergehende Befehl für das Multimedia-Steuerelement erfolgreich ausgeführt wurde.

Falls ein Fehler auftritt, nachdem Sie einen Befehl für das Multimedia-Steuerelement abgesetzt haben, werten Sie diesen wie folgt aus:

```
If (frmCD.mmcCD.Error) then
```

```
intMsg = MsgBox("CD funktioniert nicht", vbCritical)
Else
' Code, der das Abspielen auf dem CD Player fortsetzt
```

Es ist nicht sinnvoll, die Anweisung `On Error Goto` oben in der Prozedur für den CD- Player zu plazieren, weil Sie damit nicht genau erkennen, welcher Befehl die Fehlerbedingung ausgelöst hat. Besser ist es, hinter jedem Befehl eine Routine zur Fehlerprüfung auszuführen, weil man damit die Bedeutung eines Fehlers besser interpretieren kann.

Tabelle 14.8 listet mehrere **Mode**-Werte auf. Die **Mode**-Eigenschaft ermöglicht Ihnen, während der Programmausführung verschiedene Statuswerte des Multimedia-Steuerelements zu überprüfen, um festzustellen, was gerade passiert.

Tabelle 14.8: Diese Mode-Werte zeigen den aktuellen Modus des Multimedia-Steuerelements an

Konstantenname für den Modus	Beschreibung
<code>mcModeNotOpen</code>	Das Multimedia-Steuerelement-Gerät ist nicht geöffnet.
<code>mcModeStop</code>	Das Multimedia-Steuerelement-Gerät ist angehalten.
<code>mcModePlay</code>	Das Multimedia-Steuerelement-Gerät spielt ab.
<code>mcModeRecord</code>	Das Multimedia-Steuerelement-Gerät zeichnet auf
<code>mcuModeSeek</code>	Das Multimedia-Steuerelement-Gerät sucht nach Informationen.
<code>mcModePause</code>	Das Multimedia-Steuerelement-Gerät ist unterbrochen.
<code>mcModeReady</code>	Das Multimedia-Steuerelement-Gerät ist bereit.

Ein Wave-Player

Eine Wave-Datei ist eine auf dem Computer abgelegte Audiodatei. Um diese Datei abzuspielen, brauchen Sie ein zusammengesetztes Multimedia-Gerät. Zusammengesetzte Multimedia-Geräte erhalten alle ihre Daten aus einer Datei, nicht aus einer Quelle von außen, wie beispielsweise von einer Audio-CD. Ihr Computer beinhaltet standardmäßig mehrere Wave-Dateien. Das Applet **Akustische Signale** in der Systemsteuerung verwendet diese Wave-Dateien, um den Systemereignissen Sound hinzuzufügen.

Dieser Abschnitt erklärt, wie man einen Multimedia-Player für Wave-Audiodateien entwickelt. Mit Hilfe der **Mode**-Werte aus Tabelle 14.8 zeigen Sie Statusinformationen über die abzuspielende Datei an. Sie verwenden dazu die Wave-Datei `TADA.WAV`, die unter Windows im Ordner `Windows\Media` abgelegt ist.

Speichern Sie das Projekt mit dem CD-Player, falls Sie es noch geöffnet haben, und verwenden Sie es als Grundlage für dieses neue Projekt. Um beide Projekte separat zu verwalten, speichern Sie das Projekt unter einem neuen Namen und ändern es dann wie folgt ab:

1. Ändern Sie den Namen des oberen Bezeichnungsfelds in `lblWave` und die `Caption`-Eigenschaft in `Wave-Player`. Setzen Sie die `Width`-Eigenschaft auf 2415.
2. Ändern Sie den Formularnamen in `frmWav` und setzen Sie die `Caption`-Eigenschaft auf

Wave-Player.

3. Ändern Sie den Namen des Multimedia-Steuerelements auf `mmcWav` und setzen Sie die `DeviceType`-Eigenschaft auf `WaveAudio`. Diese Eigenschaft können Sie im Eigenschaftenfenster ändern, aber auch im Dialogfeld **Eigenschaftenseiten**, das erscheint, wenn Sie im Eigenschaftenfenster unter (Benutzerdefiniert) auf die Schaltfläche mit den drei Punkten klicken. Setzen Sie `Filename` auf `TADA.WAV`. Diese Datei befindet sich im Verzeichnis `\Windows\Media`.
4. Wave-Dateien haben keine Spuren wie CDs, Sie können aber die Bezeichnungsfelder unterhalb dem Multimedia-Steuerelement weiterhin nutzen. Ändern Sie den Namen von `lblTrack` auf `lblStatus` und `Caption` auf `Status`.
5. Ändern Sie den Namen des rechten Bezeichnungsfelds von `lblTrackNum` in `lblStatusValue` und löschen Sie die `Caption`-Eigenschaft. Setzen Sie die `Width`-Eigenschaft auf 2565.
6. Fügen Sie zwei weitere Bezeichnungsfelder ein, unmittelbar unterhalb der beiden ersten, und verwenden Sie dafür dieselbe Breite und dieselbe Schrift. Geben Sie dem ersten Bezeichnungsfeld den Namen `lblFile` (es sei denn, Sie haben ein Steuerelementfeld angelegt), und setzen Sie die `Caption`-Eigenschaft auf `Dateiname:` . Sie müssen das Bezeichnungsfeld etwas vergrößern, um den gesamten Titel anzeigen zu können.
7. Ändern Sie den Namen des rechten Bezeichnungsfelds auf `lblFileValue` (es sei denn, Sie haben ein Steuerelementfeld angelegt). Lassen Sie das Bezeichnungsfeld leer. Nachdem Sie die Bezeichnungsfelder unterhalb der Schaltflächen ausgerichtet haben, sollte die Ausführung Ihrer Applikation das in Abbildung 14.11 gezeigte Formular erzeugen.



Abbildung 14.11:
Der Wave-Player ist fast fertig.

Ändern Sie die Ereignisprozedur `Form_Load()` wie folgt:

```
Private Sub Form_Load ()
```

```
' Das Multimedia-Steuerelement anweisen, den WAVE-Player zu öffnen
mmcWAV.Command = "Open"
End Sub
```

Außerdem sollten Sie auch die `Form_Unload()`-Prozedur ändern:

```
Private Sub Form_Unload(Cancel As Integer)
' WAVE-Player und Formular aufräumen
mmcWAV.Command = "Close"
Unload Me ' Formular aus dem Speicher entfernen
End Sub
```

Führen Sie die Applikation aus. Wenn Sie auf die Schaltfläche **Abspielen** klicken, hören Sie einen Sound. Klicken Sie auf die Schaltfläche **Zurück** und spielen Sie den Sound erneut ab.

Der Wave-Player ist noch nicht ganz fertig. Die Bezeichnungsfelder unterhalb der Schaltflächen zeigen den Status und den Dateinamen noch nicht an. Sie ordnen den Bezeichnungsfeldern ihre Werte am besten innerhalb der Ereignisprozedur `StatusUpdate()` zu (siehe Listing 14.4). Anhand der `Mode`-Werte ermitteln Sie den richtigen Abspielmodus.

Listing 14.4: Statusinformationen in ein Bezeichnungsfeld einfügen

```
1: Private Sub mciWAV_StatusUpdate()
2: ' Status anzeigen
3: If mmcWAV.Mode = mciModeNotOpen Then
4: lblStatusValue(0).Caption = "Nicht bereit"
5: ElseIf mmcWAV.Mode = mciModeStop Then
6: lblStatusValue(0).Caption = "Abgebrochen"
7: ElseIf mmcWAV.Mode = mciModePlay Then
8: lblStatusValue(0).Caption = "Abspielen"
9: ElseIf mmcWAV.Mode = mciModeRecord Then
10: lblStatusValue(0).Caption = "Aufzeichnen"
11: ElseIf mmcWAV.Mode = mciModePause Then
12: lblStatusValue(0).Caption = "Unterbrochen"
13: ElseIf mmcWAV.Mode = mciModeReady Then
14: lblStatusValue(0).Caption = "Bereit"
15: End If
16: ' Dateinamen anzeigen
17: lblStatusValue(1).Caption = mmcWAV.FileName
18: End Sub
```



Die Ereignisprozedur `StatusUpdate()` setzt voraus, daß Sie Steuerelementfelder für die Bezeichnungsfelder angelegt haben. Falls noch nicht geschehen, ändern Sie die Namen der Bezeichnungsfelder in die Namen, die Sie für Ihre Bezeichnungsfelder verwendet haben, sonst wird die Applikation nicht korrekt ausgeführt.

Führen Sie den Wave-Player aus, um die Applikation zu testen. Während Sie die Wave-Datei abspielen, achten Sie auf die folgenden Dinge:

- Der Wave-Player zeigt keine **Stop**-Schaltfläche an, außer beim schnellen Abspielen der Audio-Datei. Die Datei beendet das Abspielen, nachdem ihr Ende erreicht ist. Wäre die Wave-Datei länger, hätten Sie mehr Gelegenheit, den Abspielvorgang abubrechen.

- Die Datei wird nur einmal abgespielt. Der Player zeigt dann auf das Dateiende. Klicken Sie auf die Schaltfläche **Rückspulen**, um zum Anfang der Audiodatei zurückzukehren, und den Sound erneut abzuspielen.
- Die Schaltfläche **Aufzeichnen** ist aktiv. Sie können am Anfang oder am Ende der Datei aufzeichnen und sie dann zum Originalsound plus Ihrer Aufzeichnung zurückspulen.
- Speichern Sie Ihr aufgezeichnetes Audio nicht in der Datei. Statt dessen sollten Sie die WAV-Datei beibehalten, die in Visual Basic enthalten war. Außerdem bräuchten Sie zum Speichern der Datei eine Möglichkeit, im Dialogfeld **Speichern** einen neuen Dateinamen einzugeben. Diese Applikation ist nicht darauf ausgelegt, Änderungen zu speichern, um die Originaldateien unverändert beizubehalten.

Abspielen von Video-Clips

Damit beherrschen Sie das Multimedia-Steuerelement! Ob Sie es glauben oder nicht, das Abspielen eines Video-Clips unterscheidet sich nicht wesentlich von dem Abspielen einer Wave-Audiodatei, wie Sie in diesem Abschnitt noch sehen werden. Sie übergeben den Namen der abzuspielenden Videodatei (das Multimedia-Steuerelement stellt als Video-Player ein zusammengesetztes Gerät dar, deshalb muß ein Dateiname übergeben werden). Außerdem richten Sie das Multimedia-Steuerelement so ein, daß es Videos abspielen kann. Damit ist Ihre Applikation in der Lage, wirklich alles an Multimedia-Unterhaltung zu bieten.

Ihr Multimedia-Steuerelement braucht zum Abspielen eines Video-Clips etwas Unterstützung. Statt nur die Schaltflächen bereitzustellen, brauchen Sie einen Projektionsbildschirm, auf dem Sie das Video anzeigen. Das gebräuchlichste Steuerelement zur Anzeige von Videos ist das Bildfeld, deshalb fügen Sie jeder Applikation, in der eine Videodatei abgespielt wird, ein Bildfeld hinzu.

Bisher haben Sie für das Multimedia-Steuerelement in der `DeviceType`-Eigenschaft den Wert `CDAudio` verwendet, um CDs abzuspielen, und `WaveAudio`, um Wave-Audiodateien abzuspielen. Sie können diese Werte zwar im Eigenschaftendialog der Eigenschaft (`Benutzerdefiniert`) eingeben, aber es ist auch möglich, sie mit Hilfe von Zuweisungen zur Laufzeit zu setzen, wenn verschiedene Arten von Multimedia-Dateien abgespielt werden sollen.

Ihre Applikation enthält möglicherweise mehrere Bildfelder. Ihr Multimedia-Steuerelement muß wissen, an welches Bildfeld es das Video senden soll. Sie teilen ihm den Namen des Bildfelds in seiner `hWndDisplay`-Eigenschaft mit.

`hWnd` ist ein allgemeines Präfix für die Windows-Programmierung, das einen Handle, oder genauer gesagt, einen Gerätekontext darstellt. Die Ausgaben von Formularen stellen keine eigentlichen Bildschirmausgaben, sondern Fensterausgaben dar. (Sie können aber mit dem grundlegenden Befehlssatz Ausgaben an alle Windows-Geräte senden. Für Ihr Programm sind aber Windows-Geräte eher virtuell als real. Windows führt all die komplizierten Umwandlungen aus, die erforderlich sind, um die Ausgaben auf einen Drucker oder einen Farbbildschirm vorzunehmen.) Das bedeutet, daß Ihr Programm den Video-Clip nicht direkt an Ihren Bildschirm sendet, sondern an einen Gerätekontext. Dieser Gerätekontext ist fast immer Ihr Bildschirm, aber das Multimedia-Steuerelement muß den richtigen Gerätekontext kennen. Auf diese Weise erkennt es, in welchem Fenster die Datei abgespielt wird, und es kann die Fenstergrenzen entsprechend anpassen, so daß das Video innerhalb des Fensters bleibt.

Legen Sie eine neue Applikation an. Geben Sie dem Formular den Namen `frmVideo` und geben Sie

als **Caption**-Eigenschaft den Wert **Video-Player** ein. (Falls noch nicht passiert, fügen Sie Ihrer Werkzeugsammlung erneut das *Microsoft Multimedia Control 6.0* hinzu.) Geben Sie dem Multimedia-Steuerelement den Namen **mmcVideo**.

Fügen Sie dem Formular ein Bildfeld hinzu. Platzieren Sie das Bildfeld unterhalb der Schaltflächen des Multimedia-Steuerelements und verwenden Sie etwa die Größe wie in Abbildung 14.12 gezeigt. Geben Sie dem Bildfeld den Namen **picVideo**.

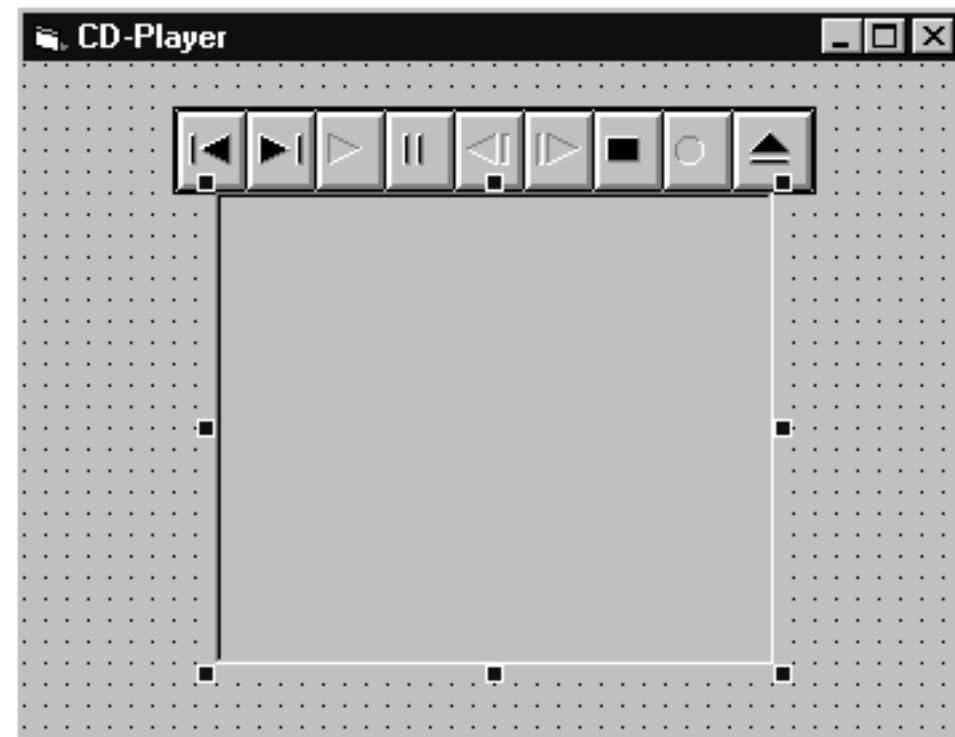


Abbildung 14.12:
Das Video wird im Bildfeld angezeigt.

Setzen Sie die **DeviceType**-Eigenschaft des Multimedia-Steuerelements auf **AVIVideo**. Geben Sie als **Filename**-Eigenschaft die AVI-Datei **Count24.AVI** an. Sie finden diese Datei im Visual-Basic-Verzeichnis **\Graphics\Videos**. Damit sind Sie fast fertig! Wie Sie sehen, ist die Einrichtung eines Video-Players kaum anders als die eines Audio-Players. Jetzt fügen Sie die Ereignisprozedur **Form_Load()** ein, die Sie in Listing 14.5 sehen.

Listing 14.5: Jetzt verbinden Sie das Video mit dem Bildfeld

```
1: Private Sub Form_Load()  
2: ' Video-Player öffnen  
3: mmcVideo.Command = "Open"  
4: ' Video-Player mit dem Bildfeld verbinden  
5: mmcVideo.hWndDisplay = picVideo.hWnd  
6: End Sub
```

Führen Sie das Programm aus, um die Zahlen im Video anzuzeigen (siehe Abbildung 14.13).



Abbildung 14.13:
Das Video wird einwandfrei abgespielt!

Bei Ihrer Arbeit mit dem Multimedia-Steuerelement werden Sie lernen, verschiedene Abkürzungen zu nutzen. Beispielsweise müssen Sie den Gerätetyp nicht angeben, wie etwa `AVIVideo`, wenn Sie zusammengesetzte Geräte programmieren, weil Visual Basic die Dateinamenerweiterung des zusammengesetzten Geräts auswertet, um damit festzustellen, welcher Gerätetyp benötigt wird.

Möglicherweise haben Sie erkannt, daß die Statusanzeigen nicht immer rechtzeitig aktualisiert werden. Mit anderen Worten, als Sie im vorigen Abschnitt die Videodateien abgespielt haben, hat das Statusfeld erst dann *Abspielen* angezeigt, als der Clip fast fertig abgespielt war. Das `StatusUpdate`-Ereignis tritt alle paar Millisekunden auf. Wenn Sie diese Bezeichnungsfelder öfter aktualisieren möchten, um genauere Informationen zu erhalten, setzen Sie die `UpdateInterval`-Eigenschaft auf einen kleineren Wert (der Standardwert ist 1000, so daß der Status jede Sekunde aktualisiert wird).



Machen Sie den Wert für die `UpdateInterval`-Eigenschaft nicht zu klein, sonst verbraucht Ihre Applikation zu viel Zeit, um das Bezeichnungsfeld zu aktualisieren und verlangsamt das gesamte System. Häufig führt eine solche Verlangsamung zu einem holprigen Abspielen der Multimedia-Dateien.

Zusammenfassung

Im heutigen Kapitel haben Sie erfahren, wie einfach es ist, Ihre Applikationen mit Grafik und Multimedia interessanter zu machen. Die Steuerelemente Linie und Figur ermöglichen Ihnen, Ihre Applikation mit Grafiken auszustatten, um wichtige Elemente auf dem Formular hervorzuheben. Außerdem können Sie Linien einfügen, die Steuerelemente voneinander abtrennen und wichtige Werte markieren.

Visual Basic unterstützt Zeichenmethoden, mit denen Sie beliebige Grafiken erstellen, indem Sie

Pixel aktivieren oder deaktivieren und Linien, Rechtecke und Ellipsen zeichnen. Diese Methoden sind relativ primitiv, bieten Ihnen aber volle Kontrolle über das Aussehen Ihrer Grafiken.

Die letzten Abschnitte dieses Kapitels haben die Verwendung des Multimedia-Steuerelements beschrieben. Das Multimedia-Steuerelement ist eines der komplexesten Steuerelemente von Visual Basic, weil es so viele verschiedene Multimedia-Geräte unterstützt. Ihre Applikationen können auf diese Weise ganz einfach Audio-CDs und Sounddateien und sogar Video abspielen. Das Multimedia-Steuerelement stellt Schaltflächen bereit, über die der Benutzer es bedient. Sie können die Steuerung aber auch ausschließlich über Ihren Code realisieren.

Im nächsten Kapitel erfahren Sie, wie Sie in Ihren Projekten Formularschablonen anlegen, so daß Sie die gebräuchlichsten Formulare in Ihren Applikationen standardisieren können.

Fragen und Antworten

Frage:
Kann ich das Multimedia-Steuerelement nutzen, um einzelne Grafiken anzuzeigen?

Antwort:
Das Multimedia-Steuerelement ist nur für Multimedia-Geräte vorgesehen. Bilder, die in einer Grafikdatei abgelegt sind, qualifizieren sich nicht für Multimedia. Nichtsdestotrotz können Sie solche Bilder auf einem Formular anzeigen. Dazu verwenden Sie Bildfelder oder Anzeigefelder.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Mit welchem Steuerelement zeichnen Sie Kreise auf einem Formular?
2. Welche Methode zeichnet Quadrate auf das Formular?
3. Welche der folgenden Methoden wird zum Zeichnen von Rechtecken verwendet?
 - PSet
 - Line
 - Box
 - Draw
4. Richtig/Falsch. Die Steuerelemente Linie und Figur erzeugen Grafiken, die Eigenschaften, aber keine Ereignisse unterstützen.
5. Welche Option der Line-Methode füllt die Rechtecke vollständig aus?
6. Warum aktiviert Visual Basic nicht jede Schaltfläche, wenn Sie das Multimedia- Steuerelement zum ersten Mal auf einem Formular plazieren?

7. Was macht die Mode-Eigenschaft?
8. Wie veranlassen Sie, daß der Status eines Multimedia-Geräts häufiger aktualisiert wird?
9. Warum brauchen Sie für die Anzeige von Videoclips ein Bildfeld?
10. Woran erkennt das Multimedia-Steuerelement ein Bildfeld, an das es das Video sendet?

Übungen

1. Fügen Sie der Applikation aus Abbildung 14.5 eine Füllfarbe und eine Farbe für die Umrißlinie hinzu. Sie wissen jetzt nicht nur, wie man den Umriß und das Füllmuster eines Steuerelements ändert, sondern auch, wie man die Füllfarbe und die Farbe der Umrißlinie festlegt. Abbildung 14.14 zeigt, wie Ihr Formular aussehen sollte, nachdem Sie die beiden Listenfelder eingefügt haben.

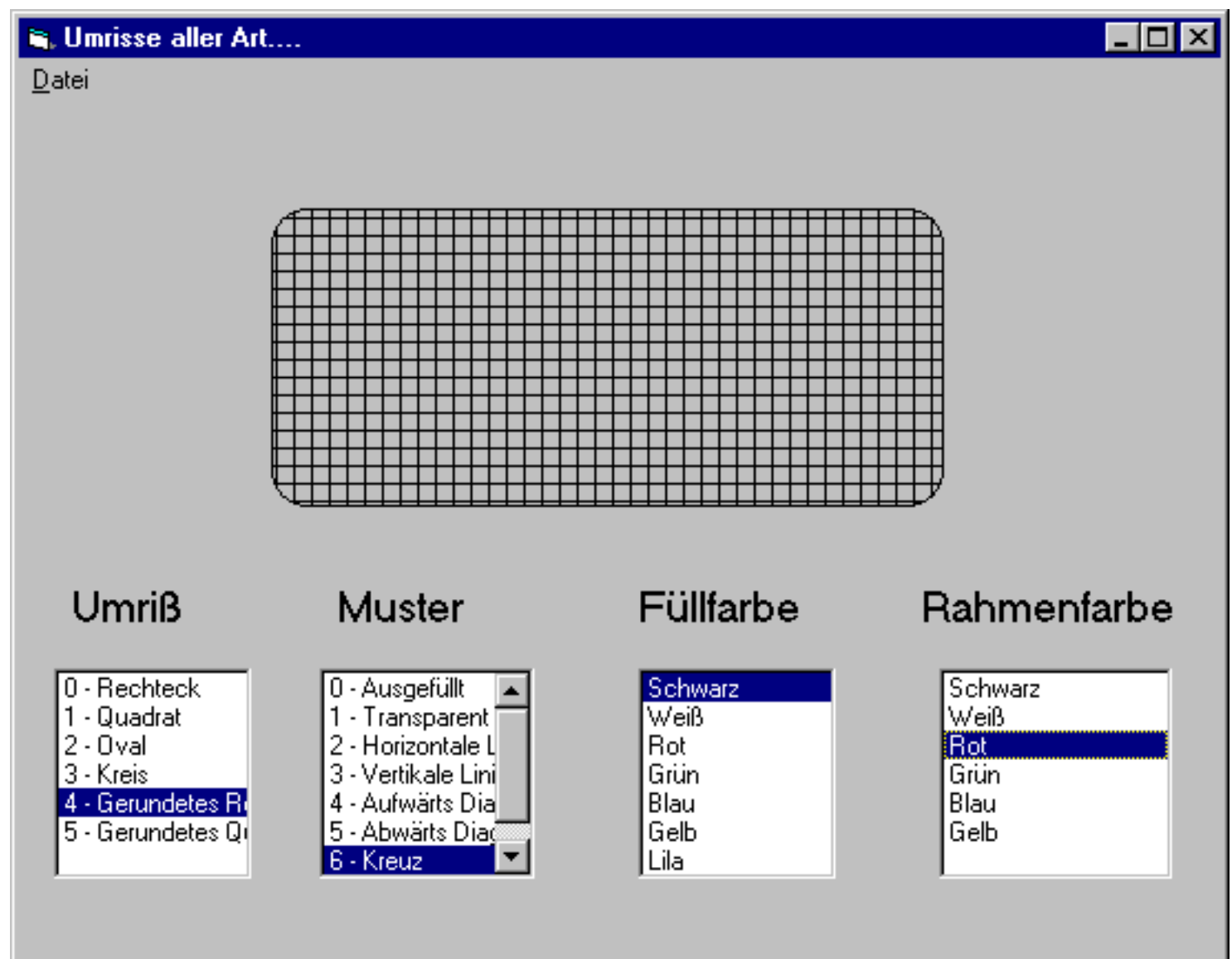


Abbildung 14.14:
Jetzt haben Sie noch mehr Kontrolle über das Aussehen einer Figur.



Setzen Sie die Farbe auf einen der Konstantennamen, beispielsweise `vbBlue`, wenn der Benutzer diese Farbe aus einem der Listenfelder auswählt. Sie müssen in den zusätzlichen Listenfeldern nicht alle Konstantenfarben angeben.

2. Üben Sie das Zeichnen von Linien und Kreisen mit den Zeichenmethoden in einem Bildfeld. Wie Sie sehen, können Sie auf das Bildfeld dieselben Zeichenmethoden anwenden wie auf das Formular.
 3. Ändern Sie den Wave-Player, den Sie in diesem Kapitel angelegt haben, so daß er das Dialogfeld **Datei öffnen** anzeigt, wo der Benutzer eine Wave-Datei auswählen kann, die abgespielt werden soll.
-

[⌂](#) [Kapitel](#) [Inhalt](#) [Index](#) [⬆](#) [Top](#) [Kapitel](#) [⌂](#)

Woche 2

Bonusprojekt 7: Bildlaufleisten

Dieses Bonusprojekt entwickelt nicht nur eine Applikation, sondern stellt Ihnen auch zwei neue Steuerelemente vor: die vertikale und die horizontale Bildlaufleiste. Horizontale und vertikale Bildlaufleisten sind ebenso einfach zu handhaben wie andere Steuerelemente, weil ihr Verhalten hauptsächlich durch die Eigenschaften bestimmt wird, die zur Laufzeit gesetzt werden. Mit Hilfe einfacher Ereignisprozeduren reagieren Sie auf die Aktionen des Benutzers.

Grundlagen

Häufig werden in Ihren Applikation automatisch Bildlaufleisten angezeigt, weil einige Steuerelemente, wie beispielsweise Listfelder, sie benötigen. Ein mehrzeiliges Textfeld, das sehr viele Daten enthält, zeigt Bildlaufleisten an, wenn Sie das in der `ScrollBars`-Eigenschaft erlauben. Das ist sinnvoll, wenn das Textfeld nicht die ganze Zeile oder alle Einträge anzeigen kann. Abbildung BP7.1 zeigt ein solches Listfeld. Die Abbildung demonstriert die beiden Bildlaufleisten und beschreibt ihre Komponenten.

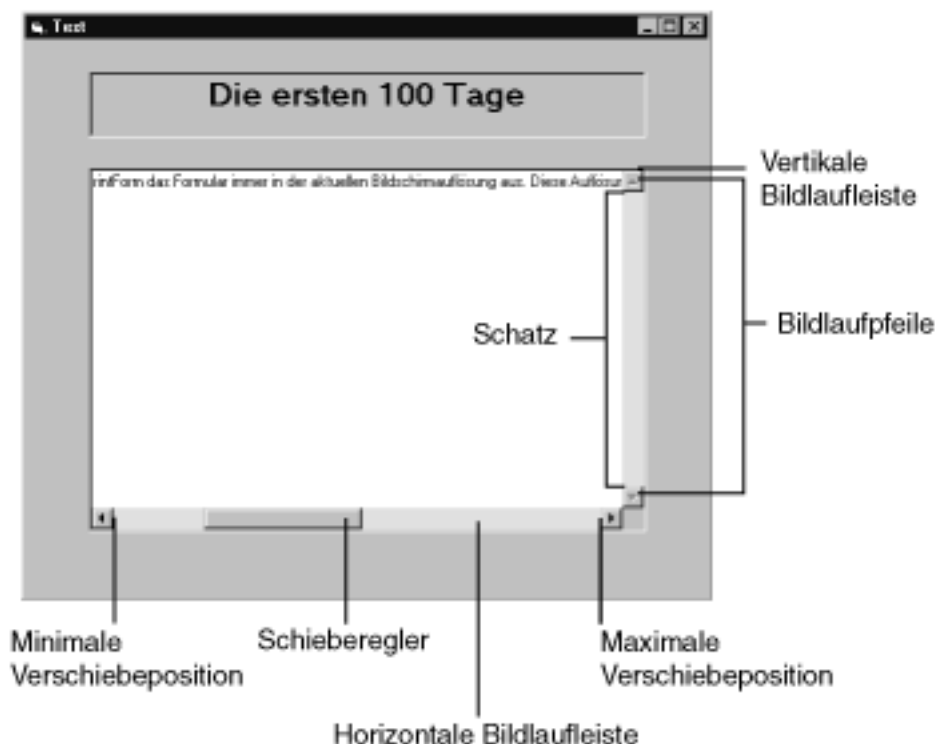


Abbildung BP7.1:
Horizontale und vertikale Bildlaufleisten können automatisch angezeigt werden.

Mehrzeilige Textfelder und Listfelder stellen nicht den einzigen Verwendungszweck für

Bildlaufleisten dar. Sie können Ihrer Applikation Bildlaufleisten hinzufügen, um dem Benutzer eine flexiblere Steuerung über bestimmte Einstellungen zu erlauben, beispielsweise zur Auswahl der Farbtiefe, einer Bildgröße, eines Textbereichs oder numerischer Wertebereiche.

Die Werkzeugsammlung stellt eine vertikale und eine horizontale Bildlaufleiste zur Verfügung. Nachdem Sie diese Steuerelemente auf einem Formular platziert haben, müssen Sie Eigenschaften setzen, die die Bereiche für die Bildlaufleisten definieren.

Die Arbeit mit Bildlaufleisten

Wenn Sie verstanden haben, was eine Bildlaufleiste macht, verstehen Sie auch ganz schnell, wie man sie programmiert. Wichtig dabei ist, daß Sie die einzelnen Komponenten der Bildlaufleisten kennen. Hier die Aktionen, die die Benutzer mit Bildlaufleisten ausführen können:

- Durch das Verschieben des horizontalen Schiebereglers an die ganz linke Position (bzw. des vertikalen Schiebereglers an die höchstmögliche Position) wird die Bildlaufleiste auf die minimale Einstellung gesetzt.
- Wird der horizontale Schieberegler nach rechts oder links (bzw. für vertikale Bildlaufleisten nach oben oder unten) geschoben, wird der entsprechende Datenbereich angezeigt.
- Wird der horizontale Schieberegler nach rechts (bzw. der vertikale Schieberegler nach ganz unten) verschoben, hat die Bildlaufleiste die maximale Einstellung.
- Durch Klicken auf einen der Schiebepfeile wird die Anzeige einen kleinen Schritt weitergeblättert.
- Durch Klicken neben dem Schieberegler im Schaft der Bildlaufleiste wird der Anzeigebereich um einen größeren Schritt weitergeblättert.

Eigenschaften der Bildlaufleiste

Tabelle BP7.1 listet die wichtigsten Eigenschaften von Bildlaufleisten auf, die Sie zur Entwurfszeit setzen können (sie stehen auch zur Laufzeit zur Verfügung, aber in der Regel werden sie alle zur Entwurfszeit gesetzt). Die Eigenschaften entsprechen der im vorigen Abschnitt beschriebenen Arbeitsweise von Bildlaufleisten.

Tabelle BP7.1: Die Eigenschaften helfen Ihnen, den Bereich und die Schrittweite von Bildlaufleisten festzulegen

Eigenschaft	Beschreibung
LargeChange	Gibt an, wie weit eine Bildlaufleiste weitergeschaltet wird, wenn der Benutzer in den Schaft klickt (egal, auf welcher Seite des Schiebereglers).
Max	Maximale Anzahl der Einheiten, die der Bildlaufleistenwert für die höchste Stellung repräsentiert. Der Standardwert ist 32767.
Min	Minimale Anzahl der Einheiten, die der Bildlaufleistenwert für die niedrigste Stellung repräsentiert. Der Standardwert ist 1.
SmallChange	Gibt an, wie weit eine Bildlaufleiste weitergeschaltet wird, wenn der Benutzer auf einen der Bildlaufpfeile klickt.
Value	Spezifiziert die durch die Bildlaufleiste dargestellte Maßeinheit.

Der Standardbereich aller Bildlaufleisten erstreckt sich von 1 bis 32767. Sie können diese Werte den Anforderungen Ihrer Applikation gemäß ändern. Angenommen, Sie schreiben ein Gehaltsprogramm, das Gehälter zwischen 500\$ und 15000\$ verwaltet. Ihr Benutzer könnte den Gehaltswert in ein Textfeld eingeben, aber auch mit Hilfe einer Bildlaufleiste einen Wert auswählen. Ihre Bildlaufleiste würde dann die folgenden Eigenschaftswerte annehmen:

Min: 500

Max: 15000



Die Breite und die Höhe einer Bildlaufleiste können angepaßt werden. Die physische Größe Ihrer Bildlaufleiste hat nichts mit dem Wertebereich zu tun, den sie darstellt.

Entwicklung einer Applikation

Die Applikation in diesem Bonusprojekt demonstriert die Leistungsfähigkeit von Bildlaufleisten. Abbildung BP7.2 zeigt den Bildschirm für die Applikation an. Wenn der Benutzer auf die Bildlaufleisten klickt, wird das Bild von dem Geldsack vergrößert oder verkleinert. Der Code, der dafür erforderlich ist, ist ganz einfach, weil die Eigenschaften der Bildlaufleisten die meiste Arbeit erledigen.



Achten Sie auf die Richtung der Bildlaufleisten. Um das Bild in der Höhe zu vergrößern, vergrößern Sie den Wert der vertikalen Bildlaufleiste, d.h. der Schieberegler der vertikalen Bildlaufleiste wird nach *unten* geschoben. Zunächst werden Sie das Gefühl haben, dadurch würde das Bild verkleinert. Sie könnten die Applikation auch so abändern, daß die Standardrichtung umgekehrt wird, aber in dieser Applikation ist das nicht der Fall.



Abbildung BP7.2:

Die Bildlaufleisten kontrollieren die Größe des angezeigten Bildes.

Die Eigenschaften helfen Ihnen, den Bereich und die Schrittweite von Bildlaufleisten festzulegen

Was Sie tun sollten
Legen Sie die Installations-CD für Visual Basic in Ihr Laufwerk ein, falls Sie den Graphics-Ordner noch nicht installiert haben. In dieser Applikation brauchen Sie Zugriff auf eine Grafikdatei. Sie können das Bild aber auch aus dem Ordner auf Ihre Festplatte kopieren.

Tabelle BP7.2 zeigt die Steuerelemente und Eigenschaften für diese Applikation.

Tabelle BP7.2: Die Steuerelemente für das Formular

Steuerelementeigenschaft	Eigenschaftswert
Formular Name	frmScoll
Formular Caption	Geld kommt und geht...
Formular Height	4650
Formular Width	5295
Bildfeld Name	picScroll
Bildfeld Height	1600
Bildfeld Left	1560
Bildfeld Picture	Common\Graphics\Metafile\Business\Moneybag
Bildfeld Top	1200
Bildfeld Width	1575
Vertikale Bildlaufleiste Name	vscScroll
Vertikale Bildlaufleiste Height	3975
Vertikale Bildlaufleiste LargeChange	100
Vertikale Bildlaufleiste Left	4920
Vertikale Bildlaufleiste Max	3750
Vertikale Bildlaufleiste Min	1600
Vertikale Bildlaufleiste SmallChange	50
Vertikale Bildlaufleiste Top	0
Vertikale Bildlaufleiste Width	255
Horizontale Bildlaufleiste Name	hscScroll
Horizontale Bildlaufleiste Height	255

Horizontale Bildlaufleiste LargeChange	100
Horizontale Bildlaufleiste Left	0
Horizontale Bildlaufleiste Max	3750
Horizontale Bildlaufleiste Min	1600
Horizontale Bildlaufleiste SmallChange	50
Horizontale Bildlaufleiste Top	3960
Horizontale Bildlaufleiste Width	4935

Immer wenn der Benutzer auf einen Bildlaufpfeil klickt, ändert sich die Bildgröße um 50 Pixel, weil der SmallChange-Wert für beide Bildlaufleisten geändert wird. Immer wenn der Benutzer links oder rechts vom Schieberegler klickt, ändert sich die Bildgröße um 100 Pixel, weil der LargeChange-Wert für Bildlaufleisten geändert wird.

Der Code

Listing BP7.1 zeigt den Ereigniscode, den Sie für dieses Projekt brauchen.

	Sie müssen nicht nur die Bildgröße anpassen, wenn der Benutzer auf die Bildlaufleiste klickt, sondern auch die Bildposition, so daß sich das Bild immer in der Formularmitte befindet.

Listing BP7.1: Die Größe des Bilds wird mit Hilfe von Bildlaufleisten angepaßt.

```

1: Private Sub hscScroll_Change()
2: ' Horizontale Größe und Position des Bildes ändern
3: picScroll.Width = hscScroll.Value
4: picScroll.Left = (frmScroll.Width / 2) - (picScroll.Width / 2) -
300
5: End Sub
6:
7: Private Sub vscScroll_Change()
8: ' Vertikale Größe und Position des Bildes ändern
9: picScroll.Height = vscScroll.Value
10: picScroll.Top = (frmScroll.Height / 2) - (picScroll.Height / 2)
-
-300
11: End Sub
12:
13: Private Sub vscScroll_Scroll()
14: ' Ändern, während der Benutzer den Schieberegler stellt
15: Call vscScroll_Change
16: End Sub

```

```

17:
18: Private Sub hscScroll_Scroll()
19: ' Ändern, während der Benutzer den Schieberegler stellt
20: Call hscScroll_Change
21: End Sub

```

Analyse

Der Code ist redundant, weil die beiden Bildlaufleisten grundsätzlich dasselbe tun (sie ändern die Breite oder Höhe des Bildes). Die erste Prozedur ändert in Zeile 3 die Breite des Bildes, so daß sie dem Wert der horizontalen Bildlaufleiste entspricht. (Die `Min`- und `Max`-Eigenschaften beider Bildlaufleisten werden zur Entwurfszeit gesetzt, so daß sich das Bild immer in einem sinnvollen Größenbereich bewegt.) Zeile 4 bestimmt die Position des Bildes, nachdem es eine neue Größe angenommen hat. Die Position wird im Verhältnis zur Formularhöhe bestimmt. Dazu wird der Formelmittelpunkt ermittelt (`frmScroll.Width/2`). Davon wird die halbe Bildgröße subtrahiert (`picScroll.Width/2`), um den Bildmittelpunkt in der Formularmitte anzulegen. Die zusätzlichen 300 Pixel werden subtrahiert, um die Größe der Bildlaufleiste rechts im Formular zu berücksichtigen.

Die Zeilen 7 bis 11 erledigen genau dieselbe Aufgabe für die horizontale Bildlaufleiste, außer daß hier anstatt der Höhe die Breite angepaßt wird.

Die beiden letzten Prozeduren sind interessant, weil sie einfach nur die anderen Prozeduren aufrufen. Die erste Prozedur wird nur ausgeführt, wenn der Benutzer auf die Bildlaufpfeile oder in den Schaft der Bildlaufleiste klickt. Was aber passiert, wenn der Benutzer den Schieberegler verschiebt? Ohne die Ereignisprozeduren für die `Change`-Ereignisse würde sich das Bild erst dann ändern, wenn der Benutzer die Maus und damit den Schieberegler losläßt. Damit ist das `Click`-Ereignis abgeschlossen. Werden die `Click`-Ereignisprozeduren von der `Change`-Ereignisprozedur aufgerufen, *ändert sich das Bild, während der Benutzer den Schieberegler verschiebt*. Mit anderen Worten, die `Click`-Ereignisprozedur wird während des Verschiebens für jeden Wert aufgerufen, an den der Benutzer den Schieberegler zieht.



Es wäre sinnvoll, den Code für das `Click`-Ereignis in der `Change`-Prozedur abzulegen und die `Click`-Ereignisse nicht mehr zu berücksichtigen. Die Ereignisprozeduren würden dann ausgeführt, sobald der Benutzer die Bildlaufleiste irgendwie ändert, egal ob durch Ziehen oder Klicken, weil beide Aktionen ein `Change`-Ereignis erzeugen. Nichtsdestotrotz muß eine Ereignisprozedur eine andere aufrufen, das Beispiel ist also treffend.

Inzwischen können Sie Visual Basic schon ganz nach Ihren Bedürfnissen einsetzen. Jetzt sollen Sie Ihre bislang erlernten Fähigkeiten weiter verbessern.

Was Sie erwartet

Die dritte und letzte Woche dieses Buchs zeigt Ihnen einige fortgeschrittenere Aspekte der Programmierung unter Visual Basic. Aber nur weil sie »fortgeschritten« sind, bedeutet das nicht, daß sie schwierig sind. Fast alles, was Sie von jetzt an lernen werden, ist fortgeschritten. Sie haben jedoch eine so starke Grundlage, daß es Ihnen ganz einfach fallen wird, Neues kennenzulernen.

In dieser Woche lernen Sie Objekte und ActiveX-Steuerelemente kennen. Diese Elemente werden Ihnen helfen, besseren Code zu entwickeln und Ihre Programme schneller zu schreiben. Außerdem lernen Sie, Komponenten wiederzuverwenden, wobei Sie genauso vorgehen wie bei der Wiederverwendung von Prozeduren. Sie werden Ihre eigenen Werkzeuge für die Werkzeugsammlung von Visual Basic entwickeln - in Form eigener ActiveX-Steuerelemente.

Wie Sie sehen werden, läßt es Visual Basic an fast nichts fehlen, was die Bereitstellung der verschiedensten Steuerelemente betrifft. In dieser letzten Woche lernen Sie, wie man auf Dateien zugreift, Daten in gebräuchlichen Datenbankformaten verwaltet und mit Hilfe der Werkzeuge von Visual Basic auf das Internet zugreift. Es wird nur selten vorkommen, daß Ihre Applikation mehr Steuerelemente benötigt, als die, die Sie bereits kennen. Aber vielleicht sind Sie beruhigter, wenn Sie wissen, daß es noch mehr gibt, und daß Sie sogar selbst welche entwickeln können. Diese ActiveX-Steuerelemente können nicht nur in der Programmierungsumgebung von Visual Basic eingesetzt werden, sondern auch auf andere Weise, beispielsweise durch Einbettung in Webseiten für Internet-Browser, die ActiveX unterstützen.

Nachdem Sie eine Applikation entwickelt haben, wollen Sie sie mit den Test- und Debugging-Werkzeugen von Visual Basic überprüfen. Einige Fehler sind nicht leicht zu erkennen, aber Visual Basic bietet viele Möglichkeiten, schwer zu verfolgende Fehler zu finden. Nachdem Ihre Applikation fertig ist, kompilieren Sie sie und geben sie an die Benutzer weiter. Zum Abschluß dieser Woche lernen Sie, wie Sie Ihre Applikationen mit ihren eigenen Installationsroutinen packen, so daß die Benutzer sie installieren können.

Woche 3

Tag 15

Formularvorlagen

In diesem Kapitel lernen Sie die Vorteile von Formularvorlagen kennen, die dafür sorgen, daß Ihre Formulare ein einheitlicheres Aussehen aufweisen, wenn Sie sie für immer wieder ähnliche Aufgaben einsetzen. Visual Basic unterstützt verschiedene Formularvorlagen, die Sie ergänzen, anpassen und in eigenen Projekten verwenden können. Viele der Formulare, die heute überall verwendet werden - beispielsweise das Dialogfeld **Info**, das mit **Hilfe, Info** aufgerufen wird -, haben ein *De-facto*-Aussehen entwickelt. Vielleicht wollen Sie diesem Trend folgen. Das Dialogfeld **Info** ist wie viele andere Standardformulare in Visual Basic enthalten.

Nachdem Sie wissen, wie man die Formularvorlagen verwendet, können Sie auch eigene erzeugen. Angenommen, Ihre Firma möchte, daß die Dialogfelder aller Applikationen einem gewissen Standard folgen - beispielsweise durch Anzeige des Firmennamens, des Logos, der Zeit und des Datums oben in jedem Formular. Nachdem Sie eine Formularvorlage mit diesen Informationen angelegt haben, enthalten alle Formulare, die Sie damit in Ihren Applikationen erzeugen, dieselben Elemente, ohne daß Sie sie für jedes Projekt neu entwickeln müßten.

Heute lernen Sie die folgenden Dinge kennen:

- Die Aufgabe von Formularvorlagen
- Wozu Formularvorlagen?
- Implementierung von Formularvorlagen in Projekten
- Mit Hilfe des Anwendungs-Assistenten Formularvorlagen hinzufügen
- Die Vorlage »Tips und Tricks«
- Eigene Formularvorlagen in die von Visual Basic bereitgestellten Vorlagen einfügen

Formularvorlagen

Eine *Vorlage* ist ein Modell. Eine Formularvorlage ist ein Modellformular. Wenn Sie Ihre Arbeit mit einer Formularvorlage beginnen, im Gegensatz zu einem leeren Formularfenster, sparen Sie Zeit und Ihre Formulare erhalten ein einheitlicheres Aussehen.



Eine *Formularvorlage* ist ein Modell für Ihre Formulare.

Angenommen, Sie möchten Ihrer Applikation das »Tips und Tricks«-Fenster hinzufügen, damit Ihre Benutzer bei jedem Aufruf der Applikation einen anderen Tip erhalten. Sie kennen das bestimmt von Windows 95. Dort erscheint das Fenster beim Starten von Windows 95. Auch andere gebräuchliche Applikationen verwenden »Tips und Tricks«, die häufig ähnlich aussehen wie das in Windows 95 verwendete Fenster. Beispielsweise findet man fast in all diesen Fenstern ein Kontrollkästchen, mit dem Sie die Anzeige dieser Tips und Tricks für alle folgende Programmaufrufe deaktivieren können.

Wie Sie bereits erfahren haben, hat die Verwendung standardisierter Applikationen viele Vorteile: Ihre Benutzer erlernen den Umgang mit Ihren Applikationen schneller, sie gewöhnen sich schneller an Ihre Programme und sie werden wahrscheinlich alle Upgrades kaufen - außerdem erhalten Sie weniger Service-Nachfragen. Wenn das noch nicht Grund genug ist, Standardbildschirme einzusetzen, dann sollten Sie berücksichtigen, daß Sie auch Zeit sparen: Sie haben Ihre Applikationen schneller fertig und müssen weniger Fehler daraus entfernen.

Abbildung 15.1 zeigt die Formularvorlage »Tips und Tricks«, die von Visual Basic bereitgestellt wird. Wenn Sie Ihrer Applikation also einen Tip des Tages hinzufügen möchten, sollten Sie dann besser mit der in Abbildung 15.1 gezeigten Formularvorlage anfangen - oder mit einem leeren Fenster? Natürlich ist die Formularvorlage die Methode der Wahl. Die Vorlage sorgt für ein einheitliches Erscheinungsbild und Sie haben weniger Arbeit, das Formular anzulegen.

	Wie die Applikationsgerüste, die der Anwendungs-Assistent erzeugt, enthält auch eine Formularvorlage Platzhalter für gebräuchliche Elemente. Sie entfernen einfach alle Elemente, die Sie nicht brauchen, und passen das Formular so an, wie Sie es in Ihrer Applikation brauchen. Trotz dieses Aufwands haben Sie das Formular mit Hilfe der Formularvorlage viel schneller fertig, als wenn Sie mit einem leeren Formularfenster anfangen würden.

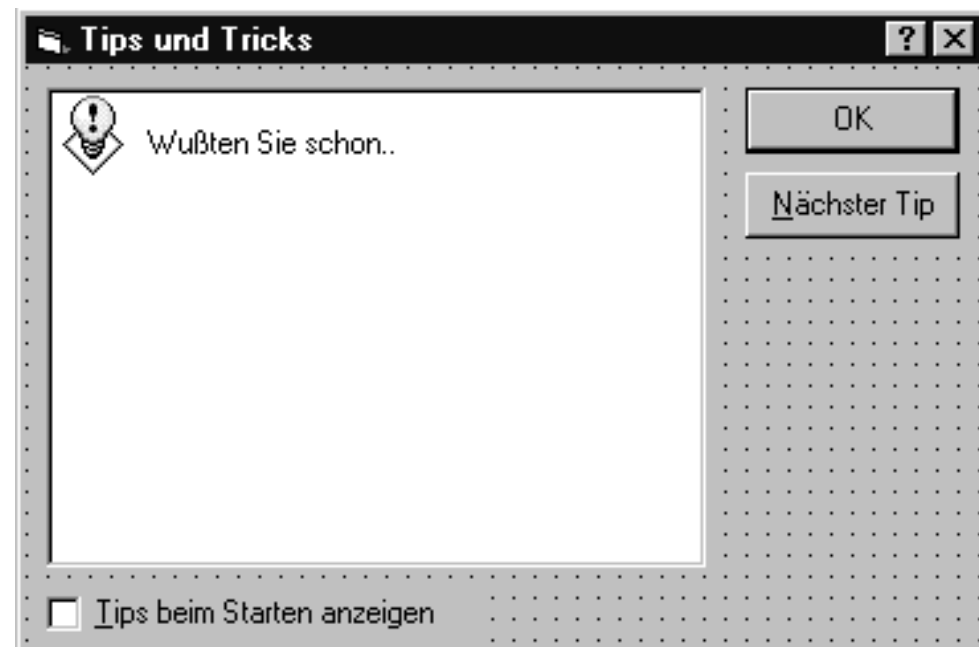


Abbildung 15.1:
Visual Basic stellt diese Formularvorlage für das Fenster »Tips und Tricks« bereit.

Formularvorlagen enthalten visuelle Elemente, beispielsweise Icons oder Steuerelemente, ebenso wie den Code für diese Elemente, der Sie dabei unterstützt, die Formularvorlage in Ihre Applikation zu integrieren.

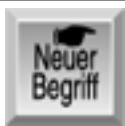


Eine Formularvorlage ist nichts anderes als ein Formular mit Elementen und Code, die bereits in ihrem Modul plaziert sind. Die Formularvorlagen sind keine besondere Formularvariante, sondern stellen vielmehr eine Menge vorbereiteter Formulare dar. Wenn Sie selbst Formulare entwickeln, die Sie immer wieder verwenden möchten, können Sie sie den Formularvorlagen Ihres Systems hinzufügen, wie in diesem Kapitel noch erklärt wird.

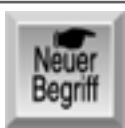
Die von Visual Basic bereitgestellten Formularvorlagen

Visual Basic stellt die folgenden Formularvorlagen bereit:

- **Info-Dialogfeld** - Wird normalerweise für die Menüoption **Hilfe, Info** bereitgestellt.
- **Web-Browser** - Für einfaches Web-Browsing.
- **VB-Datenformular** - Für die Verwaltung von Tabellen mit datenbankähnlichen Daten.
- **Dialog** - Zum Anlegen von Dialogfeldern.
- **Anmelde-Dialogfeld** - Für die Anforderung von Name und Kennwort.
- **ODBC-Anmeldung** - Für ODBC-bezogene Aktivitäten.
- **Optionen-Dialogfeld** - Für mehrseitige Dialogfelder und angepasste Einstellungen.
- **Begrüßungsbildschirm** - Erzeugt einen Startbildschirm, der angezeigt wird, während eine Applikation geladen wird. (Dieser Begrüßungsbildschirm wird in der Regel fünf Sekunden lang angezeigt.)
- **Tips und Tricks** - Zur Anzeige eines Tips beim Programmstart.



ODBC ist die Abkürzung für *Open Database Connectivity*. Es stellt einen Standardbefehlssatz für verschiedene Datenarten auf unterschiedlichen Computern bereit.



SQL ist die Abkürzung für *Structured Query Language*. Es definiert eine (theoretisch) universelle Programmiersprache für den Datenbankzugriff. *SQL*-Befehle können in Visual-Basic-Applikationen direkt ausgeführt werden.



Das restliche Kapitel beschreibt, wie viele dieser Formularvorlagen in Ihren eigenen Applikationen implementiert werden. Einige der Formularvorlagen sind zu spezifisch für dieses Kapitel. In Kapitel 19 wird die Web-Browser- Vorlage genauer beschrieben. Kapitel 18 zeigt, wie man Visual-Basic-Applikationen mit Datenbankinformationen verknüpft, beispielsweise mit dem VB-Datenformular oder der ODBC-Vorlage.

Die folgenden Abschnitte demonstrieren die allgemeineren Formularvorlagen. Sie lernen, wie man mit dem Anwendungs-Assistenten Formularvorlagen hinzufügt. Außerdem erfahren Sie, wie Sie Ihren Applikationen ein angepasstes **Info**-Feld hinzufügen. Nachdem Sie eine Formularvorlage eingefügt haben, wissen Sie auch, wie man die anderen einfügt.

Hier die allgemeinen Schritte für das Einfügen einer Formularvorlage in Ihre Applikation:

1. Fügen Sie dem Projektfenster Ihrer Applikation die Formularvorlage hinzu.
2. Passen Sie die Formularvorlage so an, daß sie die Anforderungen Ihrer Applikation erfüllt.
3. Schreiben Sie Code, der das Formular mit Ihrem Projekt verbindet.

Der Anwendungs-Assistent

Sie können Ihrer Applikation jederzeit eine der Formularvorlagen hinzufügen. Wenn Sie dazu den Anwendungs-Assistenten verwenden, müssen Sie ihm mitteilen, welche Formularvorlage Sie brauchen. Er übernimmt dann die Aufgabe, Ihrer Applikation diese Formularvorlage hinzuzufügen.

Abbildung 15.2 zeigt das Dialogfeld des Anwendungs-Assistenten, wo Sie ein Formular spezifizieren. Wie Sie sehen, bietet der Anwendungs-Assistent nur vier Formularvorlagen: Begrüßungsbildschirm, Anmelde-Dialogfeld, Optionen-Dialogfeld und Info- Dialogfeld.



Abbildung 15.2:
Der Anwendungs-Assistent fügt Ihrer Applikation ein oder mehr Formulare hinzu.

In diesem Dialogfeld werden nur vier Formularvorlagen aufgelistet, wenn Sie aber auf die Schaltfläche **Formularvorlagen** klicken, zeigt Visual Basic eine Liste mit den weiteren Formularvorlagen an.

 Hinweis	Diese Liste umfaßt nicht nur die im vorigen Abschnitt beschriebenen Formularvorlagen, sondern auch die Formularvorlagen, die Sie der Bibliothek selbst hinzugefügt haben.

Formularvorlagen für eine Applikation

Um einer Applikation eine Formularvorlage hinzuzufügen, fügen Sie diese wie ein ganz normales Formular ein:

- Wählen Sie die Menüoption **Projekt, Formular hinzufügen** und suchen Sie in dem Dialogfeld (siehe Abbildung 15.3) eine Formularvorlage aus.
- Klicken Sie mit der rechten Maustaste auf das Projektfenster, wählen Sie **Hinzufügen , Formular** und wählen Sie in dem Dialogfeld eine Formularvorlage aus.

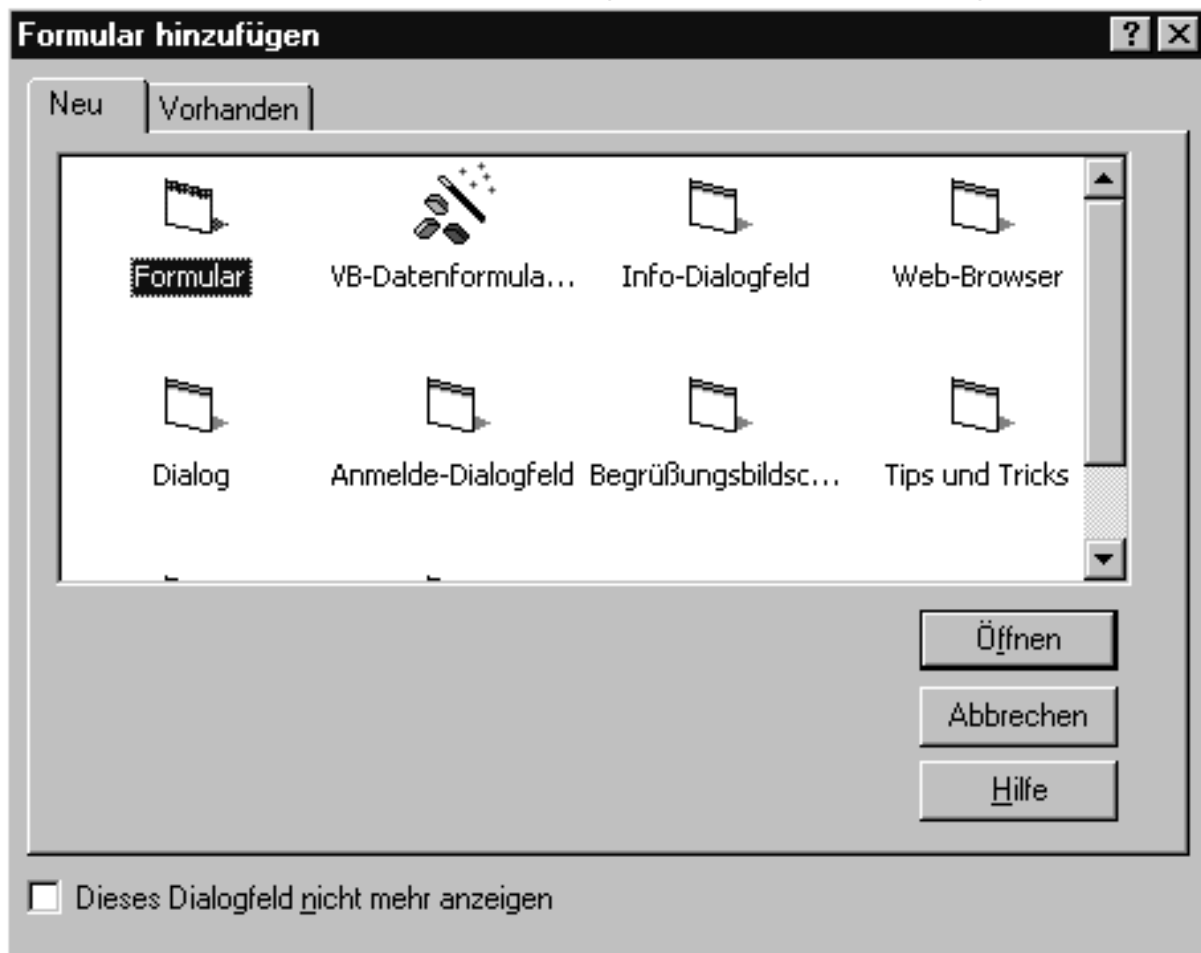


Abbildung 15.3:

Es ist ganz einfach, eine Formularvorlage hinzuzufügen.

Um ein neues Formular hinzuzufügen, wählen Sie das erste Icon des Dialogfelds (mit der Unterschrift **Formular**). Um eine Formularvorlage hinzuzufügen, wählen Sie das entsprechende Icon aus.

Abändern der Vorlagen

Wählen Sie **Extras, Optionen** und klicken Sie auf die Registerkarte **Umgebung**. Visual Basic zeigt das Dialogfeld **Umgebung** an (siehe Abbildung 15.4). Durch Aktivierung oder Deaktivierung des Kontrollkästchens **Formulare** legen Sie fest, ob Visual Basic die Liste mit den Formularvorlagen im Anwendungs-Assistenten und im Dialogfeld **Projekt, Formular hinzufügen** anbieten soll.

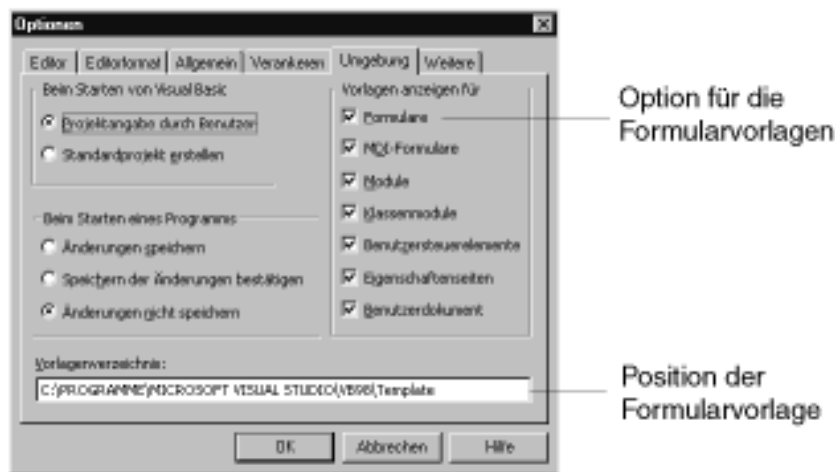


Abbildung 15.4:
Hier legen Sie fest, welche Vorlagen Visual Basic anbietet.

Wie Sie im Dialogfeld **Optionen** sehen, stellt Visual Basic Vorlagen für viel mehr Objekte als nur für Formulare bereit. Es gibt daneben auch Vorlagen für Module, Steuerelemente, Eigenschaftenseiten und anderes. Einige dieser anderen Vorlagen werden in relativ komplexen Applikationen verwendet, aber Sie sehen, daß es für fast jedes Objekt, das Sie anlegen können, ein Modell gibt.



Wenn Sie die Markierung für die Option **Formulare** entfernen, um die Formularvorlagen nicht anzuzeigen, wenn einer Applikation ein neues Projekt hinzugefügt wird, legt Formular automatisch ein neues, leeres Visual Basic in Ihrem Projekt an, ohne daß Sie die Möglichkeit haben, zu bestimmen, um was für ein Formular es sich dabei handeln soll.

Das Dialogfeld **Optionen** zeigt den Ordner an, in dem Visual Basic nach Formularvorlagen sucht. Visual Basic erwartet, daß sich alle Formularvorlagen in diesem Ordner befinden. Wenn Sie mehrere Bibliotheken für Vorlagen unterschiedlicher Arten anlegen (wie beispielsweise ein freiberuflicher Programmierer, der für verschiedene Firmen arbeitet), können Sie diese verschiedenen Vorlagensätze in unterschiedlichen Ordnern ablegen. Wenn Sie mit einem dieser Vorlagensätze arbeiten möchten, geben Sie den entsprechenden Pfad in das Dialogfeld **Optionen** ein.

Was Sie tun sollten	Was Sie lassen sollten
Ändern Sie die Formularvorlagen so ab, daß sie sich für Ihre Applikationen eignen, und speichern Sie diese Änderungen, wenn Sie feststellen, daß Sie sie häufig brauchen. Legen Sie eine Kopie der alten Formularvorlage unter einem anderen Namen an und nehmen Sie die Änderungen an der Kopie vor. Verwenden Sie die Kopie, wenn Sie das abgeänderte Formular brauchen.	Nehmen Sie die Änderungen nicht an der Original-Formularvorlage vor.

Die Formularvorlage Info-Dialogfeld

Die Aufgaben der verschiedenen Formularvorlagen sind unterschiedlich, aber die Einbindung in eine Applikation ist für alle gleich. In diesem Abschnitt fügen Sie einer Applikation eine Formularvorlage hinzu, um den Programmierprozeß zu vereinfachen. Eines der gebräuchlichsten Dialogfelder, das in allen Windows-Applikationen enthalten ist, ist das **Info**-Dialogfeld, das erscheint, wenn Sie im **Hilfe**-Menü den Eintrag **Info** auswählen. Abbildung 15.5 zeigt, wie die **Info**-Formularvorlage im Formularfenster von Visual Basic aussieht.

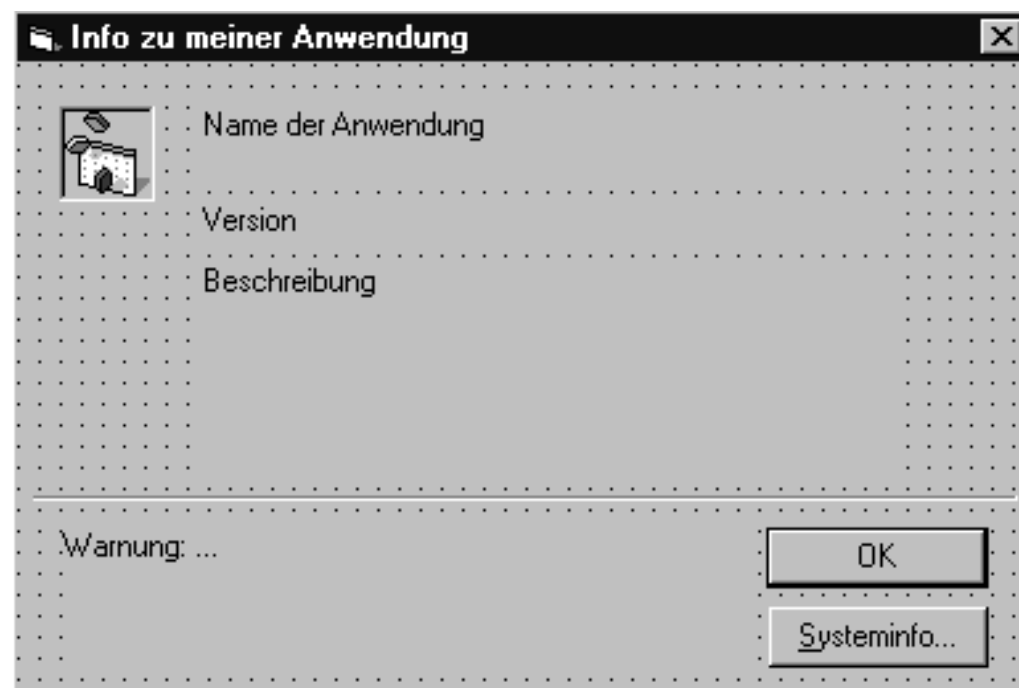


Abbildung 15.5:
Die Formularvorlage für das Info-Dialogfeld, das es in den meisten Windows-Applikationen gibt.

Um Ihnen zu zeigen, wie sich die Vorlage von dem eigentlichen Formular unterscheidet, das Sie irgendwann anlegen werden, betrachten Sie Abbildung 15.6. Sie zeigt das **Info**-Dialogfeld für Visual Basic. Wie Sie sehen, stellt die Formularvorlage Platzhalter für die Informationen bereit, die Sie für Ihre Applikation anzeigen möchten.



Abbildung 15.6:

Das Info-Dialogfeld von Visual Basic stimmt mit dem Format der Formularvorlage überein.

Jetzt werden Sie die Formularvorlage **Info**-Dialogfeld einfügen. Starten Sie ein neues Projekt und gehen Sie wie folgt vor:

1. Öffnen Sie den Menü-Editor. Fügen Sie das Menü **Hilfe** in die Menüleiste ein, mit **Info** als einziger Option. Geben Sie der Option **Info** den Namen `mnuHelpAbout`.
2. Geben Sie dem Formular für das Projekt den Namen `frmTestAbout` und setzen Sie seinen Titel auf **Beispiel für das Info-Dialogfeld**.
3. Wählen Sie im **Datei**-Menü den Eintrag **Speichern von Formular unter** und geben Sie als Dateinamen `Formular_Info_Feld` ein.
4. Wählen Sie im **Datei**-Menü den Eintrag **Projekt speichern unter** und geben Sie als Dateinamen `Info_Proj` ein. (Das **Info**-Dialogfeld, das Sie hinzufügen, verwendet den Projektnamen.)
5. Wählen Sie im **Projekt**-Menü den Eintrag **Formular hinzufügen**, um das Dialogfeld **Formular hinzufügen** anzuzeigen. Ein Dialogfeld ist nichts anderes als ein Formular mit Steuerelementen, und das **Info**-Dialogfeld wird Ihrer Applikation als neues Formular hinzugefügt, `frmAbout`.
6. Gehen Sie mit der Menüoption **Fenster** zurück zu Ihrem ursprünglichen Formular (`frmTestAbout`). Fügen Sie die Ereignisprozedur `mnuHelpAbout_Click()` ein:

```
Private Sub mnuHelpAbout_Click()  
    frmAbout.Show  
End Sub
```

7. Jetzt können Sie die Applikation ausführen. Wenn Sie im **Hilfe**-Menü den Eintrag **Info** auswählen, wird das **Info**-Dialogfeld angezeigt. Die Formularvorlage **Info**-Dialogfeld kennt den Namen des Projekts und zeigt ihn im Titelbereich an.

8. Klicken Sie auf **OK**, um das **Info**-Dialogfeld zu schließen. Schließen Sie das Hauptfenster, um die Applikation zu beenden.

Die Formularvorlage **Info**-Dialogfeld kennt den Namen des Projekts und zeigt ihn im Titelbereich an. Sie sucht an einer speziellen Position nach diesem Namen - im **App**- Objekt (Sie lernen es im nächsten Kapitel kennen).

Gehen Sie in das **Fenster**-Menü und zeigen Sie das **Info**-Formular im Arbeitsbereich an. Die Ereignisprozedur **Form_Load ()** für das Dialogfeld enthält den folgenden Code, der den Titel aus dem **App**-Objekt initialisiert:

```
1: Private Sub Form_Load ( )
2: Me.Caption = "Info " & App.Title
3: lblVersion.Caption = "Version " & App.Major & "." & App.Minor & _
  "." & App.Revision
4: lblTitle.Caption = App.Title
5: End Sub
```

Zeile 3 setzt in der **Form_Load ()**-Ereignisprozedur des Hauptformulars zur Laufzeit die Haupt- und Unterversionsnummer sowie die Revisionsnummer. Sie können den Verweis auf diese Werte entfernen (ebenso wie die Beschriftungen dafür), wenn Sie keine Versionsnumerierung brauchen.

Das Formularmodul mit dem **Info**-Dialogfeld enthält Code, der diese Werte für Sie initialisiert (Titel und Versionsnummer). Das **Info**-Dialogfeld kann jedoch keine Bezeichnungsfelder initialisieren, beispielsweise für eine Beschreibung der Applikation oder eine Warnung. Diese Beschreibung initialisieren Sie in der **Caption**-Eigenschaft des Bezeichnungsfelds (**lblDescription**). Wenn Sie keinen Bereich für Copyright oder irgendwelche Warnungen brauchen, entfernen Sie das Bezeichnungsfeld **Warnung** (**lblDisclaimer**). Zeigen Sie statt dessen beispielsweise ein Icon oder ein Bildfeld an.

Das **Info**-Dialogfeld beinhaltet mehr Code als nur für die Bereitstellung des Namens und der Versionsnummer einer Applikation. Offensichtlich ist dieser Code im Formularmodul des Dialogfelds enthalten. Darüber hinaus enthält dieses Formularmodul Code, der das Dialogfeld aus dem Speicher entfernt, wenn der Benutzer auf **OK** klickt. Die eigentliche Leistungsfähigkeit des **Info**-Dialogfelds liegt jedoch in der Ereignisprozedur für die Schaltfläche **System-Info**:

```
1: Private Sub cmdSysInfo_Click()
2: Call StartSysInfo
3: End Sub
```

StartSysInfo ist eine allgemeine Prozedur, die im Formularmodul für das **Info**-Dialogfeld unter den Ereignisprozeduren aufgelistet wird. Der Code in **StartSysInfo** führt das Systemprogramm **MSINFO32.EXE** aus, das sich in Ihrem Windows-Verzeichnis befindet. Sie könnten dieses Programm mit Ihrem eigenen Code überschreiben, aber warum sollten Sie nicht bei einem Standardprogramm bleiben, das die Benutzer von anderen Windows-Programmen her bereits kennen?

Führen Sie die Applikation wieder aus. Zeigen Sie das **Info**-Dialogfeld an und klicken Sie auf die Schaltfläche **System-Info**. Das Programm zur Ausgabe der Systeminformation wird gestartet. (Diese Applikation ist ein *Kindprozeß* Ihrer Applikation. Ihre Applikation wird fortgesetzt, sobald der Benutzer das Fenster mit der Systeminformation geschlossen hat.). Abbildung 15.7 zeigt das Fenster

System-Info, das der Code im Formularmodul für Ihre Applikation erzeugt.



Ihr System-Info-Dialogfeld unterscheidet sich von dem in Abbildung 15.7 gezeigten - abhängig von der Konfiguration Ihres Systems.

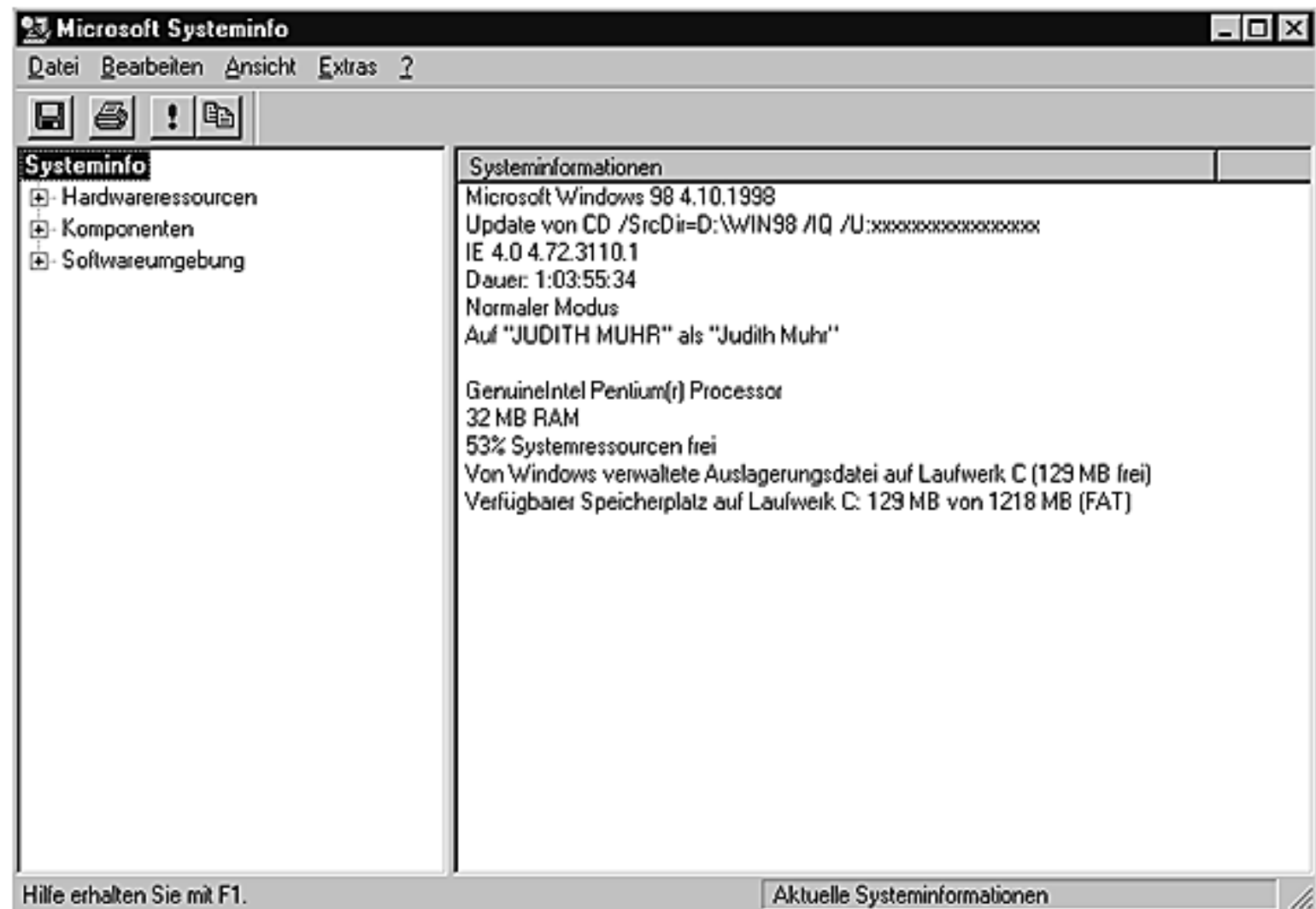


Abbildung 15.7:
Der Code für das Info-Dialogfeld zeigt das Dialogfeld Microsoft Systeminfo an.

Weitere Formularvorlagen

Nachdem Sie ein Projekt erstellt haben, das das **Info**-Dialogfeld nutzt, werden Sie keine Probleme haben, auch die drei anderen wichtigen Formularvorlagen einzufügen. Wählen Sie im **Projekt**-Menü den Eintrag **Formular hinzufügen**. Sie erhalten eine Auswahl mehrerer Formularvorlagen, die Sie Ihrem aktuellen Projekt hinzufügen können.

Dieser Abschnitt betrachtet die folgenden Formularvorlagen:

- Begrüßungsbildschirm
- Anmelde-Dialogfeld
- Optionen-Dialogfeld
- Tips und Tricks
- ODBC-Anmeldung

Nicht alle Applikationen müssen alle Formularvorlagen erhalten. Die meisten Windows-Applikationen verwenden jedoch das **Info**-Dialogfeld, so daß Sie sich angewöhnen sollten, es auch in Ihren Applikationen bereitzustellen. Welche anderen Formularvorlagen Sie verwenden, wird durch die Anforderungen Ihrer Applikation festgelegt. Dieser Abschnitt betrachtet die Formularvorlagen genauer, so daß Sie entscheiden können, welche davon Sie Ihren Projekten hinzufügen.



In diesem Abschnitt lernen Sie, wie Sie die verschiedenen Formularvorlagen mit Ihrer Applikation verbinden. Wenn Sie eine oder mehrere dieser Vorlagen im Anwendungs-Assistenten einfügen, übernimmt Visual Basic eine Menge Arbeit für Sie. Auf diese Weise ist es einfacher, die Vorlagen in Ihr Projekt zu integrieren.

Der Begrüßungsbildschirm

Ein *Begrüßungsbildschirm* zeigt einführende Informationen sowie möglicherweise Copyright-Hinweise und Kontaktpersonen für Ihr Projekt an. (Man spricht hier zwar von einem Bildschirm, aber eigentlich handelt es sich nur um ein Formularfenster in der **FORMS**-Auflistung Ihres Projekts.) Die wichtigste Aufgabe des Begrüßungsbildschirms ist, Ihre Benutzer zu begrüßen. Anders als das **Info**-Dialogfeld wird der Begrüßungsbildschirm nur beim Starten angezeigt. Der Begrüßungsbildschirm enthält ein Bild, das die Aufmerksamkeit Ihres Benutzers auf sich ziehen soll, ebenso wie Informationen über das Produkt.



Abbildung 15.8:
Excel-Benutzer sehen einen Begrüßungsbildschirm ähnlich dem hier gezeigten, wenn sie das Programm laden.

Häufig verschwindet ein Begrüßungsbildschirm, nachdem eine bestimmte Zeit vergangen ist. Sie können aber auch eine Schaltfläche bereitstellen oder Tastencodes abfragen, so daß der Benutzer den Begrüßungsbildschirm selbst schließen kann. In der Regel wird jedoch ein Timer verwendet, um den Begrüßungsbildschirm eine bestimmte Zeitdauer lang anzuzeigen. Außerdem ist es sinnvoll, einen Begrüßungsbildschirm anzuzeigen, um die Dauer des Ladevorgangs und der Dateiinitialisierung kürzer erscheinen zu lassen.

Der Begrüßungsbildschirm stellt eine Anforderung, die es für das **Info**-Dialogfeld nicht gab. Sie müssen Ihre Applikation anweisen, den Begrüßungsbildschirm anzuzeigen, bevor das normale

Formular erscheint. Dazu spezifizieren Sie ihn im **Eigenschaften**- Dialog als Startobjekt (wählen Sie im **Projekt**-Menü den Eintrag **Eigenschaften**). Nachdem Sie Ihrem Projekt die Vorlage für den Begrüßungsbildschirm hinzugefügt haben, fügen Sie eine Ereignisprozedur für Schaltflächen oder einen Timer ein, die zum nächsten Fenster springt.

Legen Sie jetzt ein Projekt mit einem Begrüßungsbildschirm an:

1. Öffnen Sie ein neues Projekt. Für das Formular **Form1** sind keine Veränderungen erforderlich.
2. Wählen Sie im **Projekt**-Menü den Eintrag **Formular hinzufügen**. Wählen Sie den Begrüßungsbildschirm aus. Visual Basic zeigt eine entsprechende Vorlage an.
3. Ändern Sie die Bezeichnungsfelder so ab, daß sie wie in Abbildung 15.9 gezeigt aussehen.

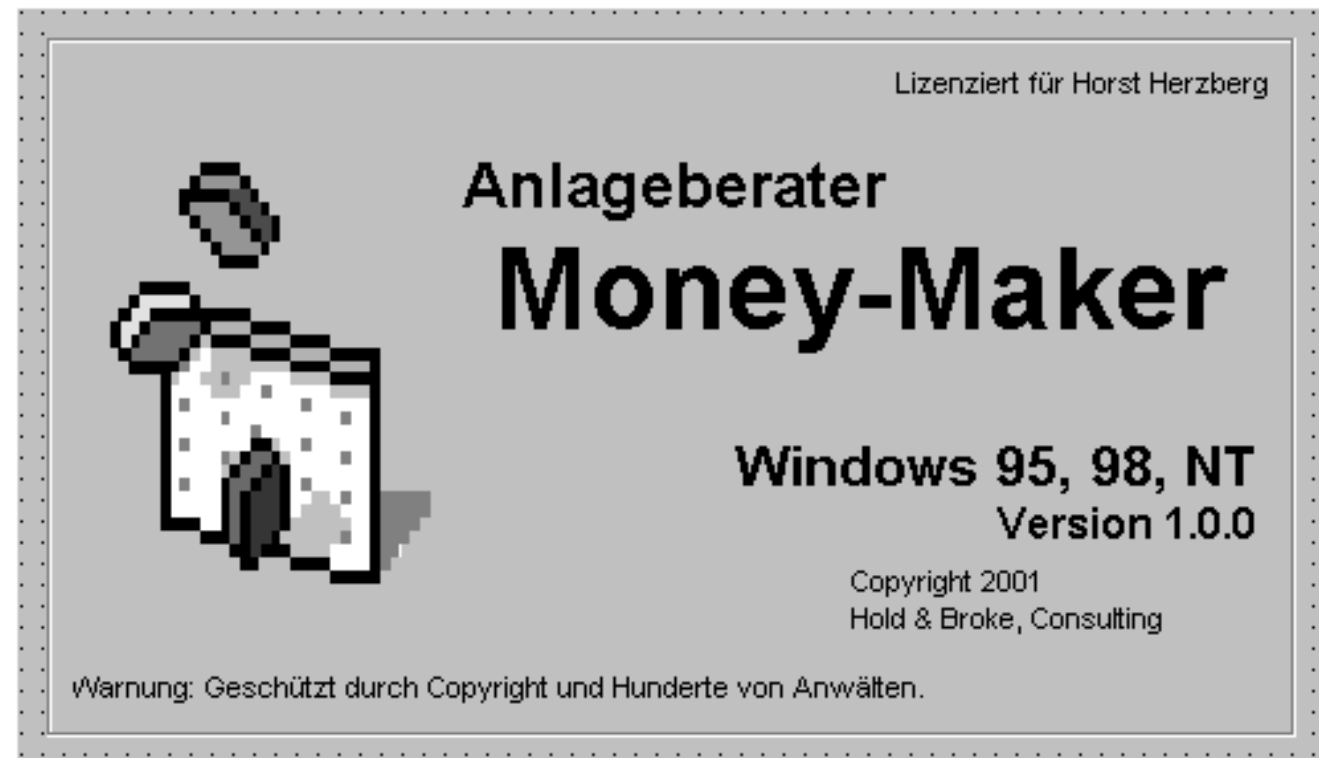


Abbildung 15.9:
Der Begrüßungsbildschirm wird an die jeweilige Applikation angepaßt.

4. Wählen Sie im **Projekt**-Menü den Eintrag **Eigenschaften**. Setzen Sie das Startobjekt auf **frmSplash** (das ist der Name des Begrüßungsbildschirms) und klicken Sie auf **OK**.
5. Fügen Sie dem Ereignis **Form_KeyPress()** und dem Ereignis **Frame1_Click()** die folgende Zeile hinzu (in beiden Prozeduren hinter der Anweisung **Unload Me**):
Form1.Show 'Das Hauptformular anzeigen
6. Führen Sie die Applikation aus. Als erstes erscheint Ihr Begrüßungsbildschirm. Drücken Sie auf eine Taste oder klicken Sie mit einer Maustaste, damit der Begrüßungsbildschirm verschwindet und das Hauptformular, **Form1**, erscheint. Schließen Sie das Formularfenster und kehren Sie in die Entwicklungsumgebung von Visual Basic zurück.



Der Begrüßungsbildschirm erhält automatisch den Namen des Programms aus dem Projektnamen, egal was Sie in das Bezeichnungsfeld in der Mitte eingeben. Sie müssen das Projekt also unter dem entsprechenden Namen ablegen oder den Code so abändern, daß Ihre Bezeichnungsfelder unverändert übernommen werden.

Das Anmelde-Dialogfeld

Weil immer mehr Computer online gehen, ist die Sicherheit zu einem immer wichtigeren Thema geworden. Das Anmelde-Dialogfeld ist eine interessante Formularvorlage, die Sie Ihrem Projekt hinzufügen können. Sie fordert den Namen und das Kennwort des Benutzers an und gibt die Werte zur Verarbeitung an Ihre Applikation weiter. Abbildung 15.10 zeigt das Anmelde-Dialogfeld.

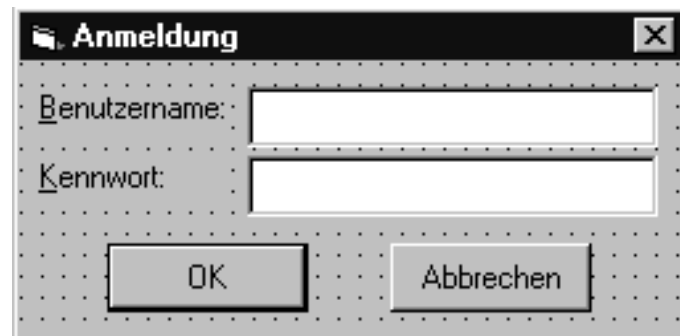


Abbildung 15.10:
Im Anmelde-Dialogfeld wird der Benutzer aufgefordert, einen Namen und ein Kennwort einzugeben.

Wenn ein Benutzer seinen Namen und sein Kennwort eingibt, erscheint der Benutzername im Dialogfeld, aber statt des Kennworts werden Sterne angezeigt (dank der `PasswordChar`-Eigenschaft des Dialogfelds), um zu verhindern, daß ein anderer das Kennwort mitliest. Trotzdem erhält das Dialogfeld das richtige Kennwort. Das erste Kennwort ist *password*, mit dem Sie das Anmelde-Dialogfeld testen können. Listing 15.1 zeigt den Code im Formularmodul für das Anmelde-Dialogfeld.

Listing 15.1: Das Anmelde-Dialogfeld ermöglicht den Benutzern, sich bei Ihrer Applikation anzumelden

```
1: Option Explicit
2: Public LoginSucceeded As Boolean
3:
4: Private Sub cmdCancel_Click()
5: ' Globale Variable auf false setzen,
6: ' um eine fehlgeschlagene Anmeldung zu melden
7: LoginSucceeded = False
8: Me.Hide
9: End Sub
10:
11: Private Sub cmdOK_Click()
12: ' Prüfen, ob das Kennwort korrekt ist
13: If txtPassword = "password" Then
14: ' Hier Code eingeben, der die erfolgreiche Anmeldung
15: ' an die aufrufende Sub-Prozedur weitermeldet
16: ' Die Verwendung einer globalen Variable ist am einfachsten
17: LoginSucceeded = True
18: Me.Hide
19: Else
20: MsgBox "Ungültiges Kennwort. Wiederholen!", , "Login"
21: txtPassword.SetFocus
```

```
22: SendKeys "{Home}+{End}" End If
```

```
23: End Sub
```

Das Formularmodul verwendet die globale Variable `LoginSucceeded` (in Zeile 2 deklariert), die Ihr Code bei der Rückgabe auf `True` oder `False` überprüft. Wenn der Benutzer auf die Schaltfläche **Abbrechen** klickt, setzt die Ereignisprozedur `cmdCancel_Click()` `LoginSucceeded` in Zeile 7 auf `False` und verbirgt das Anmelde-Formular.

Um den Code für Ihre eigenen Benutzer zu übernehmen, gehen Sie wie folgt vor:

1. Setzen Sie die Kennwortkonstante in der Ereignisprozedur `cmdOK_Click()` auf das Kennwort für Ihre Applikation. Häufig wird das Kennwort in einer Datei abgelegt und verschlüsselt. Wenn Sie das Kennwort in einer Binärdatei mit wahlfreiem Zugriff oder einer Datenbankdatei ablegen, kann es mit einem Texteditor nicht gelesen werden, wie es etwa der Fall ist, wenn Sie das Kennwort in einer Textdatei ablegen.
2. Setzen Sie den Text im Meldungsfeld auf die Meldung, den Sie dem Benutzer anzeigen möchten, falls er ein falsches Kennwort eingibt.
3. Aus Sicherheitsgründen sollten Sie die Routine zur Auswertung des Kennworts in eine **For**-Schleife schreiben, um eine bestimmte Anzahl von Anmeldeversuchen zu erlauben, bevor das Anmelde-Dialogfeld endgültig verschwindet. Damit wird es erschwert, das Kennwort zu knacken.



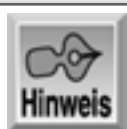
Nur weil Microsoft den Code für das Anmelde-Dialogfeld mit einer globalen Variablen ausgestattet hat, heißt das nicht, global sei gut. Wie der Kommentar in der Ereignisprozedur `cmdOK_Click()` erklärt, ist die globale Variable die einfachste Methode, die aufrufende Applikation über die erfolgreiche Anmeldung zu informieren. Guter Programmierstil wäre, die globale Variable durch lokale Variablen zu ersetzen. Die vielleicht beste Möglichkeit, diesen Code so abzuändern, daß er auf lange Sicht besser wartbar wird, ist die Umwandlung der Sub-Prozedur in eine Funktionsprozedur mit einem booleschen Rückgabewert. Die aufrufende Applikation kann dann prüfen, ob der Rückgabewert dieser Funktion gleich `True` oder `False` ist.

Der Code am Ende der Routine `cmdOK_Click()` erscheint Ihnen möglicherweise verwirrend, weil er von dem gewohnten Stil abweicht - außerdem gibt es dort ein paar neue Anweisungen. Bisher war `MsgBox()` eine Funktion, aber dieser Code enthält die folgende **MsgBox**-Anweisung:

```
MsgBox "Falsches Kennwort. Wiederholen!", , "Login"
```

Visual Basic 6 unterstützt diese **MsgBox**-Anweisung noch, aber Microsoft versucht, die Programmierer davon zu überzeugen, statt dessen die Funktion `MsgBox()` zu verwenden. Um diese Anweisung in eine Funktion umzuwandeln, weisen Sie die Funktion einer Variablen zu (am besten **Variant**) und fügen Klammern ein, nämlich so:

```
varKeys = MsgBox("Falsches Kennwort. Wiederholen!", "Login")
```



Die **MsgBox**-Anweisung erkennt nicht, auf welche Schaltfläche der Benutzer geklickt hat, um das Meldungsfeld zu schließen. Die Funktion `MsgBox()` dagegen gibt in ihrem Rückgabewert an, welche Schaltfläche der Benutzer angeklickt hat. Wenn jedoch **OK** die einzige Schaltfläche für `MsgBox()` ist, brauchen Sie keine Unterscheidungen zu treffen, weil der Benutzer das Meldungsfeld nur mit der **OK**-Schaltfläche schließen kann.

Die nächste Anweisung gibt in der **SetFocus**-Methode den Fokus an das Textfeld **Kennwort** weiter (das passiert nur, wenn der Benutzer ein ungültiges Kennwort eingegeben hat). Wenn Sie die **SetFocus**-Methode auf ein Steuerelement anwenden, das den Fokus annehmen kann, setzt die Applikation den Fokus entsprechend. Der Fokus könnte auch auf ein anderes Steuerelement übergehen, beispielsweise auf die **OK**- Schaltfläche, aber **SetFocus** gibt ihn an das Textfeld zurück, weil der Benutzer das Kennwort noch einmal eingeben muß.

Die letzte Anweisung verwendet **SendKeys**, um den Text zu markieren, der im Textfeld **Kennwort** erscheint. Egal, was der Benutzer als falsches Kennwort eingegeben hat, die **SendKeys**-Anweisung setzt den Cursor an den Anfang des Textfelds und dann an das Ende des Textfelds - so daß der gesamte Textfeldinhalt markiert wird und der nächste Tastendruck des Benutzers ihn vollständig löscht.



Die Anweisung **SendKeys** wurde in Kapitel 7 genau erklärt.

Das Optionen-Dialogfeld

Von allen Formularvorlagen erledigt das **Optionen**-Dialogfeld die wenigste Arbeit, bietet aber die meisten Verwendungszwecke. Wenn Sie ein **Optionen**-Dialogfeld einfügen, sehen Sie die in Abbildung 15.11 gezeigte Formularvorlage. Das Dialogfeld enthält vier Registerkarten. Auf diesen Registerkarten legen Sie Steuerelemente an, um dem Benutzer auf diese Weise mögliche Optionen bereitzustellen. Außerdem können Sie auch neue Registerkarten anlegen.



Abbildung 15.11:
Das Optionen-Dialogfeld zeigt Seiten für die verschiedenen Optionen an.

Viele Windows-Programme enthalten ein **Optionen**-Dialogfeld (im **Extras**-Menü), das ähnlich dem mit der Formularvorlage angelegten **Optionen**-Dialogfeld aussieht. Es handelt sich dabei zwar nur um die Oberfläche, aber immerhin haben Sie damit einen Ausgangspunkt für komplexere Dialogfelder.

Das **Optionen**-Dialogfeld verwendet ein spezielles ActiveX-Steuerelement, **TabStrip**, das ein Dialogfeld mit mehreren Registerkarten erzeugt. Wenn Sie einer Ihrer Applikationen ein **TabStrip**-Steuerelement hinzufügen möchten - d.h. Sie möchten diese Formularvorlage nicht benutzen -, fügen Sie es über **Projekt, Komponenten** mit der Option *Microsoft Custom Controls 6.0* hinzu.

Für die Verwendung des **Optionen**-Dialogfelds gehen Sie wie folgt vor:

- Fügen Sie dem **Optionen**-Dialogfeld so viele Registerkarten hinzu, wie Sie brauchen. Am einfachsten klicken Sie dazu auf eine der Registerkarten und klicken dann für die Eigenschaft **Benutzerdefiniert** auf die Schaltfläche mit den drei Punkten. Das Dialogfeld **Eigenschaftenseiten** erscheint. Es hilft Ihnen, Seiten, Registerkarten und ToolTips einzurichten (siehe Abbildung 15.12).



Abbildung 15.12:

Im Dialogfeld Eigenschaftenseiten richten Sie die Dialogfeldseiten ein.

- Fügen Sie eine allgemeine Prozedur ein, die alle Steuerelemente im **Optionen**- Dialogfeld liest und die dort vorhandenen Optionen setzt.
- Rufen Sie von der Prozedur `cmdApply_Click()` aus die Prozedur zum Setzen der Optionen auf, damit alle Optionen in Kraft treten, wenn der Benutzer auf die Schaltfläche **Übernehmen** klickt. (Sie können die **Übernehmen**-Schaltfläche und ihre Ereignisprozedur auch entfernen, wenn dem Benutzer diese Funktion nicht zur Verfügung stehen soll.)
- Ersetzen Sie die folgende Anweisung, die in der Ereignisprozedur `cmdOK_Click()` steht, durch einen Aufruf Ihrer eigenen Prozedur zum Setzen der Optionen:

`MsgBox "Code zum Setzen der Optionen und Schließen des Dialogs!"`

- Ändern Sie die Ereignisprozedur `Form_KeyDown()` so, daß sie die Fokusreihenfolge

verwaltet, die das Dialogfeld unterstützt, wenn der Benutzer auf (Strg)+(ÿ__) drückt. Dieser Code ist nicht trivial, weil Sie in Ihren Programmanweisungen genau festlegen müssen, wie der Fokus von einem Steuerelement zum nächsten weitergegeben wird.



Die Ereignisprozedur `tbsOptions_Click( )` zeigt bei der Programmausführung die entsprechende Seite im `TabStrip`-Steuerelement an (und verbirgt die anderen Seiten).

Üben Sie das Einfügen von Formatvorlagen im Anwendungs-Assistenten. Dazu legen Sie ein neues Projekt an und führen den Anwendungs-Assistenten aus. Übernehmen Sie alle Vorgabewerte, bis Sie zum Dialogfeld **Standardformulare** gelangen. Klicken Sie alle vier Formatvorlagen an (drücken Sie aber nicht die Schaltfläche **Formatvorlagen**, um zusätzliche Formatvorlagen einzufügen). Klicken Sie zweimal auf **Weiter** und dann auf **Fertig**. Visual Basic baut das Anwendungsgerüst für Sie auf. Nachdem der Anwendungs-Assistent fertig ist, klicken Sie auf **OK**, um die Setup-Anweisungen zu lesen, bevor Sie das Dialogfeld schließen.

Wie Sie aus Kapitel 1 wissen, legt der Anwendungs-Assistent nur ein Gerüst für eine Applikation an. Die Details müssen Sie selbst einsetzen. Nichtsdestotrotz enthält dieses Gerüst alle vier Formularvorlagen, die der Anwendungs-Assistent standardmäßig anbietet.

Probieren Sie die Applikation aus, indem Sie testen, was bereits möglich ist. Geben Sie kein Kennwort ein (das Standardkennwort ist leer, bis Sie dem Codemodul eines hinzugefügt haben). Sie sehen, daß das Programm den Benutzernamen (in diesem Fall *Ihren Namen*) aus dem `App`-Objekt ermittelt hat und ihn automatisch im Textfeld **Benutzername** anzeigt.

Der Eröffnungsbildschirm erscheint und wird gleich wieder ausgeblendet. Anschließend erscheint das Formular für Ihr Programm. Das **Info**-Dialogfeld wird angezeigt, wenn Sie im **Hilfe**-Menü den Eintrag **Info** auswählen. Das **Optionen**-Dialogfeld wird angezeigt, wenn Sie im **Ansicht**-Menü den Eintrag **Optionen** auswählen. Dieses Projekt stellt zwar nur ein Applikationsgerüst bereit, enthält aber bereits ausreichend viel Code, an dem Sie sich orientieren können, wenn Sie eigene Applikationen mit einer oder mehreren Formularvorlagen entwickeln.



Der Anwendungs-Assistent listet Ihre Applikationen unter den bereitgestellten Formularvorlagen auf (d.h. falls Sie selbst Formularvorlagen angelegt haben), wenn Sie auf die Schaltfläche **Formularvorlagen** klicken.

Tips und Tricks

Haben Sie Software, die Sie beim Aufruf mit einem Tip begrüßt, wie Sie sie besser nutzen können? Windows 95 bietet einen solchen Tip an (d.h. solange Sie ihn nicht deaktivieren). Immer wenn Sie Windows 95 starten, sehen Sie einen neuen Tip. Um diese Anzeige zu deaktivieren, heben Sie die Markierung im Kontrollkästchen **Tips beim Starten anzeigen** auf. Abbildung 15.13 zeigt die Formatvorlage **Tips und Tricks**.

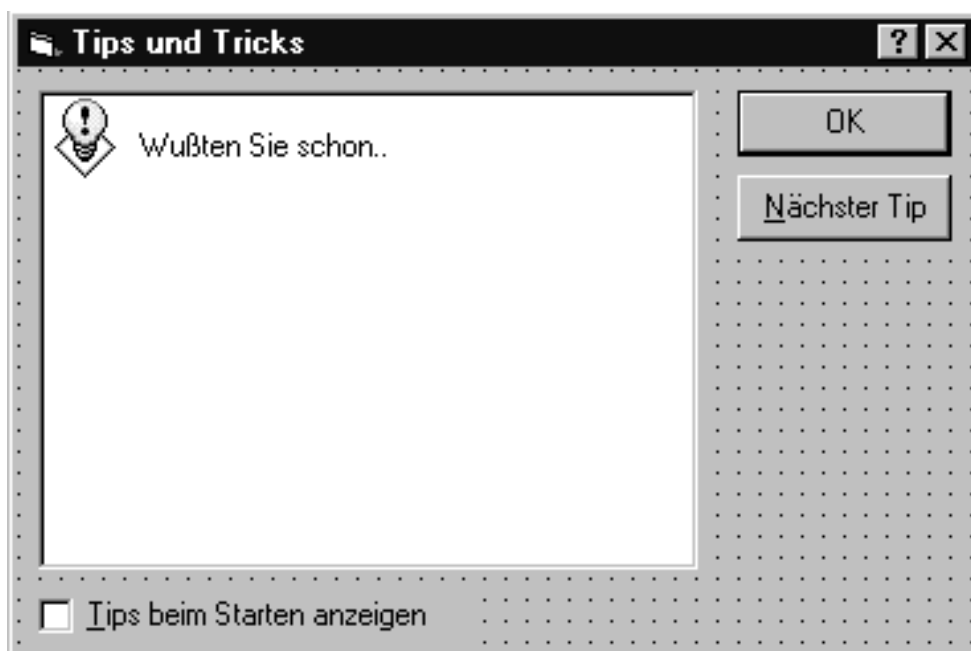


Abbildung 15.13:

Das Dialogfeld Tips und Tricks stellt Hilfe für die Neulinge unter Ihren Benutzern bereit.

Wenn Sie einem Formularfenster das Dialogfeld **Tips und Tricks** hinzufügen, fügt Visual Basic Ihrer FORMS-Auflistung das Formular **Tips und Tricks** hinzu. Abhängig von Ihrer Bildschirmgröße und der verwendeten Standardschrift müssen Sie das Bezeichnungsfeld möglicherweise vergrößern, in dem der Text **Tips beim Starten anzeigen** ausgegeben wird. Klicken Sie unterhalb des Bezeichnungsfelds **Wussten Sie schon**, um das Bezeichnungsfeld `lblTipText` anzuzeigen. Dieses Textfeld enthält die Tips, die Sie anzeigen wollen.

Der Code innerhalb des Formularmoduls richtet das Formular so ein, daß es bei jedem Programmstart einen täglichen Tip anzeigt. Die folgenden Richtlinien helfen Ihnen, die Anforderungen des Dialogfelds **Tips und Tricks** zu verstehen:

- Der Code für das Dialogfeld legt eine neue Auflistung an, **Tips**. Die Applikation holt die Tips, die beim Starten angezeigt werden, aus dieser Auflistung.
- Offensichtlich muß die **Tips**-Auflistung die Tips aus einer Datei lesen, die Sie anlegen und dem Projekt hinzufügen. Die Datei sollte pro Zeile einen Tip enthalten.
- Der Code lädt den Dateinamen in die Konstante `TIP_FILE` der Funktion `LoadTips()`. Die Prozedur fügt der Auflistung die Tips mit Hilfe der `Add`-Methode beim Lesen der Daten hinzu. Sie brauchen die Datei mit den Tips nur mit einem Texteditor anzulegen.
- Die Applikation `DoNextTip()` wählt einen zufälligen Eintrag aus der Auflistung aus und zeigt ihn mit der Methode `DisplayCurrentTip` an (das ist eine Sub-Prozedur ganz unten im Code).
- Der technischste Abschnitt des Codes ist zugleich der kürzeste. Er befindet sich in der Prozedur `chkLoadTipsAtStartup()`. Glücklicherweise stellt Microsoft den Code dafür bereit. Der Code verwendet den Befehl `SaveSetting`, um Ihre System-Registrierungsdatei zu ändern. Der Befehl `SaveSetting` speichert den Wert, der festlegt, ob der Benutzer beim Starten Tips sehen möchte. Hebt der Benutzer die Markierung für das Kontrollkästchen **Tips beim Starten anzeigen** auf, wird die Registrierungsdatei entsprechend aktualisiert und der Code zeigt die Tips in folgenden Sitzungen nicht mehr an.

Die Formatvorlage **Tips und Tricks** ist vielleicht das gebräuchlichste der heute beschriebenen Standarddialogfelder. Gehen Sie wie folgt vor, um das Dialogfeld **Tips und Tricks** auf Ihrem System

anzulegen:

1. Starten Sie den Windows-Editor. Legen Sie die Datei `Tipofday.txt` an (das ist der Standarddateiname, der vom Dialogfeld **Tips und Tricks** verwendet wird). Legen Sie diese Datei im Verzeichnis Ihrer Applikation ab. Nachdem Sie den folgenden Inhalt eingegeben haben, speichern Sie die Datei und verlassen den Editor:

Putzen Sie Ihre Zähne täglich, Ihre Schuhe einmal in der Woche.
Sparen Sie fürs Alter (niemand anderes wird sich um Sie kümmern).
Schlafen Sie aus - Ihr Spiegelbild wird es Ihnen danken.
Lesen Sie ein Buch, bevor Sie sich den Film dazu ansehen.
Fahren Sie defensiv - Ihr Leben ist zu kostbar.
Planen Sie für jeden gegessenen Nachtisch eine Stunde Sport ein.
Seien Sie nicht rechthaberisch - das macht Sie unsympathisch.

2. Legen Sie eine neue Applikation an und zeigen Sie das Standard-Formularfenster an. Bevor Sie irgend etwas anderes tun, speichern Sie das Formular und das Projekt im selben Verzeichnis wie die Tip-Datei. Wenn sich diese Dateien nicht im selben Verzeichnis befinden, findet Ihr Code die Tips nicht.

3. Wählen Sie im **Projekt**-Menü den Eintrag **Formular hinzufügen**. Wählen Sie **Tips und Tricks**.

4. Ändern Sie die **Caption**-Eigenschaft von `Wußten Sie schon...` in `Gute Ratschläge...`

5. Führen Sie die Applikation aus, um die Ergebnisse zu sehen.

Wo ist das Dialogfeld **Tips und Tricks** hingekommen? Sie haben jetzt die Tip-Datei eingerichtet und Ihrer Applikation das Dialogfeld **Tips und Tricks** hinzugefügt - aber Visual Basic zeigt beim Starten das Hauptfenster für Ihr Projekt an (`Form1`, falls Sie es nicht umbenannt haben). Sie müssen das Formular mit dem Tip des Tages als Startobjekt festlegen und Ihren Code so schreiben, daß das Hauptfenster angezeigt wird, nachdem der Benutzer das Formular **Tips und Tricks** geschlossen hat.

Wählen Sie im **Projekt**-Menü den Eintrag **Eigenschaften** und setzen Sie das Startobjekt auf `frmTip`. Klicken Sie auf **OK**, um das Dialogfeld zu schließen. Jetzt zeigt die Applikation das Dialogfeld **Tips und Tricks** an, wenn der Benutzer sie aufruft.

Sie müssen das normale Formular (immer noch `Form1`) mit dem Formular für das Dialogfeld **Tips und Tricks** verknüpfen, so daß das normale Formular erscheint, wenn das Dialogfeld geschlossen wird. Ändern Sie also die Prozedur `cmdOK_Click()` wie folgt ab:

```
1: Private Sub cmdOK_Click()  
2: Unload Me ' Tip-Dialogfeld aus dem Speicher entfernen  
3: Form1.Show ' Normales Formular anzeigen  
4: End Sub
```

Außerdem brauchen Sie noch ein zusätzliches Element. Wenn der Benutzer in folgenden Sitzungen keine weiteren Tips sehen will, gibt es keine Möglichkeit, im Formularmodul das normale Formular anzuzeigen. Deshalb fügen Sie der Prozedur `Form_Load()` die Anweisung `Form1.Show` hinzu:

```
1: Private Sub Form_Load()  
2: Dim ShowAtStartup As Long  
3:  
4: ' Prüfen, ob wir beim Starten angezeigt werden sollen  
5: ShowAtStartup = GetSetting(App.EXENAME, "Optionen", _
```

```

"Tips beim Starten anzeigen", 1)
6: If ShowAtStartup = 0 Then
7: Unload Me
8: Form1.Show ' Normales Formular anzeigen *** Neue Anweisung
9: Exit Sub
10: End If ' Der restliche Code wird hier nicht gezeigt

```

Jetzt können Sie die Applikation ausführen und die Tips nach dem Zufallsprinzip anzeigen, indem Sie auf die Funktion **Nächster Tip** klicken. Wenn Sie auf OK klicken, erscheint das normale Formular; es ist noch leer und langweilig, weil Sie noch keine Steuerelemente darauf angelegt haben. Wenn der Benutzer beschließt, die Tips in nachfolgenden Sitzungen nicht mehr anzuzeigen, zeigt die Applikation beim Starten fortan das normale Programmfenster an, Form1.

	Es gibt auch noch andere Möglichkeiten, die Applikation so einzurichten, daß entweder der Tip oder das normale Startformular angezeigt werden. Beispielsweise könnten Sie einer Sub-Prozedur namens Main die entsprechenden Show-Methoden hinzufügen, und diese Main-Routine zum Startobjekt machen.

Die ODBC-Anmeldung

ODBC stellt einen Standard-Befehlssatz für den Zugriff auf unterschiedliche Datenarten auf unterschiedlichen Computern zur Verfügung. Der ODBC-Standard ermöglicht Ihnen, einen Datenzugriff zu programmieren, der andernfalls nicht möglich wäre. ODBC hat das Ziel, unterschiedlichen Systemen den Zugriff auf Daten zu geben, die an anderer Stelle abgelegt sind.

Abbildung 15.14 zeigt die Formularvorlage, die erscheint, wenn Sie das Formular **ODBC-Anmeldung** in Ihrem Formularfenster anlegen.

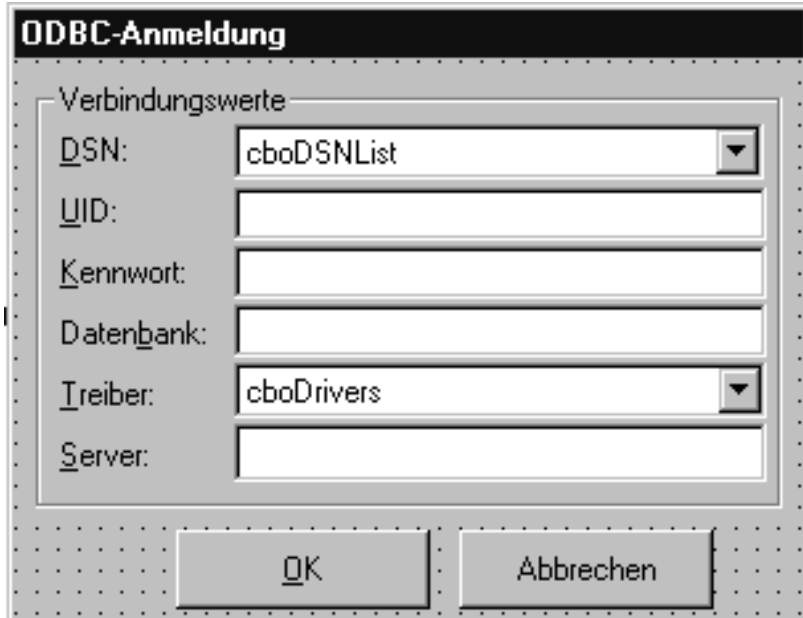


Abbildung 15.14:
Mit Hilfe der Formularvorlage ODBC-Anmeldung richten Sie den Zugriff auf externe Datenbanken ein.

Das Dialogfeld **ODBC-Anmeldung** wird Ihrer Applikation hinzugefügt, damit die Benutzer auf eine ODBC-Datenbankquelle zugreifen können. Die Quelle gibt an, auf welchen Datentyp die Applikation zugreifen soll, und wo diese Daten abgelegt sind. Tabelle 15.1 beschreibt die einzelnen ODBC-Felder.

Tabelle 15.1: Die Textfelder im Dialogfeld ODBC-Anmeldung spezifizieren ODBC-Optionen

Name	Beschreibung
DSN	Der Name der Datenquelle. Diese Option listet (in einem Drop-down-Listenfeld) die aktuell in der System-Registrierung des Benutzers registrierten ODBC-Quellen auf.
UID	Die Benutzer-ID, anhand derer die ODBC-Datenbank feststellen kann, ob der Benutzer berechtigten Zugriff auf die Daten hat.
Kennwort	Das Kennwort für den Systemzugriff.
Datenbank	Der Name der Datenbank, mit der eine Verbindung eingerichtet werden soll.
Treiber	Ein Drop-down-Listenfeld, das alle Treiber auf dem System auflistet und es dem Benutzer ermöglicht, einen neuen ODBC-Treiber zu registrieren.
Server	Der Name des Servers, der die Datenbank bereitstellt, falls der DSN nicht zur Verfügung steht.

Der Code für die Verbindung der Dialogfeldwerte für die ODBC-Anmeldung mit der entsprechenden ODBC-kompatiblen Datenbank ist relativ umfangreich. Sie müssen die ODBC-Befehle verstehen, um eine solche Verbindung einrichten zu können. Diese Befehle können im vorliegenden Buch nicht erklärt werden, weil Visual-Basic-Programmierer nur selten solche Routinen brauchen, außer bei systembezogenen Applikationen.

Eine Formularvorlage hinzufügen

Es ist ganz einfach, den Formatvorlagen von Visual Basic eigene Formulare hinzuzufügen. Nachdem Sie ein Formular angelegt haben, das Sie als Formatvorlage verwenden möchten, speichern Sie dieses im Ordner \Templates\Forms. Ab sofort wird Ihr Formular in allen Listen ebenso wie im Anwendungs-Assistent angezeigt.



Legen Sie Formatvorlagen an, die relativ allgemein gehalten sind und keine Details enthalten, es sei denn, Sie möchten daß diese Details in den Formularen erscheinen, die Sie aus der Formularvorlage erstellen möchten.



Um ein Formular aus dem Ordner für die Formatvorlagen zu entfernen, starten Sie den Windows-Explorer und gehen in den Ordner \Templates\Forms. Hier löschen Sie das Formular. Wenn Sie die Liste der Formatvorlagen zum nächsten Mal anzeigen, erscheint das Formular hier nicht mehr.

Zusammenfassung

In diesem Kapitel haben Sie gelernt, die Formatvorlagen zu nutzen, um Ihre Applikationen zu standardisieren und die Programmentwicklung zu beschleunigen. Die Formatvorlagen beinhalten verschiedene Standardformulare, die die Programmierer ihren Windows-Applikationen häufig hinzufügen. Ihre Benutzer werden es zu schätzen wissen, standardisierte Dialogfelder in Ihren

Applikationen vorzufinden.

Formularvorlagen sind allgemeine Formulare, die Sie jedem Projekt hinzufügen können. Sie enthalten Platzhalter für Text und Grafik, die Sie entsprechend anpassen. Ihre Applikation steuert, wie das Formular letztendlich angezeigt wird, und wie sie mit dem Code für die Formularvorlage zusammenarbeitet.

Kapitel 16 ist der erste Teil einer zweitägigen Betrachtung der Objekte und ihrer Rolle in der Visual-Basic-Programmierung.

Fragen und Antworten

Frage:

Warum sollen meine Formatvorlagen so allgemein gehalten sein?

Antwort:

Wenn Sie Ihre Formularvorlagen so allgemein wie möglich machen (und nur die Details vorgeben, die sich zwischen den einzelnen Applikationen nicht unterscheiden), ist es einfacher, neue Formulare aus diesen Vorlagen zu erstellen. In diesem Kapitel haben Sie mehrere Formularvorlagen gesehen, die Visual Basic bereitstellt.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welche Aufgaben haben Formularvorlagen?
2. Nennen Sie zwei Methoden, Ihren Applikationen Formularvorlagen hinzuzufügen.
3. Beschreiben Sie den Code, den Sie brauchen, um das **Info**-Dialogfeld mit Ihrer Applikation zu verknüpfen.
4. Richtig/Falsch. Sie müssen eigenen Code schreiben, um Systeminformationen anzuzeigen, wenn der Benutzer im **Info**-Dialogfeld auf die Schaltfläche **System- Info** klickt.
5. Was ist der Unterschied zwischen einem Eröffnungsbildschirm und Ihrem normalen Formular?
6. Ist das Dialogfeld **Tips und Tricks** ein Eröffnungsbildschirm?
7. Welche Aufgabe hat der Befehl `SaveSetting`?
8. Wofür steht ODBC und welche Aufgabe hat es?
9. Was tun Sie im Dialogfeld **Eigenschaften**, damit Ihre Applikation einen Eröffnungsbildschirm oder das Dialogfeld **Tips und Tricks** anzeigt?
10. Beschreiben Sie das Format der Tip-Datei, die im Dialogfeld **Tips und Tricks** gelesen wird.

Übungen

1. Folgen Sie den Empfehlungen aus dem Abschnitt über das Anmelde-Dialogfeld, um den Code zu verbessern. Ersetzen Sie die globale Variable durch lokale Variablen.
2. Entwickeln Sie eine Applikation, die bei jedem Programmaufruf einen anderen PC-Tip anzeigt. (Ändern Sie dazu die in diesem Kapitel vorgestellte Tip-Datei ab.) Fügen Sie dem normalen Formular eine Menüoption hinzu, die bewirkt, daß die Tips beim Starten wieder angezeigt werden. *Hinweis:* Probieren Sie die Prozedur `chkLoadTipsAtStartup_Click()` aus und verwenden Sie den Befehl `SaveSettings`, um die Tips zurückzusetzen. Sie haben zwar `SaveSetting` noch nicht eingesetzt, aber Sie haben alle Werkzeuge, die Sie brauchen, um dieses Projekt schnell zu vervollständigen.

Woche 3

Tag 16

Objekte in Visual Basic

Dieses Kapitel demonstriert die Verwendung von Objekten in Visual Basic. Bei der Arbeit mit Formularen, Steuerelementen und dem `Printer`-Objekt hatten Sie bereits mit Objekten zu tun. Hier soll dieses Wissen ergänzt werden. Sie lernen ein neues Steuerelement kennen, das *OLE-Steuerelement*, mit dessen Hilfe Sie Objekte von außerhalb der Visual-Basic-Umgebung verwenden können.

Darüber hinaus erfahren Sie mehr über die vordefinierten Objekte von Visual Basic, beispielsweise `Screen` und `App`. Diese vordefinierten Objekte senden Informationen an Ihre Applikationen, mit denen Sie sowohl Entscheidungen treffen als auch Titel und Benutzerinformationen auf dem Formular anzeigen. Nachdem Sie wissen, wie man mit vordefinierten Objekten arbeitet, erfahren Sie, wie Sie eine Auflistung dieser Objekte einsetzen.

Heute lernen Sie die folgenden Dinge kennen:

- Das OLE-Steuerelement
- Die Unterschiede zwischen Object Linking und Object Embedding
- Objekte aus anderen Applikationen verwenden
- In-Place-Aktivierung
- Steuerelementfelder
- Auflistungen
- Objektkatalog

OLE für Objekte, die außerhalb einer Applikation liegen

OLE ist Windows-Benutzern und -Programmierern ein vertrauter Begriff. Sie können OLE-Objekte in Ihre Applikationen einbetten, um Ihre Programme leistungsfähiger zu machen und gleichzeitig weniger Code entwickeln zu müssen. Durch die Verwendung von OLE-Objekten, die bereits in anderen Applikationen definiert sind, nutzen Sie die Vorteile der Wiederverwendung von Objekten.



OLE steht für *Object Linking and Embedding*. Viele Windows-Applikationen bieten ihre Daten als OLE-Objekte an und können solche Objekte innerhalb anderer Windows-Applikationen einbetten, die OLE unterstützen.



Die ActiveX-Technologie wird die OLE-Technologie bald ersetzt haben. Microsoft bezeichnet die ActiveX-Steuerelemente heute bereits als »frühere OLE-Steuerelemente«. Nichtsdestotrotz gibt es OLE noch. Viele Visual-Basic-Applikationen verwenden das OLE-Steuerelement weiterhin, und wie man sieht, hat Microsoft das OLE-Steuerelement in den Standardwerkzeugen der Werkzeugsammlung beibehalten. OLE stellt eine gute Einführung in die ActiveX-Technologie dar; das Thema dieses Kapitels ist also auch für das nächste Kapitel interessant, wo Sie ActiveX-basierte Objekte kennenlernen.

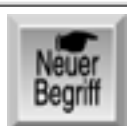
Sie verwenden OLE, um fremde Objekte in Ihren Applikationen einsetzen zu können. Das OLE-Steuerelement in Ihrer Werkzeugsammlung verwaltet die OLE-Verbindung zum OLE-Objekt Ihres Projekts. Der Benutzer kann die *In-Place-Aktivierung* nutzen, wenn er auf eingebettete OLE-Objekte zugreift.



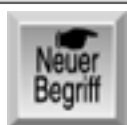
Die *In-Place-Aktivierung* bezieht sich auf die Möglichkeit, ein Objekt innerhalb einer Visual-Basic-Applikation zu bearbeiten, wozu jedoch die Menüs und Befehle der ursprünglichen Applikation des Objekts verwendet werden. Nachdem Sie ein OLE-Objekt in eine Applikation eingebettet haben, brauchen Sie keine Menüoptionen bereitzustellen, mit denen der Benutzer es bearbeitet - die Menüs der Elternapplikation erscheinen automatisch und können vom Benutzer verwendet werden.

Linking und Embedding

Das OLE-Steuerelement enthält entweder eine Verknüpfung zu einem Objekt, oder es handelt sich um ein eingebettetes Objekt, abhängig davon, wie Sie das Steuerelement einrichten. Wenn Sie eine Verknüpfung (Linking) zu einer anderen Applikation einrichten, enthält Ihr OLE-Steuerelement (das *Container-Steuerelement*) eine Verknüpfung zum Dokument einer anderen Applikation. Wenn die andere Applikation dieses Dokument verändert, zeigt auch Ihre Applikation diese Änderungen an. Das Objekt wird als *persistentes Objekt* bezeichnet, weil Sie den Objekthalt über die Verknüpfung aktuell halten können.



Ein *Container-Steuerelement* ist ein OLE-Steuerelement, das ein Datenobjekt einer anderen Applikation enthält.



Ein *persistentes Objekt* ist ein Objekt, das sich außerhalb Ihres Projekts befindet und das nicht verschwindet, nur weil Ihre Applikation beendet wird.

Wenn Sie ein OLE-Datenobjekt in Ihre Applikation einbetten, enthält Ihr OLE-Container eine Kopie des Objektdokuments, das in der anderen OLE-konformen Applikation erzeugt wurde. Es werden jedoch keine Verknüpfungen zwischen Ihrer Applikation und dem Original-OLE-Objekt verwaltet. Wenn die Originalapplikation das Objekt verändert, reflektiert Ihre Applikation diese Änderung nicht, weil Ihre Applikation eine Kopie des Objekts verwendet.

Und so beeinflussen die beiden unterschiedlichen OLE-Aktivitäten Ihr Projekt:

- Wenn Sie ein Objekt mit dem OLE-Steuerelement Ihrer Applikation verknüpfen, startet Visual

Basic die Applikation dieses anderen Objekts, sobald der Benutzer versucht, das Objekt zu ändern oder zu benutzen, und darauf doppelklickt. Der Benutzer wählt **Datei**, **Schliessen** und **Zurück zur Visual-Basic-Anwendung**, wenn er die Arbeit mit dem Objekt abgeschlossen hat.

- Wenn Sie ein Objekt in das OLE-Steuerelement Ihrer Applikation einbetten, wird das Objekt zum Teil Ihrer Applikation, weil die Kopie direkt dort eingefügt wird.

Das OLE-Steuerelement

Öffnen Sie eine neue Applikation, und doppelklicken Sie auf das OLE-Steuerelement, um es dem Formularfenster hinzuzufügen. Einen Moment lang erscheint ein großes weißes Feld auf dem Formular. Anschließend wird das Dialogfeld **Objekt einfügen** angezeigt (siehe Abbildung 16.1). Andere Steuerelemente, wie beispielsweise das Standarddialog-Steuerelement, erzeugen erst dann Dialogfelder, wenn Sie auf die Eigenschaft **Benutzerdefiniert** klicken. Das OLE-Steuerelement ist das einzige Steuerelement, das ein Dialogfeld anzeigt, sobald Sie es auf dem Formular angelegt haben.

	Das Listenfeld Objektyp enthält die OLE-konformen Applikationen, aus denen Sie auswählen können. Die Liste variiert abhängig von der im System installierten Software. Eine Applikation aktualisiert Ihre Windows- Registrierungsdatenbank mit ihren OLE-Objekten, sobald Sie sie installiert haben. Eine Windows-Applikation kann zwei, drei oder mehr solcher Einträge erzeugen. Wenn Sie beispielsweise PowerPoint 97 installiert haben, finden Sie zwei PowerPoint-Steuerelemente in der Liste.

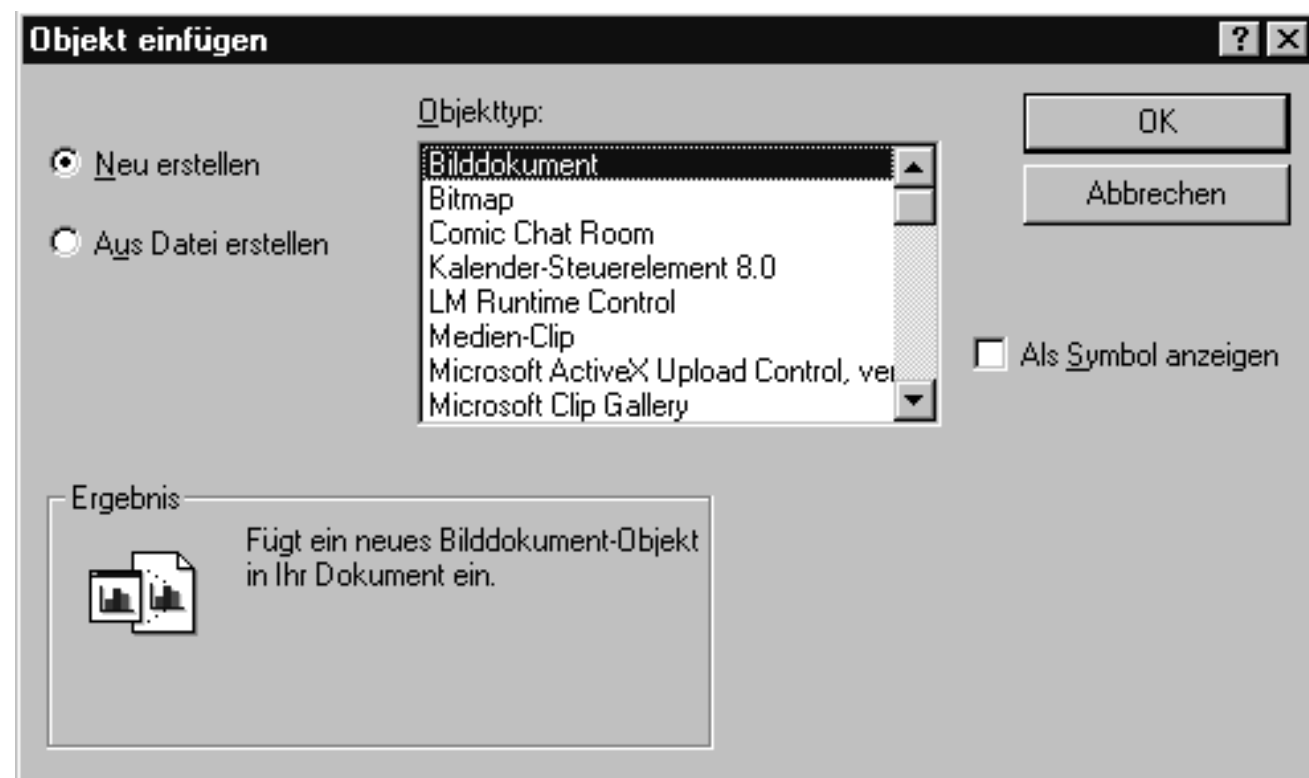


Abbildung 16.1:
Das Dialogfeld Objekt einfügen enthält alle auf Ihrem System registrierten OLE-Applikationen.

Die Option **Neu erstellen** ermöglicht Ihnen, eine Applikation zu spezifizieren, die die Objekte erzeugt, welche Sie einbetten möchten. Wenn dagegen das Datenobjekt einer Applikation bereits existiert, wählen Sie die Option **Aus Datei erstellen**, um das Dialogfeld **Objekt einfügen** anzuzeigen (siehe Abbildung 16.2). Im Dialogfeld **Objekt einfügen** betten Sie das existierende Objekt ein, oder Sie

klicken auf die Option **Link**, um einen Zeiger auf das Objekt anzulegen.

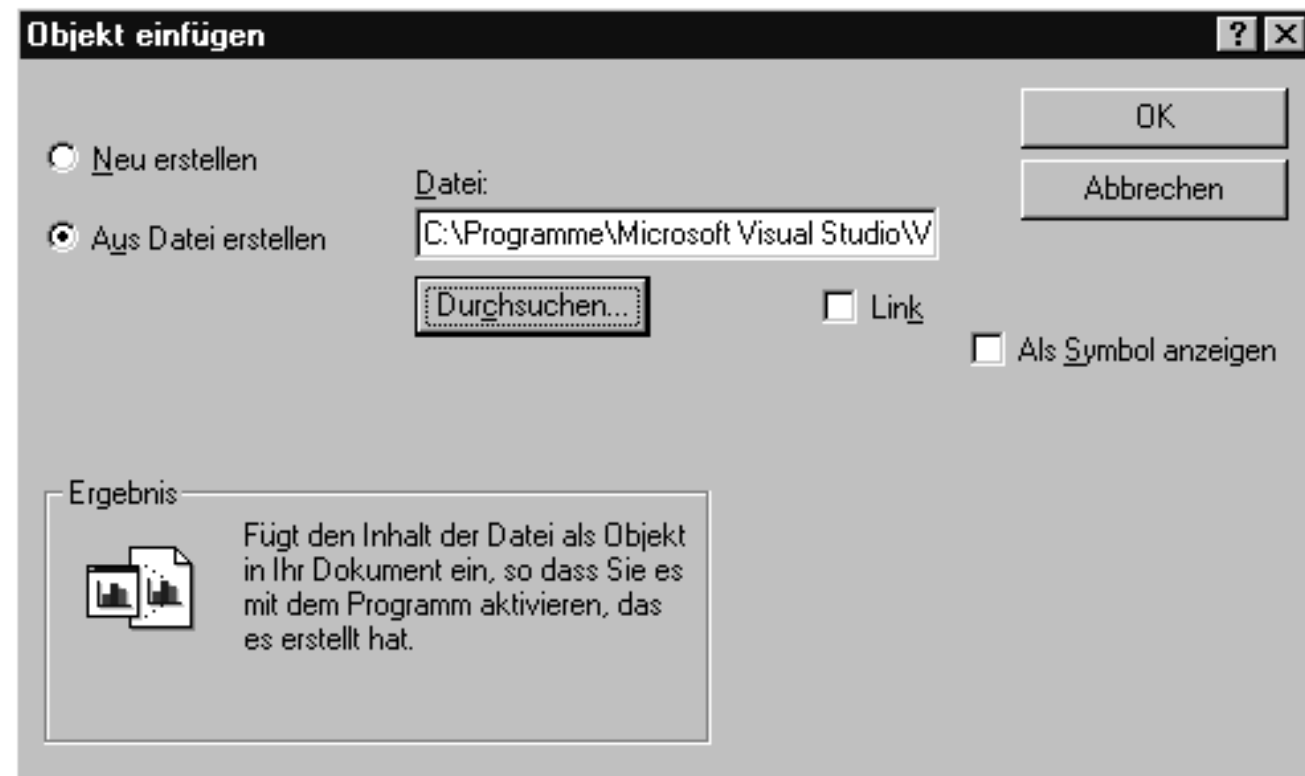


Abbildung 16.2:
Ein existierendes Objekt kann in Ihr OLE-Steuerelement eingebettet oder damit verknüpft werden.

Angenommen, Sie möchten Ihren Benutzern erlauben, Kommentare innerhalb einer Applikation abzulegen. Diese Kommentare könnten sich auf einen Kundendatensatz beziehen. Sie haben verschiedene Möglichkeiten:

- Der Benutzer gibt die Kommentare in ein einfaches Textfeld ein.
- Sie verbessern das Textfeldkonzept, indem Sie eine Routine bereitstellen, die Tastencodes erkennt und das einfache Textfeld in eine vollfunktionale Textverarbeitung umwandelt, Text formatiert und Ränder und Spalten unterstützt. (Es würde relativ lange dauern, das zu entwickeln.)
- Sie betten ein OLE-basiertes WordPad-Dokument-Objekt ein.

Die zuletzt genannte Option hört sich am besten an, weil Sie Ihren Benutzern damit die Leistungsfähigkeit einer Textverarbeitung bereitstellen, ohne Code schreiben zu müssen, der die entsprechenden Befehle verarbeitet: WordPad wurde bereits entwickelt. Nicht jeder hat Microsoft Word auf seinem System, aber WordPad ist in allen Windows-Versionen enthalten (beginnend mit Windows 95). Ihre Benutzer haben also jedenfalls Zugriff auf WordPad, was ihnen ermöglicht, die In-Place-Aktivierung für das Objekt zu nutzen.

Um ein WordPad-Dokument einzubetten, gehen Sie wie folgt vor:

1. Nachdem Sie das OLE-Steuerelement auf dem Formular angelegt haben, wählen Sie die Option **Neu erstellen** aus (das ist der Standard).
2. Blättern Sie in der Liste **Objekttyp** nach unten, bis Sie den Eintrag **WordPad- Dokument** sehen.
3. Doppelklicken Sie auf die Option **WordPad-Objekt**, um ein WordPad-Dokument-Objekt in Ihre Applikation einzubetten. Ein Teil des WordPad-Lineals wird über dem OLE-Steuerelement angezeigt

(siehe Abbildung 16.3).

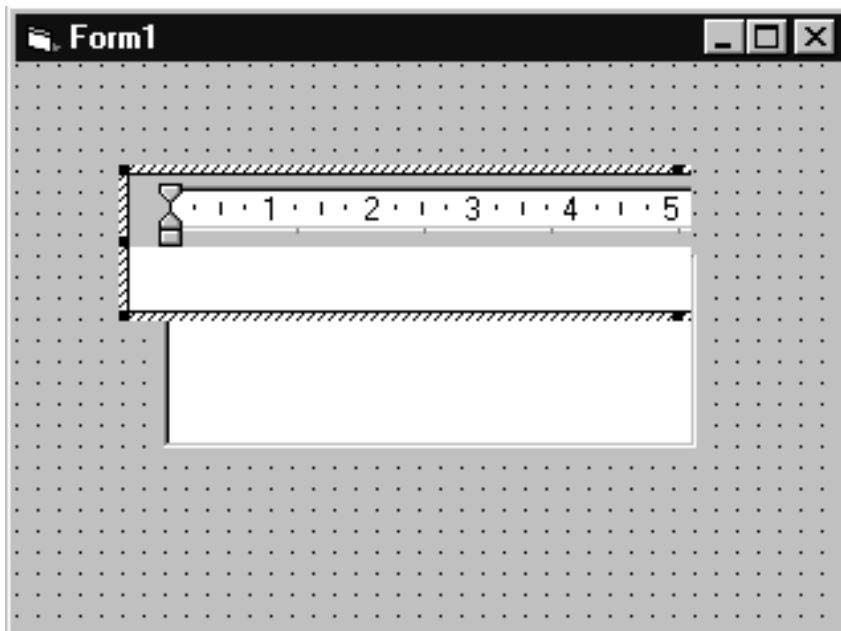


Abbildung 16.3:
Das OLE-Steuerelement enthält jetzt ein WordPad-Objekt.

4. Setzen Sie die `SizeMode`-Eigenschaft des OLE-Steuerelements auf **1 - Strecken**, damit sich das WordPad-Objekt der Höhe und der Breite des OLE-Steuerelements anpaßt.

5. Setzen Sie `Width` und `Height` für das Formular auf 6945 bzw. 5670.

6. Setzen Sie die folgenden Eigenschaften für das OLE-Steuerelement:

`Height`: 3375

`Left`: 840

`Top`: 1080

`Width`: 5055

Sie könnten diesem Projekt auch einige Extras hinzufügen und beispielsweise einen besseren Namen für das Formular verwenden, aber darum brauchen Sie sich jetzt nicht zu kümmern.

7. Führen Sie die Applikation aus. Jetzt scheint nichts Besonderes zu passieren, außer daß der Umriß des OLE-Steuerelements in der Formularmitte erscheint.

8. Doppelklicken Sie auf das OLE-Steuerelement. Voilà! Wie in Abbildung 16.4 gezeigt, passiert eine ganze Menge, wenn Sie auf das Steuerelement doppelklicken. Ein Menü erscheint (aufgrund der In-Place-Aktivierung des eingebetteten WordPad-Dokuments) ebenso wie ein Lineal.



Es gibt mehrere andere Methoden, das OLE-Objekt zu aktivieren. Beispielsweise könnte die OLE-Aktivierung einfach erfolgen, sobald das OLE-Objekt den Fokus erhält. Die `AutoActivate`-Eigenschaft steuert, wie das Objekt aktiviert wird. Der Standardwert für `AutoActivate` ist **2 - Doppelklick**.

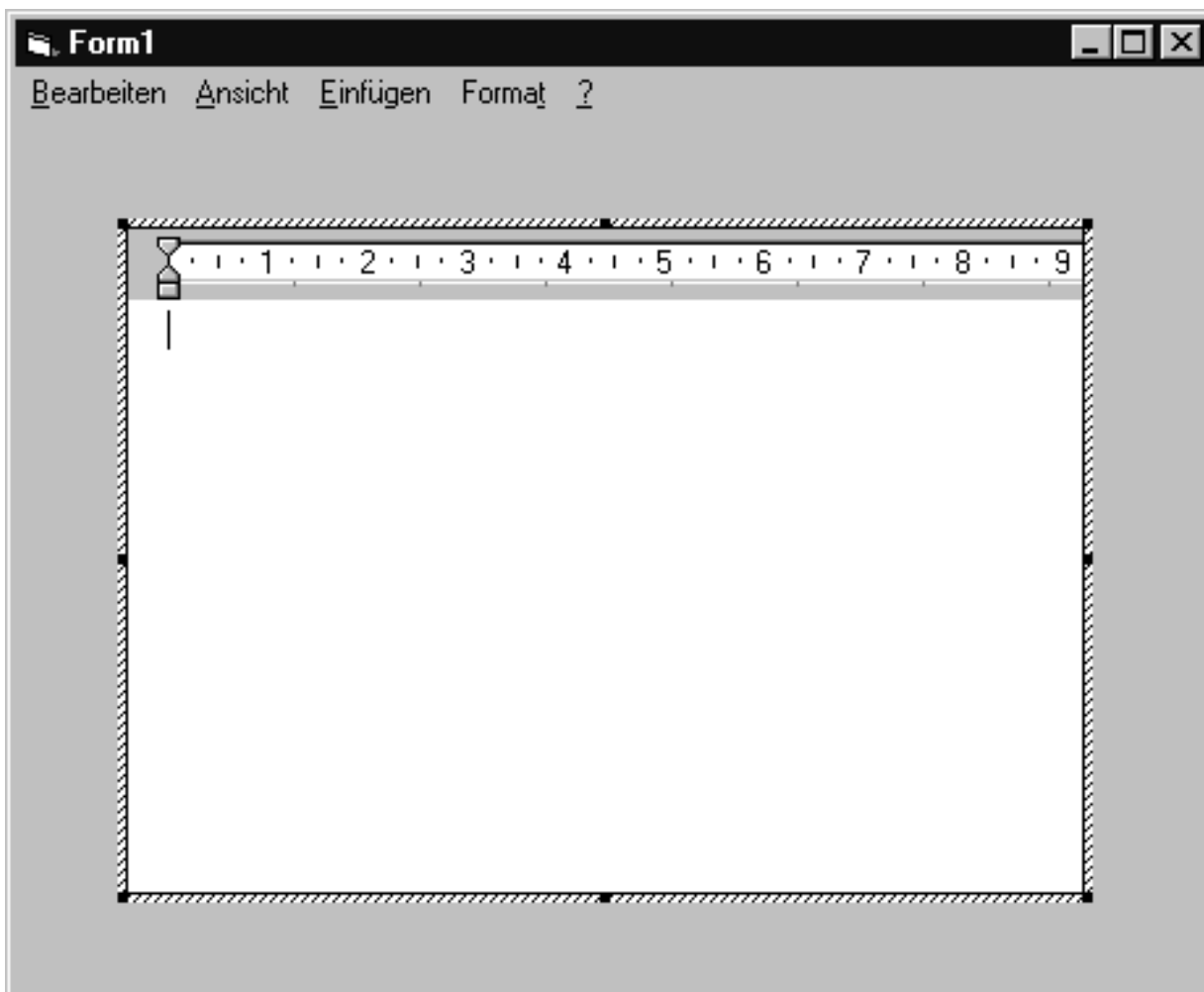


Abbildung 16.4:
Damit haben Sie eine Textverarbeitung mitten in Ihrer Visual-Basic-Applikation angelegt!

1. Geben Sie irgend etwas ein, und formatieren Sie Ihre Eingaben mit Hilfe des Menüs. In Ihrer Visual-Basic-Applikation stehen alle Optionen von WordPad zur Verfügung.
2. Schließen Sie Ihre Applikation.



Ihre Eingaben in WordPad werden nicht gespeichert! Sobald Sie Ihre Visual-Basic-Applikation schließen, verschwindet das WordPad-Dokument und wird nie gesichert. Weder Visual Basic noch WordPad weisen Sie darauf hin, daß das Dokument nicht gesichert wurde! Das Speichern des Objekts ist Ihre Aufgabe als Programmierer. Beachten Sie, daß die In-Place-Menüs von WordPad nicht den Menüeintrag **Datei** enthalten.

Objektinhalt speichern

Microsoft empfiehlt Ihnen, abgeänderte eingebettete Objekte mit einer der verschiedenen Methoden zum Speichern von Dateien zu sichern. Ihre Applikation ist dafür verantwortlich, das eingebettete Objekt zu verwalten; die Quellapplikation ist also nicht dafür zuständig, den Objektinhalt innerhalb Ihrer Applikation zu speichern.

Zum Speichern der Daten in Ihrem OLE-Containersteuerelement verwenden Sie die Methode `SaveToFile`. Angenommen, Sie möchten eine Schaltfläche bereitstellen, die ein Objekt speichert. Listing 16.1 zeigt Beispielcode, der die Änderungen in einer speziellen Binärdatei ablegt. Eine

Binärdatei weist mehr Einschränkungen auf als die sequentiellen Dateien und die Dateien mit wahlfreiem Zugriff, die Sie in Kapitel 12 kennengelernt haben. Nichtsdestotrotz bieten Binärdateien eine schnelle und effiziente Möglichkeit, Objekte zu speichern, falls Sie diese in derselben Reihenfolge wieder lesen, wie Sie sie abgelegt haben.

Listing 16.1: Speichern des OLE-Containersteuerelement-Objekts auf der Platte.

```
1: Dim intFileNum as Integer
2:
3: ' Nächste freie Dateinummer ermitteln
4: intFileNum = FreeFile
5:
6: ' Ausgabedatei öffnen
7: Open "TEST.OLE" For Binary As #intFileNum
8:
9: ' Datei speichern
10: oleObj1.SaveToFile intFileNum
11:
12: ' Datei schließen
13: Close
```

Wenn Ihre Applikation zum nächsten Mal aufgerufen wird, liest sie die neuesten Werte des Objekts (aus der vorherigen Ausführung) in das OLE-Containersteuerelement ein. Dazu wird die in Listing 16.2 gezeigte `ReadFromFile`-Methode verwendet.

Listing 16.2: Der in der vorherigen Programmausführung gespeicherte Objektkinhalt wird in das OLE-Containersteuerelement eingelesen.

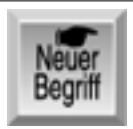
```
1: Dim intFileNum as Integer
2:
3: ' Nächste freie Dateinummer ermitteln
4: intFileNum = FreeFile
5:
6: ' Eingabedatei öffnen
7: Open "TEST.OLE" For Binary As #intFileNum
8:
9: ' Datei in Objekt einlesen
10: oleObj1.ReadFromFile intFileNum
11:
12: ' Datei schließen
13: Close
```

Legen Sie den Code zum Lesen und Speichern in Ereignisprozeduren ab, die mit Schaltflächen oder Menüeinträgen verknüpft sind. Ihre Applikation - nicht die OLE- Applikation - ist für das Speichern und Laden der Daten verantwortlich. Die OLE-Applikation übernimmt jedoch alle objektbezogenen Aufgaben.

Die Arbeit mit Objekten

Vielleicht haben Sie schon einmal den Begriff *objektorientierte Programmierung* gehört, auch als OOP bezeichnet? Visual Basic ist keine eigentliche OOP-Sprache, aber es unterstützt Objekte auf vielfältige

Weise. Nach diesem Kapitel werden Sie besser verstanden haben, was Objekte in Visual Basic sind und wie sie sich in bezug auf Arrays und Auflistungen verhalten.



Objektorientierte Programmierung oder *OOP* ist die Programmierung mit Datenelementen, die Objekte mit ihren Methoden und Eigenschaften repräsentieren, ganz ähnlich den Visual-Basic-Objekten. Echtes OOP verwendet Objekte, die von anderen Objekten erben können. Visual Basic unterstützt das OOP-Modell zwar nicht exakt und vollständig, im praktischen Einsatz scheint es jedoch die eigentlichen Ziele von OOP besser zu erfüllen, als jede echte OOP-Sprache das bisher getan hat.

Die folgenden Abschnitte beschreiben Nicht-OLE-Objekte. Sie werden Ihre Programmierfähigkeiten so ausbauen, daß Sie Steuerelementfelder und Objektauflistungen nutzen können.



Ein Steuerelementfeld ist ein Array mit Steuerelementen, auf die Sie über einen Index zugreifen.

Programmierung mit Objekten

In Kapitel 8 haben Sie erfahren, wie man mit der Anweisung `If TypeOf` prüft, welches Format ein Objekt hat. Dieses Kapitel erklärt genauer, was `If TypeOf` eigentlich macht - es stellt die *Klasse* eines Objekts fest. Wenn Sie noch nie mit Objekten und Klassen zu tun hatten (diese Begriffe werden in anderen Programmiersprachen verwendet, insbesondere in objektorientierten Sprachen), dann werden Sie heute alles lernen, was Sie brauchen.

Ein Objekt kann in Visual Basic fast alles sein. Sie haben bereits mit Objekten gearbeitet, wie beispielsweise mit Steuerelementen, Formularen oder dem `Printer`-Objekt. Außerdem haben Sie in Kapitel 13 einer Applikation den `Object`-Datentyp übergeben. Sie wissen bereits, daß *Steuerelement-Objekte* Eigenschaften, Methoden und Ereignisse umfassen. Die Steuerelemente enthalten Code und Datenelemente. Man könnte sagen, ein Objekt - wie beispielsweise eine Schaltfläche - verhält sich wie ein Paket, das Ihnen von den Visual-Basic-Entwicklern übergeben wird. Sie brauchen keinen Code zu schreiben, der eine Schaltfläche auslöst, keine Variablen deklarieren, die beschreiben, wie eine Schaltfläche aussieht oder welchen Titel sie hat oder welche Aufgaben sie erledigt. Die gekapselten Schaltflächenmethoden nehmen Ihnen all diese Arbeit ab.



Objekte sind *gekapselt*. Ähnlich einer Kapsel, die Medizin oder Astronauten enthält, kapselt ein Objekt Methoden, Ereignisse und Eigenschaften. Diese *Kapselung* ermöglicht Ihnen, die Objekte aus einer höheren Perspektive zu betrachten, als wenn Sie den ganzen Code für ihre Realisierung selbst schreiben müssen. Eine Klasse definiert das Verhalten und das Erscheinungsbild der darin enthaltenen Objekte.

Die Objektklasse

Objekte ermöglichen nicht nur eine Kapselung, sondern stellen auch einen Teil einer Objekthierarchie dar, der sogenannten *Objektklasse*. Der Vorteil einer Klasse ist, daß alle Objekte in dieser Klasse dieselben Eigenschaften aufweisen. (Ein Objekt wird als *Instanz* einer Klasse bezeichnet.)



Es ist möglich, eigene Objekte anzulegen. Wenn Sie sie als Teil einer existierenden Klasse erzeugen, *erben* Ihre Objekte automatisch viele Eigenschaften, Methoden und Ereignisse von dieser Klasse.

Bei der Auswertung eines Objekts mit `If TypeOf` gibt Visual Basic die Objektklasse zurück. Die folgende Codezeile gibt `True` oder `False` zurück, abhängig davon, ob das Objekt `myObj` Teil der Klasse `CommandButton` ist:

```
If TypeOf myObj Is CommandButton ' Klasse auswerten
```



Visual Basic unterstützt die Funktion `TypeInfo()`, die den Klassennamen zurückgibt. Beispielsweise könnte `TypeInfo(myObj)` die Klasse `CommandButton` oder `Form` zurückgeben.

Klassen machen die Programmierung mit Objekten flexibler, als es ohne Klassenstruktur möglich wäre. Beispielsweise ermöglicht Ihnen die Anweisung `With...End With`, einem Objekt ganz einfach mehrere Eigenschaften zuzuweisen. Beachten Sie die Redundanz im folgenden Code:

```
chkMaster.Caption = "Primäre Quelle"  
chkMaster.Alignment = vbLeftJustify  
chkMaster.Enabled = True  
chkMaster.Font.Bold = False  
chkMaster.Left = 1000  
chkMaster.RightToLeft = False  
chkMaster.Top = 400
```

Wenn Sie ein Objekt in einen `With...End With`-Block einschließen, können Sie auf die Wiederholung des Objektnamens verzichten. Der folgende Code erledigt dasselbe wie der oben gezeigte:

```
With chkMaster  
.Caption = "Primäre Quelle"  
.Alignment = vbLeftJustify  
.Enabled = True  
.Font.Bold = False  
.Left = 1000  
.RightToLeft = False  
.Top = 400  
End With
```



Wenn Sie `With...End With` verwenden, um zwei oder drei Eigenschaften zu setzen, ist dafür mehr Tippaufwand erforderlich als mit einfachen Zuweisungen. Wenn Sie dagegen mehr als drei Eigenschaften zuweisen, ist die `With`-Klausel eine sinnvolle Alternative, weil dadurch weniger Schreibarbeit anfällt und die Wartung einfacher wird, falls Sie später weitere Zuweisungen vornehmen möchten.



Wenn Sie glauben, daß in späteren Programmversionen zusätzliche Eigenschaften gebraucht werden, sollten Sie auch dann die `With`-Klausel verwenden, wenn vorerst nur ein oder zwei Eigenschaften gesetzt werden.

Die Systemobjekte

Anders als Objekte, die Sie deklarieren, sind die `Printer`- und `App`-Objekte, die Sie bereits in früheren Kapiteln verwendet haben, *Systemobjekte*. Es ist zwar nicht möglich, Systemobjekte zu übergeben (sie sind bereits global), aber Sie können sie ähnlich behandelt wie die von Ihnen selbst erzeugten Objekte. Systemobjekte repräsentieren bestimmte Elemente Ihrer Applikation.

Tabelle 16.1 beschreibt die Systemobjekte und listet einige wichtige Methoden auf, die Sie darauf anwenden können.

Tabelle 16.1: Systemobjekte unterstützen mehrere Methoden

Systemobjekt	Methoden	Beschreibung
App		Die aktuelle Applikation.
	<code>EXENAME</code>	Gibt den Dateinamen der Applikation zurück.
	<code>Path</code>	Gibt den Pfad zur Applikation zurück.
	<code>Title</code>	Gibt den Text der Titelleiste des Startformulars an.
	<code>PrevInstance</code>	Gibt <code>True</code> oder <code>False</code> zurück, womit angezeigt wird, ob noch eine Kopie der Applikation ausgeführt wird.
Clipboard		Die Windows-Zwischenablage.
	<code>Clear</code>	Löscht die Zwischenablage.
	<code>GetData</code>	Gibt die in der Zwischenablage enthaltene Grafik zurück.
	<code>GetFormat</code>	Gibt das Format des Objekts in der Zwischenablage zurück.
	<code>GetText</code>	Gibt den Text aus der Zwischenablage zurück.
	<code>SetData</code>	Kopiert eine Grafik in die Zwischenablage.
	<code>SetText</code>	Kopiert Text in die Zwischenablage.
	<code>SelStart</code>	Wird für Auswahloperationen in der Zwischenablage verwendet.
	<code>SelLength</code>	Wird für Auswahloperationen in der Zwischenablage verwendet.
Debug	<code>SelText</code>	Wird für Auswahloperationen in der Zwischenablage verwendet.
		Das Debug-Fenster
	<code>Print</code>	Kopiert zur Laufzeit Informationen in das Debug-Fenster (das ist nur in Visual-Basic-Programmen möglich, die noch nicht kompiliert wurden und die in der Entwicklungsumgebung ausgeführt werden).
Err		Enthält Informationen über den aktuellen Fehlerstatus einer Applikation

		Die Schlüsseleigenschaft Number enthält einen Fehlercode für den zuletzt verursachten Systemfehler. (Null, falls kein Fehler aufgetreten ist.)
Printer		Der Systemdrucker. In Kapitel 13 haben Sie die Methoden für das Printer -Objekt kennengelernt. Außerdem wurde dort gezeigt, wie diese Methoden den Drucker unterstützen.
Screen		Der Bildschirm des Benutzers.
	FontCount	Gibt die Anzahl der Schriften zurück, die der aktuelle Bildschirm unterstützt.
	Fonts	Enthält eine Liste aller Schriftnamen, die für den Bildschirm möglich sind.
	Height	Gibt die Höhe des Bildschirms in Twips zurück.
	MousePointer	Enthält den Umriß des Mauszeigers oder bestimmt seinen Umriß, wenn Sie einen neuen Mauscursor spezifizieren.
	TwipsPerPixelX	Gibt die Anzahl der möglichen horizontalen Twips zurück.
	TwipsPerPixelY	Gibt die Anzahl der möglichen vertikalen Twips zurück.
	Width	Gibt die Bildschirmbreite in Twips zurück.

Sie haben bereits mit vielen dieser Systemobjekte gearbeitet - insbesondere **Printer** und **Screen**. Mit dem **App**-Objekt werden zur Laufzeit Informationen über den Pfad und den Dateinamen des Programms ermittelt. Das **Clipboard**-Objekt bietet einige interessante Funktionen für Ihre Arbeit. Das **Debug**-Objekt ermöglicht Ihnen, beim Debuggen mit Ihrem Programm zusammenzuarbeiten, um mögliche Fehler zu eliminieren.

Abhängig von den Anforderungen Ihrer Applikation ist das **Clipboard**-Objekt relativ einfach zu programmieren. Das **Clipboard**-Objekt repräsentiert dieselbe Zwischenablage wie die von Windows. Außerdem kann Ihre Applikation Informationen aus der Windows-Zwischenablage in das **Clipboard**-Objekt einfügen.

Mit Hilfe des **Clipboard**-Objekts und seiner Eigenschaften markieren Sie Text in Ihrem Programm und stellen fest, welchen Text Ihr Benutzer markiert hat. Die **SelStart**-Eigenschaft beispielsweise markiert die Anfangsposition des Auswahlcursors im Textfeld (oder in welchem Steuerelement Sie sich befinden). Der Wert 0 platziert den Cursor vor dem ersten Zeichen. **SelLength** legt fest, wie viele Zeichen selektiert werden. Wenn Sie Text mit Hilfe der Werte **SelStart** und **SelLength** selektieren, wird dieser Text in das **Clipboard**-Objekt kopiert, wenn der Benutzer (Strg)+(C) (Kopieren) oder (Strg)+(X) (Ausschneiden) drückt. **SelText** ist ein String mit dem selektierten Text, der durch **SelStart** und **SelLength** bestimmt wird.

Wenn Sie in einem Textfeld Text selektiert haben (oder den Benutzer aufgefordert haben, Text zu selektieren), erscheint dieser Text im **SelText**-String. Der folgende Code löscht den aktuellen Inhalt des **Clipboard**-Objekts (das **Clipboard**-Objekt kann jeweils nur einen Wert enthalten) und sendet den selektierten Text an das Zwischenablage- Objekt:

```
Clipboard.Clear ' Aktuellen Inhalt der Zwischenablage löschen
Clipboard.SetText txtName.SelText ' Text kopieren
```

Mit `GetText()` kopieren Sie den Text aus der Zwischenablage in eine Variable:

```
strInfo = Clipboard.GetText()
```

Wenn Sie den in einem Steuerelement selektierten Text durch den im `Clipboard`-Objekt enthaltenen Text ersetzen möchten, gehen Sie wie folgt vor:

```
txtName.SelText = Clipboard.GetText()
```

Die Methode `GetText()` verwendet manchmal Argumente. Aber auch wenn keine Argumente angegeben sind, müssen die Klammern geschrieben werden. Wenn Sie die Zwischenablage für Text einsetzen, brauchen Sie keine Argumente bereitzustellen.

Arrays mit Objekten und Steuerelementen

Es ist möglich, Arrays (Felder) mit Objekten anzulegen. Beispielsweise können Sie ein Array mit Schaltflächen oder Formularen anlegen. Darüber hinaus müssen diese Objekte noch nicht einmal existieren. Sie brauchen also nicht alle Formulare zur Entwurfszeit deklarieren, weil Sie zur Laufzeit ein Array mit Formularen anlegen können.

Die `Forms`- und `Printers`-Auflistungen kennen Sie bereits. Visual Basic unterstützt außerdem die `Controls`-Auflistung, mit deren Hilfe Sie alle Steuerelemente durchlaufen können, als handelte es sich dabei um Arrayvariablen. Der folgende Code beispielsweise verbirgt alle Steuerelemente auf einem Formular:

```
For intCtr = 0 to Controls.Count - 1  
Controls(intCtr).Visible = False  
Next intCtr
```

Für eine Applikation mit mehreren Formularen können Sie alle Steuerelemente auf allen Formularen verbergen, indem Sie eine verschachtelte Schleife einsetzen (beachten Sie, daß durch `For Each` die Anweisung `Count - 1` wegfallen kann):

```
1: Dim frmAForm As Form  
2: Dim ctlAControl As Control  
3: For Each frmAForm In Forms ' Alle Formulare durchlaufen  
4: For Each ctlAControl In frmAForm.Controls  
5: ctlAControl.Visible = False  
6: Next ctlAControl  
7: Next frmAForm
```



In einer `Controls`-Auflistung wird ein Menü als Steuerelement betrachtet. Häufig möchte man die Menü-Steuerelemente in so einer Schleife nicht berücksichtigen, deshalb sollte mit der `GetTypeOf()`-Funktion ermittelt werden, ob es sich bei einem Steuerelement um ein `Menu`-Objekt handelt, bevor es verborgen dargestellt wird.

Die `Controls`-Auflistung enthält alle Steuerelemente Ihres aktuellen Formulars; Sie können jedoch ein Steuerelementfeld deklarieren, das nur einen bestimmten Typ von Steuerelementen aufnimmt. Ein Steuerelementfeld wird wie folgt deklariert:

```
Dim ctlManyLabels(1 To 4) As Label
```

Der nächste Abschnitt wird diese Auflistungen noch genauer beschreiben. Auflistungen verhalten sich ein bißchen wie Arrays, weil sie auf die einzelnen Elemente ebenfalls über einen Index zugreifen.

Beispielsweise könnten Sie Arrays mit Objekten wie etwa Formularen und Steuerelementen zugreifen. Statt die Objekte zur Entwurfszeit zu erzeugen, legen Sie sie in einem Array an, nämlich wie folgt (beachten Sie das Schlüsselwort `New`):

```
Dim frmArray(1 To 10) As New frmFirstForm
```

Diese `Dim`-Anweisung setzt voraus, daß es das Formular `frmFirstForm` bereits gibt. Nach der Deklaration gibt es 10 neue Formulare, mit den Indexwerten `frmArray(1)` bis `frmArray(10)`. Der folgende Code kann die Formulareigenschaften dieser Formulare im Array ändern, um sie vom Basisformular zu unterscheiden, `frmFirstForm`.



Keines dieser Formulare wird angezeigt, bevor Sie nicht die `Show`-Methode ausgeführt haben.

Angenommen, Sie möchten die Steuerelemente eines Formulars verkleinern, wenn der Benutzer ein maximiertes Formular verkleinert. Dazu verwenden Sie die `Controls`-Auflistung:

```
1: Private Sub Form_Resize ()
2: ' Schriftgröße aller Steuerelemente verkleinern
3: Dim intCtr As Integer
4: For intCtr = 0 to Controls.Count - 1
5: Controls(intCtr).FontSize = Controls(intCtr).FontSize * .75
6: Next intCtr
7: End Sub
```

Die Schriftgröße wird damit um 25 Prozent verkleinert.

Die Visual-Basic-Programmierer verwenden in der Regel keine Steuerelementfelder, wenn es für dasselbe Objekt eine Auflistung gibt (Visual Basic stellt eine vordefinierte `Forms`-Auflistung bereit). Wenn Sie jedoch Steuerelementfelder verwenden möchten, müssen Sie Speicher deklarieren, der den Inhalt des Arrays aufnimmt, und die Arrays initialisieren.

Visual Basic unterstützt eine Technik für Steuerelementfelder, die Sie häufig einsetzen werden. Wenn Sie ein Steuerelement kopieren und auf dem Formular eine solche Kopie einfügen, zeigt Visual Basic das in Abbildung 16.5 gezeigte Meldungsfeld an.

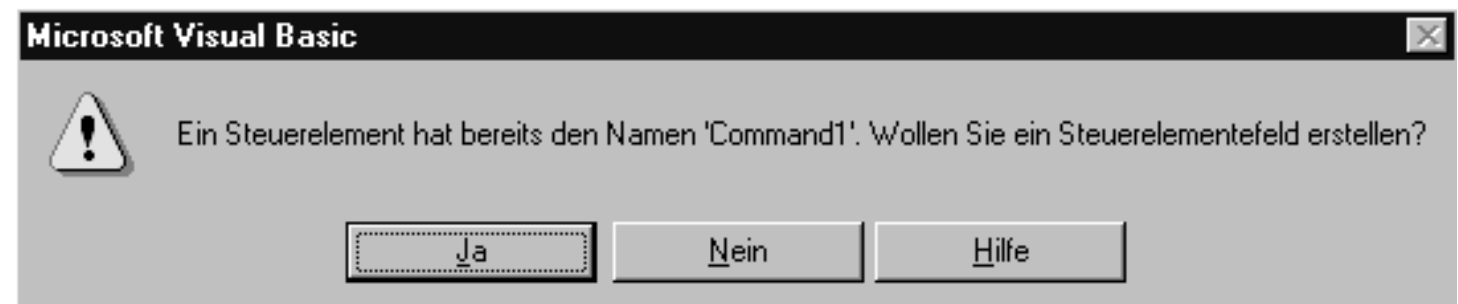


Abbildung 16.5:
Visual Basic legt für Sie ein Steuerelementfeld an.

Sie fragen sich vielleicht, wieso Sie ein Steuerelement kopieren und einfügen sollten. Das kann sehr praktisch sein, wenn Sie mehrere Schaltflächen verwenden möchten, die alle dasselbe Format haben, beispielsweise die Schriftgröße und die Ausrichtung. Sie legen einfach ein solches Steuerelement an, setzen alle seine Eigenschaften und kopieren dieses Steuerelement in die Zwischenablage. Der Inhalt der Zwischenablage wird in das Formular kopiert, so daß Sie so viele dieser Steuerelemente erhalten,

wie Sie brauchen.

Sobald Sie das kopierte Steuerelement eingefügt haben, zeigt Visual Basic das oben beschriebene Meldungsfeld an. Wenn Sie mit Ja antworten, legt Visual Basic automatisch ein Steuerelementefeld an, dessen Name gleich dem Namen des ersten Steuerelements ist. Handelt es sich beim ersten Steuerelement beispielsweise um die Schaltfläche `Command1`, heißt das Feld `Command1`, und die Elemente werden beginnend mit `Command1(0)` abgelegt. Der Index wird bei jedem Einfügen dieses Steuerelements inkrementiert.

Ihr Code kann die Elemente dieses Arrays von `Command1(0)` bis `Command1(n)` durchlaufen, wobei `n` die Gesamtzahl der `Command1`-Steuerelemente auf dem Formular darstellt, und die gewünschten Eigenschaften dafür setzen.

Auflistungen

Auflistungen spielen eine wichtige Rolle in der Visual-Basic-Programmierung, wie Sie bereits in früheren Kapiteln ebenso wie in den ersten Abschnitten dieses Kapitels erfahren haben. Die Auflistungen stehen immer zur Verfügung, und Visual Basic aktualisiert sie automatisch. Wenn Sie beispielsweise mit der Deklaration `New Form` zur Laufzeit ein neues Formular anlegen, aktualisiert Visual Basic die `Count`-Eigenschaft der `Forms`-Auflistung entsprechend.

Zweifellos sind die vordefinierten Auflistungen sehr praktisch. Warum sollten Sie also nicht auch eigene Auflistungen anlegen? Das ist in Visual Basic möglich. Wenn Sie jedoch eine Auflistung erzeugen, müssen Sie diese selbst verwalten - und das ist mühsamer als die Verwaltung vordefinierter Auflistungen.

Wie Sie in den vorigen Abschnitten erfahren haben, gehören alle Objekte zu einer Klasse. Wenn Sie etwas über eine Klasse wissen, wissen Sie dasselbe über alle Objekte innerhalb dieser Klasse. Ist ein Steuerelement beispielsweise Element der Klasse `CommandButton`, wissen Sie, daß das Steuerelement das `Click`-Ereignis unterstützt, weil alle Elemente der Klasse `CommandButton` das `Click`-Ereignis unterstützen.

Ihre eigenen Auflistungen müssen Objekte der Klasse `Collection` sein. Auflistungen werden auf Modulebene mit den Schlüsselwörtern `Private` oder `Public` definiert, abhängig davon, welche Prozeduren Zugriff darauf benötigen. Die folgende Anweisung deklariert die Auflistung `colMyCol`:
`Private colMyCol As New Collection`

Eine Auflistung kann mit einem leeren Bücherschrank verglichen werden. Es ist möglich, Objekte einzufügen (beispielsweise Bücher), sie entfernen, sie zu zählen usw. Natürlich kann ein Bücherschrank auch andere Dinge enthalten. Eine Auflistung kann immer nur eine Art von Elementen enthalten, aber Sie können mehrere Auflistungen deklarieren, die alle verschiedene Arten von Elementen aufnehmen. Hier die Methoden, die Ihren Auflistungen zur Verfügung stehen:

- `Add` - fügt Ihrer Auflistung ein Element hinzu.
- `Count` - Gibt die Anzahl der Elemente in Ihrer Auflistung zurück.
- `Item` - Dient als Indexzahl für die Elemente Ihrer Auflistung.
- `Remove` - Entfernt ein Element aus Ihrer Auflistung.

Listing 16.3 zeigt, daß Visual Basic `Count` aktualisiert und Elemente in Ihre Auflistungen einfügt. Der Code in Listing 16.3 erzeugt die Auflistung `Cities` und fügt vier Elemente (Städtenamen) ein.

Listing 16.3: Mit `Add` werden Ihrer neuen Auflistung Elemente hinzugefügt.

```

1: Dim Cities As New Collection
2: Dim intCtr As Integer
3:
4: ' Elemente einfügen
5: Cities.Add "Tulsa"
6: Cities.Add "Miami"
7: Cities.Add "New York"
8: Cities.Add "Seattle"
9:
10: ' Anzeige, daß es vier Städte in der Auflistung gibt
11: frmMyForm.Print "Es gibt "; Cities.Count; " Städte:"
12:
13: ' Städtenamen ausgabe
14: For intCtr = 1 To Cities.Count
15: frmMyForm.Print " "; Cities(intCtr)
16: Next

```

Bei der Ausführung dieses Codes erscheint die folgende Ausgabe auf dem Formular:

```

Es gibt 4 Städte:
Tulsa
Miami
New York
Seattle

```

Dieses Kapitel konnte nur kurz auf die Leistungsfähigkeit von Auflistungen hinweisen. Nichtsdestotrotz sollten Sie wissen, daß Sie Elemente in Auflistungen in jeder beliebigen Reihenfolge einfügen und auch daraus entfernen können. Sie wissen, daß die Elemente einer Auflistung durch Indexwerte angesprochen werden, beginnend bei 1. In dem oben gezeigten Beispiel ist `Cities(1)` die erste in der Auflistung `Cities` enthaltene Stadt. Der Indexwert beginnt bei 1 und nicht bei 0 (wie es für Steuerelementfelder gefordert wird).

Mit dem *benannten Argument* (das ist ein Argument, wobei Sie den Argumentnamen gefolgt von dem Zuweisungsoperator `:=` verwenden) `Before` fügen Sie einer Auflistung an einer bestimmten Position Elemente hinzu. Die folgende Zeile fügt am Anfang der `Cities`-Auflistung eine Stadt ein, unabhängig davon, wie viele Städte sich zuvor dort befunden haben:

```
Cities.Add "St. Louis", Before:=1
```

Die `Before`-Position 1 fügt Elemente am Anfang der Auflistung ein. Mit anderen Worten, Visual Basic fügt das neue Element vor dem Element mit dem angegebenen Index in die Auflistung ein. Wenn Sie diese `Add`-Methode am Ende des oben gezeigten Codes einfügen, erhalten Sie die folgende Ausgabe:

```

Es gibt 4 Städte:
St. Louis
Tulsa
Miami
New York
Seattle

```

Wenn Sie diese Codezeile ohne das benannte Argument `Before:=1` verwenden, erscheint St. Louis am Ende der Auflistung.

Mit der **Remove**-Methode entfernen Sie bestimmte Elemente. Dabei werden die verbleibenden Indizes so korrigiert, daß sie immer mit 1 beginnen und fortlaufend weitergezählt werden. Die folgende Anweisung entfernt das zweite Element (Tulsa) aus der Auflistung:

```
Cities.Remove 2
```

Der Objektkatalog

Je mehr Sie über Visual Basic wissen, desto bessere Werkzeuge brauchen Sie. Visual Basic beinhaltet den sogenannten *Objektkatalog*, der Ihnen ermöglicht, Variablen, Steuerelemente und andere Objekte in Ihrer Applikation zu beobachten. Visual-Basic- Programmierer, die den Objektkatalog noch nicht kennen, werden ihn öfter nutzen, als sie vielleicht glauben, weil er die Programmierung unter Visual Basic sehr viel einfacher macht.



Der *Objektkatalog* ermöglicht Ihnen, Objekte innerhalb Ihrer Applikationen zu finden und zu verwalten.

Der Objektkatalog ist eine umfassende Online-Referenz - aber nicht im Sinne der Online-Hilfe. Vielmehr stellt er eine zentrale Position dar, an der Sie nach Objekten und Objektinformationen suchen können. Außerdem ermöglicht er Ihnen, direkt in den Code zu springen, mit dem Sie weiterarbeiten möchten.



Der Objektkatalog beschreibt die *Typbibliotheken* Ihrer Applikation, das sind die Behälter, wo Ihre Klasseninformationen abgelegt sind. Mit Hilfe des Objektkatalogs greifen Sie auf alle Objekteigenschaften, Ereignisse und Methoden für Ihre Applikation zu, auch auf die von Ihnen selbst geschaffenen Objekte.

Das Fenster des Objektkatalogs

Wenn Sie den Objektkatalog aus dem **Ansicht**-Menü auswählen oder in der Symbolleiste auf die Schaltfläche **Objektkatalog** klicken, sehen Sie das Objektkatalogfenster (siehe Abbildung 16.6). Sie müssen Ihr Fenster möglicherweise vergrößern und das Eigenschaftenfenster und die Werkzeugsammlung schließen, um den gesamten Objektkatalog anzeigen zu können.

Tabelle 16.2 beschreibt die Bestandteile des Objektkatalogfensters.

Tabelle 16.2: Das Objektkatalogfenster enthält mehrere Komponenten, mit denen Sie sich vertraut machen sollten.

Komponente	Beschreibung
Listenfeld Projekt/ Bibliotheken	Zeigt die Quelle der Objekte an, die Sie anzeigen möchten. (Im allgemeinen wählen Sie hier die Option <Alle Bibliotheken>, aber Sie könnten beispielsweise auch die Objekte eines bestimmten Projekts ansehen, indem Sie hier den Projektnamen angeben.)
Suchtext	Hier geben Sie ein Objekt, eine Methode oder eine Eigenschaft ein, nach dem bzw. der gesucht werden soll.

Navigationsschaltflächen	Mit diesen Schaltflächen springen Sie in einem zuvor durchlaufenen Pfad vor und zurück.
Klassen	Enthält die Klassennamen für das ausgewählte Projekt bzw. die ausgewählte Bibliothek.
Mitglieder	Enthält die Mitglieder der ausgewählten Klasse.

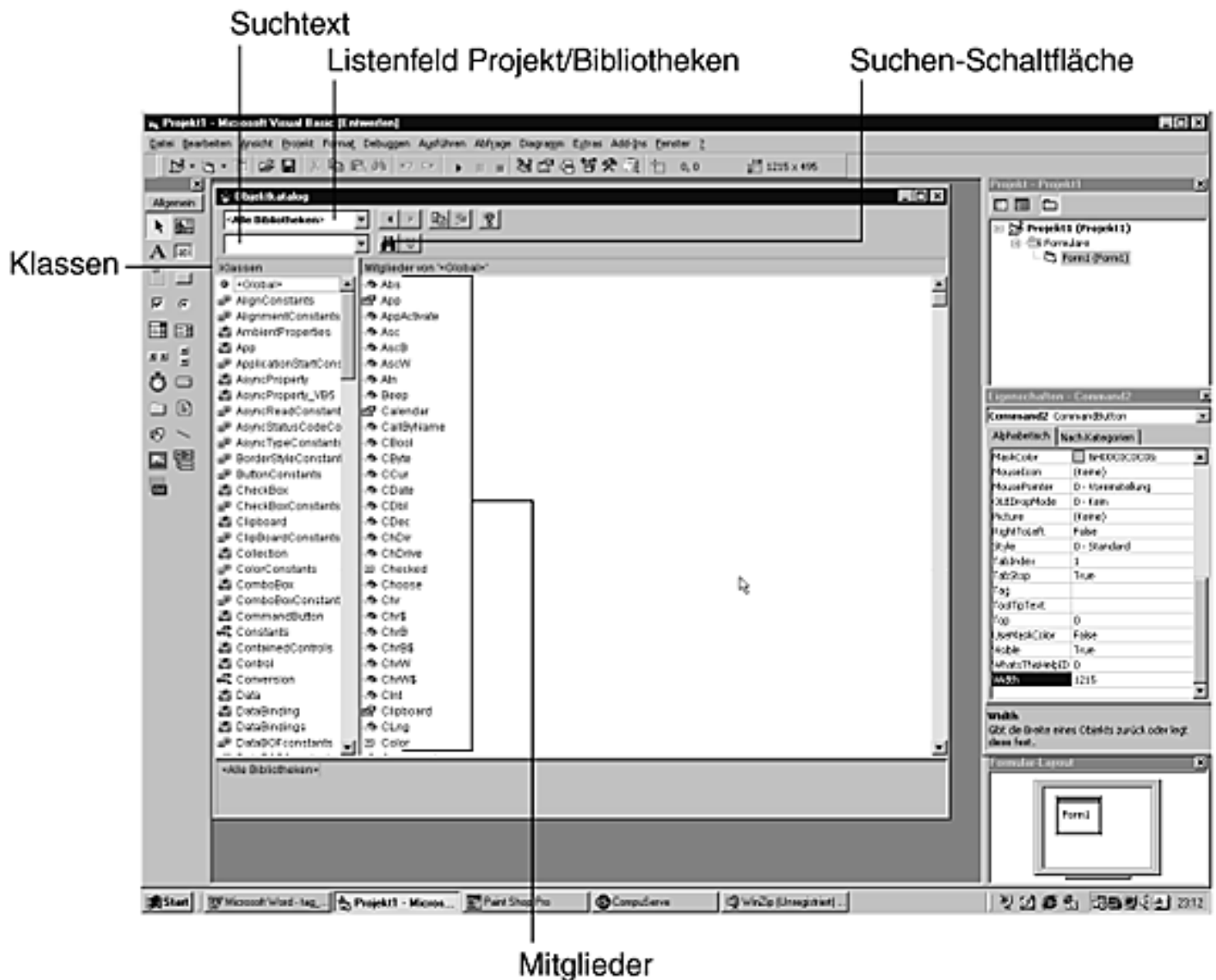


Abbildung 16.6:
Der Objektkatalog zeigt die Objekte Ihrer Applikation an.

Die Arbeit im Objektkatalog

Der Objektkatalog enthält viele Informationen, die auch im Online-Hilfesystem enthalten sind. Er wendet sich jedoch insbesondere an Sie als Visual-Basic-Programmierer und bietet Ihnen genau die Informationen an, die Sie brauchen. Der Eintrag <Global> in der Klassenliste beispielsweise beschreibt alle in Visual Basic eingebauten Funktionen. Wenn Sie Informationen über die Funktion `Left()` suchen, blättern Sie bis zum Eintrag `Left`.



Wie Sie in Kapitel 8 gelernt haben, gibt die Funktion `Left ( )` einen Stringabschnitt von links zurück.

Wenn Sie den Eintrag `Left` selektieren, zeigt Visual Basic unten im Fenster des Objektkatalogs eine Beschreibung dafür an. Der Text erklärt nicht nur die Aufgabe der Funktion, sondern zeigt auch ihr Format an. In der Dropdown-Liste **Mitglieder** erkennen Sie anhand der Icons, um welche Objekte es sich jeweils handelt. Das kleine grüne Icon weist darauf hin, daß es sich bei dem Mitglied um eine Funktion handelt. Auch die Auflistungen (siehe `Forms`-Eintrag) und Konstantennamen werden durch spezielle Icons gekennzeichnet. Blättern Sie in der Liste nach unten, um alle Konstantennamen anzuzeigen, die unterhalb der Funktionen und Auflistungen dargestellt sind.

Wenn Sie mit der rechten Maustaste in eine der Listen klicken und im Popup-Menü den Eintrag **Mitglieder gruppieren** auswählen, gruppiert Visual Basic alle Mitglieder und Klassen nach ihrer jeweiligen Aufgabe. Die Mitglieder werden dann also nicht mehr in alphabetischer Reihenfolge, sondern es werden zuerst alle benannten Konstanten, dann alle Ereignisse usw. angezeigt.



Nachdem Sie im Objektkatalog einen Eintrag selektiert haben, können Sie in der Symbolleiste auf die **Hilfe**-Schaltfläche (mit dem Fragezeichen-Icon) klicken, um Online-Hilfeinformationen über dieses Objekt zu erhalten.

Der Objektkatalog bietet noch detailliertere Informationen. Die Klassenliste beispielsweise enthält mehrere Einträge, die auf Konstantennamen verweisen. Wenn Sie beispielsweise auf den Eintrag `ColorConstants` klicken, erscheinen nur die Farb-Konstantennamen von Visual Basic in der Mitgliederliste (siehe Abbildung 16.7).

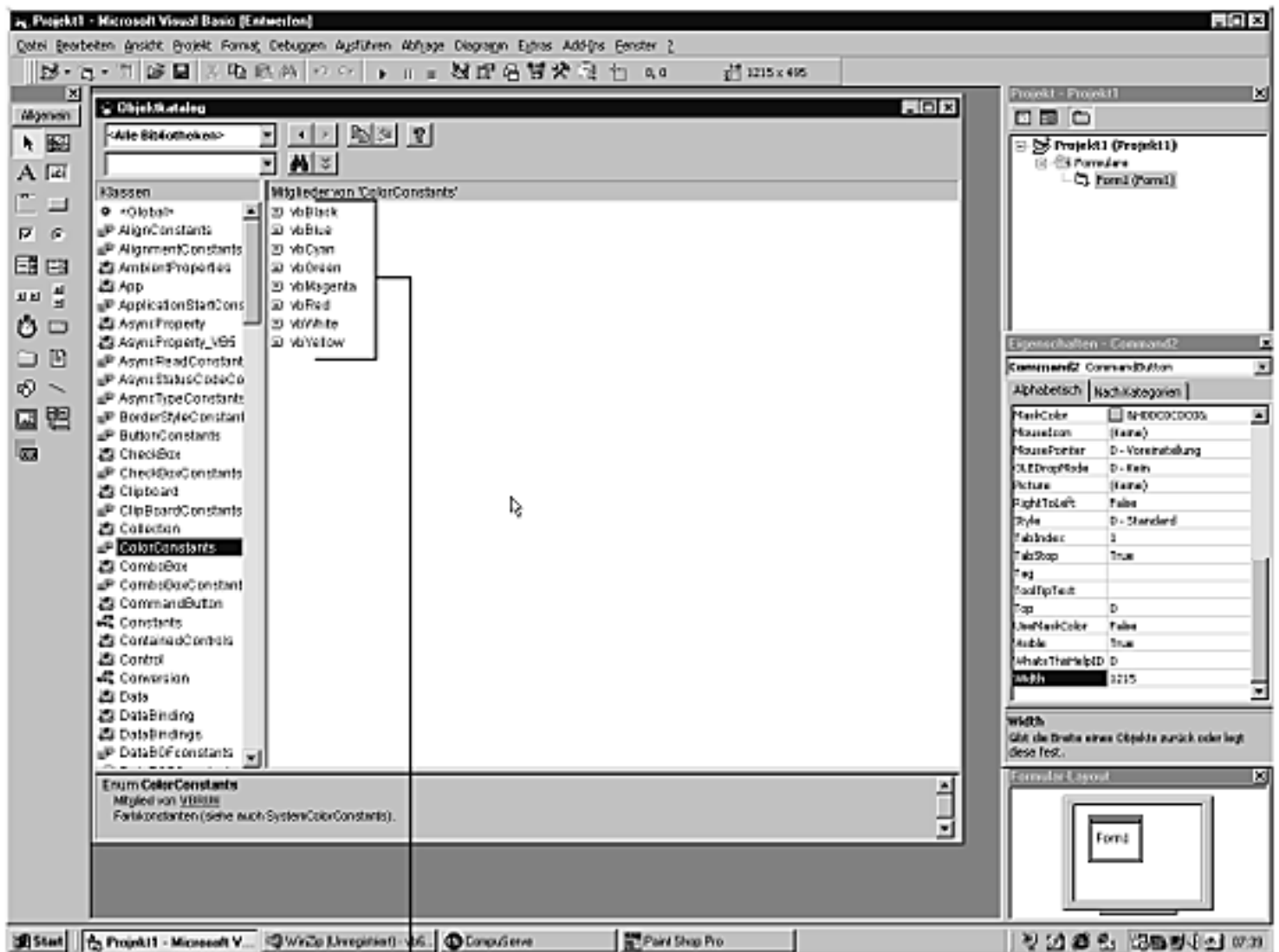


Sie können diese Konstantennamen überall dort im Code verwenden, wo Sie eine Farbe benötigen. Beispielsweise könnten Sie die Hintergrundfarbe für ein Formular wie folgt setzen:

```
frmMyForm.BackColor = vbRed ' Formular rot anzeigen
```

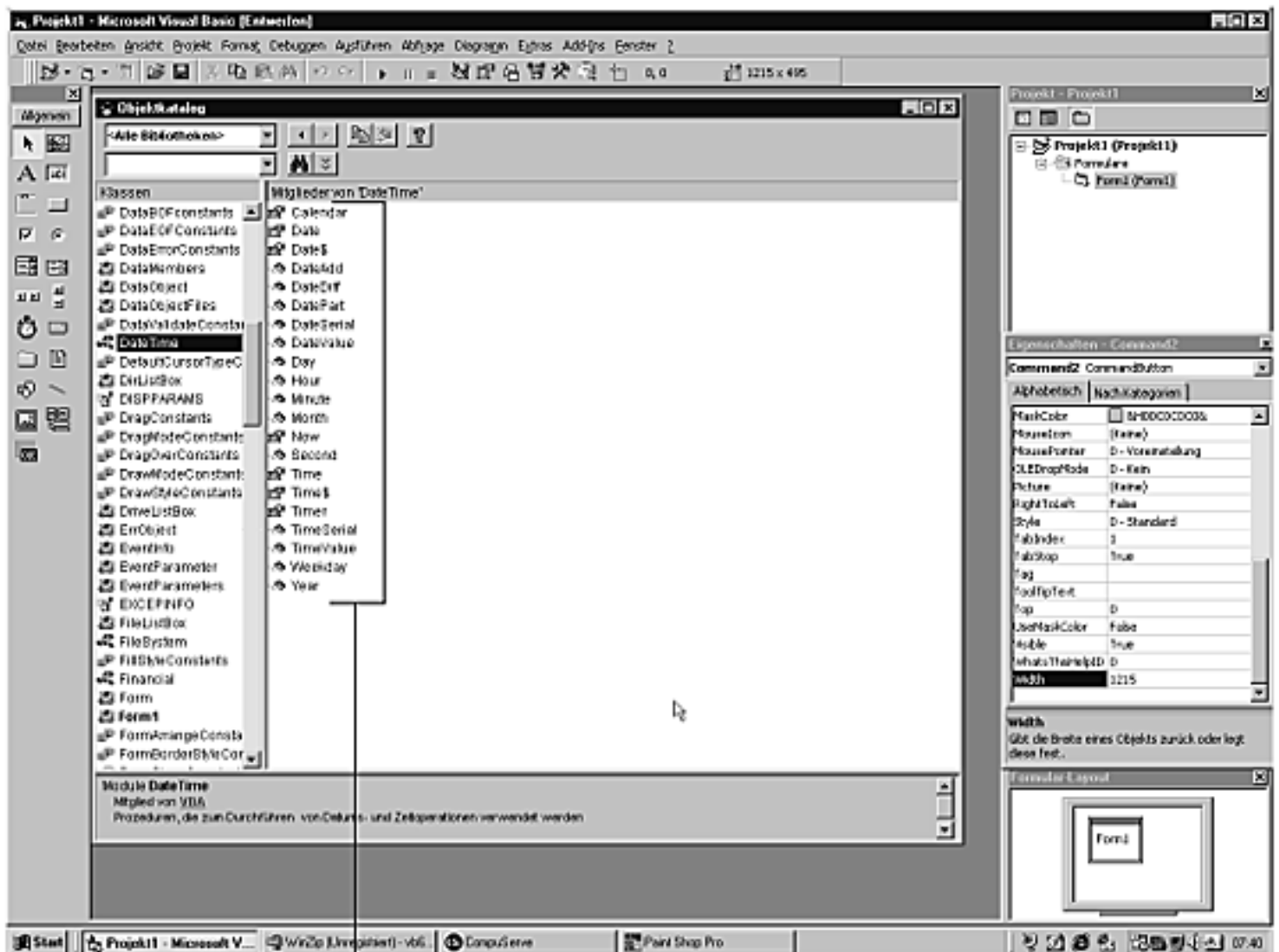
Beachten Sie, daß alle Steuerelemente, die in der Werkzeugsammlung zur Verfügung stehen, auch in der Klassenliste erscheinen. Wenn Sie beispielsweise auf den Eintrag `ComboBox` klicken, zeigt der Objektkatalog alle Informationen für Kombinationsfelder an, unter anderem ihre Eigenschaften, Ereignisse und Methoden. Wenn Sie in der Mitgliederliste auf einen der Einträge klicken, erhalten Sie eine Beschreibung dieser Methode, dieses Ereignisses oder dieser Eigenschaft.

Programmierer nutzen den Objektkatalog für die unterschiedlichsten Zwecke. Merken Sie sich, daß Ihnen der Objektkatalog die Objektinformation strukturiert anzeigt. Neben den Objekten berücksichtigt er auch alle Ihre Programmiergewohnheiten. Wenn Sie beispielsweise Code schreiben, der die eingebauten Datums- und Zeitfunktionen verwendet, klicken Sie in der Klassenliste auf den Eintrag `DateTime`. Wie Abbildung 16.8 zeigt, wird die Mitgliederliste entsprechend aktualisiert und zeigt nur die für das Datum und die Zeit eingebauten Funktionen an.



Konstantennamen für Farben

Abbildung 16.7:
Die Suche nach einem Objekt wurde auf ein paar bestimmte Konstanten eingegrenzt.



Datums- und Zeitfunktionen

Abbildung 16.8:

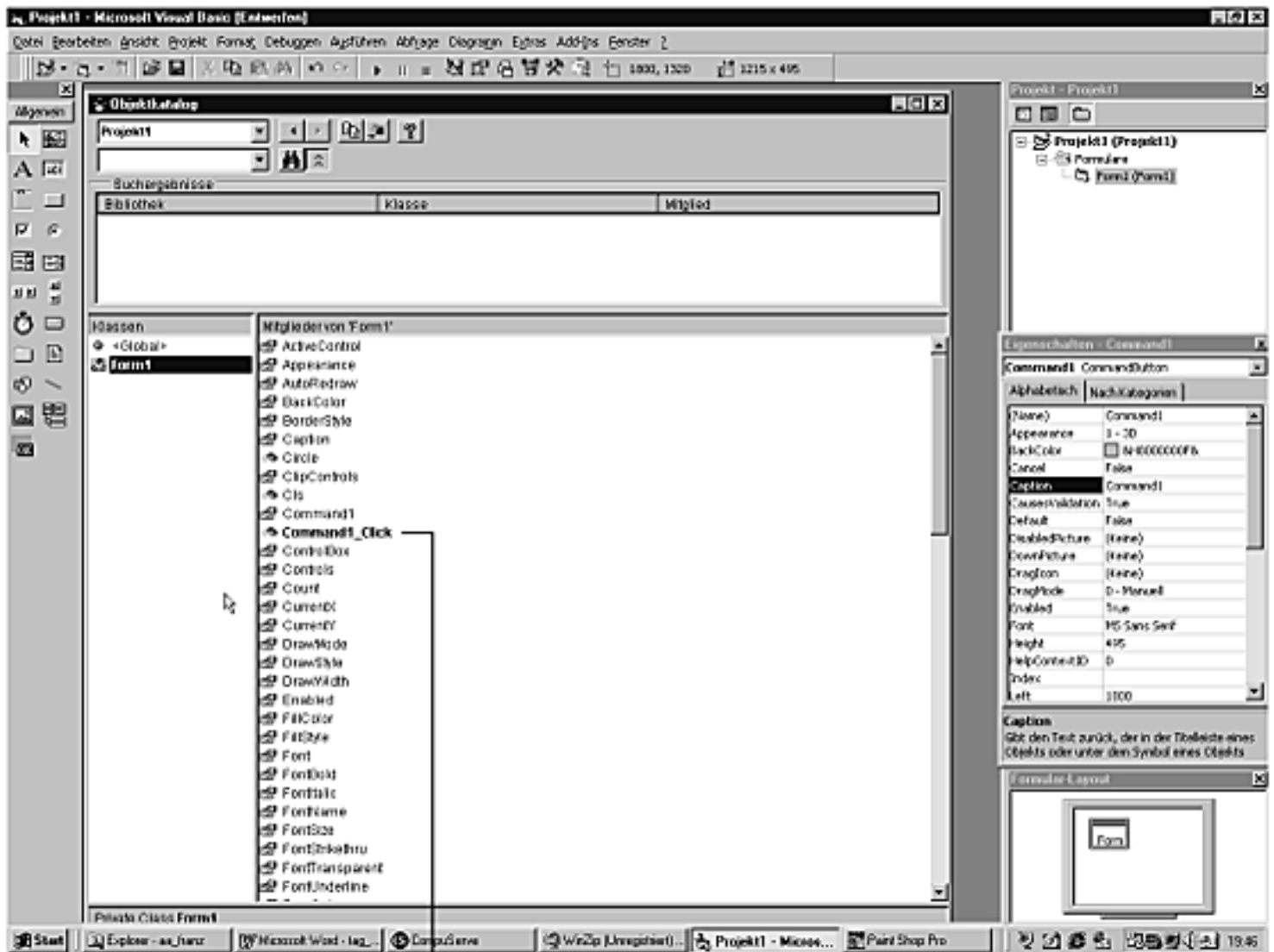
Es ist ganz einfach, die eingebauten Funktionen für ein bestimmtes Thema zu finden.

Diese Funktionen sind zwar auch in der Online-Hilfe beschrieben und stehen an anderen Stellen im Objektkatalog zur Verfügung (wie beispielsweise im Klasseneintrag <Global>), aber durch die Gruppierung von Datums- und Zeitfunktionen wird die Programmierung mit diesen Funktionen vereinfacht, weil alle an zentraler Stelle zusammengefaßt sind.



Beachten Sie, daß Sie die Schaltfläche **Zurück** im Objektkatalog einen Schritt zurück bringt, so daß es ganz einfach ist, sich mit der Maus vor und zurück zu bewegen. Diese Aktion simuliert die heute gebräuchlichen Internet-Browser.

Der Objektkatalog ist sehr praktisch, weil er Ihr eigenes Projekt beschreibt. Er beobachtet im Hintergrund, wie Sie Ihrer Applikation Objekte, Variablen und Ereignisprozeduren hinzufügen. Abbildung 16.9 zeigt den Objektkatalog mit **Projekt1** in der Projekt-/Bibliotheken-Liste. Wenn **Form1** (das einzige bisher in diesem Projekt plazierte Objekt) angeklickt wird, zeigt der Objektkatalog eine Liste aller Ereignisprozeduren, Methoden und Eigenschaften für dieses Formular an. Nur ein Eintrag, **Command1_Click**, wird fett angezeigt, d.h. bisher wurde nur dieser Ereignisprozedur Code hinzugefügt.



Die einzige Ereignisprozedur mit Code

Abbildung 16.9:
Der Objektkatalog zeigt nur die aktiven Komponenten der Applikation an.

Einer der leistungsfähigsten Aspekte des Objektkatalogs ist die Option **Definition anzeigen**. Wenn Sie ein Mitglied markieren, bei dem es sich um eines Ihrer eigenen Objekte handelt (beispielsweise eine Ereignisprozedur, die Sie für ein von Ihnen deklariertes Objekt geschrieben haben), mit der rechten Maustaste auf das Objekt klicken und dann **Definition anzeigen** wählen, springt Visual Basic direkt in den Code, wo Sie dieses Objekt definiert haben! Mit Hilfe des Objektkatalogs finden Sie also auch bestimmten Code innerhalb einer umfangreichen Applikation. Sie müssen nicht wissen, in welchem Projekt ein Objekt definiert wurde, weil Sie es im Objektkatalog jederzeit auffinden.

Bei der Suche nach einem bestimmten Thema zeigt der Objektkatalog eine Liste mit allen Vorkommen dieses Themas in der Applikation an. Wenn Sie beispielsweise nach **Click** suchen, zeigt Visual Basic ein zusätzliches Fenster an (siehe Abbildung 16.10), das jedes relevante Vorkommen des Suchthemas im gesamten Projekt (und im gesamten Visual-Basic-Repertoire, weil in der Dropdown-Liste Projekt/Bibliotheken der Eintrag **<Alle Bibliotheken>** steht) anzeigt.

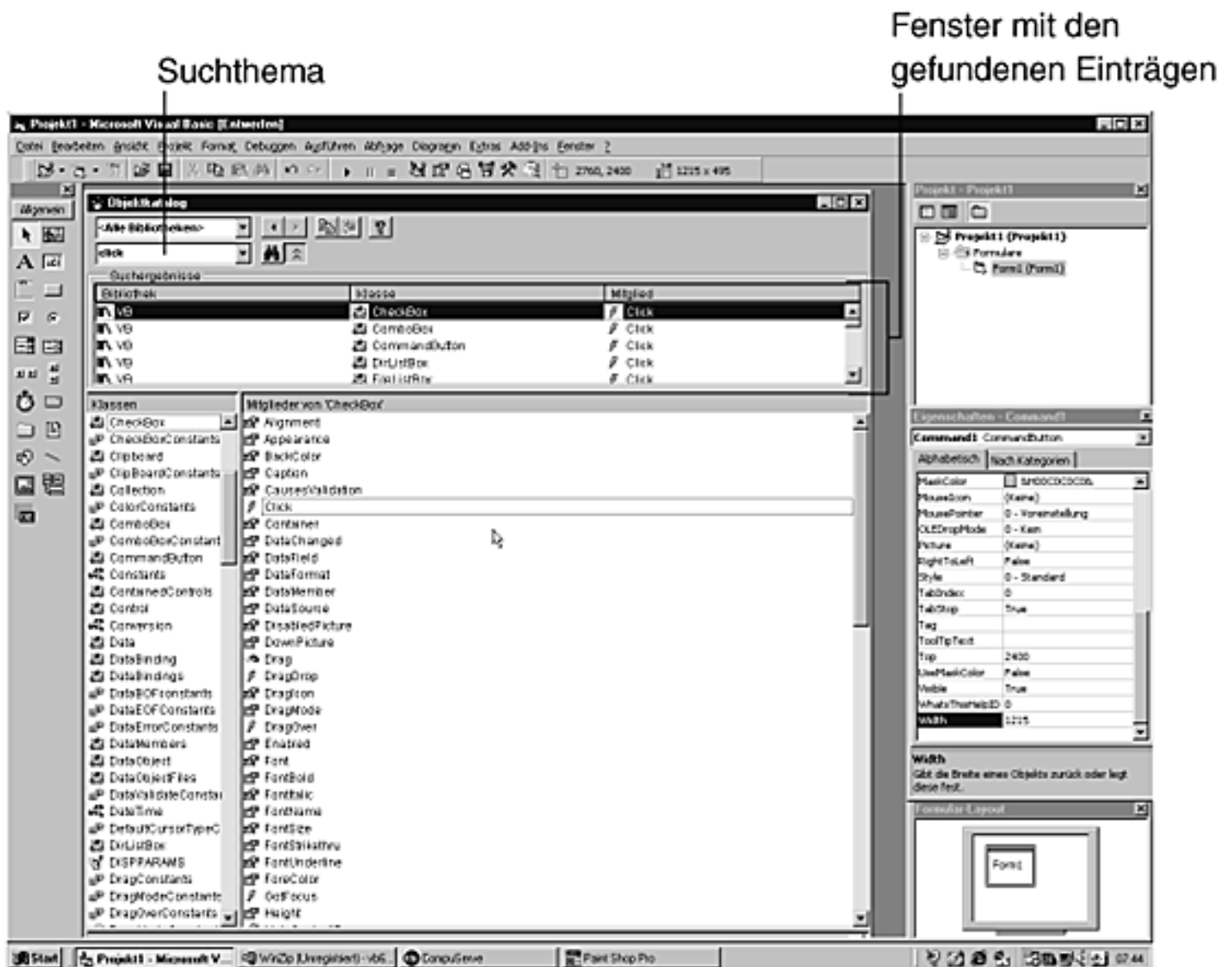


Abbildung 16.10:
Die Suchfunktion hat alle Vorkommen von **Click** gefunden.

Das **Click**-Ereignis tritt für verschiedene Objekte auf; deshalb zeigt der Objektkatalog ein Fenster an, das alle Objekte auflistet, die **Click** unterstützen. Sie grenzen die Suche ein, indem Sie auf das Objekt oder das Projekt klicken, in dem Sie nach dem **Click** suchen möchten.

Zusammenfassung

In diesem Kapitel hat sich alles um Objekte gedreht. Sie haben erfahren, wie OLE-Objekte in Visual Basic eingesetzt werden. Das OLE-Steuerelement macht überhaupt nichts, außer Objekte aus anderen Applikationen aufzunehmen (oder eine Verknüpfung zu einem Objekt). Nachdem diese Objekte eingefügt wurden, können die Benutzer Ihrer Applikation sie direkt innerhalb der Originalapplikation manipulieren und bearbeiten. Sie brauchen keine Befehle zu schreiben, um auf diese anderen Applikationen zuzugreifen. OLE-Objekte sind intelligente Objekte, die alle Bearbeitungswerkzeuge, unter anderem auch Menüs, aus der anderen Applikation mitbringen.

Neben den Auflistungen erlaubt Ihnen Visual Basic auch, Steuerelementfelder zu deklarieren, die sich häufig wie Objekt-Arrays verhalten. Sie können Steuerelemente auf einem Formular vervielfältigen, so

daß sie alle gemeinsame Eigenschaften aufweisen und Sie nur noch die Unterschiede implementieren müssen (beispielsweise Titel oder Farben). Die Systemobjekte stellen allgemeine vordefinierte Objekte zur Verfügung, mit deren Hilfe Ihre Applikation mit den Ressourcen außerhalb der typischen Programmumgebung zusammenarbeiten kann. Durch den Zugriff auf das App-Objekt ermittelt Ihre Applikation zur Laufzeit den Pfad, über den der Benutzer die Applikation gestartet hat. Das Clipboard-Objekt ermöglicht Ihrer Applikation, mit der Windows- Zwischenablage zu arbeiten, um Text dort abzulegen bzw. von dort einzufügen.

Zur Unterstützung des Programmierers stellt Visual Basic den sogenannten Objektkatalog bereit. Dabei handelt es sich im wesentlichen um einen Behälter für Daten. Sie suchen dort nach bestimmten Ereignissen, Eigenschaften und Methoden oder nach einer ganzen Objektklasse. Der Objektkatalog beobachtet, welche Objekte Ihr Code initialisiert hat und welche Ereignisprozeduren Sie geschrieben haben.

Im nächsten Kapitel lernen Sie Objekte der nächsten Generation kennen - ActiveX- Steuerelemente.

Fragen und Antworten

Frage:

Muß mein Benutzer die Originalapplikation für ein OLE-Objekt installiert haben, wenn er in einer Visual-Basic-Applikation mit einem solchen Objekt arbeiten soll?

Antwort:

Ja. Für die In-Place-Automatisierung ist es erforderlich, daß Ihr Benutzer die Applikation installiert hat. Andernfalls kann er das OLE-Objekt nicht bearbeiten. Beispielsweise ist Microsoft Excel, eine leistungsfähige Tabellenkalkulation, die viel Plattenspeicher und Hauptspeicher benötigt, um ihre Aufgabe zu erfüllen, OLE-kompatibel. Sie können also eine Excel-Tabellenkalkulation in eine von Ihnen entwickelte Applikation einbinden, und Ihrem Benutzer steht die Leistungsfähigkeit von Excel in Ihrer Applikation zur Verfügung! Wenn man die Größe und die ganzen Menüs und Funktionen von Excel betrachtet, ist klar, daß es niemals völlig in Ihre Applikation aufgenommen werden kann, um das Objekt zu unterstützen. Ihr Benutzer muß Excel installiert haben, um mit dem Excel-OLE-Objekt in Ihrer Applikation arbeiten zu können. Sie müssen die Anforderungen Ihrer Applikation klar formulieren, unter anderem auch die Notwendigkeit von unterstützenden Programmen wie Microsoft Excel, damit Ihr Benutzer sie sinnvoll nutzen kann.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollen, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist der Unterschied zwischen Linking und Embedding?
2. Welche OLE-Technik (Linking oder Embedding) verbraucht Ihrer Meinung nach den meisten Plattenspeicher?
3. Richtig/Falsch. Visual Basic speichert die Änderungen, die Ihre Benutzer an dem eingebetteten

OLE-Objekt vornehmen, automatisch.

4. Welche Methode speichert ein OLE-Objekt in einer Datei?

5. Welche Methode lädt ein OLE-Objekt aus einer Datei?

6. Nennen Sie zwei Möglichkeiten, die Klasse eines Objekts zu ermitteln.

7. Richtig/Falsch. Sie müssen Systemobjekte übergeben, wenn Sie in mehreren Prozeduren darauf Zugriff brauchen.

8. Nennen Sie drei Dinge, die in der Mitgliederliste des Objektkatalogs angezeigt werden.

9. Was passiert, wenn Sie die Mitgliederliste im Objektkatalog gruppieren?

10. Richtig/Falsch. Der Objektkatalog sucht nicht nach Objekten, die Sie selbst erzeugt haben.

Übungen

1. Warum ist `With . . . End` in dieser Situation möglicherweise keine gute Wahl?

```
With chkMaster  
.Caption = "Datenquelle"  
.Alignment = vbLeftJustify  
End With
```

2. Schreiben Sie eine einfache Applikation, die ein OLE-Steuerelement verwendet. Betten Sie ein Paint-Objekt in das Steuerelement ein. Fügen Sie eine Menüoption ein, die Bilder speichert, und eine, die Bilder lädt. Bei der Ausführung der Applikation doppelklicken Sie auf das Steuerelement und zeichnen ein Bild. Speichern Sie das Bild, verlassen Sie die Applikation, und starten Sie sie neu. Laden Sie das gespeicherte Bild, um sicherzustellen, daß Ihre Prozeduren zum Speichern und Laden funktionieren.

Woche 3

Tag 17

ActiveX- Steuerelemente

Dieses Kapitel stellt Ihnen ActiveX-Steuerelemente vor. Durch die Ergänzung Ihrer Werkzeugsammlung um ActiveX-Steuerelemente fügen Sie Ihrer Visual-Basic-Umgebung zusätzliche Funktionalität hinzu. Außerdem wird die Entwicklungszeit weiterer Applikationen dadurch verkürzt.

Die von Visual Basic unterstützte Objekttechnologie ermöglicht Ihnen, die Funktionalität anderer Applikationen zu nutzen, die ActiveX und die ActiveX-Automatisierung unterstützen. Beispielsweise kann Ihre Visual-Basic-Applikation ein Word-Dokument oder eine Excel-Tabelle anlegen, indem es die ActiveX-Technologie dieser beiden Applikationen übernimmt.

Nachdem Sie erkannt haben, wie wichtig die ActiveX-Technologie ist und wie Sie davon profitieren, werden Sie erfahren, wie Sie eigene ActiveX-Steuerelemente entwickeln.

Heute lernen Sie die folgenden Dinge kennen:

- Die Geschichte der ActiveX-Steuerelemente
- VBX- und OCX-Steuerelemente im Vergleich zu ActiveX-Steuerelementen
- ActiveX-Steuerelemente in Ihren Projekten
- Die Automatisierung von Objekten in Ihren Applikationen
- Excel-Tabellen in Visual-Basic-Applikationen
- Eigene ActiveX-Steuerelemente
- Unterklassen neuer ActiveX-Objekte
- Eigene Eigenschaftswerte für Ihre Steuerelemente anlegen

Wozu ActiveX?

Die ActiveX-Technologie ist die aktuelle Version der Add-in-Steuerelemente, die noch vor ein paar Jahren als VBX-Steuerelemente bezeichnet wurden. Die Dateinamenerweiterung .VBX bedeutet, daß es sich dabei um ein erweitertes Visual-Basic- Steuerelement handelt. Mit anderen Worten, ActiveX-Steuerelemente waren zunächst Visual-Basic-Steuerelemente, die Sie früheren Versionen von Visual Basic hinzufügen konnten, um damit die Werkzeugsammlung zu erweitern.

Damals waren die Visual-Basic-Steuerelemente nicht mit der Browser-Technologie, dem Internet und den anderen Programmierwerkzeugen von Windows, wie beispielsweise Visual C++, kompatibel. Nichtsdestotrotz waren sie in frühen Versionen von Visual Basic wichtig, um dem Programmierer eine

verbesserte Codeentwicklung zu ermöglichen. Je mehr Steuerelemente der Programmierer zur Verfügung hatte, desto weniger Arbeit mußte er selbst leisten. Es entstanden eine ganze Programmierergemeinde und ganze Unternehmenszweige, die Visual-Basic-Steuerelemente entwickelten, mit denen Grafik, Daten, Tabellen, Multimedia usw. manipuliert werden konnten. Weil die Visual-Basic-Steuerelemente so beliebt waren, mußte Microsoft sie überarbeiten (wenn sie nur mit Visual Basic und nicht mit anderen Windows-Produkten kompatibel waren).



OLE ist ein enger Verwandter der VBX-Steuerelemente, aber OLE steht in Windows-Applikationen eher allgemein zur Verfügung - es wurde nicht ausschließlich für Visual-Basic-Programmierer entwickelt. Wie Sie gestern jedoch gesehen haben, ist ein OLE-Objekt ein Datenobjekt (und kein Steuerelement), das Sie verwenden wie ein zusätzliches Visual-Basic-Steuerelement.

Zwischen den Visual-Basic-Steuerelementen und den ActiveX-Steuerelementen hat Microsoft die speziellen 32-Bit-OCX-Steuerelemente entwickelt. Diese neuen Visual-Basic-Steuerelemente erweiterten Visual Basic und andere Programmiersprachen wie Visual C++. OCX-Steuerelemente hatten die Dateinamenerweiterung .OCX. Die älteren VBX-Steuerelemente unterstützten nur 16-Bit-Applikationen.



Wenn Sie bereits mit früheren Ausgaben von Visual Basic gearbeitet haben, die 16-Bit-VBX-Steuerelemente unterstützten, beispielsweise Versionen vor 5.0, ist es jetzt nicht möglich, diese Steuerelemente mit dem aktuellen 32-Bit-Visual-Basic-System zu nutzen, es sei denn, das System enthält einen Ersatz für diese alten Steuerelemente. Beispielsweise stellt Visual Basic eine 32-Bit-Version für das Kalibrierungs-Steuerelement bereit (zuvor gab es 16-Bit-Versionen davon). Sie können also eine ältere Visual-Basic-Applikation, die dieses VBX-Steuerelement verwendet, laden. Visual Basic ersetzt das Steuerelement durch die 32-Bit-Version, und alles funktioniert wieder. Wenn es jedoch keinen solchen Ersatz gibt und der Hersteller des VBX-Steuerelements auch keinen solchen anbietet, müssen Sie das Steuerelement aus der Applikation entfernen und nach einer möglichen Alternative suchen.

Die OCX-Steuerelemente sind zwar mit Visual C++ kompatibel, konnten aber nicht über das Internet eingesetzt werden. Microsoft hat sie deshalb zu ActiveX-Steuerelementen gemacht, so daß auch Internet-Browser und andere Applikationen und Programmiersprachen damit arbeiten konnten.

Gibt es bald schon einen Ersatz für ActiveX? Vielleicht. Das ist davon abhängig, ob eine neue Technologie irgendwann Anforderungen stellt, die ActiveX nicht erfüllen kann.



Die aktuellen Versionen des Internet Explorer und des Netscape Navigator (mit einem entsprechenden Plug-in von der Support-Seite von Netscape) unterstützen ActiveX auf Web-Seiten. Das bedeutet, die Benutzer können auf allen Web-Seiten mit ActiveX-Steuerelementen arbeiten.

ActiveX-Steuerelemente in einem Projekt

Sie wissen, daß ActiveX-Steuerelemente (für Sie als Visual-Basic-Programmierer) einfach nur zusätzliche Steuerelemente sind, die Sie Ihrer Werkzeugsammlung hinzufügen, um damit Programme zu entwickeln. Es gibt alle möglichen Steuerelemente. Einige sind in Visual Basic enthalten. Sie finden sie über **Projekt, Komponenten** (oder (Strg)+(T)), was Sie bereits aus früheren Kapiteln kennen. Darüber hinaus gibt es auch zahlreiche ActiveX-Steuerelemente im Internet. Und es gibt Softwarefirmen, die ActiveX-Steuerelemente verkaufen, welche nahtlos in die Visual-Basic-Umgebung übernommen werden können.

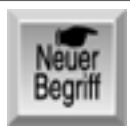
Was Sie tun sollten

Suchen Sie auf der Web-Site von Microsoft (www.microsoft.com/activex) nach ActiveX-Steuerelementen, die Sie auf Ihren PC herunterladen können.



Fügen Sie Ihrer Werkzeugsammlung nicht mehr ActiveX-Steuerelemente hinzu, als Sie in Ihrer Applikation brauchen. Jedes Steuerelement, das Sie einfügen, wird in die kompilierte Version für Ihre Benutzer aufgenommen. (Wie Sie die Applikationen verpacken und weitergeben, erfahren Sie in Kapitel 21.) Je mehr Steuerelemente die Applikation enthält, desto größer wird sie, desto langsamer wird sie und desto mehr Ressourcen verbraucht sie auf dem System des Benutzers. Um nicht benötigte ActiveX-Steuerelemente zu entfernen, zeigen Sie das Dialogfeld **Projekt, Komponenten** an und heben die Markierung für alle überflüssigen Steuerelemente auf.

Egal ob eine Applikation die zum Zeitpunkt der Kompilierung geladenen ActiveX-Steuerelemente nutzt oder nicht, müssen Sie alle *Laufzeit-ActiveX-Steuerelementdateien* zusammen mit dem Projekt und den dazugehörigen Dateien weitergeben.



Eine *Laufzeit-ActiveX-Steuerelementdatei* ist eine Hilfsdatei, die Sie einer Applikation hinzufügen, die das entsprechende ActiveX-Steuerelement verwendet. Die Laufzeitdatei enthält die Anweisungen, die benötigt werden, damit das ActiveX-Steuerelement zur Laufzeit funktioniert.

Es gibt für fast alle Programmieraufgaben ActiveX-Steuerelemente. Wenn Sie mehr Kontrolle über den Sound bereitstellen möchten, als das eingebaute Multimedia-Steuerelement anbietet, finden Sie online viele weitere ActiveX-Steuerelemente. Außerdem gibt es 3D-Grafik-Steuerelemente, Internet-Steuerelemente, mathematische Steuerelemente, Drucker-Steuerelemente, Scanner-Steuerelemente und eine riesige Auswahl anderer Steuerelemente. Sie müssen nur entscheiden, was Sie programmieren möchten, und dann nach den Steuerelementen suchen, die Sie am besten dabei unterstützen. Wenn Sie beispielsweise Software zur Textverarbeitung entwickeln, sollten Sie versuchen, so viele Steuerelemente zur Textbearbeitung und -manipulation wie möglich zu finden.

Wie nutzen Sie die neuen ActiveX-Steuerelemente in Ihrer Werkzeugsammlung? Zunächst können Sie voraussetzen, daß das ActiveX-Steuerelement die folgenden Dinge unterstützt:

- Eigenschaften
- Ereignisse
- Methoden

Mit anderen Worten, Sie verwenden das ActiveX-Steuerelement so wie jedes andere Steuerelement. Das ActiveX-Steuerelement integriert sich nahtlos in Ihre Visual-Basic- Applikation. Sie brauchen eine Liste aller von dem Steuerelement unterstützten Eigenschaften, Ereignisse und Methoden, damit Sie es richtig programmieren können. Viele der Eigenschaften des ActiveX-Steuerelements werden im Eigenschaftenfenster angezeigt, aber nicht alle (insbesondere solche, die nur zur Laufzeit zur Verfügung stehen). Wenn Sie ActiveX-Steuerelemente kaufen oder aus dem Internet herunterladen, erhalten Sie gleichzeitig Anweisungen mit einer Auflistung aller unterstützten Eigenschaften, Ereignisse und Methoden. Für die ActiveX-Steuerelemente in Visual Basic verwenden Sie die Online-Hilfe, um Informationen zu erhalten.

Abbildung 17.1 zeigt das Dialogfeld **Komponenten**, das Sie mit **Projekt, Komponenten** aufrufen. Dieses Dialogfeld ist Ihnen schon einige Male begegnet, weil Sie Ihrer Werkzeugsammlung auch schon früher zusätzliche ActiveX-Steuerelemente hinzugefügt haben, beispielsweise in Kapitel 14, wo Sie mit dem Multimedia-Steuerelement gearbeitet haben.

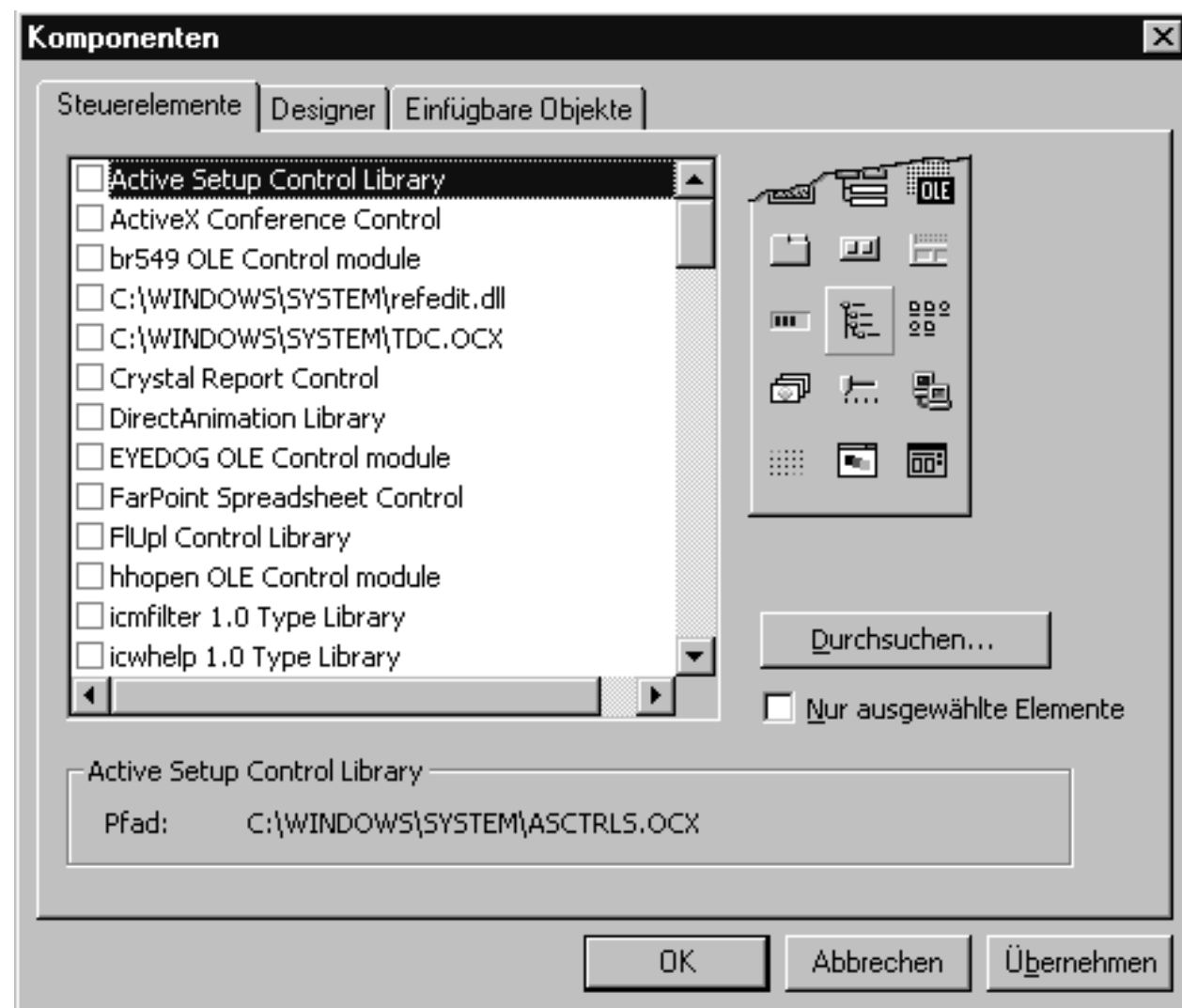


Abbildung 17.1:
Im Dialogfeld Komponenten fügen Sie Ihrem Projekt neue ActiveX-Steuerelemente hinzu.

Bei der Installation von Visual Basic werden Ihrem System mehrere ActiveX-Steuerelemente hinzugefügt, die Sie einem Projekt mit Hilfe des Dialogfelds **Komponenten** zur Verfügung stellen. Darüber hinaus durchsucht die Installationsroutine von Visual Basic Ihr System nach weiteren Steuerelementen. Viele dieser Steuerelemente werden im Dialogfeld **Komponenten** angezeigt. Die Visual-Basic-Installation kann also auf unterschiedlichen Computern unterschiedlich aussehen, weil das Dialogfeld **Komponenten** für beide Systeme möglicherweise unterschiedliche Einträge enthält.

Klicken Sie auf die Schaltfläche **Durchsuchen**, um Ihre Festplatte nach ActiveX-Steuerelementen zu durchsuchen.



Die Liste mit den ActiveX-Steuerelementen im Dialogfeld **Komponenten** kann relativ umfangreich werden, wenn Sie mehrere ActiveX-Steuerelemente auf Ihrem System haben. Nachdem Sie einige der Steuerelemente ausprobiert haben, sollten Sie das Kontrollkästchen **Nur ausgewählte Elemente** anklicken, so daß nur die von Ihnen ausgewählten ActiveX-Steuerelemente in der Liste erscheinen. Wenn Sie jedoch weitere Elemente einfügen möchten, entfernen Sie die Markierung dieses Kontrollkästchens, damit Sie die gewünschten Steuerelemente finden können.

Die Werkzeugsammlung kann sehr schnell sehr voll werden, wenn Sie Steuerelemente hinzufügen. Betrachten Sie die Schaltfläche **Allgemein** oben in Ihrer Werkzeugsammlung. Klicken Sie mit der rechten Maustaste auf einen leeren Bereich in der Werkzeugsammlung. Damit legen Sie neue Gruppen für Ihre ActiveX-Steuerelemente in der Werkzeugsammlung an. Um einer neuen Gruppe Steuerelemente hinzuzufügen, klicken Sie auf die entsprechende Schaltfläche und fügen das Steuerelement ein. (In dem Popup-Menü, das bei einem Klick mit der rechten Maustaste für die Werkzeugsammlung angezeigt wird, steht auch das Dialogfeld **Komponenten** zur Verfügung.). Abbildung 17.2 zeigt eine solche Registerkarte zur Gruppierung von Internet- Steuerelementen.



Abbildung 17.2:
Gruppen helfen, die Werkzeugsammlung
übersichtlicher zu machen.

ActiveX-Automatisierung

Bisher haben Sie Steuerelemente eingefügt; Sie wissen, wie man Eigenschaften setzt, auf Ereignisse reagiert und Methoden auslöst. Einige ActiveX-Steuerelemente ermöglichen Ihnen noch einen weiteren Schritt: Sie können ein eingebettetes Steuerelement innerhalb Ihrer Applikation verwenden und sich seine Funktionalität für Ihre eigene Applikation borgen.

Visual Basic unterstützt die Automatisierung von Steuerelementen zwischen Applikationen. Sie könnten beispielsweise Excel öffnen, eine Tabelle laden, die Daten mit den Excel-Befehlen manipulieren, Excel schließen und die resultierende Tabelle im Formularfenster Ihrer Applikation verwenden, ohne daß Ihre Benutzer merken, daß Sie sich die Funktionalität von Excel zunutze gemacht haben.



Bei der *Automatisierung* nutzt eine Applikation die Daten einer anderen Applikation und manipuliert diese auch mit Hilfe der anderen Applikation. Die Benutzer merken nicht, daß die andere Applikation gestartet wurde, ihre Unterstützung bereitstellte und dann wieder geschlossen wurde.

Die Automatisierung kann nur für ActiveX-Applikationen eingesetzt werden, die in der Registrierungsdatenbank Ihres Systems registriert sind. Eine ActiveX-fähige Applikation wird jedoch im allgemeinen bei der Installation in der Registrierungsdatenbank Ihres Systems registriert.



Normalerweise brauchen Sie für die Automatisierung umfassendes Wissen über die Objekthierarchie der anderen Applikation - und so eine Hierarchie kann verhältnismäßig kompliziert sein. Das nächste Beispiel demonstriert die Automatisierung anhand eines ActiveX-Dokuments. Um die Automatisierung wirklich zu verstehen, müssen Sie sich mit der »geborgten« Applikation auskennen. Dieses Buch kann Ihnen nicht erklären, wie andere Applikationen arbeiten. Die meisten Konzepte der Automatisierung gelten jedoch für alle Applikationen; die im Beispiel vorgestellten Vorgehensweisen können also auch auf andere Applikation übertragen werden.

Als erstes definieren Sie eine Variable, die die Applikation repräsentiert, die Sie innerhalb Ihrer eigenen Applikation automatisieren möchten. Dazu verwenden Sie eine Variable mit dem Visual-Basic-Datentyp `Object`. Sie definieren also ein Applikationsobjekt:

```
Dim obExcelApp As Object
```

Jetzt verbinden Sie die Variable für das Applikationsobjekt mit der Applikation. Falls die Applikation noch nicht läuft, starten Sie sie mit Hilfe der Funktion `CreateObject()` im Hintergrund. `CreateObject()` startet die Applikation nicht nur, sondern verbindet sie gleichzeitig mit Ihrer Objektvariablen:

```
Set obExcelApp = CreateObject("Excel.Application")
```

Setzen Sie für das Funktionsargument von `CreateObject()` den Namen der Applikation ein.

Was Sie tun sollten

Verwenden Sie statt einer einfachen Zuweisung den Befehl `Set`, um Ihre Visual- Basic-Applikation mit der automatisierten Applikation zu verknüpfen. Eine Variable kann keine ganze Applikation aufnehmen; Variablen enthalten nur Werte wie Zahlen oder Strings.

Verwenden Sie `Set`, um eine Referenzvariable für die außenliegenden Objekte zu erstellen. `Set` weist nichts zu, sondern setzt in der Variablen einen Verweis auf das Objekt, das sie repräsentieren soll.

Ein Problem kann auftreten, falls die Applikation bereits läuft. In einem Multitasking- Betriebssystem mit mehreren Benutzern können mehrere Excel-Instanzen gleichzeitig auf demselben Computer ausgeführt werden. Falls die Applikation bereits läuft, verwenden Sie statt `CreateObject()` die Funktion `GetObject()`:

```
Set obExcelApp = GetObject(, "Excel.Application")
```

Beachten Sie das Komma am Anfang der Argumenteliste. In der Regel kann das erste Argument wegfallen, weil das zweite Argument das Objekt beschreibt. Fehlt das zweite Argument, müssen Sie das erste Argument angeben, das einen Pfad zu einer Datei enthält, die das gewünschte Objekt beschreibt.

Falls Excel bereits läuft, wollen Sie keine neue Instanz davon aufrufen. Mit Hilfe der Fehlerverarbeitung können Sie feststellen, ob Excel bereits läuft. Die Funktion `GetObject()` löst einen Fehler aus, wenn Excel noch nicht läuft (weil dann nichts zu ermitteln ist). Wenn Sie feststellen, daß Excel noch nicht läuft, starten Sie mit `CreateObject()` eine Instanz von Excel.

Der folgende Code zeigt, wie Sie prüfen können, ob Excel bereits läuft:

```
1: ' Fehler auffangen
2: On Error Resume Next
3: '
4: ' Verweis auf die Excel-Applikation setzen
5: Set obExcelApp = GetObject(, "Excel.Application")
6: If Err.Number <> 0 Then
7: Set obExcelApp = CreateObject("Excel.Application")
8: blnRunning = False ' Excel läuft nicht
9: Else
10: blnRunning = True
11: End If
```

Sie kennen die Anweisung `On Error Goto` bereits aus früheren Kapiteln. Hier sehen Sie jedoch zum erstenmal die Option `Next`. Zu den bisherigen Beispielen gab es immer eine Sprungmarke, zu der die Anweisung `OnError` verweist, falls ein Fehler auftrat. Die Option `Next` teilt Visual Basic einfach mit, im Falle eines Fehlers zur nächsten Anweisung weiterzugehen und das Programm fortzusetzen. In diesem Code hätte zwar ein Label verwendet werden können, aber hier sollte gezeigt werden, daß bei jedem Auftreten eines Fehlers ein Fehlercode zurückgegeben wird. Der Fehlercode ist eine der Eigenschaften des vordefinierten Systemobjekts `Err` (siehe Kapitel 16).

Bevor in einem Programm ein Fehler auftritt, ist `Err.Number` gleich 0. Enthält `Err.Number` einen Wert ungleich Null, ist ein Fehler aufgetreten. In diesem Fall zeigt Zeile 6 einen Fehlercode in `Err.Number` an, wenn die Funktion `GetObject()` nicht korrekt ausgeführt werden konnte. Zeile 7 sieht also, daß `GetObject()` nicht funktioniert hat, weil Excel noch nicht lief, und startet mit der Funktion `CreateObject()` eine Excel- Instanz. (Wenn Sie später im Programm weitere Fehler

auffangen möchten, setzen Sie den Fehlerstatus zurück, indem Sie `Err.Number` wieder den Wert 0 zuweisen.) Die boolesche Variable `blnRunning` wird auf `False` gesetzt, so daß der Code später im Programm weiß, wie Excel gestartet wurde.



Wenn Excel bereits läuft, soll Ihr Code diese Instanz nicht schließen!



Objektvariablen für die Automatisierung von Applikationen stellen eine Ausnahme zur allgemeinen Regel dar, nur lokale Variablen zu verwenden. Die Applikation liegt außerhalb Ihrer eigenen Applikation. Deshalb verwenden Sie eine globale Objektvariable, damit Ihre Prozeduren diese nicht weitergeben müssen.

Wenn Sie eine andere Applikation öffnen und die Automatisierung verwenden, muß Ihre Applikation die Schnittstelle zur anderen Applikation genau kennen. Ihre Applikation ist quasi der Benutzer der anderen Applikation. Wenn Sie die Excel-Applikation öffnen, arbeiten Sie mit Excel und verwenden die normale Spalten- und Zeilennotation, müssen aber auch die für Excel spezifische Notation für die Objekteigenschaften verwenden.

Jetzt deklarieren Sie ein Tabellen-Objekt, damit die Applikation Daten erzeugen kann:

```
Dim obWorkSheet As Object ' Tabellen-Objekt
```

Der folgende Code fügt Daten in die Zellen einer Tabelle ein:

```
' Werte in Zellen einfügen
obWorkSheet.Cells(1, 1).Value = "Verkäufe"
obWorkSheet.Cells(1, 2).Value = "Monat"
obWorkSheet.Cells(2, 1).Value = 21913.44
obWorkSheet.Cells(2, 2).Value = "April"
```

Wenn man nun alles zusammenfaßt und außerdem Code zur Verfügung stellt, der Aufräumarbeiten übernimmt, die Tabelle speichert und das Excel-Objekt schließt, erhalten Sie etwa Listing 17.1.

Listing 17.1: Ihre Applikation kann Excel nutzen, um eine Tabelle anzulegen

```
1: Private Sub cmdSendToExcel_Click()
2: Dim obExcelApp As Object ' Applikationsobjekt
3: Dim obWorkSheet As Object ' Tabellen-Objekt
4: Dim blnRunning As Boolean ' Falls bereits ein Excel-Objekt
5: ' ausgeführt wird
6: ' Fehler auffangen
7: On Error Resume Next
8: '
9: ' Verweis auf Excel einrichten
10: Set obExcelApp = GetObject(, "Excel.Application")
11: If Err.Number <> 0 Then
12: Set obExcelApp = CreateObject("Excel.Application")
13: blnRunning = False ' Excel lief nicht
14: Else
15: blnRunning = True
16: End If
```

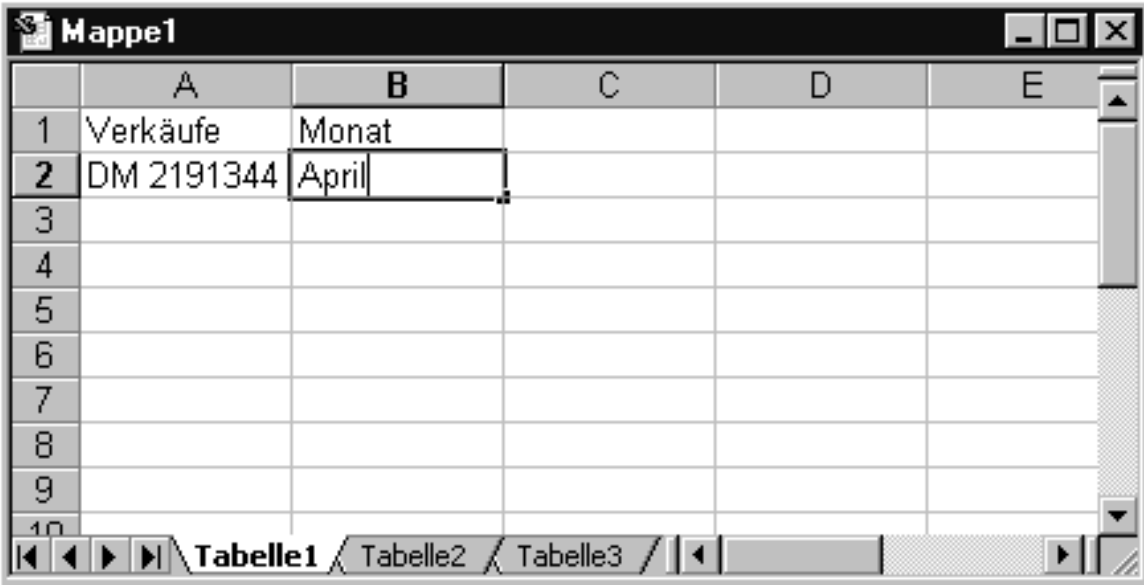


```

17:
18: ' Neue Mappe einfügen
19: obExcelApp.Workbooks.Add
20:
21: ' Verweis auf das aktive Blatt
22: Set obWorkSheet = obExcelApp.ActiveSheet
23:
24: ' Werte in Zellen eintrage
25: obWorkSheet.Cells(1, 1).Value = "Verkäufe"
26: obWorkSheet.Cells(1, 2).Value = "Monat"
27: obWorkSheet.Cells(2, 1).Value = 21913.44
28: obWorkSheet.Cells(2, 2).Value = "April"
29:
30: ' Zweite Zeile formatieren
31: obWorkSheet.Rows("2:2").Select
32: obExcelApp.Selection.NumberFormat = "$##,###.##"
33:
34: ' Mappe speichern (ändern Sie diesen Namen gegebenenfalls)
35: obExcelApp.Save ("c:\VBCreated.XLS")
36:
37: ' Excel nicht verlassen, wenn es bereits lief!
38: obExcelApp.ActiveWorkBook.Close False
39:
40: If Not (blnRunning) Then ' Falls es nicht lief...
41: obExcelApp.Quit ' Excel beenden
42: End If
43:
44: End Sub

```

Falls Ihre Tests ergeben haben, daß Excel vor der Ausführung Ihres Codes noch nicht lief, schließen Sie es (siehe Zeile 41). Lief Excel dagegen schon (d.h. die Funktion `GetObject()` gab keinen Fehler zurück), sollten Sie Excel nicht beenden, weil dort vielleicht noch ein Hintergrundprozeß ausgeführt wird. Der Code in diesem Beispiel erzeugt eine einfache Tabellenkalkulation, wie in Abbildung 17.3 gezeigt.

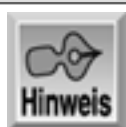


	A	B	C	D	E
1	Verkäufe	Monat			
2	DM 2191344	April			
3					
4					
5					
6					
7					
8					
9					
10					

Abbildung 17.3:

Ihre Visual-Basic-Applikation kann Excel-Tabellen anlegen!

Diese Mappe ist ganz einfach, um das Beispiel nicht zu umfangreich zu machen. Normalerweise kann Ihre Visual-Basic-Applikation Werte verändern und sogar ein Diagramm erzeugen und einen Bericht ausdrucken. Merken Sie sich, daß Visual Basic dazu die Funktionalität von Excel nutzt, um eine formatierte Tabelle zu erzeugen, ohne daß der Benutzer vor dem Bildschirm überhaupt erkennt, daß Excel zu Hilfe geholt wurde.

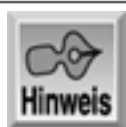


Excel enthält eine eigene Automatisierungssprache, ebenso wie Word und alle anderen ActiveX-fähigen Automatisierungssaplikationen. Nichtsdestotrotz unterstützen die meisten Applikationen die hier demonstrierten Funktionen zum Öffnen und Schließen, und es ist möglich, einer Objektvariablen in Visual Basic das primäre Datenobjekt zuzuweisen. Außerdem unterstützen die Applikationen Methoden und Eigenschaften wie die hier gezeigten, so daß Sie kaum Probleme haben werden, wenn Sie Visual Basic beherrschen. Sie brauchen jedoch Zugriff auf die interne Sprache, die die Applikation für die Automatisierung bereitstellt. Suchen Sie in der Online-Hilfe nach der Objekthierarchie der Applikation. Dort erfahren Sie, welche Objekte zur Verfügung stehen.

Nachdem Sie Excel oder Word oder eine andere ActiveX-kompatible Automatisierungs-Applikation verwendet haben, müssen Sie dieses Objekt in Ihre Visual-Basic- Applikation aufnehmen.

Eigene ActiveX-Steuerelemente

Im ersten Moment erscheint es vielleicht erschreckend kompliziert, ein eigenes ActiveX-Steuerelement anzulegen. Überlegen Sie jedoch einmal, wozu Sie es nutzen könnten. Sie könnten Ihre Steuerelemente nicht nur an andere Entwickler weitergeben (d.h. *verkaufen*), sondern diese Steuerelemente auch in Ihren eigenen Applikationen wiederverwenden. Angenommen, Sie stellen fest, daß Sie immer wieder dieselben Änderungen an Visual-Basic-Steuerelementen vornehmen, damit sie so arbeiten, wie Sie sich das vorstellen. Hier sollten Sie in Betracht ziehen, *neue* Steuerelemente zu entwickeln, die nicht nur die bereits vorhandenen Steuerelemente nachbilden, sondern auch eingebaute Eigenschaften und Methoden aufweisen, durch die sie sich ihren Aufgaben besser anpassen. Wenn Sie später wieder eine Applikation schreiben, in der Sie dieses Steuerelement brauchen, fügen Sie der Werkzeugsammlung Ihr eigenes Steuerelement hinzu und setzen die entsprechenden Eigenschaften. Statt immer wieder denselben Code zu schreiben, damit sich ein Steuerelement so verhält, wie Sie es erwarten, setzen Sie einfach seine Eigenschaftswerte und kümmern sich dann wieder um die eigentliche Applikation.



Ein weiterer Vorteil bei der Entwicklung eigener ActiveX-Programmierungswerkzeuge ist, daß Sie diese Werkzeuge auch in anderen Programmiersprachen einsetzen können, die ActiveX unterstützen, beispielsweise Visual C++ (obwohl sich die Frage stellt, warum Sie je etwas anderes benutzen sollten als Visual Basic).

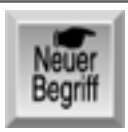
Entwurf eines Steuerelements

Visual Basic stellt Werkzeuge bereit, die Ihnen helfen, eigene ActiveX-Steuerelemente zu entwickeln. Dazu sind jedoch mehrere Schritte erforderlich. Das restliche Kapitel demonstriert, wie ein neues Textfeld entwickelt wird.

Das neue Textfeld - Details

Das neue Textfeld, das Sie hier anlegen werden, erweitert die Funktionalität des in Visual Basic bereitgestellten generischen Textfelds:

- Das Textfeld unterstützt alle Eigenschaftswerte, die auch das normale Textfeld unterstützt.
- Das Textfeld enthält eine neue Eigenschaft, `AutoSize`, die vier mögliche Werte annehmen kann: 1 - `NA`, 2 - `Klein`, 3 - `Mittel` und 4 - `Groß`. Diese Werte erscheinen als Aufzählung in einem Dropdown-Listefeld im Eigenschaftenfenster. Mit 1, 2, 3 oder 4 weisen Sie dem Textfeld im Code einen Wert zu. Für 1 - `NA` (das ist der Standard, der verwendet wird, wenn Sie das Textfeld anlegen), übernimmt das Textfeld die aktuelle Schriftgröße aus dem Eigenschaftswert `Font.Size`. Für 2 - `Klein` wird der Text im Textfeld auf 25% des `Height`-Werts gesetzt. Für 3 - `Mittel` wird der Text auf 50% des `Height`-Werts gesetzt. Für 4 - `Groß` wird der Text auf 75% des `Height`-Werts gesetzt. (Die `Font.Size`-Eigenschaft reflektiert diese neue Größe.) Damit haben Sie eine einfache Methode an der Hand, für den Text im Textfeld eine von vier Schriftgrößen zu verwenden.



Eine *Aufzählung* ist eine Liste fester Werte, die ein Steuerelement annehmen kann, beispielsweise `True` und `False`. Diese Werte erscheinen für die betreffende Eigenschaft in einer Dropdown-Liste.

- Das Textfeld erhält außerdem noch zwei neue Eigenschaften, `UCase` und `LCase`. Dabei handelt es sich um boolesche Eigenschaften. Ist `UCase` gleich `True`, wird der Text im Textfeld in Großbuchstaben umgewandelt. Ist `LCase` gleich `True`, wird der Text im Textfeld in Kleinbuchstaben umgewandelt. `UCase` und `LCase` werden standardmäßig auf `False` gesetzt und können nicht gleichzeitig `True` sein. Ihr Steuerelement muß also sicherstellen, daß, wenn eine dieser Eigenschaft auf `True` gesetzt ist, die andere auf `False` gesetzt wird.

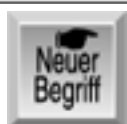
Das neue ActiveX-Steuerelement, das Sie anlegen, verhält sich wie die anderen Steuerelemente. Sie können es in die Werkzeugsammlung einfügen, darauf doppelklicken, um es dem Formularfenster hinzuzufügen, und seine Eigenschaften im Eigenschaftenfenster setzen. Dieses neue Textfeld ist zur Laufzeit sichtbar. Es ist jedoch auch möglich, ein Hintergrund-Steuerelement zu schaffen, das zur Laufzeit nicht im Formularfenster erscheint (wie beispielsweise der Timer). Dazu setzen Sie die `InvisibleAtRunTime`-Eigenschaft auf `True`. Ein solches Steuerelement arbeitet im Hintergrund und erscheint nicht auf dem Formular des Benutzers. (Die `Visible`-Eigenschaft gibt nur an, ob das Steuerelement angezeigt wird, aber `InvisibleAtRunTime` stellt sicher, daß das Steuerelement nie sichtbar ist und nicht angezeigt werden kann.)



Der Name Ihres neuen Steuerelements ist `TextSizeUL`.

Wozu Klassen?

Alle Visual-Basic-Objekte, auch Variablen und Steuerelemente, sind Elemente einer *Klasse*. Die Klasse gruppiert ähnliche Objekte. Außerdem können Sie durch eine Unterklasse von einer übergeordneten Klasse ein neues Objekt ableiten, das alle Eigenschaften der restlichen Klasse haben soll, aber auch Eigenschaften, die für das neue Objekt eindeutig sind. Sie müssen also das Rad nicht neu erfinden, zumal das Objekt (in diesem Fall das ActiveX-Steuerelement) automatisch die Eigenschaften, Methoden und Ereignisse der übergeordneten Klasse übernimmt; Sie fügen nach Belieben neue Eigenschaften, Methoden und Ereignisse ein.



Eine *Klasse* ist eine Sammlung aller Eigenschaften, Methoden und Ereignisse eines Objekts. Ein Steuerelement ist nichts anderes als eine Instanz eines Objekts einer bestimmten Klasse. Ein Formular, das Sie einem Projekt hinzufügen, ist beispielsweise eine Instanz der *Form*-Klasse. Als Mitglied der *Form*-Klasse hat das Formular alle Eigenschaften, Ereignisse und Methoden, die es von anderen Objektklassen unterscheiden, beispielsweise von denen der *CommandButton*-Klasse.

Beim Anlegen einer Unterklasse wird ein Objekt aus einer Klasse existierender Objekte angelegt. Das neue Objekt übernimmt die Eigenschaften, Methoden und Ereignisse seiner Elternklasse, und Sie können ihm eigene Eigenschaften, Methoden und Ereignisse hinzufügen.



Eingebaute Datentypen, wie beispielsweise *String*, sind keine Mitglieder einer Visual-Basic-Klasse, weil sie keine Ereignisse unterstützen.

Angenommen, Sie möchten ein Steuerelement anlegen, das ein anderes Steuerelement bis zu einem gewissen Grad nachbildet. Wenn Sie für dieses neue Steuerelement eine Unterklasse anlegen, übernehmen Sie damit automatisch die Eigenschaften, Methoden und Ereignisse des übergeordneten Steuerelements. Es ist jedoch auch möglich, ein Steuerelement anzulegen, ohne dazu eine Unterklasse anzulegen.

Visual Basic bietet drei Möglichkeiten, ein neues ActiveX-Steuerelement anzulegen:

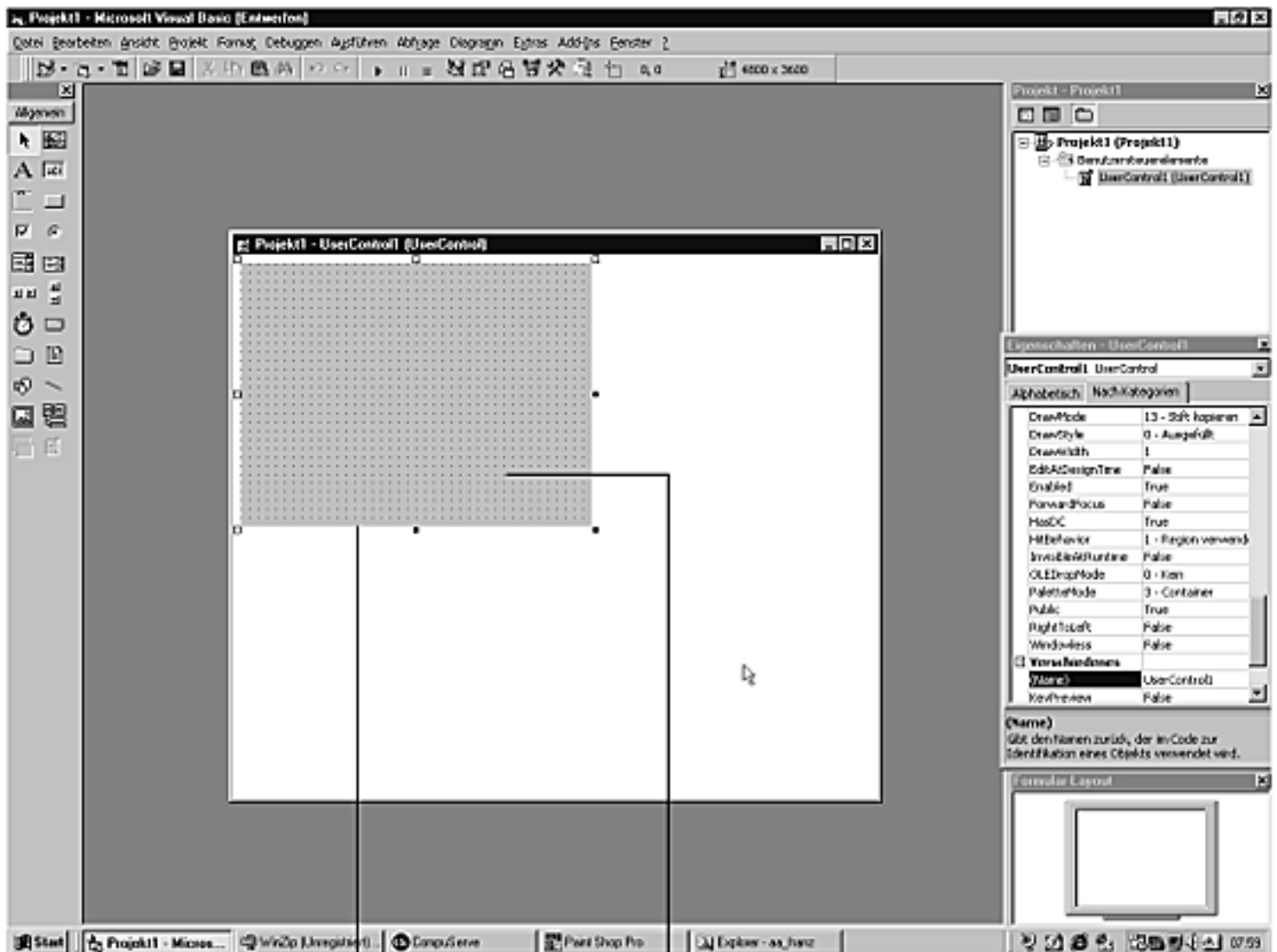
- **Unterklassen für einfache Steuerelemente** - Als Grundlage für Ihr neues Steuerelement verwenden Sie ein existierendes ActiveX-Steuerelement. Ihr neues Steuerelement ist eine Unterklasse des übergeordneten Steuerelements. Es übernimmt die gesamte Funktionalität des übergeordneten Steuerelements und erweitert diese nach Bedarf. Das Anlegen von Unterklassen ist die einfachste Methode, ein ActiveX-Steuerelement anzulegen. Sie ändern die Oberfläche des Steuerelements entsprechend ab, um seine neue Funktionalität zu realisieren.
- **Unterklassen für zusammengesetzte Steuerelemente** - Sie können Ihr neues Steuerelement von mehreren bereits existierenden Steuerelementen ableiten. Mit anderen Worten, wenn Ihr neues Steuerelement ähnlich einem Dialogfeld ist, das Schaltflächen, Textfelder und Bezeichnungsfelder enthält, verwenden Sie existierende Schaltflächen, Textfelder und Bezeichnungsfelder, um den Entwicklungsaufwand für das neue Steuerelement zu reduzieren. Anschließend fügen Sie Ihrem neuen Steuerelement die zusätzlich erforderlichen Funktionen hinzu.
- **Benutzerdefinierte Steuerelemente** - Wenn Ihr neues Steuerelement überhaupt nichts mit bereits existierenden Steuerelementen gemeinsam hat, legen Sie es von Grund auf neu an, indem Sie alle Eigenschaften, Ereignisse und Methoden dafür definieren und das Steuerelement dann so zeichnen, daß es genau Ihren Bedürfnissen entspricht. Ein vom Benutzer entworfenes Steuerelement ist etwas aufwendig zu realisieren, weil Sie sich keine Funktionalität von bereits existierenden Steuerelementen ausleihen können.

Das ActiveX-Steuerelement anlegen

Die folgenden Abschnitte beschreiben, wie Sie das neue ActiveX-Steuerelement *TextSizeUL* anlegen.

Die Vorbereitung von Visual Basic

Bei der Entwicklung eines ActiveX-Steuerelements folgen Sie nicht den Standardprozeduren der Entwicklung neuer Visual-Basic-Applikationen. Nach der Auswahl von **Neu, Projekt** wählen Sie im Dialogfeld **Neues Projekt** nicht das Icon **Standard- EXE**, sondern **ActiveX-Steuerelement**. Visual Basic bereitet die Entwicklung eines neuen Steuerelement vor. Ihr Bildschirm sieht ähnlich aus wie bei der Entwicklung einer neuen Applikation (außer daß das Formularfenster ohne seinen üblichen Rahmen angezeigt wird, wie Sie in Abbildung 17.4 sehen). Das Fenster zeigt jedoch kein Formular mehr an. Vielmehr stellt es den Hintergrund für das ActiveX-Steuerelement dar, das Sie entwickeln wollen. Visual Basic weist diesem Steuerelement standardmäßig den Namen **UserControl1** zu. Das Wort »User« ist etwas irreführend, weil Sie, der Programmierer, das Steuerelement anlegen. Nachdem Sie das Steuerelement fertig haben, wird es jedoch dem Benutzer bereitgestellt.



Kein Rahmen

Hier legen Sie das
Steuerelement an

Abbildung 17.4:

Bei der Entwicklung eines ActiveX-Steuerelements arbeiten Sie in gewohnter Umgebung.

Der Begriff *Laufzeit* bedeutet bei der Entwicklung von Steuerelementen etwas anderes als bei der Entwicklung normaler Applikationen. Während die Entwurfszeit eines Steuerelements dann ist, wenn Sie es anlegen und bearbeiten, können für die Laufzeit zwei Situationen unterschieden werden:

- Wenn ein Programmierer das Steuerelement zur Entwurfszeit einer Applikation in diese einfügt, handelt es sich um ein kompiliertes Steuerelement, das auf die Setup-Anweisungen des Programmierers reagiert.
- Wenn der Programmierer die Applikation kompiliert und ausführt, wird auch das Steuerelement ausgeführt, aber jetzt reagiert es auf die Aktionen des Benutzers.

Häufig unterscheiden die Programmierer zwischen diesen verschiedenen Laufzeitsituationen, indem sie von *Ausführung zur Entwurfszeit* und *Ausführung zur Laufzeit* sprechen. Im allgemeinen ist der Ausführungsmodus aus dem Kontext erkenntlich.

	Der Begriff <i>Ausführung zur Entwurfszeit</i> bezieht sich auf die Situation, wenn Sie ein Steuerelement »ausführen«, um es während der Entwicklung zu testen. Die <i>Ausführung zur Laufzeit</i> beschreibt die Situation, wo ein Steuerelement zusammen mit einer Applikation vom Benutzer ausgeführt wird.

Eine erste Anpassung

Sie können Ihr neues ActiveX-Steuerelement in die Werkzeugsammlung einfügen. Nachdem Sie festgelegt haben, welche Aufgaben es übernimmt, sollten Sie also ein Icon dafür auswählen. Die `ToolboxBitmap`-Eigenschaft ermöglicht Ihnen, ein passendes Icon anzulegen. Das graue Feld, das ähnlich wie das Formularfenster aussieht, ist Ihr eigentliches Steuerelement, das aber noch leer ist. Das Eigenschaftenfenster zeigt also Eigenschaften für Ihr Steuerelement an.

Blättern Sie im Eigenschaftenfenster nach unten, um die `ToolboxBitmap`-Eigenschaft zu finden. Wenn Sie auf die Eigenschaft doppelklicken, wird das Dialogfeld **Bitmap laden** angezeigt. Dort suchen Sie sich ein Icon für Ihr Steuerelement aus. Für unser hier entwickeltes Steuerelement wählen Sie das Icon `\Graphics\Bitmaps\Assorted\Plan` aus.

	In Windows Paint (oder einem vergleichbaren Zeichenprogramm) können Sie eigene Bitmapbilder erstellen. Verwenden Sie dazu 15x16 Pixel, so daß sie dieselbe Größe haben wie die anderen Icons in der Werkzeugsammlung.

Die Bitmap erscheint in der oberen linken Ecke des Steuerelement-Formulars, nachdem Sie es in der `ToolboxBitmap`-Eigenschaft spezifiziert haben.

In Applikationen, die Ihr Steuerelement verwenden, erscheinen Informationen darüber. Sie sollten es also dokumentieren. Wählen Sie **Projekt, Eigenschaften**, und setzen Sie den Projektnamen auf `TextSizeUL`. Tragen Sie in das Textfeld **Projektbeschreibung** eine Beschreibung ein, die in anderen Applikationen angezeigt werden kann. Geben Sie folgendes ein:

Ein Textfeld-Steuerelement, das seine Größe automatisch anpaßt und die Umwandlung in Groß- und Kleinbuchstaben ermöglicht.

Setzen Sie die `Name`-Eigenschaft des Steuerelements auf `NewControl`. Nachdem Sie dem Steuerelement und dem Projekt einen Namen zugewiesen haben, speichern Sie beides. Visual Basic speichert das Steuerelement unter dem Namen `NewControl.ct1`, auch wenn Sie es irgendwann in eine ActiveX-Datei umwandeln.

Eine Unterklasse für das Steuerelement anlegen

Jetzt legen Sie von einem normalen Textfeld ausgehend eine Unterklasse für Ihr Steuerelement an, damit Sie dessen Eigenschaften, Ereignisse und Methoden übernehmen können. Der *ActiveX-Schnittstellen-Assistent* ermöglicht Ihnen, die Funktionalität des Steuerelements zu erweitern.



Der *ActiveX-Schnittstellen-Assistent* hilft Ihnen dabei, eine Unterklasse für ein neues Steuerelement anzulegen.

Der ActiveX-Schnittstellen-Assistent ist nicht Teil der Standardumgebung von Visual Basic. Sie fügen ihn wie folgt ein:

1. Gehen Sie in das Menü **Add-Ins**.
2. Wählen Sie den Eintrag **Add-In-Manager** aus, um das in Abbildung 17.5 gezeigte Dialogfeld aufzurufen.

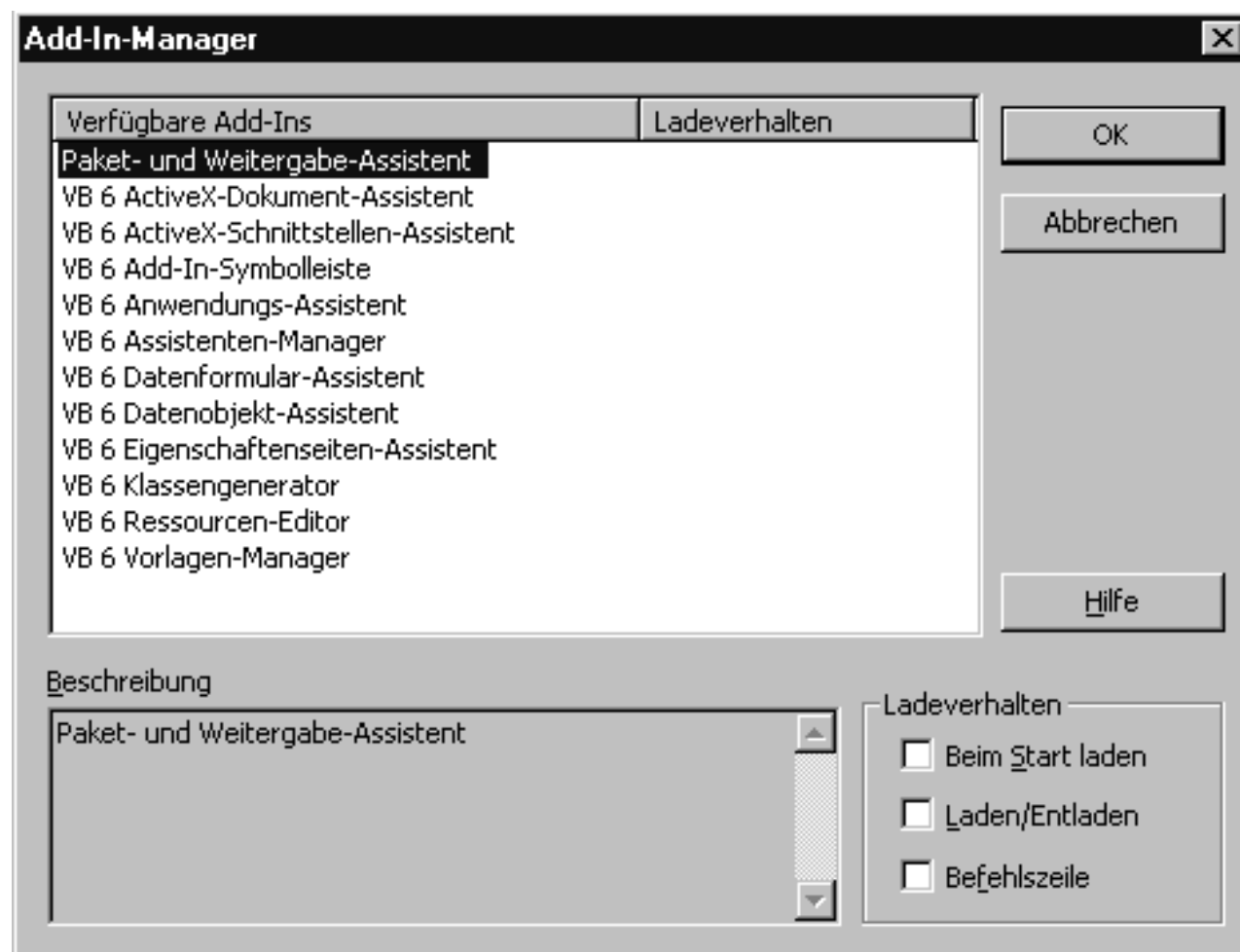


Abbildung 17.5:
Fügen Sie Ihrer Umgebung den ActiveX-Schnittstellen-Assistenten von Visual Basic hinzu.

3. Doppelklicken Sie auf den Eintrag VB 6 ActiveX-Schnittstellen-Assistent. Neben dem Eintrag erscheint die Meldung »Ge laden« und weist darauf hin, daß der Assistent jetzt Teil der Umgebung ist.



Wenn Sie häufig ActiveX-Steuerelemente entwickeln, wählen Sie die Option **Beim Start laden** aus, damit der ActiveX-Schnittstellen-Assistent immer zusammen mit Visual Basic geladen wird.

4. Klicken Sie auf **OK**, um das Dialogfeld zu schließen.

Sie haben den Assistenten nun zwar geladen, können ihn aber noch nicht starten, weil Sie Ihrem leeren ActiveX-Steuerelement zuerst ein Textfeld hinzufügen müssen, von dem aus Sie eine Unterklasse anlegen. Sie erkennen jedoch, ob Visual Basic Ihrer Umgebung den Assistenten hinzugefügt hat, indem Sie in das **Add-Ins**-Menü sehen. Die letzte Option sollte jetzt **ActiveX-Schnittstellen-Assistent** sein. Immer wenn Sie Ihrer Visual-Basic-Umgebung Komponenten hinzufügen, erscheinen diese im **Add-Ins**-Menü.

Bevor Sie den Assistenten starten, fügen Sie Ihrem Projekt ein Textfeld-Steuerelement hinzu. Geben Sie dem Textfeld den Namen `txtParent`. Dieses Textfeld wird von dem neuen ActiveX-Steuerelement gesteuert, das Sie hinzufügen, und nicht von einem Programmierer. Der Programmierer, der Ihr ActiveX-Steuerelement einsetzt, kann dessen Größe abändern. Die Ereignisprozeduren für das neue Steuerelement ändern das interne Textfeld entsprechend ab. Das Textfeld-Steuerelement stellt die Funktionalität zur Verfügung, für die Sie eine Unterklasse anlegen.



Das eingebaute Textfeld verhält sich gewissermaßen wie ein verborgenes Steuerelement (ähnlich einem lokalen Objekt eines öffentlichen Steuerelements), das andere Projekte nutzen. Ihr Steuerelement verwendet das eingebaute Textfeld, ändert dessen Verhalten ab und präsentiert dieses veränderte Verhalten anderen Applikationen.

Der Assistent

Jetzt starten Sie den ActiveX-Schnittstellen-Assistenten aus dem **Add-Ins**-Menü. Das erste Fenster zeigt eine Begrüßung an, die Sie für alle folgenden Sitzungen deaktivieren, indem Sie die Markierung im Kontrollkästchen ganz unten auf dem Bildschirm entfernen. Nachdem Sie diesen Eröffnungsbildschirm gelesen haben, klicken Sie auf **Weiter**, um das Fenster **Schnittstellenmitglieder auswählen** zu öffnen, wie in Abbildung 17.6 gezeigt.



Abbildung 17.6:
Hier wählen Sie Elemente des übergeordneten Steuerelements aus, die in Ihrem ActiveX-Steuerelement verwendet werden sollen.

Der Assistent zeigt zwei Listen an. In der linken Liste werden die Eigenschaften, Ereignisse und Methoden angezeigt, die Sie für Ihr neues ActiveX-Steuerelement aufnehmen können. Der Assistent legt diese Liste an, indem er Ihr aktuelles Steuerelement analysiert und dabei erkennt, daß es sich um das eingebaute Textfeld handelt.

Die rechte Liste enthält mehrere Eigenschaften, Methoden und Ereignisse, die der Assistent aus der vollständigen Liste links ausgewählt hat. Sie wählen zusätzliche Elemente aus, indem Sie sie im linken Fenster anklicken und dann auf die Schaltfläche > klicken, um dieses Element auch in das rechte Fenster einzutragen. Sie können aber auch in der rechten Liste Elemente auswählen und auf die Schaltfläche < klicken, um sie aus der Liste zu entfernen.



Die Steuerelement >> kopiert alle Elemente aus der linken in die rechte Liste; die Schaltfläche << entfernt alle Elemente aus der rechten Liste, so daß Sie ganz von vorne beginnen können.

Überraschenderweise sendet der Assistent mehrere Eigenschaften, die Sie für Textfeld-Unterklassen brauchen, nicht in die rechte Liste. Sie müssen sie also selbst auswählen und dorthin kopieren:

Alignment
 Change
 FontBold
 FontItalic
 FontName

FontSize
FontStrikethru
FontUnderline
MultiLine
PasswordChar
ScrollBars
Text
ToolTipText

Sie können alle diese Eigenschaften auf Ihr neues ActiveX-basiertes Textfeld anwenden, ebenso wie alle neuen Eigenschaften, die Sie definieren. Nachdem Sie die Elemente hinzugefügt haben, klicken Sie auf **Weiter**, um das Fenster **Benutzerdefinierte Schnittstellenelemente erstellen** anzuzeigen, das Sie in Abbildung 17.7 sehen.



Abbildung 17.7:
Jetzt fügen Sie eigene Eigenschaften, Methoden und Ereignisse ein.

Das Listenfeld in der Fenstermitte ist leer, weil Sie erst noch eigene Elemente eintragen müssen. Sie können hier nicht einfach das Element eingeben, wie beispielsweise die Eigenschaft, die die Größe des neuen ActiveX-Textfelds automatisch anpaßt. Statt dessen klicken Sie auf die Schaltfläche **Neu**, um die neue Eigenschaft (oder die neue Methode oder das neue Ereignis) zu beschreiben, die Sie einfügen möchten.

Nachdem Sie auf **Neu** geklickt haben, wird das in Abbildung 17.8 gezeigte Dialogfeld geöffnet, in das Sie ein neues Element eingeben. Neben dem Elementnamen wählen Sie hier auch seinen Typ (Eigenschaft, Methode oder Ereignis) aus. Dazu klicken Sie auf eines der Optionsfelder unten im Dialogfeld. Geben Sie einen Namen ein, und wählen Sie einen Typ aus. Klicken Sie auf **OK**, um das

Element in das Listenfeld des Assistenten für Ihre eigenen Elemente einzufügen.

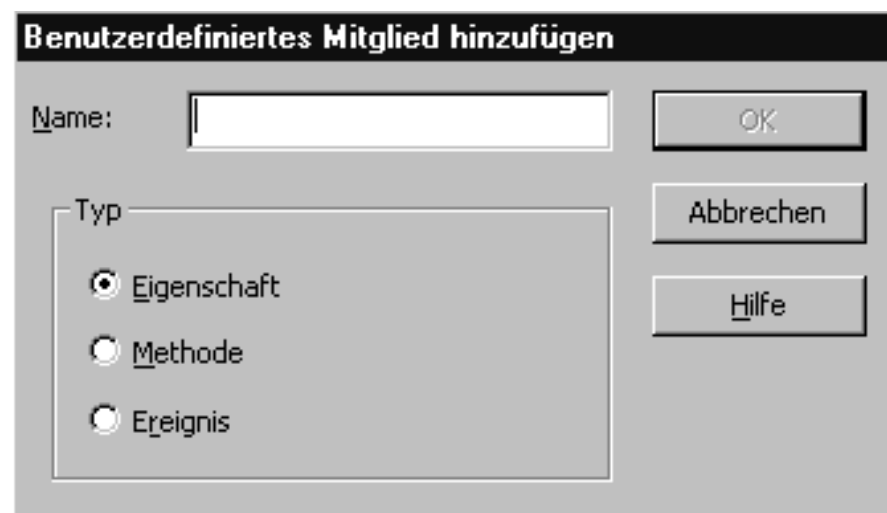


Abbildung 17.8:

Hier geben Sie den Namen des neuen Steuerelements ein und legen seinen Typ fest.

Für das ActiveX-Steuerelement, das wir hier entwickeln, geben Sie die beiden folgenden neuen Elemente ein:

- AutoTSize (Eigenschaft)
- ULText (Eigenschaft)

Klicken Sie auf **Weiter**, um in das nächste Fenster des Assistenten zu gelangen. Dieses Fenster, **Attribute festlegen**, verbindet die Eigenschaften, Ereignisse und Methoden mit den Textfeld-Äquivalenten. Mit anderen Worten, die Liste, die Sie im Feld **Öffentlicher Name** sehen, wurde aus den zuvor ausgewählten und den hier neu hinzugefügten Elementen erstellt.

Teilen Sie dem Assistenten mit, wie sich die Eigenschaften, Ereignisse und Methoden verhalten sollen. Bei unserem neuen Textfeld möchten wir, daß sich *alle* Eigenschaften, Methoden und Ereignisse so verhalten, wie es für Textfelder üblich ist. (Sie könnten diese Zuordnung auch abändern. Beispielsweise könnten Sie einem MouseDown-Ereignis ein MouseClick-Ereignis zuordnen, so daß ein MouseClick-Ereignis erzeugt wird, wenn der Benutzer auf Ihr Steuerelement klickt und eigentlich ein MouseDown-Ereignis resultieren sollte.)

Die beiden einzigen Eigenschaften, die Sie nicht dem normalen Verhalten des Textfelds zuordnen möchten, sind AutoTSize und ULText. Selektieren Sie also alle Elemente in der Liste, bis auf AutoTSize und ULText.

	Die Liste unterstützt Mehrfachauswahlen. Eine schnelle Methode, alle bis auf die beiden neuen Eigenschaften auszuwählen, ist, das oberste Element in der Liste anzuklicken und dann ^(a))+(Ende) zu drücken, um alle Elemente zu markieren. Halten Sie dann die (Strg)-Taste gedrückt, während Sie auf ULText und AutoTSize klicken, um die Auswahl dafür aufzuheben.

Wählen Sie das Element aus, dem Sie diese Eigenschaften zuordnen möchten. Öffnen Sie das Listenfeld **Zugeordnet zu Steuerelement**, und wählen Sie txtParent. Alle diese öffentlichen Elemente, bis auf die beiden nicht selektierten, werden auf das eingebettete Textfeld-Steuerelement angewendet, das Sie auf dem Formular angelegt haben. Sie möchten, daß sich das interne Textfeld normal verhält, damit Ihr ActiveX- Steuerelement wie gewohnt darauf zugreifen kann. Deshalb wurden die beiden neuen Eigenschaften nicht auf das interne Textfeld angewendet.

Klicken Sie auf **Weiter**, um das Fenster **Attribute festlegen** anzuzeigen, das Sie in Abbildung 17.9 sehen. Hier ordnen Sie dem neuen Steuerelement die neuen Eigenschaften zu.

ActiveX-Schnittstellen-Assistent - Attribute festlegen

Sie können die Attribute der nichtzugeordneten Mitglieder festlegen. Jedes nicht geänderte Mitglied verwendet die Standardwerte. Wählen Sie den öffentlichen Namen des Mitglieds aus, dessen Attribute Sie ändern möchten, und ändern Sie die zugehörigen Werte. Wiederholen Sie dies für alle zu ändernden Mitglieder.

Wenn Sie das Ändern der Attribute abgeschlossen haben, klicken Sie auf 'Weiter'.

Öffentlicher Name:

Öffentlicher Name	Property
AutoSize	Property
BackColor	Property
BackStyle	Property
BorderStyle	Property
Click	Event
DbClick	Event

Beschreibung

Attributinformationen

Datentyp	Standardwert
Variant	0

Laufzeit	Entwurfszeit
Lesen/Schreib	Lesen/Schreib

Argumente

Hilfe Abbrechen < Zurück Weiter > Fertig

Abbildung 17.9:
Jetzt ordnen Sie Ihrem Steuerelement die neuen Eigenschaften zu.

Wenn Sie dem neuen Steuerelement Methoden und Ereignisse hinzufügen, erscheinen auch sie im Fenster Attribute festlegen. Sie haben die bekannten Eigenschaften, Methoden und Ereignisse bereits dem internen Steuerelement zugeordnet. Jetzt ordnen Sie dem neuen Steuerelement die neuen Eigenschaften zu. Bereiten Sie die neuen Eigenschaften wie folgt vor:

Ändern Sie für die Eigenschaft **AutoTSize** den Standardwert von 0 in 1. (Alle anderen Felder sind bereits richtig belegt.) Die **AutoTSize**-Eigenschaft nimmt einen von vier Werten einer Aufzählung an, die zwischen 1 und 4 liegen. Der Wert 1 ist der Vorgabewert, deshalb ersetzen Sie hier 0 durch 1.

Geben Sie die folgende Beschreibung für die **AutoTSize**-Eigenschaft ein:

Bestimmt die Schriftgröße in bezug auf die **Height**-Eigenschaft, nämlich 25%, 50% oder 75%.

Selektieren Sie die **ULText**-Eigenschaft. Alle Felder sind bereits richtig gesetzt, Sie müssen aber die folgende Beschreibung einfügen:

Zeigt den Text in Großbuchstaben, Kleinbuchstaben oder unverändert an.

Klicken Sie auf **Weiter**, um den Assistenten fortzusetzen. Der Assistent hat jetzt alle Informationen, die er braucht, um das Steuerelement anzulegen. Klicken Sie auf **Fertig**, um das Steuerelement zu erzeugen. Wenn Sie die Option **Zusammenfassung anzeigen** markiert haben, zeigt der Assistent an,

was er jetzt machen wird.

Offensichtlich ist das ActiveX-Steuerelement noch nicht fertig. Sie haben ihm noch nicht gesagt, wie es sich verhalten soll, wenn die Eigenschaften `AutoSize` oder `ULText` ausgewählt werden. Der Assistent legt die Parameter für das Steuerelement an, aber Sie müssen den Code schreiben und das ActiveX-Steuerelement manuell vervollständigen, damit es seine Arbeit aufnehmen kann.

Vervollständigen des ActiveX-Steuerelements

Der Assistent konnte keinen Code erstellen, der die Eigenschaften `AutoSize` oder `ULText` aktiviert, weil er nicht weiß, was die beiden neuen Eigenschaften bewerkstelligen sollen. Andererseits konnte der Assistent die existierenden Eigenschaften, Methoden und Ereignisse des übergeordneten Steuerelements zuordnen, weil diese Elemente bereits definiert sind.

Um das ActiveX-Steuerelement zu vervollständigen, brauchen Sie Code. Dazu öffnen Sie das Codefenster. Wählen Sie **Ansicht, Code**, um das Codefenster für das ActiveX-Steuerelement anzuzeigen. Während Sie den Code betrachten, sehen Sie, daß ein Großteil des Codefensters mit Zuordnungen der Eigenschaften, Ereignisse und Methoden des neuen Steuerelements zu den Eigenschaften, Ereignissen und Methoden des zugrundeliegenden Textfelds `txtParent` gefüllt ist. Mit anderen Worten, wenn ein Programmierer dieses neue Steuerelement verwendet und zur Entwurfszeit (d.h. wenn er das fertige Steuerelement in seiner eigenen Applikation nutzt) die `BackColor`-Eigenschaft setzt, setzt der Code die `BackColor`-Eigenschaft des zugrundeliegenden Textfelds. Der Code ist an manchen Stellen sehr kompliziert; erwarten Sie also nicht, sofort alles zu verstehen.

Was Sie tun sollten

Betrachten Sie im Code alle `Let`- und `Get`-Ereignisprozeduren. Wie Sie aus Kapitel 11 wissen, werden diese beiden speziellen Befehle verwendet, um Eigenschaftswerte zu setzen und zu lesen. Der Assistent hat Eigenschaften für das übergeordnete Textfeld-Steuerelement erzeugt, die Sie dann Ihrem neuen Steuerelement zugeordnet haben.



Sie können diesen Code direkt bearbeiten, sollten aber nie etwas ändern, was mit dem folgenden Kommentar gekennzeichnet ist:

ACHTUNG! DIE FOLGENDEN KOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER VERÄNDERN!

Der nachfolgende Code ist wichtig für die Funktionalität Ihres neuen Steuerelements und sollte deshalb nicht verändert werden.

Was Sie tun sollten

Vervollständigen Sie alle Codeabschnitte, die mit dem Kommentar »Noch zu ergänzen« gekennzeichnet sind. Solche Codeabschnitte begegnen Ihnen insbesondere, wenn Sie neue ActiveX-Steuerelemente anlegen, die auf Listefeldern basieren. Der Assistent kann die Listen nicht vollständig bearbeiten; Sie müssen also den erforderlichen Code für die Listenverarbeitung selbst einfügen.

Listing 17.2 zeigt die ersten paar Zeilen des Codes. Diese Zeilen sind den neuen Eigenschaften

gewidmet, die Sie dem Projekt hinzugefügt haben.

Listing 17.2: Hinzufügen von Eigenschaften

```
1: 'Standard-Eigenschaftswerte:  
2: Const m_def_AutoTSize = 1  
3: Const m_def_ULText = 0  
4: 'Eigenschaftsvariablen  
5: Dim m_AutoTSize As Variant  
6: Dim m_ULText As Variant
```

Das Schlüsselwort **Const** deklariert Konstantennamen, keine Variablen. `m_def_AutoTSize` in Zeile 2 ist also keine Variable, sondern ein Konstantenname. Visual Basic beinhaltet diverse Konstantennamen, wie beispielsweise `vbWhite` oder `vbInformation`, Sie können aber auch eigene Konstanten deklarieren, egal ob diese lokal oder global sind. (Konstanten sind häufig global, weil sie sich nicht ändern; es besteht also nicht die Gefahr, daß sie versehentlich von einer Prozedur geändert werden, die eigentlich keinen Zugriff darauf haben sollte.)

Die Zeilen 2 und 3 deklarieren Konstanten für die Standardwerte Ihrer beiden neuen Eigenschaften. Sie setzen diese Standardwerte im Fenster **Attribute festlegen**. Es handelt sich dabei um die Standardwerte, die im Eigenschaftenfenster für diese beiden Eigenschaften angezeigt werden, wenn ein Programmierer das ActiveX-Steuerelement auf einem Formular verwendet. Wenn der weitere Code auf eine der Konstanten zugreift, wird statt 0 der Wert 1 verwendet, weil das der Wert ist, der in den Zeilen 2 und 3 in Listing 17.2 dafür gesetzt wird.

Die Zeilen 5 und 6 deklarieren **Variant**-Variablen, die den aktuellen Wert der Eigenschaften darstellen. wenn ein Programmierer einen dieser Eigenschaftswerte zur Entwurfszeit setzt (oder wenn die Prozedur, in der das Steuerelement eingesetzt wird, im Code diese Werte unter Verwendung von Zuweisungen setzt), werden diese Werte in den Variablen aufbewahrt.

Sie definieren die Werte für die beiden Eigenschaften, die im Eigenschaftenfenster erscheinen. Die Aufzählung wird in einem Codeblock definiert, der mit der Anweisung **Enum** beginnt. Unmittelbar unterhalb des allgemeinen Deklarationsbereichs im Codefenster geben Sie den in Listing 17.3 gezeigten Code ein.

Listing 17.3: Hier definieren Sie die Werte der Aufzählung, die im Eigenschaftenfenster erscheinen.

```
1: Public Enum AutoTSizeEnum  
2: NA = 1  
3: Klein = 2  
4: Mittel = 3  
5: Groß = 4  
6: End Enum  
7: Public Enum ULTextEnum  
8: AsIs = 0  
9: Uppercase = 1  
10: Lowercase = 2  
11: End Enum
```

Deklarieren Sie alle Aufzählungswerte als öffentlich, so daß der gesamte Code Zugriff darauf hat. Die Definition von Aufzählungswerten beginnt mit der **Enum**-Anweisung. Es handelt sich dabei um die Werte, die in den Dropdown-Listenfeldern für diese beiden Eigenschaftswerte im Eigenschaftenfenster

angezeigt werden. Die `AutoSize`-Werte erscheinen in dem folgenden gebräuchlichen Format (das Sie schon von anderen Eigenschaftswerten her kennen):

- 1 - NA
- 2 - Klein
- 3 - Mittel
- 4 - Groß

Um einen Ausgangswert zu setzen, wählt der Programmierer, der mit dem ActiveX- Steuerelement arbeitet, einen dieser Werte aus dem Eigenschaftenfenster aus. Darüber hinaus kann der Code der Applikation, in der dieses ActiveX-Steuerelement eingesetzt wird, der Eigenschaft die Werte 1, 2, 3 oder 4 zuweisen, um das Steuerelement auf einen dieser Eigenschaftswerte zu setzen. Die Zuweisung kann ebenfalls Aufzählungswerte zuweisen, wie hier erfolgt:

```
NewControl.AutoSize = Medium ' Weist 3 zu
```

Auf dieselbe Weise definiert der Aufzählungstyp `ULTextEnum` die Aufzählungswerte für die `ULText`-Eigenschaft. Die Zuweisungen für diese Eigenschaft arbeiten wie gewohnt.

Der Code für die Größenänderung des neuen Steuerelements ist extrem einfach, weil sie so erfolgt wie für das übergeordnete Textfeld. Häufig ändert ein neues ActiveX- Steuerelement seine Größe anders als die Steuerelemente seiner Oberklasse, insbesondere bei zusammengesetzten ActiveX-Steuerelementen. Wenn für die Größe des übergeordneten Steuerelements und die des neuen Steuerelements jedoch eine 1:1- Beziehung besteht, fügen Sie die Ereignisprozedur `UserControl_Resize()` hinzu, indem Sie den in Listing 17.4 gezeigten Code in das Codefenster eingeben:

Listing 17.4: Die Positionierung und Größenänderung für das neue ActiveX-Steuerelement erfolgt genau wie für das interne Textfeld.

```
1: Private Sub UserControl_Resize()  
2: ' Höhe und Skalierung des zugrundeliegenden Textfelds verwenden  
3: ' Steuerelement auf die vorgegebene Höhe und Breite bringen  
4: If UserControl.Height <> txtParent.Height Then  
5: txtParent.Height = UserControl.Height  
6: End If  
7: txtParent.Move 0, 0, UserControl.ScaleWidth  
8: End Sub
```

Zeile 4 stellt sicher, daß der Programmierer, der dieses neue Steuerelement verwendet, mit der Größenänderung für das Steuerelement auch das eingebettete Textfeld aktualisiert, weil das interne Textfeld als Behälter für das neue Steuerelement dient. Zeile 7 verschiebt das übergeordnete Textfeld mit der `Move`-Methode an die obere linke Koordinate (0,0) des neuen Steuerelements und setzt dann für beide Eigenschaften dieselbe Skalierung. Damit wird sichergestellt, daß das Textfeld verschoben wird, wenn der Programmierer das neue Steuerelement verschiebt. Auch hier bestimmt das neue Steuerelement, was das interne Steuerelement macht, wenn der Programmierer das neue ActiveX-Steuerelement verschiebt und seine Größe ändert. Die `Move`-Methode ist sowohl für die Verschiebung als auch für die Skalierung für das existierende Steuerelement zuständig, um so beide synchron zu halten. Das interne Textfeld verhält sich also wie eine Folie, die über dem ActiveX-Steuerelement liegt und immer darauf liegen bleibt, um Text entgegenzunehmen und anzuzeigen.

Nachdem die Größenänderung realisiert ist, richten Sie die Aufzählungsliste ein, schreiben Code, der die Auswahl der `AutoSize`-Eigenschaft ermöglicht und der es dem ActiveX-Steuerelement erlaubt,

die Eingabe in Groß- oder in Kleinbuchstaben umzuwandeln, abhängig von der `ULText`-Eigenschaft. Der Assistent hat Platzhaltercode für die `AutoSize`- und `ULText`-Eigenschaften geschaffen, aber Sie müssen die Details eintragen.

Damit können Sie die `Get`- und `Let`-Prozeduren für die neuen Eigenschaftswerte einrichten. Wenn der Benutzer (d.h. der Programmierer, der irgendwann Ihr `ActiveX`- Steuerelement in seinen Applikationen einsetzt) einen Eigenschaftswert setzt, wird die `Let`-Prozedur ausgeführt. Wenn der Benutzer auf einen Eigenschaftswert zugreift, wird die `Get`-Prozedur ausgeführt.

Die einfachere der beiden Methoden ist `Get`. Der Assistent erzeugt `Get`-Funktionsprozeduren für beide Eigenschaftswerte, die Sie jedoch noch ändern müssen. Statt des Rückgabetyps `Variant` brauchen Sie die Aufzählungsdatentypen, wie in Listing 17.5 gezeigt.

Listing 17.5: Die `Get`-Prozeduren für die neuen Eigenschaften müssen die entsprechenden Aufzählungswerte zurückgeben.

```
1: Public Property Get AutoTSize() As AutoTSizeEnum
2: AutoTSize = m_AutoTSize
3: End Property
4:
5: Public Property Get ULText() As ULTextEnum
6: ULText = m_ULText
7: End Property
```

Listing 17.5 weist der Statureigenschaft den Wert des aktuellen Elements zu. Die Umwandlung des `Variant` in eine Aufzählung ist die einzige Änderung, die Sie ausführen müssen.

Außerdem sollten Sie die Rückgabedatentypen der beiden `Let`-Prozeduren ändern. Auch für sie ist zusätzlicher Code erforderlich. Wenn für eine der beiden neuen Eigenschaften ein Wert zugewiesen wird, müssen mehrere Dinge erfolgen, beispielsweise die Größenänderung für den Text oder die Umwandlung in Groß- oder Kleinbuchstaben.

Jetzt stehen zwei weitere Prozeduren zur Vervollständigung an: die entsprechenden `Let`-Prozeduren für die beiden Eigenschaften. Der Assistent erzeugt ein Codegerüst für diese Prozeduren. Dieser Code ist in Listing 17.6 gezeigt.

Listing 17.6: Sie müssen die `Let`-Prozeduren für die beiden Eigenschaften vervollständigen.

```
1: Public Property Let AutoTSize(ByVal New_AutoTSize As
AutoTSizeEnum)
2: m_AutoTSize = New_AutoTSize
3: ' Überprüfung des Eigenschaftsstatus und Änderung der Größe
4: ' (abhängig von dem Eigenschaftswert)
5: '
6: Select Case New_AutoTSize
7: Case 1: ' Keine Änderung erforderlich
8: Case 2: Font.Size = 72 * 0.25 * (Height / 1440)
9: Case 3: Font.Size = 72 * 0.5 * (Height / 1440)
10: Case 4: Font.Size = 72 * 0.75 * (Height / 1440)
11: End Select
12: PropertyChanged "AutoTSize"
13: End Property
14:
```



```

15: Public Property Let ULText(ByVal New_ULText As ULTextEnum)
16: m_ULText = New_ULText
17: ' Steuerelementstatus auswerten und
18: ' das Textfeld entsprechend ändern
19: ' (ULText 0 ignorieren)
20: If New_ULText = 1 Then
21: Text = UCase(txtParent.Text)
22: ElseIf New_ULText = 2 Then
23: Text = LCase(txtParent.Text)
24: End If
25: PropertyChanged "ULText"
26: End Property

```

Was Sie tun sollten

Ändern Sie die übergebenen Datentypen in diesen `Get AutoTSize()`- und `Get ULText()`-Prozeduren ab, so daß statt des `Variant`-Datentyps, den sie standardmäßig entgegennehmen, Aufzählungsdaten empfangen werden.

Damit ist das ActiveX-Steuerelement fertig. Jetzt bereiten Sie es darauf vor, daß es in eine andere Applikation eingefügt wird. Außerdem prüfen Sie, ob es so funktioniert, wie Sie das erwartet haben.

Implementierung des ActiveX-Steuerelements

Sie können Ihr neues Steuerelement jetzt nicht nur in eine Applikation einfügen und es im Formularfenster anlegen wie jedes andere Steuerelement - das neue ActiveX- Steuerelement hat wirklich jeden Vorteil, den auch interne Steuerelemente zu bieten haben. Das Eigenschaftenfenster verhält sich also wie gewohnt. Wenn ein Programmierer, der das ActiveX-Steuerelement anwendet, im Codefenster eine Zuweisung eingibt, um einen Eigenschaftswert zu setzen, wird sogar die Quick Info angezeigt, um ihm bei der Auswahl einer Eigenschaft zu helfen. Die Werkzeugsammlung die das ActiveX-Steuerelement enthält, zeigt automatisch einen ToolTip dafür an. Sie werden stolz auf Ihr neues ActiveX-Steuerelement sein, wenn Sie es erst einmal einsetzen, weil es sich genau so verhält, als hätte es Microsoft in Visual Basic eingebaut.

Wenn Sie das ActiveX-Steuerelement kompilieren, kompiliert Visual Basic es in eine ActiveX-Datei, die Sie in ein Projekt einfügen können wie jedes andere ActiveX-Steuerelement. Wenn Sie die OCX-Datei nicht anlegen, können Sie das Steuerelement nicht in Applikationen nutzen.

Speichern Sie Ihr Steuerelement, bevor Sie es kompilieren. Wählen Sie **Datei, Projekt speichern**, um sowohl das Steuerelement als auch das Projekt zu speichern. Es ist nicht möglich, das ActiveX-Steuerelement mit (F5) zu starten, weil alle ActiveX- Steuerelemente kompiliert werden müssen, bevor sie ausgeführt werden können. Der Begriff »ausführen« bedeutet hier natürlich, »sich verhalten wie die anderen Steuerelemente«, wenn ein Programmierer das ActiveX-Steuerelement in einer Applikation einsetzt.

Um das ActiveX-Steuerelement zu kompilieren, wählen Sie **Datei, Projekt erstellen** . Visual Basic zeigt das Dialogfeld Projekt erstellen an (siehe Abbildung 17.10). Legen Sie fest, wo das kompilierte ActiveX-Steuerelement gespeichert werden soll. Sie könnten es in Ihrem Windows\System-Ordner ablegen oder in einem Ordner, den Sie dafür angelegt haben. (Diesen Ordner durchsuchen Sie, wenn Sie das ActiveX- Steuerelement im Dialogfeld **Projekteigenschaften** in die Werkzeugsammlung einer

anderen Visual-Basic-Prozedur laden möchten.) Wenn der Compiler Fehler erkennt, legt Visual Basic das Steuerelement nicht an und markiert die fehlerhaften Zeilen im Code. Sobald Sie die Fehler eliminiert haben, bringt Sie der Compiler wieder in die Entwicklungsumgebung.

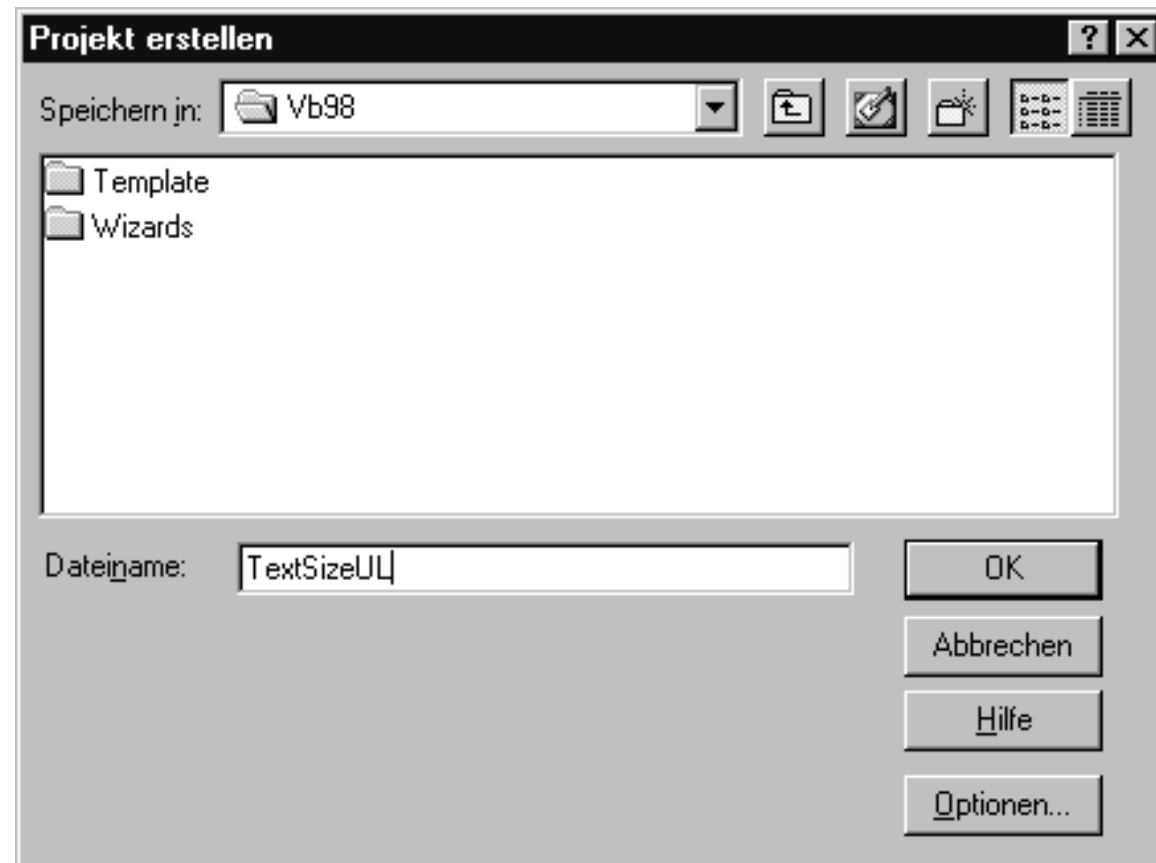


Abbildung 17.10:
Geben Sie den Dateinamen für das ActiveX-Steuerelement ein. Visual Basic speichert es mit der Dateinamenerweiterung .OCX.

Visual Basic bietet zwei Möglichkeiten, das Steuerelement zu testen:

- Öffnen Sie ein neues Projekt, und testen Sie das Steuerelement dort. Es gibt eine Entwicklungsumgebung für Mehrfachprojekte, die speziell für das Testen von ActiveX-Steuerelementen bereitgestellt wird.
- Öffnen Sie ein neues Standard-EXE-Projekt, und legen Sie das Steuerelement in diesem neuen Projekt an.

Jetzt testen Sie Ihr neues Steuerelement, indem Sie es wirklich einsetzen. Wählen Sie **Datei, Neues Projekt**, und legen Sie eine neue Standard-EXE-Datei an. Drücken Sie (Strg)+(T), um das Dialogfeld **Komponenten** zu öffnen. In Abbildung 17.11 sehen Sie das Steuerelement etwa in der Mitte des Auswahlfelds.

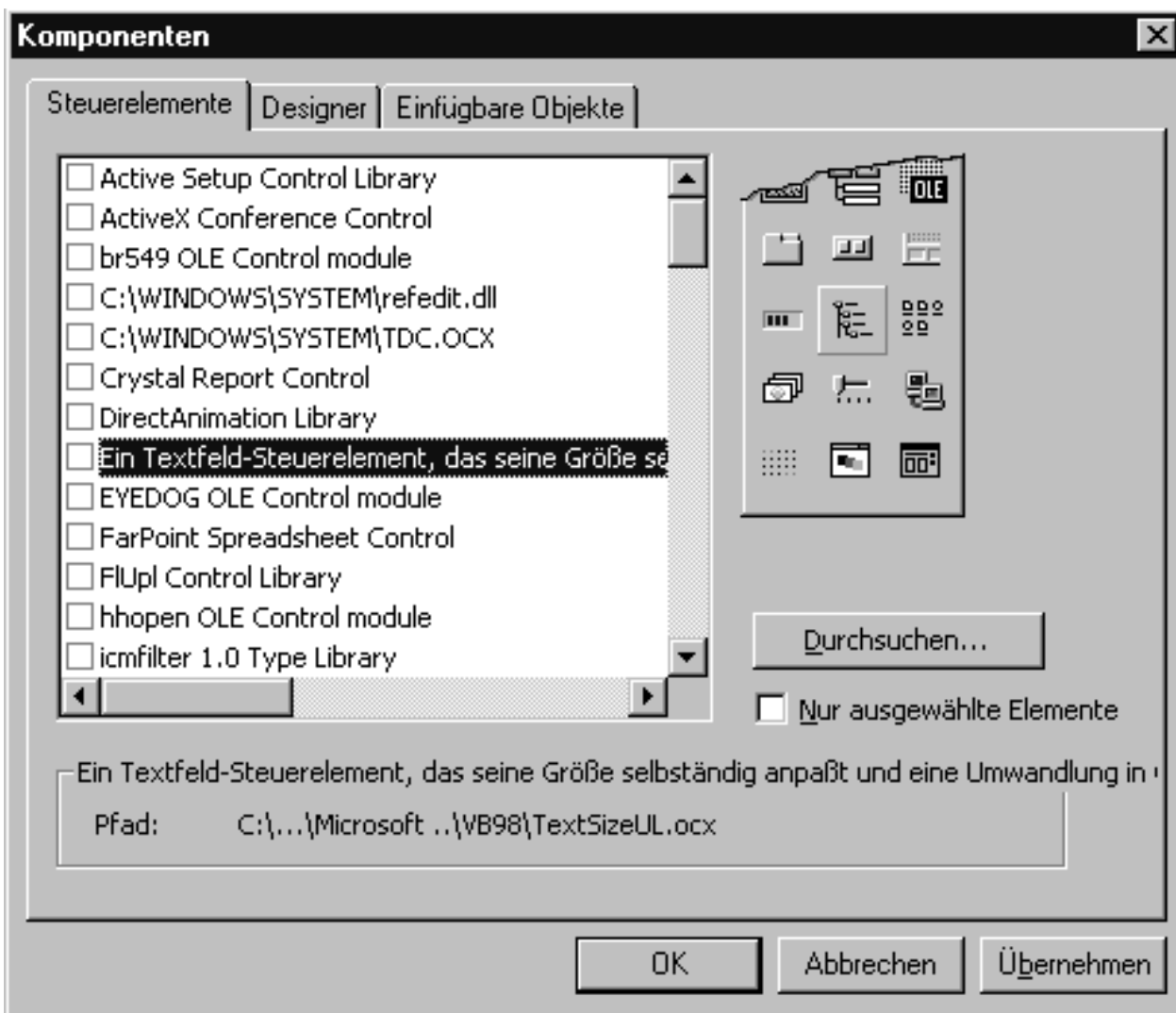


Abbildung 17.11:

Der Text für die Beschreibung Ihres ActiveX-Steuerelements wird im Dialogfeld Komponenten angezeigt.

Wählen Sie das ActiveX-Steuerelement aus, und schließen Sie das Dialogfeld. Das Bitmapbild, das Sie dafür angelegt haben, erscheint in der Werkzeugsammlung. Um das Steuerelement in einer einfachen Applikation einzusetzen, gehen Sie wie folgt vor:

Setzen Sie den Formularnamen auf `frmActiveX`, und verwenden Sie den Titel `Test` des ActiveX-Steuerelements. Setzen Sie `Width` und `Height` auf 7575 bzw. 5775.

Zeigen Sie auf das `TextSizeUL`-Steuerelement, und lesen Sie den ToolTip, den der ActiveX-Schnittstellen-Assistent für Sie angelegt hat. Der Name ist recht langweilig: `NewControl`. Dieser Name wird benutzt, weil Sie das Steuerelement unter diesem Projektnamen abgelegt haben. In dieser Übung sind wir jedoch mehr daran interessiert, was das Steuerelement macht, und nicht daran, wie es heißt.

Doppelklicken Sie auf das neue Steuerelement, um es dem Formular hinzuzufügen. (Sie könnten es auch auf das Formularfenster ziehen.) Beachten Sie, daß das Steuerelement wie ein normales Textfeld aussieht, bis auf die beiden zusätzlichen Eigenschaften. Setzen Sie es auf eine Größe von etwa 4815 Twips (`Width`) mal 1215 Twips (`Height`). Ändern Sie die `FontSize`-Eigenschaft auf 18, die `FontBold`-Eigenschaft auf `True`.

Klicken Sie für die `ULText`-Eigenschaft auf den Pfeil, um das Dropdown-Listenfeld zu öffnen. Sie sehen drei Aufzählungswerte, `Gleichbleibend`, `Großbuchstaben` und `Kleinbuchstaben`,

so wie Sie sie programmiert haben. Behalten Sie für dieses Beispiel den Vorgabewert bei.

Klicken Sie auf die `AutoSize`-Eigenschaft, um die Aufzählungswerte anzuzeigen. Verwenden Sie auch hier den Vorgabewert.

Setzen Sie die `Name`-Eigenschaft auf `MyFirstCtl`, und löschen Sie die `Text`-Eigenschaft.

Legen Sie fünf Schaltflächen an, und weisen Sie ihnen die in Tabelle 17.1 gezeigten Werte zu.

Tabelle 17.1: Weisen Sie den Schaltflächen auf dem Formular diese Eigenschaften zu.

Steuerelement	Eigenschaftswert
Befehlsschaltfläche #1 Name	<code>cmdSmall</code>
Befehlsschaltfläche #1 Caption	<code>&Klein</code>
Befehlsschaltfläche #1 Left	<code>1320</code>
Befehlsschaltfläche #1 Top	<code>2640</code>
Befehlsschaltfläche #2 Name	<code>cmdMedium</code>
Befehlsschaltfläche #2 Caption	<code>&Mittel</code>
Befehlsschaltfläche #2 Left	<code>3120</code>
Befehlsschaltfläche #2 Top	<code>2640</code>
Befehlsschaltfläche #3 Name	<code>cmdLarge</code>
Befehlsschaltfläche #3 Caption	<code>&Groß</code>
Befehlsschaltfläche #3 Left	<code>4920</code>
Befehlsschaltfläche #3 Top	<code>2640</code>
Befehlsschaltfläche #4 Name	<code>cmdUpper</code>
Befehlsschaltfläche #4 Caption	<code>Groß&buchstaben</code>
Befehlsschaltfläche #4 Left	<code>2160</code>
Befehlsschaltfläche #4 Top	<code>3600</code>
Befehlsschaltfläche #5 Name	<code>cmdLower</code>
Befehlsschaltfläche #5 Caption	<code>K&leinbuchstaben</code>
Befehlsschaltfläche #5 Left	<code>3960</code>
Befehlsschaltfläche #5 Top	<code>3600</code>

Fügen Sie die in Listing 17.7 gezeigten Ereignisprozeduren ein.

Listing 17.7: Diese Ereignisprozeduren testen das neue ActiveX-Steuerelement

```
1: Private Sub cmdSmall_Click()  
2: ' Umwandlung in kleinen Text prüfen  
3: MyFirstCtl.AutoSize = Small
```

```
4: End Sub
5:
6: Private Sub cmdMedium_Click()
7: ' Umwandlung in mittelgroßen Text prüfen
8: MyFirstCtl.AutoTSize = Medium
9: End Sub
10:
11: Private Sub cmdLarge_Click()
12: ' Umwandlung in großen Text prüfen
13: MyFirstCtl.AutoTSize = Large
14: End Sub
15:
16: Private Sub cmdUpper_Click()
17: ' Umwandlung in Kleinbuchstaben prüfen
18: MyFirstCtl.ULText = Uppercase
19: End Sub
20:
21: Private Sub cmdLower_Click()
22: ' Umwandlung in Großbuchstaben prüfen
23: MyFirstCtl.ULText = Lowercase
24: End Sub
```

Beachten Sie bei der Codeeingabe, daß Visual Basic Ihnen hilft, die Eigenschaftswerte des neuen ActiveX-Steuerelements einzugeben. Sobald Sie das Gleichheitszeichen gedrückt haben (siehe Abbildung 17.12), erscheint ein Listenfeld mit Auswahlmöglichkeiten. Die hier angebotenen Auswahlen sind die einzigen möglichen Werte, die Visual Basic dem Steuerelement zuordnet. Diese Auswahlliste ist wirklich sehr praktisch - insbesondere wenn man bedenkt, daß Sie nichts tun mußten, um sie anzulegen.

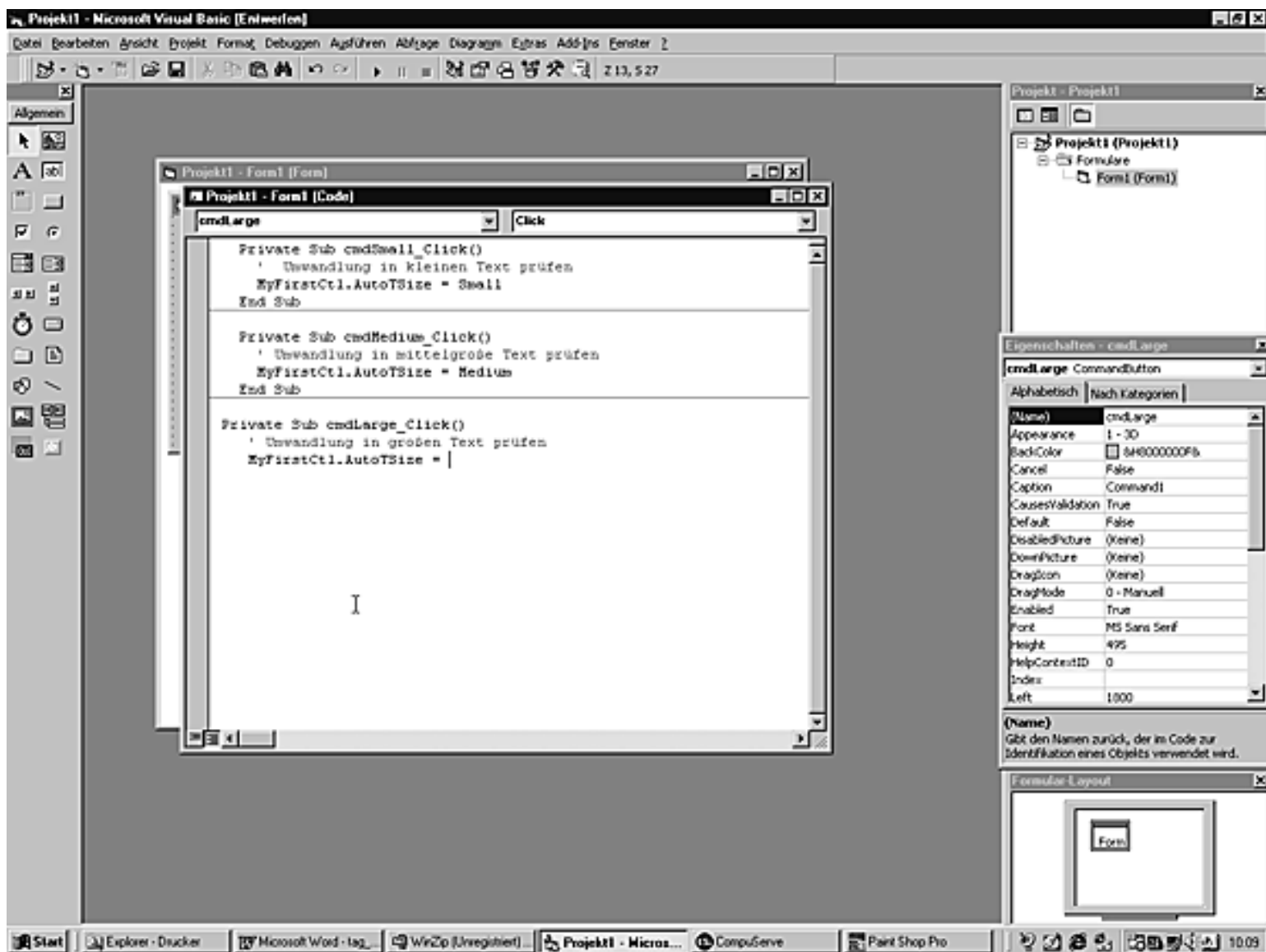


Abbildung 17.12:
Ihr ActiveX-Steuerelement unterstützt das Popup-Listenfeld für Quick Info.

Kompilieren Sie die Applikation, und führen Sie sie aus. Geben Sie in das Textfeld des ActiveX-Steuerelements einen Wert ein, der sich aus Groß- und Kleinbuchstaben zusammensetzt. Klicken Sie auf die drei Schaltflächen für die Schriftgrößen, und beobachten Sie, was passiert. Sie wissen, daß diese Schaltflächen nicht mit einer `Font . Size`-Eigenschaft verknüpft sind, sondern neue Eigenschaftswerte verwenden, die Sie für das Steuerelement angelegt haben. Klicken Sie auf die Schaltflächen, die die Schrift in Klein- bzw. Großbuchstaben umwandeln, um zu sehen, wie sich der Text ändert.


Hinweis

Nachdem Sie den Text im ActiveX-Steuerelement in Groß- oder Kleinbuchstaben umgewandelt haben, ist die ursprüngliche Aussehen des Texts verloren.

Abbildung 17.13 zeigt die Applikation bei der Ausführung.



Abbildung 17.13:
Die Eigenschaften des ActiveX-Steuerelements nehmen jetzt Umwandlungen vor.

Zusammenfassung

Im heutigen Kapitel wurde erklärt, wie man mit speziellen ActiveX-Objekten umgeht. ActiveX-Objekte fügen Ihrer Werkzeugsammlung nicht nur neue Steuerelemente hinzu, sondern können auch in anderen Windows-Applikationen eingesetzt werden, beispielsweise in Visual C++ oder Internet-Browsern. Aus diesem Grund werden von den verschiedensten Anbietern ActiveX-Steuerelemente bereitgestellt. Nachdem Sie gelernt haben, wie man ein ActiveX-Steuerelement verwendet, können Sie es auch in anderen Applikationen einsetzen.

Mit dem ActiveX-Schnittstellen-Assistenten können Sie eigene ActiveX-Steuerelemente entwickeln. ActiveX-Steuerelemente können als Unterklassen von bereits existierenden Steuerelementen angelegt werden (auch wenn es sich bei diesen Steuerelementen um ein ActiveX-Steuerelement handelt, das Sie oder jemand anderer entwickelt hat). Irgendwann werden Sie eine ganze Bibliothek aus Steuerelementen besitzen, die Ihnen hilft, andere Steuerelemente und Applikationen anzulegen.

Im nächsten Kapitel erfahren Sie, wie Visual-Basic-Applikationen in Datenbanken integriert werden, damit Ihre Programme auf große Datenmengen zugreifen und diese verarbeiten können.

Fragen und Antworten

Frage:
Warum sollte ich in meiner Werkzeugsammlung neue Gruppenregisterkarten anlegen?

Antwort:

Die Registerkarten sollen Ihnen helfen, Ihre Steuerelemente sinnvoll anzuordnen. Durch die Gruppierung der Steuerelemente finden Sie die Werkzeuge schneller, die Sie für eine bestimmte Aufgabe brauchen. Wenn Sie beispielsweise eine Visual-Basic-Applikation entwickeln, die mit Datenbanken zu tun hat, gruppieren Sie die betreffenden Steuerelemente auf einer eigenen Registerkarte, um einen gezielteren Zugriff darauf zu haben. Sie müssen dann nicht alle Steuerelemente auf der Registerkarte **Allgemein** betrachten, um dasjenige zu finden, das Sie gerade brauchen. Trotz der Verteilung auf Registerkarten stehen Ihnen alle Steuerelemente der Werkzeugsammlung ständig zur Verfügung.

Frage:

Kann ich die Registerkarten wieder entfernen?

Antwort:

Natürlich. Klicken Sie mit der rechten Maustaste auf einen Gruppennamen. In dem Popup-Menü gibt es die Option **Gruppe entfernen**. Hier können Sie die Registerkarten gegebenenfalls auch umbenennen.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist *Automatisierung*?
2. Was passiert, wenn Ihre Applikation `CreateObject()` für ein Word-Dokument verwendet und Word bereits läuft?
3. Warum können Sie Objektvariablen keine Applikationen direkt zuweisen?
4. Welche Aufgabe hat das Systemobjekt `Err.Number`?
5. Nennen Sie drei Möglichkeiten, ActiveX-Steuerelemente zu erstellen.
6. Wie erstellt man am einfachsten ein ActiveX-Steuerelement?
7. Richtig/Falsch. Wenn Sie eine Unterklasse für ein Steuerelement anlegen, übernimmt das neue ActiveX-Steuerelement die Eigenschaften, Ereignisse und Methoden von dem übergeordneten Steuerelement.
8. Wofür werden Aufzählungen verwendet?
9. Welche Dateinamenerweiterung gibt Visual Basic kompilierten ActiveX-Steuerelementen?
10. Welche beiden Prozeduren braucht man für die Eigenschaften von ActiveX-Steuerelementen?

Übungen

1. Suchen Sie im Dialogfeld Komponenten nach ActiveX-Steuerelementen auf Ihrer Festplatte. Sie finden möglicherweise auch ActiveX-Steuerelemente, die nicht in Ihrem Windows-Ordner abgelegt sind. Wenn Sie beispielsweise Mitglied des Online-Dienstes Microsoft Network sind, finden Sie im Ordner Microsoft Network mehrere ActiveX-Steuerelemente.

[❖Kapitel](#) [Inhalt](#) [Index](#) [❖Top](#) [Kapitel❖](#)

Woche 3

Bonusprojekt 8: Elemente, die Ihre Applikationen verschönern

Dieses Bonusprojekt beschreibt eine Applikation, die die folgenden Elemente enthält:

- Ein **Info**-Dialogfeld, das mit der Menüoption **Hilfe, Info** angezeigt wird.
- Eine Sounddatei, die automatisch abgespielt wird, sobald das **Info**-Dialogfeld erscheint.
- Ein Bildfeld, das eine einfache Animation anzeigt.
- Einen Timer, der die Animation regelt.
- Ein Array mit Bildfeldern.

Dieses Projekt enthält nichts Außergewöhnliches. Trotz des animierten Bildfelds sehen Sie nur eine recht einfache Animation. Das ist jedoch nicht weiter schlimm, weil Sie hier das Prinzip kennenlernen sollen, wie Objekte in Bildfeldern bewegt werden. Außerdem werden Sie üben, ein **Info**-Dialogfeld anzulegen und WAV-Dateien in Ihren Applikationen zu nutzen.

Das Ziel der Applikation

Abbildung BP8.1 zeigt das Formular, wie es während der Animation aussieht. (Wenn Sie die Applikation zum ersten Mal ausführen, ist der Umschlag noch geschlossen, und der Brief befindet sich nicht auf dem Bildschirm.) Das Formular ist einfach und groß genug, so daß der Brief aus dem geöffneten Umschlag fliegen kann, sobald der Benutzer auf die Steuerelement-Animation klickt.



Nachdem der Benutzer die Animation gestartet hat, fliegt der Brief aus dem Umschlag, und auf der Schaltfläche erscheint der Titel »Stop«.



Abbildung BP8.1:
Die Applikation zeigt einfach nur einen Brief, der aus einem Umschlag herausfliegt.

Abbildung BP8.2 zeigt das **Info**-Dialogfeld, das erscheint, wenn der Benutzer auf **Hilfe** , **Info** klickt.



Abbildung BP8.2:

Wenn der Benutzer dieses Info-Dialogfeld anzeigt, wird eine WAV-Datei abgespielt.

Das Hauptformular

Tabelle BP8.1 beschreibt die Steuerelemente und Eigenschaften für das Hauptformular beim Starten der Applikation. Sie müssen den Graphics-Ordner von Visual Basic installiert haben; andernfalls legen Sie Ihre Visual-Basic-CD ein und öffnen den Graphics-Ordner, oder Sie installieren den Graphics-Ordner mit Setup nachträglich. Um das Multimedia-Steuerelement in die Werkzeugsammlung einzufügen, drücken Sie (Strg)+(T) und öffnen damit das Dialogfeld **Komponenten**.

	Viele der Steuerelemente, die Sie zur Entwurfszeit in diesem Projekt anlegen, beispielsweise der Timer oder das Bildfeld, befinden sich am Formularrand, um sie damit aus dem Weg zu schaffen, während Sie die weiteren Steuerelemente einfügen. Der Timer wird Ihrem Benutzer nie angezeigt, Sie können ihn also an einer beliebigen Position ablegen. Die Bildfelder werden bewegt, um eine Animation darzustellen, deshalb ist es auch egal, wo sie zu Anfang platziert werden.
	Dieses Projekt enthält drei Bildfelder, die Teil eines Steuerelementfelds sind. Sie könnten auch drei separate Bildfelder anlegen, aber die Verwendung des Steuerelementfelds ist eine gute Übung. Dazu legen Sie das erste Bildfeld an, <code>picAni2</code> (das Bildfeld <code>picAni1</code> ist nicht Teil des Felds, sondern ein eigenständiges Steuerelement). Nachdem Sie <code>picAni2</code> angelegt und ihm seine Eigenschaften zugeordnet haben, kopieren Sie es mit Bearbeiten, Kopieren in die Zwischenablage. Nachdem Sie Bearbeiten, Einfügen ausgeführt haben, antworten Sie mit Ja, wenn Sie in einem Dialogfeld gefragt werden, ob Sie ein Steuerelementfeld anlegen möchten. Visual Basic macht <code>picAni2</code> zu <code>pinAni2(0)</code> , das erste Element im Steuerelementfeld. Fügen Sie das Bildfeld noch einmal ein, um das dritte Element im Steuerelementfeld <code>picAni2</code> anzulegen.

Damit Sie sich besser vorstellen können, wie das Formular aussehen soll, betrachten Sie Abbildung BP8.3. Dort sehen Sie, wie das Formular aussieht, nachdem alle Steuerelemente aus Tabelle BP8.1 eingefügt wurden. Das Programm verbirgt einige der Steuerelemente bei der Ausführung und animiert andere, so daß es aussieht, als flöge ein Brief aus einem Umschlag heraus.

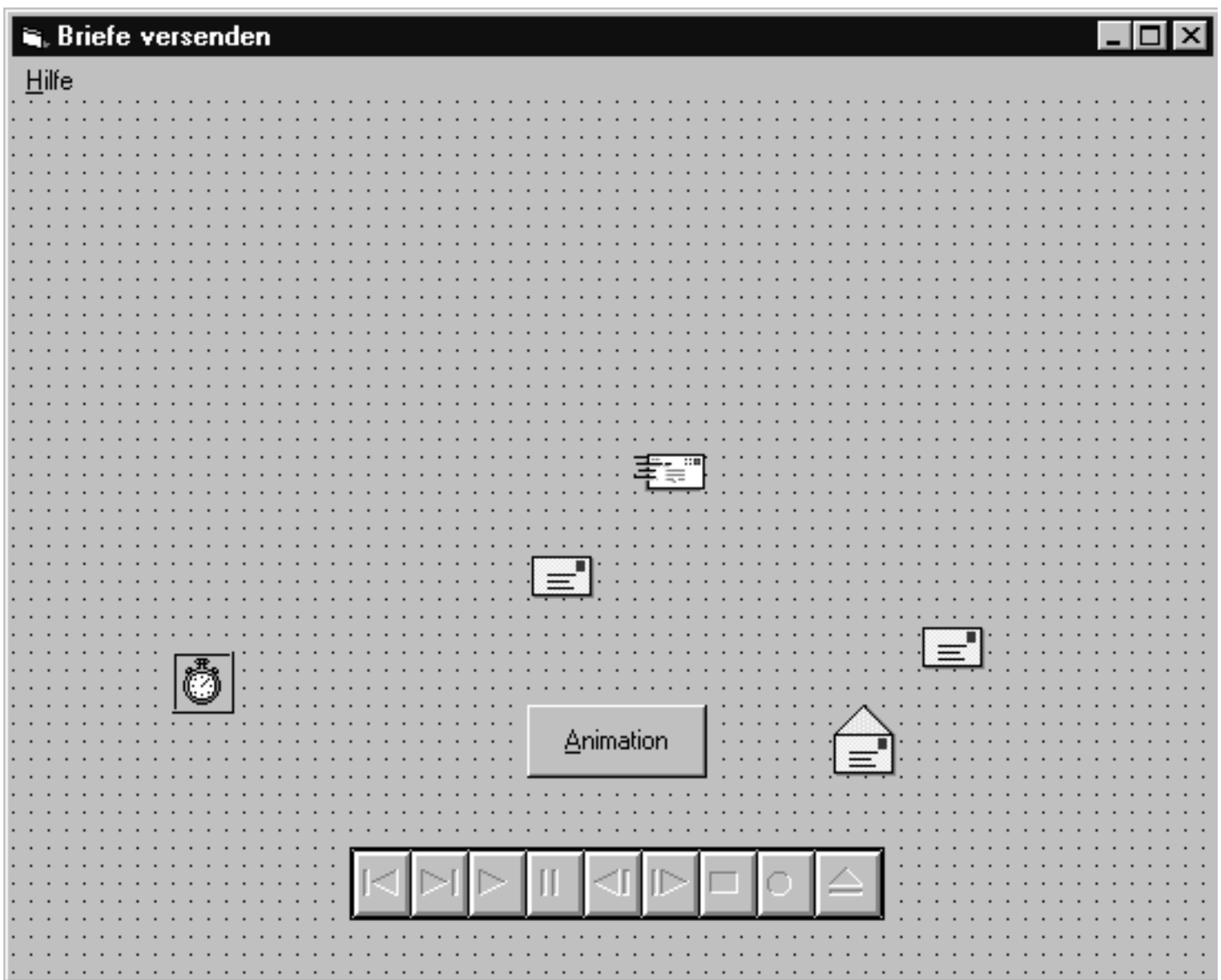


Abbildung BP8.3:
Ihr Formular sieht so aus, nachdem Sie die Steuerelemente darauf angelegt haben.

Tabelle BP8.1: *Legen Sie die Steuerelemente mit den hier beschriebenen Eigenschaften auf dem Formular an.*

Steuerelementeigenschaft	Eigenschaftswert
Formular Name	frmEnvelope
Formular Caption	Briefe versenden
Formular Height	5790
Formular Width	7845
Menüoption #1 Name	mnuHelp
Menüoption #1 Caption	&Hilfe
Menüoption #2 Name	mnuHelpAbout
Menüoption #2 Caption	&Info
Befehlsschaltfläche Name	cmdAni

Befehlsschaltfläche Caption	&Animation
Befehlsschaltfläche Left	2940
Befehlsschaltfläche Top	2880
Timer Name	tmrAni
Timer Enabled	False
Timer Interval	300
Timer Left	1410
Timer Top	3405
Bildfeld #1 Name	picAni1
Bildfeld #1 Height	495
Bildfeld #1 Left	3330
Bildfeld #1 Picture	Common\Graphics\Icons\Mail\Mail01a
Bildfeld #1 Top	1485
Bildfeld #1 Width	1215
Bildfeld #2 Name	picAni2(0)
Bildfeld #2 Height	495
Bildfeld #2 Left	5895
Bildfeld #2 Picture	Common\Graphics\Icons\Mail\Mail01a
Bildfeld #2 Top	2520
Bildfeld #2 Width	1215
Bildfeld #3 Name	picAni2(1)
Bildfeld #3 Height	495
Bildfeld #3 Left	5520
Bildfeld #3 Picture	Common\Graphics\Icons\Mail\Mail01b
Bildfeld #3 Top	3240
Bildfeld #3 Visible	False
Bildfeld #3 Width	1215
Bildfeld #4 Name	picAni2(2)
Bildfeld #4 Height	495
Bildfeld #4 Left	3960
Bildfeld #4 Picture	Common\Graphics\Icons\Mail\Mail03
Bildfeld #4 Top	1080

Bildfeld #4 Visible	False
Bildfeld #4 Width	1215
Multimedia-Steuerelement Name	mmcEnv
Multimedia-Steuerelement DeviceType	WaveAudio
Multimedia-Steuerelement PlayEnabled	True
Multimedia-Steuerelement Filename	\Windows\Media\Chimes.wav
Multimedia-Steuerelement Left	2520
Multimedia-Steuerelement Top	4080
Multimedia-Steuerelement Visible	False
Multimedia-Steuerelement Width	3540



Der Sound Chimes.wav, der abgespielt wird, wenn der Benutzer **Hilfe, Info** auswählt, ist eine Standard-Sounddatei von Windows.

Der Code für das Hauptformular

Listing BP8.1 zeigt den Code, den Sie brauchen, um das Formular zu öffnen. Er aktiviert das Formular und steuert die Animation, die stattfinden soll. Die Animation ist einfach. Sie verschiebt ein Bildfeld an drei verschiedene Positionen auf dem Bildschirm.

Listing BP8.1: Der Code für die Animation ist ganz einfach

```

1: Private Sub cmdAni_Click()
2: ' Schaltfläche zur Steuerung der Animation
3: If cmdAni.Caption = "&Animation" Then
4: cmdAni.Caption = "&Stop"
5: tmrAni.Enabled = True
6: Else
7: cmdAni.Caption = "&Animation"
8: tmrAni.Enabled = False
9: End If
10: End Sub
11:
12: Private Sub mnuHelpAbout_Click()
13: mmcEnv.Command = "Open"
14: mmcEnv.Command = "Play"
15: frmAbout.Show
16: End Sub
17:
18: Private Sub tmrAni_Timer()
19: ' Die richtige Position auswählen, an der
20: ' das Bild angezeigt werden soll

```

```

21: '
22: ' Die folgende Variable beginnt bei Null und
23: ' behält ihren Wert für jede Ausführung der
24: ' Prozedur bei.
25: Static intCounter As Integer
26:
27: Select Case intCounter
28: Case 0:
29: picAni1.Picture = picAni2(1).Picture
30: picAni2(2).Visible = True
31: picAni2(2).Left = 3840
32: picAni2(2).Top = 1220
33: intCounter = 1
34: Case 1:
35: picAni1.Picture = picAni2(1).Picture
36: picAni2(2).Visible = True
37: picAni2(2).Left = 4040
38: picAni2(2).Top = 1120
39: intCounter = 2
40: Case 2:
41: picAni1.Picture = picAni2(1).Picture
42: picAni2(2).Visible = True
43: picAni2(2).Left = 4240
44: picAni2(2).Top = 1220
45: intCounter = 3
46: Case 3:
47: picAni1.Picture = picAni2(0).Picture
48: picAni2(2).Left = 4440
49: picAni2(2).Top = 1320
50: intCounter = 4
51: Case 4:
52: ' Animation beenden
53: picAni1.Visible = True
54: intCounter = 0
55: picAni2(2).Visible = False
56: End Select
57: End Sub

```

Analyse

Die Ereignisprozedur `cmdAni_Click()` wechselt den Titel der Schaltfläche von **Animation** auf **Stop**. Wenn die Schaltfläche den Titel Animation hat, ändert Zeile 4 diesen in Stop und aktiviert in Zeile 5 das Timer-Steuerelement. Durch die Aktivierung des Timers wird durch das Click-Ereignis das `tmrAni_Timer()`-Ereignis in Zeile 18 ausgelöst. Die `Interval`-Eigenschaft 300 für den Timer bedeutet, daß die Ereignisprozedur `tmrAni_Timer()` alle 300 Millisekunden ausgeführt wird und die Animation erzeugt. Die Animation wird so lange fortgesetzt, bis der Benutzer auf die Stop-Schaltfläche klickt. Zeile 7 ändert den Titel wieder und Zeile 8 deaktiviert den Timer.

Die Ereignisprozedur `tmrAni_Timer()`, die in Zeile 18 beginnt, wird durch das Timer- Ereignis alle 300 Millisekunden ausgeführt. Diese Prozedur führt der vier Aktionen der **Select Case**-Anweisung in Zeile 27 aus. Die Variable, die bestimmt, welche dieser Aktionen stattfindet, ist statisch. Die statische Variable, die in Zeile 25 deklariert ist, hat den Anfangswert 0 (wie alle statischen Variablen). Nachdem ihr ein Wert zugewiesen wurde, erhalten alle folgenden Ausführungen von `tmrAni_Timer()` den Wert, der `intCounter` in der vorhergehenden Ausführung der Prozedur zugewiesen wurde. `intCounter` kann einen von fünf Werten annehmen, von 0 bis 4, die bei jeder Ausführung der Prozedur aktualisiert werden.

Jeder der ersten drei Fälle führt die folgenden Dinge aus:

- Der geschlossene Umschlag wird zum geöffneten Umschlag (Zeilen 29 und 35).
- Der Brief wird sichtbar (Zeilen 30 und 36).
- Der Brief wird verschoben, indem die **Left**- und **Top**-Eigenschaften des Bildfelds geändert werden (Zeilen 31 und 32).
- Die statische Variable `intCounter` wird inkrementiert, so daß bei der nächsten Ausführung von `tmrAni_Timer()` andere **Case**-Anweisungen in Kraft treten.

Dieser Prozeß wird fortgesetzt, bis der **Case**-Wert gleich 4 ist. Sobald das der Fall ist, zeigt die Prozedur in Zeile 53 den geschlossenen Umschlag noch einmal an und verbirgt das Icon mit dem Brief (Zeile 55). Wenn der Benutzer jetzt nicht auf die **Stop**- Schaltfläche klickt, fliegt der Brief nach 300 Millisekunden wieder aus dem Umschlag, weil Zeile 54 der statischen Variablen den Wert 0 zuweist, so daß der erste **Case**-Fall wieder ausgeführt wird.

Die andere Ereignisprozedur, `mnuHelpAbout_Click()`, steuert die Anzeige des **Info**-Dialogfelds in Zeile 15, nachdem die Zeilen 13 und 14 das Wave-Gerät geöffnet und die WAV-Datei abgespielt haben. Die Eigenschaften des **Info**-Formulars werden im nächsten Abschnitt beschrieben.



Das Info-Dialogfeld enthält zwar Code, der aber aus der Formularvorlage stammt, die Sie benutzen (siehe nächster Abschnitt). Sie brauchen diesen Code nicht zu ergänzen oder zu verändern. Der Code stellt sicher, daß die Routine zur Ermittlung der Systeminformation ausgeführt wird, wenn der Benutzer auf diese Schaltfläche klickt.

Das Info-Dialogfeld

Verwenden Sie für das **Info**-Dialogfeld die Formularvorlage. Nachdem Sie das erste Formular angelegt haben, klicken Sie mit der rechten Maustaste in das Projektfenster und wählen dort **Hinzufügen, Formular**. Wählen Sie **Info**-Dialogfeld, um Ihrem Formular das **Info**-Dialogfeld hinzuzufügen. (Damit werden sowohl das Formular als auch der entsprechende Code erzeugt.)

Tabelle BP8.2 enthält die Steuerelementwerte, die Sie für das **Info**-Dialogfeld brauchen. Diese Steuerelemente befinden sich bereits auf dem Formular. Tabelle BP8.2 beschreibt nur die Eigenschaften, die Sie ändern sollten.

Tabelle BP8.2: Setzen Sie für das Info-Dialogfeld diese Eigenschaften.

Steuerelementeigenschaft	Eigenschaftswert
<code>lblDescription Caption</code>	Eine einfache Animation mit Sound

lblDescription	Schriftgröße	14
lblDescription	Schriftstil	Fett
lblDisclaimer	Caption	Warnung: Der Programmierer war hier!

Woche 3

Tag 18

Auf Daten zugreifen

Dieses Kapitel erklärt, wie Sie innerhalb Ihrer Visual-Basic-Applikationen auf Datenbanken zugreifen. Eine *Datenbank* besteht aus mehreren Dateien, die in ihrer Kombination ein vollständiges System zur Datenverwaltung bilden. Ein Datenbanksystem, wie beispielsweise Microsoft Access, erzeugt eine Datenbank, auf die Ihre Visual- Basic-Applikation zugreifen kann. Mit Hilfe spezieller Datenbank-Steuerelemente und -Methoden kommunizieren Ihre Applikationen mit der Datenbank.

Heute lernen Sie die folgenden Dinge kennen:

- Auf welche Datenbanken kann Visual Basic zugreifen?
- Datenbankterminologie
- Die Bedeutung von Indexfeldern
- Der Visual Data Manager
- Das Daten-Steuerelement
- Das ADO-Steuerelement im Vergleich zum Daten-Steuerelement
- Im Anwendungs-Assistenten von Visual Basic Tabelle analysieren und Formulare erstellen

Datenbankdaten und Visual Basic

Durch die Möglichkeit, mit Datenbanken zu arbeiten, bietet Ihnen Visual Basic die Leitungsfähigkeit, große Datenmengen innerhalb eines Visual-Basic-Programms zu manipulieren. Visual Basic kann auf Datenbanken der verschiedensten Arten und Formate zugreifen. Es unterstützt die folgenden gebräuchlichen Datenbankformate:

- Microsoft Access
- dBASE
- FoxPro
- Lotus-Tabellenkalkulations-Datenbanken
- ODBC-konforme Datenbanken
- Paradox
- Textdateien mit durch Komma voneinander abgetrennten Daten



Eine *Datenbank* ist eine organisierte Datenmenge. Die Daten sind in der Regel nicht textbasiert, so wie Daten in einer Textverarbeitung, sondern basieren auf Elementgruppen, die Sie verwalten wollen, etwa für Personal, Kunden, Hersteller, Inventar, Bücher oder Software (d.h. alles, was verwaltet werden muß, wofür Sie Berichte ausdrucken wollen oder was aktuell gehalten werden soll). Häufig verwenden Sie ein Datenbanksystem, wie beispielsweise Microsoft Access, um die Datenbankstruktur anzulegen und zu verwalten. Visual Basic kann auf die Datenbank zugreifen, die Sie im Datenbanksystem anlegen.



Visual Basic kann auf Daten verschiedener Datenbanksysteme zugreifen und sie verwalten. Solange Sie eine Version benutzen, die unter Windows 95 läuft, können Sie davon ausgehen, daß Visual Basic das Datenformat unterstützt (es unterstützt außerdem einige Versionen der aufgelisteten Datenbanksysteme, die aus der Zeit vor Windows 95 stammen).

Um zu verstehen, wie Visual Basic die Verwendung von Datenbanken unterstützt, müssen Sie die Datenbankterminologie kennen. Dieses Kapitel kann Ihnen nur einige Grundlagen vermitteln. Sie lesen dieses Buch, um Visual Basic zu erlernen und nicht die Datenbanktechnologie.

Sie brauchen jedoch kein Datenbankexperte zu sein, um die Techniken zu beherrschen, die Visual Basic für den Zugriff auf Datenbanken nutzt. Visual Basic beinhaltet zahlreiche Steuerelemente, die Datenbankzugriff in einer strukturierten Umgebung unterstützen. Das restliche Kapitel stellt Ihnen diese Werkzeuge und die Datenbankprogrammierung im allgemeinen vor.

Die Begriffe

Bei der Arbeit mit Datenbanken sollte man zunächst die Dateien im allgemeinen betrachten. Sie haben hier einen Vorteil, weil Sie bereits in Kapitel 12 gelernt haben, was Datensätze und Felder sind. Abbildung 18.1 demonstriert ein typisches Szenario, das das Konzept von Datensätzen und Feldern verdeutlichen soll. Ein Datensatz ist eine Zeile mit Informationen einer Datendatei, wobei sich ein logischer Datensatz auf zwei oder drei Zeilen in der physischen Datei erstrecken kann. Datensätze sind in Felder oder *Spalten* zerlegt, die helfen, die dort abgelegten Datenelemente zu unterscheiden.



Die Daten in Abbildung 18.1 zeigen 9 Datensätze und 5 Felder. Wenn der Inventarliste weitere Daten hinzugefügt werden, steigt die Anzahl der Datensätze, die Anzahl der Felder bleibt jedoch gleich. Sie können die Anzahl der Felder möglicherweise erhöhen, aber dazu müssen Sie den Entwurf der Datei entsprechend abändern. Das Einfügen von Feldern ist nicht dasselbe wie das Einfügen von Datensätzen.



Die Feldnamen sind nicht Teil der Daten. Sie dienen nur der Feldbeschriftung, so wie Variablennamen den Inhalt der Variablen bezeichnen.

Inventur-Datei					
Datensätze	Teile- Nummer	Beschreibung	Anzahl	Einkaufs- preis	Verkaufs- preis
	XG12	Hammer	47	0.52	1.32
	C1-98	Safeblatt	3	16.73	26.99
	W2A	Schraube	5	9.38	14.95
	KL7	Schraube	62	9.38	14.95
	AT8E	Seilwinde	38	124.30	149.95
	MVP9	Motoröl	4	1.98	4.15
	MVP8	Pinzette	2	4.67	12.20
	ZX3	Radkette	8	19.12	33.20
	A17	Außenspiegel	7	67.13	85.50
Felder					

Abbildung 18.1:
Eine typische Datendatei ist in Datensätze und Felder aufgeteilt.

Die in Abbildung 18.1 gezeigte Datei ist eine sogenannte flache Datenbank, weil sie eigenständig ist und ein Programm ganz einfach auf die Information zugreifen kann - entweder sequentiell oder durch wahlfreien Zugriff. Die Datensätze müssen nicht in einer bestimmten Reihenfolge abgelegt werden. Viele Datenbankentwickler ermöglichen jedoch, die Tabelle nach bestimmten Feldern zu sortieren, beispielsweise nach der Teilenummer, weil es dadurch einfacher wird, mit der Datei zu arbeiten. Die heutigen Datenbanksysteme haben das in der Abbildung gezeigte flache Datenbankkonzept längst überholt.

In der heute verwendeten Datenbankterminologie ist eine *Tabelle* eine Datendatei, und die Datenbank ist die Gesamtheit dieser Tabellen. Ihre Datenbank könnte beispielsweise aus einer Kundentabelle, einer Lieferantentabelle, einer Angestelltentabelle und einer Inventartabelle bestehen. Wenn diese Tabellen in einer einzigen Datenbank angelegt werden, können Programme, die Zugriff darauf haben, alle diese Tabellen gleichzeitig nutzen. Das Programm könnte etwa ermitteln, von welchem Lieferanten ein bestimmter Artikel stammt oder welcher Angestellter ein bestimmtes Produkt verkauft hat. Mit anderen Worten, ein Datenbanksystem kombiniert die einzelnen Tabellen und erzeugt komplexe Informationen daraus.

Zur Ermittlung von Informationen aus einer Datenbank verwendet man eine *Abfrage*. Ein Programm fragt die Datenbank nach Daten ab, wenn es einen bestimmten Datensatz benötigt. Für eine Datenbanktabelle gibt es normalerweise mindestens ein Indexfeld. Ein *Index* ist ein Schlüsselfeld, das für alle Datensätze eindeutige Werte enthält. Der Index verhält sich wie der Index eines Buchs: Wenn Sie auf einen bestimmten Datensatz in einer Tabelle zugreifen möchten, geben Sie den Indexwert an, und die Datenbank springt direkt in diese Zeile, ohne daß die gesamte Tabelle durchsucht werden muß (wie es bei Routinen mit sequentielltem Zugriff der Fall wäre). In Tabelle 18.1 wäre das Feld Teilenummer am besten für den Index geeignet, weil jedes Teil eine eindeutige Teilenummer hat.



Haben Sie auch manchmal das Gefühl, eine Nummer zu sein? Können Sie sich auch alle diese Kontennummern nicht mehr merken (Girokonto, Sparkonto, Autoversicherung, Sozialversicherungsnummer usw.)? Für den Computer sind Zahlen wichtig: Er kann Sie damit schneller erkennen. Wenn eine Firma Informationen dem Namen nach ablegen würde, wäre diese Information häufig schwer wiederzufinden. Beispielsweise könnten mehrere Leute, die ein Konto bei Ihrer Bank haben, denselben Namen wie Sie tragen. Oder Ihr Name ist Mueller, und man sucht nach Müller oder MUELLER - der Computer findet keinen übereinstimmenden Namen. (Computer sind nicht so gewandt!) Bei der Identifikationsnummer im Indexfeld kann man weniger Fehler machen. Und die Firmen sparen Zeit, was letztlich zu besseren Preisen oder günstigeren Zinsbedingungen führt.

Bei der Arbeit mit Datenbanken (anstelle von einzelnen Datendateien) verwenden die Programmierer häufig den Begriff *Tabelle* statt *Datei*, wie Sie bereits gelesen haben. Darüber hinaus bezeichnen sie *Spalten* und *Zeilen* als *Felder* und *Datensätze*. Das ist sinnvoll, weil Dateien, also Tabellen, theoretisch rechteckig sind (in der Realität sehen sie ganz anders aus) und aus Zeilen und Spalten bestehen, wie in Abbildung 18.1 gezeigt. Mehrere Dateien in einer einzelnen Datenbankdatei könnten verwirrend sein, daher der Begriff *Tabelle* für eine Datenmenge (eine Datendatei) in der Datenbank. Darüber hinaus sind die meisten der heutigen Datenbanksysteme *relational*, d.h. die Tabellen enthalten nie dieselben Daten, so daß so wenig Redundanz wie möglich entsteht. Microsoft Access ist eine solche relationale Datenbank. Für Datenbankdateien, die nicht relational sind, wie beispielsweise Dateien vor dBASE 4.0, mußten Sie relativ komplizierten Visual-Basic-Code schreiben, um einen relationalen Zugriff nachzubilden, bevor ein I/O mit den Datenbankwerkzeugen von Visual Basic möglich war.

Die meisten Datenbanken bieten eine eigene Benutzeroberfläche. Beispielsweise können Datenbankdateien vordefinierte Berichte enthalten, die Ausgaben für die Daten erzeugen, Bildschirmformulare, auf denen Dateien angezeigt und vom Benutzer eingegeben werden, gespeicherte Abfragen, so daß der Benutzer nicht jede Abfrage neu erstellen muß, sowie Datenbankdefinitionen, mit deren Hilfe Programme die Datenbank analysieren und das Format unter Verwendung von Standardprozeduren lesen können.

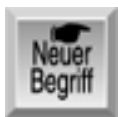
Beispieldaten

Visual Basic enthält die beiden folgenden Beispieldatenbankdateien, die beide im Format von Microsoft Access vorliegen.

- BIBLIO.MDB - Enthält eine Datenbank mit Verlagen und Titeln von Computerbüchern.
- NWIND.MDB - Enthält das vollständige Datenbanksystem einer imaginären Firma, unter anderem mit Tabellen für Inventar, Kunden, Lieferanten, Angestellten, Verkaufsstatistiken und anderem. Der Firmenname ist Northwind Traders, Inc.

Alle Access-Datenbankdateien haben die Dateinamenerweiterung .MDB (für Microsoft Datenbank). Eine Access-Datenbankdatei kann riesengroß sein - eine einzige Datei enthält alle Tabellen, Berichte, Formulare, Bildschirme und gespeicherten Abfragen. Der Vorteil einer einzigen Datei ist, daß Sie die gesamte Datenbank einfacher sichern können (d.h. Sie müssen bei einem Backup nicht immer darauf achten, daß alle Dateien berücksichtigt werden).

Dieses Kapitel verwendet der Einfachheit halber die Beispieldatenbank NWIND.MDB, um einige der datenbezogenen Steuerelemente und Befehle zu demonstrieren. Nicht jeder Visual-Basic-Programmierer hat jedoch Zugriff auf ein externes Datenbanksystem. Glücklicherweise bietet Visual Basic ein spezielles Add-in-Werkzeug, den *Visual Data Manager*, mit dem Sie Datenbankdateien erzeugen und abändern.



Der *Visual Data Manager* ist ein Add-in-Programm, das in der Visual-Basic-Umgebung zur Verfügung steht, um Datenbanken zu erzeugen, Daten einzugeben und zu bearbeiten und die Datenstruktur zu verändern oder darzustellen. Die Dateien, die Sie im Visual Data Manager anlegen, helfen Ihnen, Visual-Basic-Programme zu schreiben und zu testen, die mit ähnlichen Daten zurechtkommen müssen.

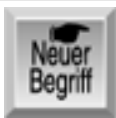


Wenn Sie die Standard-Edition von Visual Basic verwenden, steht Ihnen der Visual Data Manager nicht zur Verfügung. Ihre Version enthält jedoch auf alle Fälle die Datenbankdatei NWIND.MDB, so daß Sie die folgenden Beispiele nachvollziehen können. Leider können Sie jedoch den Visual Data Manager nicht nutzen.

Der Visual Data Manager ist das einzige Standardwerkzeug im Menü **Add-Ins**. Wählen Sie **Add-Ins, Visual Data Manager**. Visual Basic startet den Visual Data Manager und öffnet das **VisData**-Fenster (siehe Abbildung 18.2).

Mit dem Visual Data Manager können Sie die folgenden Aufgaben wahrnehmen:

- Neue Datenbankdateien anlegen
- Neue Daten in Datendateien eingeben
- Existierende Strukturen von Datenbankdateien ändern
- Existierende Datenbankdaten bearbeiten
- Nach Datenbankfeldern, Abfragen und Recordsets suchen



Ein *Recordset* ist eine Menge aus mehreren Datensätzen. Es gibt verschiedene Arten von Recordsets. Das Standard-Recordset umfaßt einfach alle Datensätze (Zeilen) einer Tabelle. Sie können ein neues Recordset anlegen, indem Sie nur Datensätze berücksichtigen, die bestimmte Kriterien erfüllen (beispielsweise »Alle Datensätze, die einen Kontenstand größer DM 500 enthalten«). Ein *Dynaset* ist ein Recordset, das sich ändert, um bestimmte Kriterien zu erfüllen, wenn Sie das Recordset anfordern. Ein *Snapshot* ist ein Recordset für einen bestimmten Zeitpunkt, beispielsweise alle Datensätze, die am letzten Tag im Monat ein bestimmtes Kriterium erfüllen.

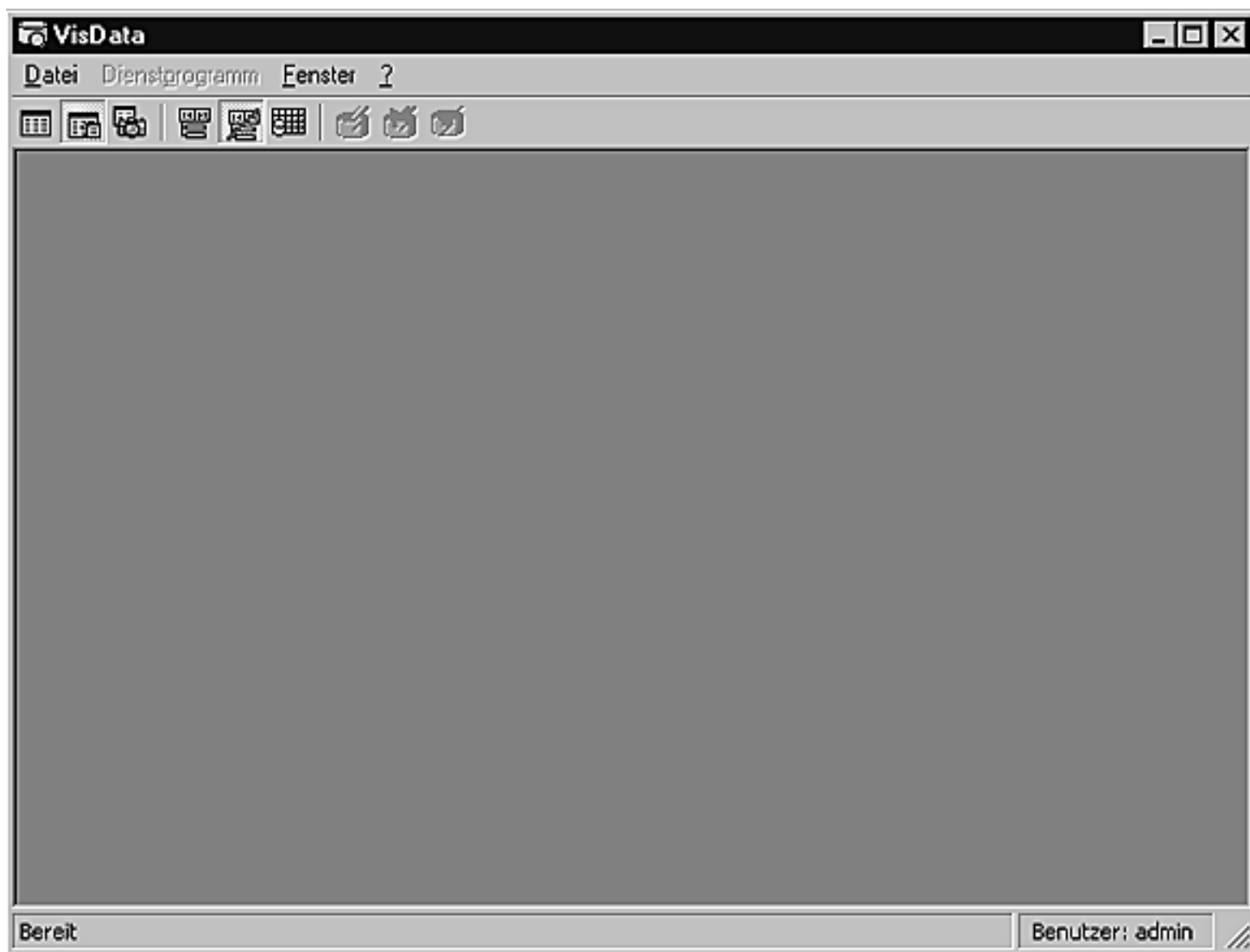


Abbildung 18.2:

Der Visual Data Manager hilft Ihnen, Datenbankdateien zu erzeugen und zu analysieren.

Mit anderen Worten, der Visual Data Manager verhält sich wie ein Datenbanksystem, ganz ähnlich wie Microsoft Access. Beachten Sie jedoch, daß der Visual Data Manager extreme Einschränkungen aufweist und eher eine administrative Möglichkeit darstellt, Datenbankdateien zu analysieren - keinesfalls handelt es sich dabei um ein Datenbanksystem im herkömmlichen Sinn. Sie können beispielsweise keine Berichte und Formulare für den Benutzer der Daten anlegen.



Dieses Kapitel kann nur eine kurze Einführung in die Datenbanktechnologie und -terminologie darstellen. Die Beherrschung eines vollständigen Datenbanksystems wie beispielsweise Microsoft Access dauert fast so lange, wie die Programmierumgebung Visual Basic kennenzulernen.

Hier die allgemeinen Schritte, wie Sie mit dem Visual Data Manager eine neue Datenbank anlegen:

1. Legen Sie die Datenbankdatei an, indem Sie **Datei, Neu** auswählen und dann die gewünschte Datenbank auswählen. Der Visual Data Manager kann alle im ersten Abschnitt dieses Kapitels aufgelisteten Datenbankformate anlegen.
2. Legen Sie die einzelnen Tabellen in der Datenbank an, indem Sie mit der rechten Maustaste in das Datenbankfenster klicken und im Popup-Menü den Eintrag **Neue Tabelle** auswählen. Der Visual Data Manager zeigt das Formular **Tabellenstruktur** an, das Sie in Abbildung 18.3 sehen. Dort geben Sie die Namen der einzelnen Tabellen, die Felder, die Feldtypen (jedes Datenbankfeld unterstützt einen bestimmten Datentyp so wie Variablen in Visual Basic), Sicherheits- und Validierungsinformationen (wie beispielsweise kennwortgeschützte Felder) und Indexfelder (eine Tabelle kann mehrere Indizes haben, abhängig davon, wie

die Applikation auf die Daten zugreift) ein.

Tabellenstruktur

Tabellenname:

Felder:

Feld hinzufügen

Feld entfernen

Name:

Type:

☐ FixedLength

Size:

☐ VariableLength

CollatingOrder:

☐ AutoIncrement

☐ AllowZeroLength

OrdinalPosition:

☐ Required

ValidationText:

ValidationRule:

DefaultValue:

Indizes:

Index hinzufügen

Index entfernen

Name:

☐ Primary

☐ Unique

☐ Foreign

☐ Required

☐ IgnoreNull

Felder:

Tabelle erstellen

Schließen

Abbildung 18.3:
Im Formular Tabellenstruktur werden die Tabelle und ihre Feldstruktur angelegt.

- 3. Klicken Sie auf die Schaltfläche **Tabelle erstellen**, um die Tabellenstruktur aufzubauen und zusätzliche Tabellen hinzuzufügen.
- 4. Klicken Sie auf die Schaltfläche **Daten-Steuerelement in neuem Formular verwenden**, damit der Visual Data Manager ein Werkzeug hat (das Daten-Steuerelement), mit dem die Daten in die Tabelle eingegeben werden.
- 5. Doppelklicken Sie im Datenbankfenster auf einen der Tabellennamen, und geben Sie die verschiedenen Datensätze in das angezeigte Formular ein (siehe Abbildung 18.4).



Hinweis

Wenn Sie zuvor auf die Schaltflächen **Recordset vom Typ Dynaset** oder **Recordset vom Typ Snapshot** geklickt haben, nehmen die Daten, die Sie eingeben, die Form eines dieser komplexen Recordset-Typen an.

Abbildung 18.4:
Geben Sie Daten in die Tabelle ein, um eine funktionierende Datenbank anzulegen.

6. Wählen Sie **Datei, Schliessen**, um die Datei zu schließen und die neue Datenbank zu speichern.

In der Realität ist es viel komplizierter, eine Datenbank anzulegen. Aber Sie haben jetzt eine Vorstellung davon, wie man mit den von Visual Basic bereitgestellten Werkzeugen eine Datenbank anlegt.

	Wenn Sie in Ihren Programmen sehr viel mit Datenbanken arbeiten, sollten Sie sich nicht ausschließlich auf den Visual Data Manager verlassen. Er ist zwar praktisch, um Datenbankformate zu entziffern und einfache Datenbanken zum Testen und Debuggen anzulegen, aber bei weitem nicht so leistungsfähig wie Microsoft Access, FoxPro oder die anderen Datenbanksysteme auf dem Markt. Es ist nicht sinnvoll, mit dem Visual Data Manager die vollständige Datenbank für ein Unternehmen zu entwerfen und anzulegen. Der Visual Data Manager ist nicht flexibel und einfach genug für die tägliche Arbeit mit der Datenbank. Für eine ernsthafte Datenbankprogrammierung sollten Sie ein Datenbanksystem einsetzen.

Der Visual Data Manager bietet mehrere Abfragewerkzeuge, die Ihnen ermöglichen, nach Daten zu suchen und komplexe Abfragen auszuführen, aber das Ganze ist auf die Datenbankadministration beschränkt. Sie benutzen ihn, um Visual Basic auf die Arbeit mit echten Datenbanksystemen vorzubereiten. Nachdem Sie den Visual Data Manager kennengelernt haben, werden Sie erfahren, wie Visual Basic auf die Datenbankdateien zugreift, die Sie angelegt haben - egal, ob mit dem Visual Data Manager oder mit einem Standalone-Datenbanksystem wie dBASE. In diesem Kapitel werden wir zur Demonstration der Arbeit mit Datenbanken die Beispieldatenbankdatei NWIND.MDB verwenden, die in Visual Basic enthalten ist.

Das Daten-Steuerelement

Das Daten-Steuerelement wird von vielen als langsames und müßiges Werkzeug für die Datenverwaltung bezeichnet. Nichtsdestotrotz bietet es die folgenden Vorteile:

- Das Daten-Steuerelement ist einfach, so daß Sie sehr schnell verstehen, wie Visual Basic mit Datenbankdateien arbeitet.
- Das Daten-Steuerelement ist immer in der Werkzeugsammlung enthalten - Sie brauchen es nicht als ActiveX-Steuerelement zusätzlich zu laden.
- Die Standard-Edition von Visual Basic beinhaltet die komplexen Datenbankwerkzeuge nicht, die in der Professional- und der Enterprise-Edition bereitgestellt werden. (In den letzten Abschnitten dieses

Kapitels werden Sie lernen, diese komplexen Datenbankwerkzeuge und -techniken einzusetzen, die dem Visual-Basic- Programmierer helfen, auf andere Arten auf die Daten zuzugreifen, als es das Daten-Steuerelement erlaubt.) Nicht jeder hat die Professional- oder die Enterprise-Edition. Die komplexeren Techniken stehen also nicht jedermann zur Verfügung - im Gegensatz zum Daten-Steuerelement.

Die folgenden Abschnitte beschreiben, wie das Daten-Steuerelement eingesetzt wird, um auf die Datenbank NWIND.MDB zuzugreifen. Wenn Sie die Arbeitsweise dieses Steuerelements kennengelernt haben, wird es Ihnen auch leichterfallen, mit den komplexeren Steuerelementen umzugehen.

Einrichtung des Daten-Steuerelements

Wenn Sie auf eine Datenbank zugreifen möchten, verwenden Sie am einfachsten das Daten-Steuerelement. Wenn Sie das Daten-Steuerelement auf einem Formular anlegen, sieht es ein bißchen aus wie das Multimedia-Steuerelement, das sie in Kapitel 14 kennengelernt haben. Verbreitern Sie das Steuerelement, wie in Abbildung 18.5 gezeigt.

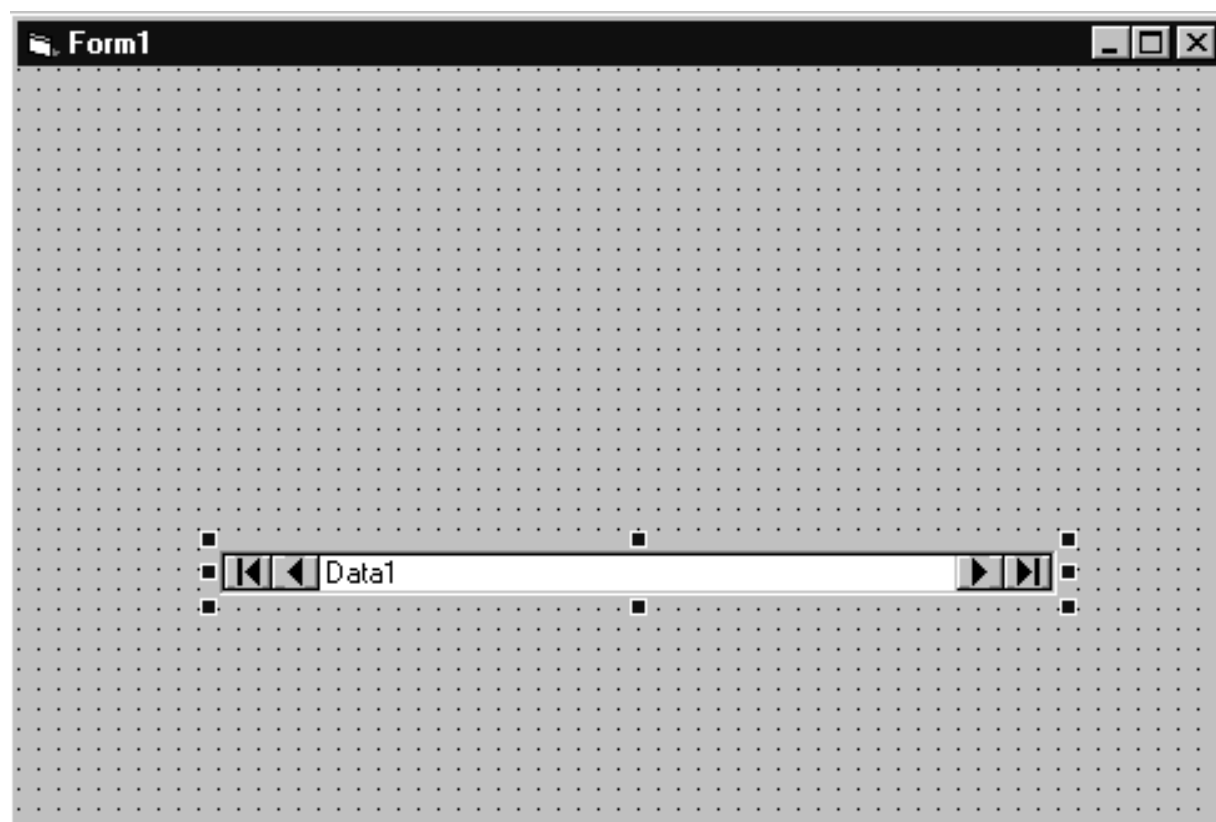


Abbildung 18.5:
Das Daten-Steuerelement enthält Schaltflächen, mit denen Sie eine Datenbank durchlaufen können.

Ein Daten-Steuerelement besteht aus den folgenden Komponenten:

- Zwei inneren Pfeilen, mit denen Sie sich in einer Datenbanktabelle jeweils um einen Datensatz vor oder zurück bewegen.
- Äußeren Pfeilen, die Sie zum ersten und zum letzten Datensatz in einer Datenbanktabelle bringen.
- Einem Bereich in der Mitte, der die Informationen aus der Datenbank enthält, die Sie anzeigen möchten.

Das Daten-Steuerelement ist ein *gebundenes Steuerelement*. Sie können viele Visual-Basic-Steuerelemente zu einer Datenbank binden, beispielsweise ein Textfeld. Wenn ein Benutzer eine Datenbank durchläuft, zeigt das Textfeld die verschiedenen Felder an. Wenn Sie das Steuerelement zur Datenbank binden, müssen Sie sich nicht darum kümmern, wie die Daten aus den Feldern angezeigt werden, weil Visual Basic das für Sie erledigt.



Ein *gebundenes Steuerelement* ist zu Ihrer Datenbank gebunden, so daß es in einem Visual-Basic-Programm einfacher ist, die Datenbank zu durchlaufen.

Wenn Sie jeweils einen Datensatz aus einem Recordset Ihrer Datenbank anzeigen möchten, verwenden Sie ein gebundenes Steuerelement. Gebundene Steuerelemente zeigen in der Regel nur den aktuellen Datensatz oder ein Feld innerhalb des aktuellen Datensatzes an. Wenn der Benutzer die Datenbank durchläuft und dazu die Schaltfläche auf dem Daten-Steuerelement anklickt, reflektiert der aktuelle Datensatz die betreffende Position in der Datenbanktabelle.

Verwendung des Daten-Steuerelements

Um die Arbeitsweise des Daten-Steuerelements kennenzulernen, legen Sie eine Applikation an, die Datensätze aus den Datenbanktabellen von NWIND.MDB anzeigt. Die folgenden Schritte beschreiben, wie diese Applikation angelegt wird:

1. Legen Sie eine neue Applikation an, und fügen Sie dem Formular das Daten-Steuerelement hinzu. Setzen Sie den Wert von `Width` für das Daten-Steuerelement auf 4620, um Platz für den Text zu schaffen, der im mittleren Bereich des Steuerelements angezeigt wird. Verschieben Sie das Steuerelement unten in das Fenster, so daß seine `Top`-Eigenschaft gleich 3240 ist.
2. Geben Sie dem Daten-Steuerelement den Namen `dtaBooks`.
3. Verbinden Sie das Daten-Steuerelement mit Hilfe der `DatabaseName`-Eigenschaft mit der Datenbank. Doppelklicken Sie auf die `DatabaseName`-Eigenschaft, um die Dateien NWIND.MDB aus Ihrem Visual-Basic-Ordner auszuwählen. Das Daten-Steuerelement ist ein intelligentes Steuerelement, das zwischen mehreren Tabellen in einer einzelnen Datenbank unterscheiden kann. Das Daten-Steuerelement kann jeweils nur auf die Datensätze einer einzelnen Tabelle zugreifen (genauer gesagt, auf je ein Recordset gleichzeitig). Nachdem Sie die `DatabaseName`-Eigenschaft spezifiziert haben, erkennt das Daten-Steuerelement die Tabellen innerhalb der Datenbank.
4. Doppelklicken Sie auf die Eigenschaft `RecordSource`, und wählen Sie aus der Liste die Tabelle **Kunden** aus (Kunden ist der Name einer der Datenbanktabellen): Die `RecordSource`-Eigenschaft spezifiziert, welches Recordset im Daten-Steuerelement angezeigt wird, wenn die Applikation auf die Datenbank zugreift.
5. Setzen Sie den Titel des Daten-Steuerelements auf **Klicken Sie hier, um den nächsten Kunden anzuzeigen**. Der Titel erscheint in der Mitte des Daten-Steuerelements.
6. Fügen Sie ein Bezeichnungsfeld mit den folgenden Eigenschaften ein:
Name: `lblCust`
Alignment: 2-Center
FontSize: 18
Height: 915
Left: 1680
Top: 1440
Width: 4320
7. Öffnen Sie die `DataSource`-Eigenschaft für das Bezeichnungsfeld. Diese Eigenschaft mußten Sie bisher nicht setzen. Sie binden damit das Bezeichnungsfeld zum Daten-Steuerelement. (Ein Bezeichnungsfeld ist eines von mehreren Steuerelementen, die gebunden werden können.) Die `DataSource`-Eigenschaft ermöglicht Ihnen, den Wert `dtaBooks` auszuwählen, weil das die Datenbanktabelle ist, die dem Daten-Steuerelement zugeordnet ist. Wenn sich auf dem Formular mehrere Daten-Steuerelemente befinden und diesen bereits Datenbanken zugeordnet sind, listet die `DataSource`-Eigenschaft sie alle auf, damit Sie

die richtige Quelle für das Bezeichnungsfeld auswählen können.

8. Nachdem Sie das Bezeichnungsfeld zur richtigen Tabelle gebunden haben, müssen Sie ihm noch das richtige Feld aus der Tabelle zuordnen. Öffnen Sie die `DataField` -Eigenschaft, und wählen Sie den Eintrag `Kontaktperson`. Die Daten dieses Felds erscheinen im Bezeichnungsfeld.

9. Jetzt führen Sie die Applikation aus. Drücken Sie (F5). Der Name des ersten Kunden erscheint automatisch im Bezeichnungsfeld. Beachten Sie, daß das Bezeichnungsfeld an das Daten-Steuerelement gebunden ist.

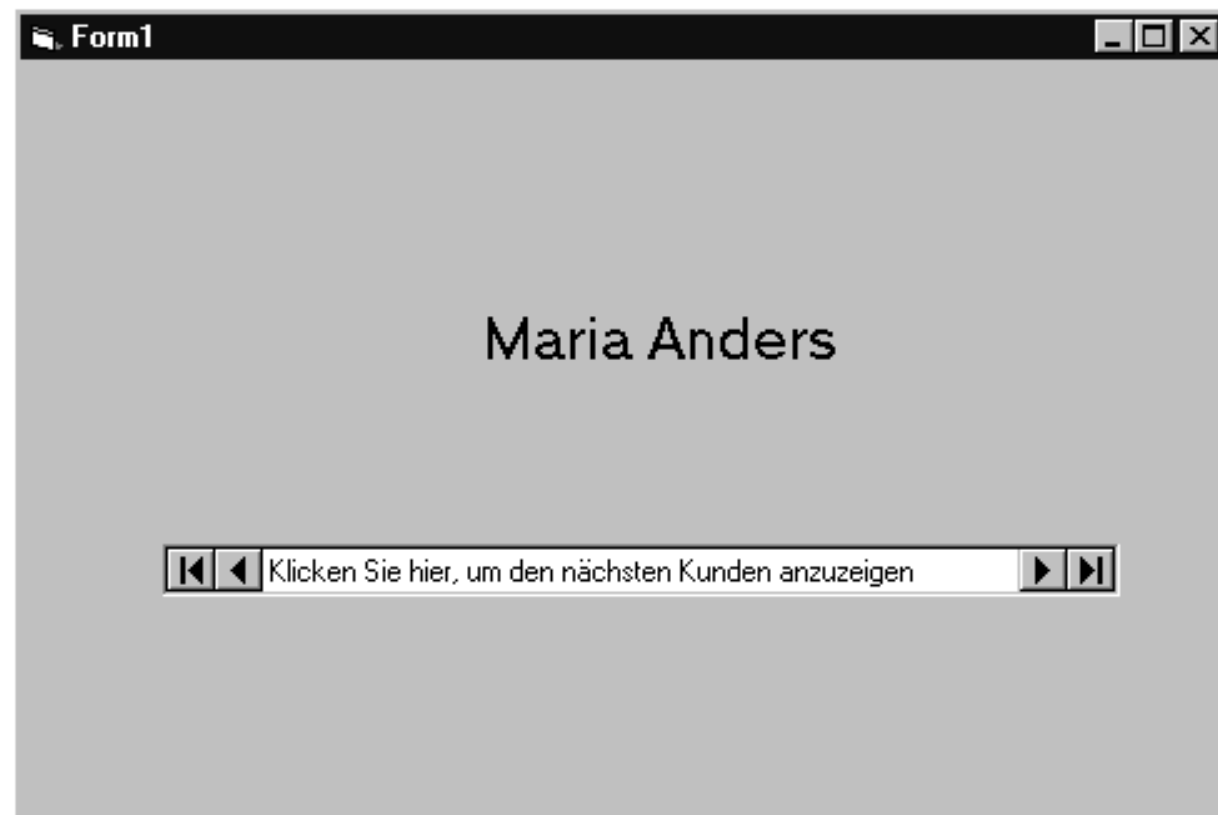


Abbildung 18.6:

Jetzt können Sie die Datensätze der Datenbank durchlaufen - vorwärts oder rückwärts.

10. Klicken Sie auf die **Schliessen**-Schaltfläche des Programmfensters, und beenden Sie das Programm. Speichern Sie Ihr Projekt.

Sie sind natürlich nicht auf eine einzige Spalte der Tabelle beschränkt. Um mehrere Spalten anzuzeigen, legen Sie beliebig viele Bezeichnungsfelder an. Abbildung 18.7 zeigt einen vollständigen Kundendatensatz an. Für die Felder wurden einfach Bezeichnungsfelder angelegt, die zu der zugehörigen Tabellenspalte gebunden wurden. Darüber hinaus helfen zusätzliche Bezeichnungsfelder, die angezeigten Daten zu beschreiben.



Wenn Sie Spalten aus mehreren Tabellen anzeigen möchten, müssen Sie ein Recordset anlegen, beispielsweise ein Dynaset, das die entsprechenden Spalten aus den Tabellen ermittelt. Komplexere Steuerelemente und Befehle ermöglichen Ihnen, eine genauere Datenbankauswahl zu erzielen, wie Sie in den letzten Abschnitten dieses Kapitels sehen werden.



Um Daten anzuzeigen, die als boolescher Datentyp formatiert sind, verwenden Sie ein Kontrollkästchen oder ein Optionsfeld, um die Werte `True` oder `False` auszudrücken. Diese Richtig/Falsch- oder Ja/Nein-Werte kommen in Datenbanktabellen häufig vor.

Kunden-ID:	<u>FAMIA</u>
Kundenname:	<u>Ana Cruz</u>
Firmenname:	<u>Familia Arquibaldo</u>
Kontaktperson:	<u>Marketing-Assistent</u>
Adresse:	<u>Rua Oros, 92</u>
Stadt:	<u>Sao Paulo</u>
Region:	<u>SP</u>

⏮ ⏪
Klicken Sie hier, um den nächsten Kunden anzuzeigen
⏩ ⏭

Abbildung 18.7:
Für die Anzeige mehrerer Tabellenspalten brauchen Sie nur zusätzliche Bezeichnungsfelder.

Komplexer Einsatz des Daten-Steuerelements

Ihre Datenbank-Applikationen sollen natürlich nicht schreibgeschützt sein. Sie möchten, daß Ihre Benutzer die Datenbank aktualisieren. Es gibt mehrere Möglichkeiten, das zu realisieren. Der einfachste Ansatz ist, die Daten des Daten-Steuerelements in Textfeldern statt in Bezeichnungsfeldern auszugeben. Ihre Benutzer sehen die Information wie in den Bezeichnungsfeldern, können sie aber innerhalb der Textfelder abändern, um die Datenbank so zu aktualisieren.



Warnung

Achten Sie immer auf die Sicherheit. Vielleicht fügen Sie ein Kennwort-Dialogfeld ein, um nur bestimmten Benutzern Zugriff auf die Datenbank zu bieten (siehe Kapitel 15). Darüber hinaus könnte auch die Datenbank selbst festlegen, welche Änderungen stattfinden dürfen. Die **Exclusive**-Eigenschaft des Daten-Steuerelements legt fest, ob ein Benutzer exklusive Rechte für eine Datenbank erhalten kann und damit der einzige Benutzer wird, der Zugriff darauf hat (in vernetzten Systemen), aber das Daten-Steuerelement selbst unterstützt nur wenig Sicherheit. Wenn die Sicherheit für Ihre Applikationen wichtig ist, sollten Sie die komplexeren Steuerelemente und Befehle einsetzen, die im restlichen Kapitel vorgestellt werden.

Je besser Sie die Programmiersprache kennen, desto mehr Möglichkeiten werden Sie erkennen, das Daten-Steuerelement für einen komplexen Datenbankzugriff einzusetzen. Beispielsweise unterstützt das Daten-Steuerelement mehrere **Move**-Methoden, mit denen Sie den aktuellen Datensatzzeiger auf dieselben Positionen setzen wie mit den Schaltflächen im Steuerelement. Die folgenden Methoden verschieben den Datensatzzeiger auf den ersten, letzten, nächsten oder vorhergehenden Datensatz in der Datenbank, auf die die **DataSource**-Eigenschaft des Daten-Steuerelements verweist:

```
dtaCust.Recordset.MoveFirst ' Zum ersten Datensatz
dtaCust.Recordset.MoveLast  ' Zum letzten Datensatz
dtaCust.Recordset.MoveNext  ' Zum nächsten Datensatz
dtaCust.Recordset.MovePrevious ' Zum vorhergehenden Datensatz
```



Der Datensatzzeiger verweist auf den aktuellen Datensatz in einer Tabelle der geöffneten Datenbank. Wenn Sie eine Datenbanktabelle zum ersten Mal öffnen, zeigt der Datensatzzeiger auf den ersten Datensatz. Wenn Sie die Tabelle sequentiell durchlaufen, wird der Zeiger in der Tabelle entsprechend weitergeschaltet. Die **Move**-Methoden setzen den Datensatzzeiger so, daß Sie auf beliebige Datensätze in der Tabelle zugreifen können.

Das Standard-Recordset des Daten-Steuerelements wird durch die von Ihnen gesetzten Eigenschaftswerte festgelegt. Beispielsweise können Sie eine Schaltfläche einfügen, die in einer **Click**-Ereignisprozedur eine dieser Methoden verwendet und als Titel den ausgewählten Datensatz anzeigt, wenn der Benutzer auf die Schaltfläche klickt. Neben den Methoden zur Manipulation des Datensatzzeigers gibt es auch Methoden, die Datensätze in die Datenbank einfügen oder daraus entfernen.



Verwenden Sie die booleschen Eigenschaften **BOF** und **EOF**, um zu prüfen, ob sich der Datensatzzeiger am Anfang oder am Ende einer Tabelle befindet. Visual-Basic-Programmierer verwenden häufig **Do . . . While**-Schleifen, um die Datensätze einer Tabelle zu durchlaufen. Die Schleife wird beendet, wenn `dataControl.Recordset.EOF` gleich **True** ist.

Komplexe Datenbank-Steuerelemente

Die Visual-Basic-Editionen Professional, Enterprise und Visual Studio unterstützen zahlreiche Steuerelemente, Eigenschaften, Methoden und Ereignisse, die Sie für Applikationen mit Datenbankzugriff einsetzen. Hier können nicht alle komplexen Datenbankkonzepte vorgestellt werden, aber die Grundlagen sind geschaffen. Wenn Sie in Ihren Applikationen viel mit Datenbanken zu tun haben, wissen Sie damit wenigstens, was Visual Basic dafür tun kann, und Sie werden die Terminologie besser verstehen, nachdem Sie dieses Buch gelesen haben.



In Bonusprojekt 9 werden Sie ein ADO-basiertes Datenbankprojekt kennenlernen.

Beginnend mit Version 6, unterstützt Visual Basic zahlreiche *ADO-Objekte* (ADO steht für *ActiveX Data Objects*). Weil diese Objekte ActiveX-basiert sind, können sie auf unterschiedlichen Plattformen und unter verschiedenen Programmiersprachen eingesetzt werden (anders als das Daten-Steuerelement, das nur in der Visual-Basic-Umgebung funktioniert). Die ADO-Objekte unterstützen Datenbankzugriff sowohl für lokale als auch für Remote-Datenobjekte (*Remote Data Objects*, RDO). Remote-Daten können über ein Netzwerk oder eine Kommunikationsleitung übertragen werden.

Es ist wichtig, daß Sie die ADO-Steuerelemente verstehen, weil sie zahlreiche Vorteile gegenüber dem Daten-Steuerelement aufweisen. Trotz des Hintergrunds, den Sie für die Arbeit mit ADO-Steuerelementen brauchen (Sie müssen noch mehr lernen, um sie wirklich nutzen zu können), werden sie heute von Visual-Basic-Datenbankprogrammierern bevorzugt eingesetzt - weil sie so leistungsfähig und flexibel sind.

Die ADO-Technologie unterstützt einen schnelleren Datenbankzugriff als das Daten-Steuerelement. Die heutigen Computer sind zwar schnell, aber Sie werden den Geschwindigkeitsverlust erkennen, wenn Sie das Daten-Steuerelement für sehr große Datenbanktabellen einsetzen, insbesondere für ODBC-basierte Datenbanken.

Wenn Sie ADO einsetzen, schreiben Sie in der Regel mehr Programmcode als für das Daten-Steuerelement. Sie können zwar Code schreiben, der auf die verschiedenen Methoden des Daten-Steuerelements zugreift, aber ADO ermöglicht Ihnen, den Datenzugriff genauer zu steuern. Das Daten-Steuerelement ist einfach - und deshalb unflexibel. Außerdem werden ADO-basierte Programme nicht durch den Overhead des

Daten-Steuerelements belastet.

Der vielleicht wichtigste Vorteil von ADO ist die Möglichkeit, auf die unterschiedlichsten Daten zuzugreifen. Sie sind damit nicht auf relationale oder nichtrelationale Datenbankinformationen beschränkt. ADO-Steuerelemente können durch eine komplexe Programmierung auch auf Internet-Browser, E-Mail-Text und sogar auf Grafiken zugreifen.



Mehrere ADO-Steuerelemente auf demselben Formular können eine Performanceverschlechterung der Applikation verursachen, weil die Steuerelemente unabhängig voneinander arbeiten und deshalb eigene Ressourcen beanspruchen.

Die ADO-Technologie unterstützt die folgenden Daten-Steuerelemente (in allen Editionen bis zu einem gewissen Ausmaß, vollständig in den Editionen Professional, Enterprise und Visual Studio):

- **ADO-Daten-Steuerelement** - Verhält sich wie das Daten-Steuerelement und stellt eine Verbindung zu einer Datenbank her, wo der Benutzer die Datensätze durchlaufen kann.
- **ADO-DataCombo-Steuerelement** - Erinnert an ein Standard-Kombinationsfeld, bietet aber Ihren Benutzern Zugriff auf eine Spalte mit mehreren Datensätzen für diese Spalte.
- **ADO-DataList-Steuerelement** - Erinnert an ein Standard-Listenfeld, bietet aber Ihren Benutzern Zugriff auf eine Spalte mit mehreren Datensätzen für diese Spalte.



Für diese Steuerelemente gibt es auch Nicht-ADO-Versionen.

Der VB-Anwendungs-Assistent

Visual Basic kann Ihnen einen Großteil Ihrer Arbeit mit Datenbanken abnehmen, wenn Sie den Anwendungs-Assistenten verwenden, um Applikationen mit Datenbankzugriff zu erstellen. Der Assistent bietet zwar nicht den flexiblen Zugriff, den Sie durch die ADO-Programmierung erhalten könnten, aber Sie können eine erste Applikation anlegen und sie dann mit eigenem Code verbessern.

Der Code des Assistenten ist relativ vollständig und bildet die Grundlage für eine echte Datenbankanwendung. Abbildung 18.8 zeigt das Assistentenfenster, das den Datenbankzugriff anbietet.



Abbildung 18.8:
Der Anwendungs-Assistent von Visual Basic erlaubt Ihnen, Datenbankprogramme zu entwickeln.

Angenommen, Sie möchten eine Applikation schreiben, die Datenbankzugriff für die Datei BIBLIO.MDB bietet, so daß Ihre Benutzer diese Datenbank ansehen und aktualisieren könnten. Nachdem Sie den Anwendungs-Assistenten gestartet und die ersten Fenster durchlaufen haben, sehen Sie das in Abbildung 18.8 gezeigte Fenster. Hier teilen Sie dem Assistenten mit, daß Sie Formulare erzeugen möchten, die auf eine Datenbank zugreifen. Sie könnten diese Formulare (mit der SDI-Option) auch selbst anlegen, aber der Assistent kann Ihnen einen Teil der Arbeit abnehmen, so daß Sie nur noch die entsprechenden Anpassungen vorzunehmen brauchen.

Nachdem Sie auf die Schaltfläche **Neues Formular erstellen** geklickt haben, fragt Sie der Assistent nach einer Profileinstellung, falls Sie benutzerdefinierte Profile verwenden möchten. In diesem Beispiel klicken Sie hier auf **Weiter**, um einen Datenbanktyp für die Applikation auszuwählen. BIBLIO.MDB ist eine Microsoft-Access- Datenbank; wählen Sie also **Access**, und klicken Sie auf **Weiter**.

Als nächstes fordert der Assistent einen Datenbanknamen an (siehe Abbildung 18.9). Geben Sie den Pfad und den Dateinamen für die Datei ein. (Sie können auch auf die Schaltfläche **Durchsuchen** klicken, um die Datei zu suchen.)



Abbildung 18.9:
Der Assistent fragt, welche Datenbank verwendet werden soll.

Nachdem Sie eine Datenbank benannt und auf **Weiter** geklickt haben, fragt der Assistent nach dem Namen des Hauptformulars (geben Sie in diesem Beispiel frmADO ein) und nach dem Layout des Formulars. Für das Layout stehe die folgenden Auswahlen zur Verfügung:

- **Einzelner Datensatz** - Ermöglicht den Benutzern, auf jeweils einen einzelnen Datensatz zuzugreifen, ihn anzuzeigen und ihn zu bearbeiten.
- **Tabelle (Datenblatt)** - Ermöglicht den Benutzern, auf mehrere Datensätze gleichzeitig zuzugreifen, sie anzuzeigen und zu bearbeiten, ähnlich wie in einer Tabelle.
- **Master/Detail** - Ermöglicht den Benutzern, auf Detail-Datensätze zuzugreifen, anzuzeigen und zu bearbeiten, die einem einzigen Master-Datensatz entsprechen; beispielsweise alle Produkte (der Detail-Datensatz), die ein Händler (der Master- Datensatz) in der Vergangenheit gekauft hat, nämlich in einer *1:n-Beziehung*.
- **Microsoft HFlexGrid** - Ermöglicht den Benutzern, auf mehrere Datensätze zuzugreifen, sie anzuzeigen und zu bearbeiten, wobei ein tabellarisches Format verwendet wird.
- **Microsoft Chart** - Ermöglicht den Benutzern, auf mehrere Datensätze zuzugreifen, sie anzuzeigen und zu bearbeiten, wobei ein Diagramm-Format verwendet wird.



In den meisten Datenbanken besteht für einige Datensätze in den Tabellen eine *1:n-Beziehung*. Ein Datensatz könnte einen Spaltenwert enthalten, der sich auch in mehreren Datensätzen einer anderen Datei befindet. Beispielsweise könnte eine Autorendatenbank eine Tabelle mit Autoren enthalten, und eine Buch-Datenbank könnte mehrere Bücher enthalten, die vom selben Autor geschrieben wurden. Der Autor würde als der Master- Datensatz erscheinen, die Bücher als die Detail-Datensätze. Bei dieser Autor:Buch-Beziehung handelt es sich ebenfalls um eine *1:n-Beziehung*.

Außerdem müssen Sie den *Bindungstyp* angeben (d.h. wie Visual Basic die Datenbankinformationen zu den Steuerelementen bindet). Der Assistent bietet drei Auswahlmöglichkeiten an:

- **ADO-Datensteuerelement** - Verwendet das ADO-Datensteuerelement, um Steuerelemente an die Daten zu binden.
- **ADO-Code** - Verwendet ADO-Code, um Steuerelemente an die Daten zu binden.
- **Klasse** - Erzeugt eine spezielle Datenklasse für die Datenbank und bindet die Daten über diese Klasse an die Steuerelemente.

In diesem Beispiel wählen Sie **ADO-Datensteuerelement** und klicken auf **Weiter**. Im nächsten Fenster muß der Assistent wissen, auf welche Tabelle (oder auf welches Recordset) Sie in der Datenbank zugreifen möchten. Öffnen Sie das Dropdown-Listenfeld **Datensatzquelle**, um die Publishers-Tabelle auszuwählen. Wie in Abbildung 18.10 gezeigt, werden in der Liste **Verfügbare Felder** sofort alle Felder oder *Spalten* aus der Publishers-Tabelle angezeigt.

Datenformular-Assistent - Datensatzquelle

Wählen Sie eine Datensatzquelle, und danach die Felder, die auf dem Formular erscheinen sollen.

Datensatzquelle:
Lieferanten

Verfügbare Felder:

Firma
Homepage
Kontaktperson
Land
Lieferanten-Nr
Ort
PLZ

Ausgewählte Felder:

Nach folgender Spalte sortieren:
(Keine)

Hilfe Abbrechen < Zurück Weiter > Fertig

Abbildung 18.10:
Visual Basic analysiert die Tabelle und stellt Ihnen die Felder zur Auswahl bereit.

Die Felder, die Sie auswählen, werden auf dem vom Assistenten erzeugten ADO-basierten Formular angelegt. Wählen Sie eines der Felder in der linken Liste aus, und klicken Sie auf die Schaltfläche mit dem Pfeil nach rechts, um dieses Feld in die rechte Liste einzutragen. Der Assistent erzeugt auf dem resultierenden Formular Felder, die der rechten Liste entsprechen. In diesem Beispiel tragen Sie alle Felder in die rechte Liste ein.

Ordnen Sie die Feldreihenfolge so an, daß der Firmenname ganz oben in der Liste erscheint. Klicken Sie auf das Feld **Firma** und dann dreimal auf die Schaltfläche mit dem Pfeil nach oben, um das Feld ganz oben in der Liste anzuzeigen. Um sicherzustellen, daß das Formular die Datensätze in alphabetischer Reihenfolge nach dem Firmennamen anzeigt, wählen Sie in der **Listen Nach folgender Spalte sortieren** die Spalte **Firma** aus.

Klicken Sie auf die Schaltfläche **Weiter**, um das Fenster **Steuerelementauswahl** anzuzeigen (siehe

Abbildung 18.11). Dieses Fenster ermöglicht Ihnen, die Schaltflächen festzulegen, die auf dem Formular erscheinen und über die der Zugriff auf die Datenbank erfolgt. Die Schaltflächen reflektieren die Möglichkeiten, die Sie dem Benutzer bereitstellen. Wenn Sie nicht möchten, daß die Benutzer Datensätze aus der Tabelle entfernen können, heben Sie die Markierung für die Option **Schaltfläche 'Löschen'** auf. In diesem Beispiel behalten Sie die Markierungen für alle Schaltflächen bei.

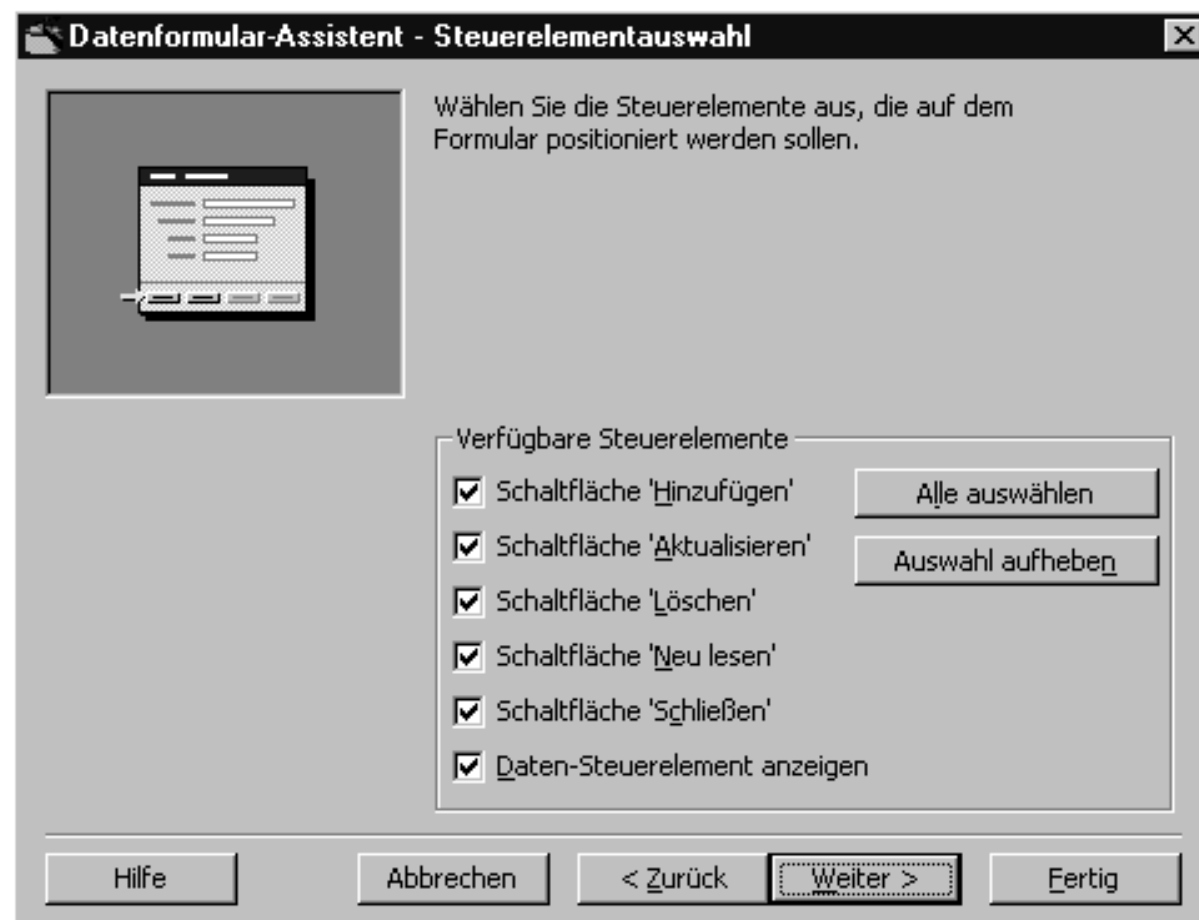


Abbildung 18.11:
Sie bestimmen, welche Möglichkeiten dem Benutzer zur Manipulation der Daten bereitgestellt werden.

Wenn Sie auf die Schaltfläche **Fertig** klicken, legt Visual Basic die Applikation an.

 Warnung	Die Schaltfläche Fertig beendet nicht den Assistenten, sondern bewirkt nur, daß das Datenbankformular angelegt wird. Sie müssen den Assistenten auf die übliche Weise abschließen. In diesem Beispiel klicken Sie auf die Schaltfläche Fertig im Assistenten, so daß die Applikation fertiggestellt wird.

Das Publishers-Formular sieht aus wie in Abbildung 18.12 gezeigt. Der Assistent hat alle erforderlichen Textfelder und Schaltflächen auf dem Formular angelegt, so daß ein Zugriff auf alle Daten möglich ist. Nachdem Sie sich davon überzeugt haben, daß alles funktioniert, können Sie das Formular beliebig ergänzen.

Abbildung 18.12:
Der Assistent hat das Formular mit allen erforderlichen Feldern und Schaltflächen angelegt.

Zusammenfassung

Dieses Kapitel hat Ihnen gezeigt, wie unter Visual Basic ein Datenbankzugriff erfolgt. Am einfachsten greift man auf eine Datenbank zu (vorausgesetzt, man verwendet nicht den Anwendungs-Assistenten), indem man einem Formular das Daten-Steuerelement hinzufügt. Das Daten-Steuerelement sorgt dafür, daß die zugrundeliegende Datenbank in den gebundenen Steuerelementen angezeigt und entsprechend aktualisiert wird. Das Daten-Steuerelement wird mit Hilfe von Methoden mit zusätzlicher Funktionalität ausgestattet.

Die ADO-Schnittstelle kann sehr viel Programmieraufwand erforderlich machen, aber Sie erhalten dadurch mehr Kontrolle über den Zugriff auf Ihre Datenbank und sind damit außerdem flexibler. Ihre Applikation muß die Steuerelemente aktualisieren und bei der Auslösung von Ereignissen die entsprechenden Datensätze anzeigen, aber die Datenbank-Applikation ist insgesamt schneller.

Im nächsten Kapitel erfahren Sie, wie man die Arbeit mit dem Internet in Visual Basic integriert. Wir leben in einer Online-Welt; deshalb sollten Ihre Applikationen in der Lage sein, Internet-Fähigkeiten anzubieten.

Fragen und Antworten

Frage:
Existieren in allen Datenbanktabellen 1:n-Beziehungen?

Antwort:
Es gibt verschiedene Beziehungen zwischen den Daten. Abhängig von der Datenbankstruktur können einige oder alle davon in einer Datenbank vorliegen. Dieses Kapitel konnte die Datenbanktheorie nicht detailliert beschreiben. Für viele Visual-Basic-Applikationen wird jedoch die 1:n-Beziehung benötigt; Sie sollten also wissen, worum es sich dabei handelt. Häufig will man alle Datensätze zu einem bestimmten Schlüsselwert anzeigen, und die 1:n-Beziehung ermöglicht diese Konstellation.

Die Master/Detail-Ansicht des Assistenten stellt eine praktische Methode dar, Applikationen anzulegen, die diese Beziehung zwischen den Daten bereitstellen. Es muß mindestens ein Feld beider Tabellen übereinstimmen, damit eine 1:n-Beziehung existieren kann. Eine 1:1-Beziehung gibt es, wenn eine Spalte in einer Tabelle einer Spalte in einer anderen Tabelle entspricht. Das könnte beispielsweise der Fall sein, wenn ein Produkt von zwei Herstellern angeboten wird. Es gibt auch n:n-Beziehungen. Alle diese Beziehungen sind Teil des relationalen Datenbankentwurfs. Man muß sie verstanden haben, um zu verstehen, wie eine relationale Datenbank arbeitet. Wichtig ist dabei nicht, wie man darauf zugreift. Diese Beziehungen werden beim Anlegen der Datenbank eingerichtet. Ihre Aufgabe als Visual-Basic-Programmierer ist es, auf diese Beziehungen zuzugreifen, aber Sie brauchen sie in den Daten nicht rekonstruieren, es sei denn, Sie haben die Datenbank selbst eingerichtet.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollen, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welches Werkzeugs stellt Visual Basic bereit, mit dem Sie Datenbankdateien bearbeiten und sichten können?
2. Was ist der Unterschied zwischen einer Datei und einer Tabelle?
3. Richtig/Falsch. Wenn Sie einer Tabelle Datensätze hinzufügen, wächst auch die Anzahl der Spalten in der Tabelle an.
4. Richtig/Falsch. Eine Tabelle ist eine Untermenge eines Recordsets.
5. Was ist ein gebundenes Steuerelement?
6. Was ist der Unterschied zwischen Recordset, Dynaset und Snapshot?
7. Nennen Sie zwei Vorteile von ADO gegenüber dem Daten-Steuerelement.
8. Was legen die Werte EOF und BOF fest?
9. Was ist der Unterschied zwischen einer Master- und einer Detail-Ansicht.
10. Welches Werkzeug bietet Visual Basic, das Datenbankformulare unmittelbar aus Ihrer Datenbankstruktur anlegt?

Übungen

1. Ändern Sie das Programm zum Datenbankzugriff, das Sie in diesem Kapitel erstellt haben (siehe Abbildung 18.6), so daß alle Felder der Kunden-Tabelle angezeigt werden. Ihr Formular sollte ähnlich aussehen wie das in Abbildung 18.7 gezeigte Formular. (Legen Sie unbedingt die Bezeichnungsfelder für die Beschriftung an, so daß der Benutzer erkennt, was die einzelnen Spalten enthalten.)
 2. Erzeugen Sie mit dem Anwendungs-Assistenten eine Master-/Detail-Ansicht der Datenbank BIBLIO.MDB, die den Autorennamen in der Master-Ansicht und die ISBNs aller seiner Bücher in der Detail-Ansicht darstellt.
-

Woche 3

Bonusprojekt 9: ADO-Steuerelemente

Dieses Bonusprojekt beschreibt eine ADO-basierte Datenbank-Applikation. Dazu verwenden Sie die Datenbank BIBLIO.MDB, die in Visual Basic bereitgestellt wird. Die ADO-Datenbank-Steuerelemente unterstützen zusätzliche Programmieranweisungen (wie Sie in diesem Abschnitt noch sehen werden), aber grundsätzlich sind sie einfach zu verstehen.

Das Ziel dieses Beispiels

Abbildung BP9.1 zeigt das Formular, das Sie hier entwickeln werden. Es enthält mehrere Linien und Steuerelemente und ist relativ aufwendig. Das Formular bietet ein vollständiges System zur Datenbankverwaltung für die Datenbank BIBLIO.MDB, die in Visual Basic enthalten ist. Dieses Bonusprojekt kann nicht jede Aktion des ADO-Steuerelements beschreiben, aber es zeigt, wie die Datenbank BIBLIO.MDB mit dem ADO-Steuerelement manipuliert werden kann.



Diese Applikation könnte auch mit Hilfe des Anwendungs-Assistenten angelegt werden, aber Sie sollen in diesem Bonusprojekt die ADO-Steuerelemente und ihre Arbeitsweise kennenlernen. Sie werden Programmanweisungen kennenlernen, die auf diesem ADO-Steuerelement basieren und die Sie einsetzen können, um auf Datenbanktabellen zuzugreifen, ohne daß der Benutzer eingreifen muß.



Dieses Bonusprojekt kann Ihnen nicht alle Details zum ADO-Steuerelement oder die dafür erforderliche Programmiersprache vermitteln. Sie lernen hier nur die Grundlagen der ADO-Programmierung kennen.

Buchtitel - ADO-Applikation

Buchtitel

Titel:

Erscheinungsjahr ISBN Verlags-ID Thema

Speichern Einfügen Neu Löschen Abbrechen < > Beenden

Abbildung BP9.1:
Ihre ADO-basierte Applikation verwaltet die Datenbank BIBLIO.MDB

Das Formular

Legen Sie das Formular an, wie in Tabelle BP9.1 beschrieben. Drücken Sie (Strg)+(T), um das Dialogfeld Komponenten zu öffnen, und wählen Sie dort *Microsoft ADO Data Control 6.0* aus, um das ADO-Steuerelement in Ihrer Werkzeugsammlung anzulegen.

Tabelle BP9.1: Legen Sie das Formular mit den hier beschriebenen Steuerelementen an.

Steuerelementeigenschaft	Eigenschaftswert
Formular Name	frmBookTitle
Formular Caption	Buchtitel ADO-Applikation
Formular Height	4590
Formular Width	7740
ADO Name	adoBooks
ADO Height	735
ADO Left	5400
ADO Top	0

ADO Width	2055
Bezeichnungsfeld #1 Name	lblApp
Bezeichnungsfeld #1 Alignment	Zentriert
Bezeichnungsfeld #1 BorderStyle	Fest einfach
Bezeichnungsfeld #1 Caption	Book Titles
Bezeichnungsfeld #1 FontStyle	Bold
Bezeichnungsfeld #1 FontSize	18
Bezeichnungsfeld #1 Left	2520
Bezeichnungsfeld #1 Height	495
Bezeichnungsfeld #1 Top	240
Bezeichnungsfeld #1 Width	2535
Bezeichnungsfeld #2 Name	lblTitle
Bezeichnungsfeld #2 Alignment	Rechts
Bezeichnungsfeld #2 Caption	Titel:
Bezeichnungsfeld #2 FontSize	10
Bezeichnungsfeld #2 Left	720
Bezeichnungsfeld #2 Height	255
Bezeichnungsfeld #2 Top	840
Bezeichnungsfeld #2 Width	495
Bezeichnungsfeld #3 Name	lblYear
Bezeichnungsfeld #3 Alignment	Rechts
Bezeichnungsfeld #3 Caption	Erscheinungsjahr
Bezeichnungsfeld #3 FontSize	10
Bezeichnungsfeld #3 Left	120
Bezeichnungsfeld #3 Height	255
Bezeichnungsfeld #3 Top	2400
Bezeichnungsfeld #3 Width	1455
Bezeichnungsfeld #4 Name	lblISBN
Bezeichnungsfeld #4 Alignment	Rechts
Bezeichnungsfeld #4 Caption	ISBN:
Bezeichnungsfeld #4 FontSize	10
Bezeichnungsfeld #4 Left	2880
Bezeichnungsfeld #4 Height	255

Bezeichnungsfeld #4 Top	2400
Bezeichnungsfeld #4 Width	495
Bezeichnungsfeld #5 Name	lblPubID
Bezeichnungsfeld #5 Alignment	Right Justify
Bezeichnungsfeld #5 Caption	Publisher's ID:
Bezeichnungsfeld #5 FontSize	10
Bezeichnungsfeld #5 Left	120
Bezeichnungsfeld #5 Height	255
Bezeichnungsfeld #5 Top	3000
Bezeichnungsfeld #5 Width	1455
Bezeichnungsfeld #6 Name	lblSubject
Bezeichnungsfeld #6 Alignment	Rechts
Bezeichnungsfeld #6 Caption	Thema:
Bezeichnungsfeld #6 FontSize	10
Bezeichnungsfeld #6 Left	3480
Bezeichnungsfeld #6 Height	255
Bezeichnungsfeld #6 Top	3000
Bezeichnungsfeld #6 Width	855
Textfeld #1 Name	txtTitle
Textfeld #1 DataField	Title
Textfeld #1 DataSource	adoBooks
Textfeld #1 Height	1095
Textfeld #1 Left	1320
Textfeld #1 Top	840
Textfeld #1 Width	5535
Textfeld #2 Name	txtPub
Textfeld #2 DataField	Year Published
Textfeld #2 DataSource	adoBooks
Textfeld #2 Height	345
Textfeld #2 Left	1680
Textfeld #2 Top	2400
Textfeld #2 Width	975
Textfeld #3 Name	txtTitle

Textfeld #3 DataField	Title
Textfeld #3 DataSource	adoBooks
Textfeld #3 Height	345
Textfeld #3 Left	1680
Textfeld #3 Top	2400
Textfeld #3 Width	975
Textfeld #4 Name	txtISBN
Textfeld #4 DataField	ISBN
Textfeld #4 DataSource	adoBooks
Textfeld #4 Height	345
Textfeld #4 Left	3480
Textfeld #4 Top	2400
Textfeld #4 Width	3495
Textfeld #5 Name	txtPubID
Textfeld #5 DataField	PubID
Textfeld #5 DataSource	adoBooks
Textfeld #5 Height	345
Textfeld #5 Left	1680
Textfeld #5 Top	3000
Textfeld #5 Width	1575
Textfeld #6 Name	txtSubject
Textfeld #6 DataField	Subject
Textfeld #6 DataSource	adoBooks
Textfeld #6 Height	345
Textfeld #6 Left	4440
Textfeld #6 Top	3000
Textfeld #6 Width	1575
Befehlsschaltfläche #1 Name	cmdSave
Befehlsschaltfläche #1 Caption	&Speichern
Befehlsschaltfläche #1 Left	240
Befehlsschaltfläche #1 Top	3600
Befehlsschaltfläche #1 Width	735
Befehlsschaltfläche #2 Name	cmdAdd

Befehlsschaltfläche #2 Caption	&Hinzufügen
Befehlsschaltfläche #2 Left	1200
Befehlsschaltfläche #2 Top	3600
Befehlsschaltfläche #2 Width	735
Befehlsschaltfläche #3 Name	cmdNew
Befehlsschaltfläche #3 Caption	&New
Befehlsschaltfläche #3 Left	2160
Befehlsschaltfläche #3 Top	3600
Befehlsschaltfläche #3 Width	735
Befehlsschaltfläche #4 Name	cmdDelete
Befehlsschaltfläche #4 Caption	&Löschen
Befehlsschaltfläche #4 Left	3120
Befehlsschaltfläche #4 Top	3600
Befehlsschaltfläche #4 Width	735
Befehlsschaltfläche #5 Name	cmdCancel
Befehlsschaltfläche #5 Caption	&Abbrechen
Befehlsschaltfläche #5 Left	4080
Befehlsschaltfläche #5 Top	3600
Befehlsschaltfläche #5 Width	735
Befehlsschaltfläche #6 Name	cmdPrec
Befehlsschaltfläche #6 Caption	&<
Befehlsschaltfläche #6 Left	5160
Befehlsschaltfläche #6 Top	3600
Befehlsschaltfläche #6 Width	495
Befehlsschaltfläche #7 Name	cmdNext
Befehlsschaltfläche #7 Caption	&>
Befehlsschaltfläche #7 Left	5760
Befehlsschaltfläche #7 Top	3600
Befehlsschaltfläche #7 Width	495
Befehlsschaltfläche #8 Name	cmdExit
Befehlsschaltfläche #8 Caption	&Beenden
Befehlsschaltfläche #8 Left	6600

Befehlsschaltfläche #8 Top	3600
Befehlsschaltfläche #8 Width	855
Linie #1 Name	Line1
Linie #1 X1	120
Linie #1 X2	7440
Linie #1 Y1	2160
Linie #1 Y2	2160
Linie #2 Name	Line2
Linie #2 X1	0
Linie #2 X2	7800
Linie #2 Y1	3480
Linie #2 Y2	3480
Linie #3 Name	Line3
Linie #3 X1	5040
Linie #3 X2	5040
Linie #3 Y1	3480
Linie #3 Y2	4280
Linie #4 Name	Line4
Linie #4 X1	6480
Linie #4 X2	6480
Linie #4 Y1	3480
Linie #4 Y2	4280

Beachten Sie, daß die Textfelder an die entsprechenden Felder der Tabelle BIBLIO.MDB gebunden werden, auf die das ADO-Steuererelement `adoBooks` verweist. In Kapitel 18 haben Sie erfahren, wie das Binden für das Daten-Steuererelement erfolgt. Das ADO-Steuererelement ist jedoch viel leistungsfähiger als das Daten-Steuererelement.

Verbindung des ADO-Steuererelements mit den Daten

Wenn Sie ein ADO-Steuererelement einsetzen wollen, verbinden Sie es mit den Daten, auf die es zugreifen soll. Dazu gehen Sie ähnlich vor wie das Daten-Steuererelement; das ADO-Steuererelement kann jedoch mit Daten verbunden werden, die nicht unbedingt in einer Datenbanktabelle abgelegt sein müssen. Das ADO-Steuererelement kann auf E-Mail, Grafik und fast jede andere Datenquelle außerhalb Ihrer Applikation zugreifen.

Größtenteils wird das ADO-Steuerelement jedoch mit Datenbanken verbunden, sei es mit einer lokalen Datenbank auf dem aktuellen PC oder mit einer vernetzten Datenbank, an die die Maschine des Benutzers angeschlossen ist. Ihr erster Schritt, das ADO-Steuerelement einzurichten, ist also die Verbindung zur Datenquelle. In diesem Projekt ist das **BIBLIO.MDB**, eine Access-97-Datenbank, die in Visual Basic enthalten ist.



Die Verbindung mit einem ADO-Steuerelement ist viel schwieriger als mit einem normalen Daten-Steuerelement. Seine Vorteile überwiegen jedoch diesen Aufwand bei weitem, insbesondere, weil es sehr viel effizienter ist und außerdem mit fast allen Arten von Daten verbunden werden kann.

Die Verbindung mit einer Datenquelle kann auf zweierlei Arten erfolgen:

- Durch Setzen der **ConnectionString**-Eigenschaft
- Im Code

Wenn Sie das Eigenschaftsfenster nutzen, um eine Verbindung zu den Daten herzustellen, werden Sie durch mehrere Dialogfelder dabei unterstützt. Im Code dagegen müssen Sie relativ komplizierte Anweisungen schreiben (dieser Abschnitt zeigt Ihnen beides).

Klicken Sie im Eigenschaftsfenster auf die Eigenschaft **ConnectionString**. Das in Abbildung BP9.2 gezeigte Dialogfeld erscheint. Hier legen Sie fest, welche Option für Ihren Datenquellentyp zutrifft.

Hier werden Sie eine einfache Verbindungszeichenfolge anlegen, die auf Ihre Datenbankdatei verweist. Die ersten beiden Optionen sind für unser Beispiel nicht erforderlich. Klicken Sie auf die dritte Option, um eine Verbindungszeichenfolge anzugeben.

Abbildung BP9.2:
Im Dialogfeld Eigenschaftenseiten geben Sie eine Verbindungszeichenfolge an.

Verbindungszeichenfolgen können relativ umfangreich sein, aber Visual Basic hilft Ihnen, sie anzulegen. Gehen Sie wie folgt vor:

1. Klicken Sie auf die Schaltfläche **Erstellen**.
2. Doppelklicken Sie im Dialogfeld auf die erste Option, **Microsoft Jet 3.51 OLE DB Provider**. Access-97-Datenbanken verwenden die Jet-Technologie, die Microsoft für den schnellen Datenbankzugriff bereitstellt. Das Dialogfeld **Data Link Eigenschaften** erscheint, wo Sie die Datenbank auswählen.
3. Klicken Sie rechts neben dem ersten Textfeld auf die Schaltfläche mit den drei Punkten, und suchen Sie BIBLIO.MDB auf Ihrer Platte.
4. Klicken Sie auf **Öffnen**, um die Datenbank mit Ihrem ADO-Steuerelement zu verbinden.



Um sicherzustellen, daß Sie die Verbindung zur Datenbank eingerichtet haben, klicken Sie auf die Schaltfläche **Verbindung testen**. Wenn Sie die richtige Verbindungszeichenfolge eingegeben haben, zeigt Visual Basic ein Meldungsfeld an, das Ihre Verbindung bestätigt. Falls Sie keine Verbindung eingerichtet haben, teilt Ihnen das Meldungsfeld das ebenfalls mit. Gehen Sie zurück auf die Registerkarten des anderen Dialogfelds, um das Problem zu beheben.

5. Klicken Sie auf die Registerkarte **Alle**, um einen Überblick über die von Ihnen erstellten Verbindungszeichenfolgen zu erhalten. Die anderen Elemente in dem Dialogfeld werden von den anderen Registerkarten aus manipuliert.
6. Klicken Sie auf **OK**, um zum Eigenschaftendialog zurückzukehren. Anschließend klicken Sie auf **OK**, um dieses Dialogfeld zu schließen und zu Ihrem Formular zurückzukehren.

Statt diese Auswahl im Dialogfeld vorzunehmen, könnten Sie auch im Code eine Verbindungszeichenfolge erzeugen, beispielsweise wenn das Formular geladen wird oder erst wenn der Benutzer wirklich auf die Datenbank zugreifen will. Je länger Sie warten, um eine Verbindung zu den Daten einzurichten, desto weniger wahrscheinlich ist es, daß Ihre Daten durch einen Stromausfall oder einen Systemabsturz in Mitleidenschaft gezogen werden. Ihre Applikation wird schneller gestartet, wenn Sie die Verbindungszeichenfolge erst aufbauen, wenn der Datenbankzugriff erforderlich ist.



Auch wenn Sie über den Code auf die Daten zugreifen, müssen Sie das ADO-Steuerelement auf Ihrem Formular anlegen. Setzen Sie seine **Visible** -Eigenschaft auf **False**, wenn der Benutzer keinen Zugriff darauf haben soll, Sie aber das Steuerelement brauchen, um im Code auf Daten zuzugreifen. Es ist nicht möglich, die Verbindungszeichenfolge sowohl im Code als auch im Dialogfeld zu setzen. Achten Sie also darauf, daß der Wert für **ConnectionString** leer sein muß, wenn Sie die Verbindungszeichenfolge im Programmcode setzen wollen.



Abbildung BP9.3:
Visual Basic bietet einen Überblick über die Einstellungen in Ihrer Verbindungszeichenfolge.

Um das ADO-Steuerelement im Code mit der Datenbank zu verbinden, schreiben Sie beispielsweise die folgende Anweisung in die Ereignisprozedur `Form_Load()`:

```
adoBooks.ConnectionString = "Provider=Microsoft.Jet.OLEDB.3.51;" & _
"Persist Security Info=False;" & _
"Data Source=C:\Program Files\Microsoft Visual" & _
"Studio\VB98\Biblio.mdb"
```

Das Steuerelement ist jedoch noch nicht bereit, Daten aus der Datenbank anzuzeigen. Sie müssen genau vorgeben, welche Zeilen aus welcher Tabelle in der Datenbank der restlichen Applikation zur Verfügung gestellt werden sollen. Dazu verwenden Sie SQL (eine Datenbankprogrammiersprache, die Visual Basic als `RecordSource`-Eigenschaft unterstützt). Die folgende Anweisung legt fest, daß alle Datensätze aus der Tabelle `Title` in der Datenbank `BIBLIO.MDB` bereitgestellt werden sollen:

```
adoBooks.RecordSource = "Select * From Titles"
```

Code oder Steuerelement?

Sollten Sie den Benutzern ermöglichen, das ADO-Steuerelement zu bedienen, oder sollten Sie es

verbergen und die Daten im Code durchlaufen? In diesem Projekt werden beide Vorgehensweisen demonstriert.

Welche Vorgehensweise Sie auswählen, hängt von den Anforderungen Ihrer Applikation ab. Häufig aktualisiert eine Applikation einen einzigen Datensatz oder einen Datensatz, der bestimmten Kriterien entspricht. Ihre Applikation muß in der Lage sein, einen solchen Datensatz zu finden oder etwa alle Datensätze zu durchlaufen und Gesamtsummen oder Durchschnittswerte zu berechnen. Für diese Art der Verarbeitung brauchen Sie Code, der die Daten im Hintergrund durchläuft.

Daten suchen

Um sicherzustellen, daß das Titelfeld die Buchtitel anzeigt, verbinden Sie die **DataSource**-Eigenschaft des Titels mit dem ADO-Steuerelement und die **DataField**-Eigenschaft mit dem Titelfeld der Tabelle. Dazu könnten Sie beispielsweise die beiden folgenden Anweisungen ausführen:

```
Set txtTitle.DataSource = adoBooks  
txtTitle.DataField = "Title"
```

Es ist nicht möglich, dieses Feld im Eigenschaftenfenster mit einem Wert zu belegen, wenn Sie das ADO-Steuerelement zur Laufzeit mit der Datenbank verbunden haben.

Nachdem Sie ein Textfeld mit dem ADO-Steuerelement verbunden haben, können Sie das entweder auch für alle weiteren Textfelder tun, oder Sie verwenden die in Tabelle BP9.1 beschriebenen **DataField**-Eigenschaften, die funktionieren, nachdem Sie ein Text-Steuerelement auf dem Formular mit der Datenbanktabelle verbunden haben.

Daten durchlaufen

Wenn Sie die Applikation ausführen, nachdem Sie dem ADO-Steuerelement das erste Textfeld hinzugefügt haben, werden der erste Datensatz und seine Felder angezeigt, wie Sie in Abbildung BP9.4 sehen.



Es kann sein, daß nicht alle Felder der Tabelle vollständige oder genaue Daten enthalten. Der erste Datensatz in **BIBLIO.MDB** beispielsweise hat keinen Eintrag für das Themenfeld. Für manche Datensätze werden ganz seltsame Ausgaben erzeugt, beispielsweise 2nd, was vielleicht die Auflage ist, nicht aber das Thema.

Buchtitel

Titel: dBASE III: Ein praktischer Leitfaden

Erscheinungsjahr: 1985 ISBN: 0-0038307-6-4

Verlags-ID: 469 Thema:

Speichern Einfügen Neu Löschen Abbrechen < > Beenden

Abbildung BP9.4:
Der erste Datensatz erscheint.

Hier erscheint nur der erste Datensatz, weil der Benutzer keinen Zugriff auf das unsichtbare ADO-Steuerelement hat und die Schaltflächen noch nicht mit dem Code verbunden sind. Doppelklicken Sie auf die Schaltfläche &>, um Code dafür bereitzustellen. Der folgende Code verschiebt den Datensatzzeiger auf den nächsten Datensatz in der Tabelle, damit die gebundenen Steuerelemente darauf zugreifen können:

```

1: Private Sub cmdNext_Click()
2: ' Einen Datensatz weiterschalten
3: If Not adoBooks.Recordset.EOF Then
4:   adoBooks.Recordset.MoveNext
5: If adoBooks.Recordset.EOF Then
6:   adoBooks.Recordset.MovePrevious
7: End If
8: End If
9: End Sub

```

Das Recordset, das in den Zeilen 3 bis 6 verwendet wird, ist die Liste der Datensätze, mit denen Sie arbeiten. Sie haben das Recordset zuvor mit der SQL-Anweisung spezifiziert, die die Anzeige auf Datensätze der Tabelle *Titles* beschränkt. Die Recordset- Methode `MoveNext` verschiebt den Datensatzzeiger in der Tabelle um einen Datensatz weiter (Zeile 4). Alle anderen zur Tabelle gebundenen Steuerelemente werden ebenfalls aktualisiert.

Die folgenden Situationen können zu Problemen führen:

- Wenn es in der Datenbanktabelle keine Datensätze gibt, tritt ein Fehler auf, sobald Sie versuchen, den Datensatzzeiger weiterzuschalten.

- Wenn der aktuelle Datensatz der letzte Datensatz in der Tabelle ist, tritt ein Fehler auf, sobald Sie versuchen, den Datensatzzeiger weiterzuschalten.

Die If-Anweisung in Zeile 3 stellt sicher, daß das Tabellenende noch nicht erreicht ist. Dazu wird mit dem Not-Operator die EOF-Eigenschaft des Recordsets ausgewertet. Diese Anweisung bedeutet »wenn das Dateiende des Recordsets noch nicht erreicht ist, soll der Datensatzzeiger weitergeschaltet werden«. Zeile 5 stellt sicher, daß die MovePrevious-Methode noch einmal ausgeführt wird, wenn MoveNext das Dateiende erreicht, so daß stets verhindert wird, daß der Datensatzzeiger das Dateiende erreicht. MovePrevious durchläuft die Tabelle rückwärts.

Mit Hilfe der MovePrevious-Methode können Sie der Schaltfläche **Vorheriger** (&<) eine Ereignisprozedur hinzufügen, die die Tabelle rückwärts durchläuft:

```
1: Private Sub cmdPrev_Click()  
2: ' Einen Datensatz zurückschalten  
3: If Not adoBooks.Recordset.BOF Then  
4:   adoBooks.Recordset.MovePrevious  
5: If adoBooks.Recordset.BOF Then  
6:   adoBooks.Recordset.MoveNext  
7: End If  
8: End If  
9: End Sub
```

Die Recordset-Eigenschaft BOF prüft, ob der Dateianfang erreicht ist, um sicherzustellen, daß der Leser den Datensatzzeiger nicht vor den Dateianfang setzt.



Geben Sie in der Click-Ereignisprozedur für die **Beenden**-Schaltfläche die Anweisung End ein, damit der Benutzer das Programm verlassen kann.

Tabellen aktualisieren

Das ADO-Steuerelement aktualisiert den zugrundeliegenden Datensatz, wenn der Benutzer in einem gebundenen Textfeld eine Änderung vornimmt. Wenn Sie also einen Titel oder ein Thema ändern, wird diese Änderung in der Tabelle aufgezeichnet. Bei der nächsten Ausführung der Applikation erscheinen die geänderten Daten im Formular.

Nichtsdestotrotz müssen Sie auch in der Lage sein, im Code selbst Daten in einer Datenbank zu speichern. Dieser Code kann jedoch relativ umfangreich und komplex sein und soll hier nicht beschrieben werden.

Um Ihnen zu demonstrieren, wie kompliziert es sein kann, die Daten mit Hilfe des ADO-Steuerelements zu aktualisieren, betrachten Sie den Code in Listing BP9.1, der die Daten aus den Textfeldern in die Datenbank schreibt. (Hier zeigt sich natürlich eine Redundanz, weil die Textfelder bereits an die Tabelle gebunden sind. Wenn die Textfelder oder außenliegende Daten jedoch nicht an die Tabelle gebunden sind, verwenden Sie ähnlichen Code wie hier gezeigt.)

Listing BP9.1: Es gibt Methoden, im Code Daten in die Tabelle zu schreiben.

```
1: Private Sub cmdSave_Click()  
2: ' Zuordnung der Felder und Textfelder
```

```

3: ' Nur Daten zuweisen, die nicht Null sind
4: ' (lange Zeilen werden umbrochen)
5: adoBooks.Recordset!Title = _
6: IIf(txtTitle = "", "N/A", txtTitle)
7: adoBooks.Recordset![Year Published] = _
8: IIf(txtPub = "", "N/A", txtPub)
9: adoBooks.Recordset!ISBN = _
10: IIf(txtISBN = "", "N/A", txtISBN)
11: adoBooks.Recordset!PubID = _
12: IIf(txtPubID = "", "N/A", txtPubID)
13: adoBooks.Recordset!Subject = _
14: IIf(txtSubject = "", "N/A", txtSubject)
15:
16: ' Aktualisierung des Recordsets
17: adoBooks.Recordset.Update
18:
19 : End Sub

```

In Listing BP9.1 wird überall die `IIf( )`-Funktion verwendet, so daß der Wert N/A (für *Not Applicable*) in alle gespeicherten Felder geschrieben wird, die keine Daten enthalten. Das `IIf( )` stellt sicher, daß keine Werte geschrieben werden, die Null sind. Das geklammert Feld in Zeile 7 ist erforderlich, weil das Feld ein Leerzeichen enthält. Visual Basic behandelt `[Year Published]` als einzelnen Feldnamen; ohne die Klammern könnte es das Leerzeichen nicht erkennen.

Listing BP9.1 trägt nur die geänderten Daten in die Tabelle ein. Sie können den Code jedoch einfach abändern, um Daten aus anderen Quellen in der Tabelle abzulegen. Beispielsweise könnten Sie in Zeile 14 die externen Daten berücksichtigen, etwa Eingaben von einem Benutzer, statt die Daten aus dem Textfeld im Formular zu speichern. In Zeile 17 erfolgt die eigentliche Aktualisierung, weil die Tabelle die neuen Daten ohne die `Update`-Methode nicht übernimmt.

Der Code in Listing BP9.1 speichert also die Änderungen, die der Benutzer an den Daten vornimmt, auch wenn diese Werte aufgrund der gebundenen Textfelder in diesem Beispiel sowieso gespeichert werden. Wenn Sie möchten, daß völlig neue Daten in die Tabelle geschrieben werden (hinter dem letzten Datensatz), löschen Sie die Felder auf dem Formular, so daß der Benutzer Daten eintragen kann. (Sie könnten diesen Code mit der Schaltfläche **Neu** verbinden, was hier nicht erfolgt ist.)

Bevor der neue Datensatz geschrieben werden kann, muß der Datensatzzeiger an das Dateiende gesetzt werden, damit neue Daten eingetragen werden können (und nicht der aktuelle Datensatz überschrieben wird):

```
adoBooks.Recordset.AddNew ' Vorbereitung auf den neuen Datensatz
```

Alle Daten, die Sie in der Tabelle ablegen, erscheinen als neue Datensätze.

Fazit

Wie Sie sehen, ist das ADO-Steuerelement nicht so einfach zu programmieren wie das Daten-Steuerelement, aber es ist leistungsfähiger und schneller. Sie haben in diesem Bonusprojekt bereits einige Eigenschaften kennengelernt, aber längst nicht alle.

Woche 3

Tag 19

Internet-Zugriff

Dieses Kapitel zeigt Ihnen, wie Sie von Ihren Visual-Basic-Applikationen aus auf das Internet zugreifen. Visual Basic ist eines der einfachsten Werkzeuge, mit dem Sie eine Verbindung zum Internet einrichten können. Beachten Sie jedoch, daß die Programmierung für den Internet-Zugriff selbst unter Visual Basic nicht trivial ist. Dieses kurze Kapitel kann Ihnen nur einen ersten Eindruck verschaffen, wie Visual Basic das Internet sieht und damit arbeitet.

Heute lernen Sie die folgenden Dinge kennen:

- Internet-Verbindung in Visual Basic
- Web-Browser in Ihrer Applikation
- Internet-Steuerelemente
- Kapselung
- ActiveX-Dokumente
- Wie Sie fast *jede* Visual-Basic-Applikation zu einer Internet-Applikation machen können.

Der Internet-Assistent

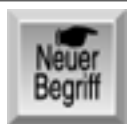
Der Anwendungs-Assistent von Visual Basic nimmt Ihnen Arbeit ab, wenn Sie Ihre Applikation für den Internet-Zugriff einrichten möchten. Sie wählen einfach ein paar der vorgegebenen Auswahlmöglichkeiten aus, und der Assistent fügt Ihrer Applikation Internet-Zugriff hinzu und ermöglicht damit eine weltweite Kommunikation.



Die hier beschriebenen Internet-Werkzeuge funktionieren sowohl für Internet- als auch für Intranet-Applikationen. Internet- und die Intranet-Technologien unterstützen ein gemeinsames Protokoll, so daß sie innerhalb derselben Applikationen eingesetzt werden können.



Ein *Intranet* ist ein lokal vernetztes System (möglicherweise innerhalb eines einzigen Gebäudes oder innerhalb eines einzelnen Stockwerks), das dieselben Funktionen wie das Internet bereitstellt.

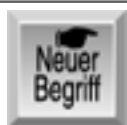


Ein *Protokoll* ermöglicht die Kommunikation zwischen zwei Computern. Internet- und Intranet-Verbindungen verwenden *TCP/IP* (*Transfer Control Protocol / Internet Protocol*), das sehr gebräuchlich ist. Es wird für allgemeine Internet-Verbindungen eingesetzt.

Dieser Abschnitt erklärt, was der Visual Basic Anwendungs-Assistent macht, wenn Sie ihn nutzen, um Ihrer Applikation Internet-Zugriff hinzuzufügen. Der Assistent bietet die Möglichkeit, in Ihren Applikationen WWW-Seiten (World Wide Web) zu lesen.



Die Benutzer Ihrer Applikation brauchen einen Internet Service Provider, sonst haben sie auch über Ihre Applikation keinen Zugriff auf das Web. Außerdem muß der Internet Explorer 4 oder neuer auf Ihrem eigenen Entwicklungssystem installiert sein, damit Sie mit der Internet-Unterstützung von Visual Basic arbeiten können. Bei der Installation von Visual Basic wird der Internet Explorer 4 also Option bereitgestellt.



Ein *Internet Service Provider*, auch als ISP bezeichnet, ist ein Unternehmen, das Accounts für den Internet-Zugang bereitstellt.

Im Anwendungs-Assistenten von Visual Basic brauchen Sie das sechste Dialogfeld (siehe Abbildung 19.1), **Internet-Verbindung**, das den Web-Zugriff für Ihre Applikation einrichtet.

Anwendungs-Assistent - Internet-Verbindung

Der Assistent kann Ihnen einen individuellen Web Browser zur Verfügung stellen und einen Sprung zu Ihrer Homepage hinzufügen!

Anmerkung: Dieses Funktionsmerkmal setzt voraus, daß Ihre Benutzer bereits einen Internet-Anbieter haben!

Möchten Sie, daß Ihre Benutzer von Ihrer Anwendung aus auf das Internet zugreifen können?

☐ Ja ☒ Nein

Wenn Sie 'Ja' wählen, können Sie eine Standard-URL angeben, die beim Starten Ihres Web Browsers angezeigt wird.
(Beispiel: <http://www.microsoft.com>)

<http://www.microsoft.com>

Hilfe Abbrechen < Zurück Weiter > Fertig

Abbildung 19.1:
Im Dialogfeld Internet-Verbindung bestimmen Sie den Web-Zugriff.

Der Assistent stellt eine Standard-URL bereit - die Homepage von Microsoft. Ändern Sie diese

Standard-URL, wenn Ihre Benutzer etwas anderes sehen sollen. Wenn ein Benutzer den Browser in der Applikation aufruft, meldet sich der Browser an und stellt eine Verbindung zu der im Assistenten angegebenen URL her (dazu wird der ISP des Benutzers verwendet). Beispielsweise könnten Sie die URL Ihrer Homepage in das URL-Textfeld eingeben. Wenn Sie das Optionsfeld Ja auswählen, fügt der Assistent in Ihre Applikation die Engine für einen Web-Browser ein.



Eine *URL (Uniform Resource Locator)* ist die Adresse einer Web-Site. Jede Web-Site hat eine eindeutige URL.

Eine URL beginnt immer mit `http://`. (Die Buchstaben `http` stehen für *Hypertext Transfer Protocol* und bezeichnen die Standardkommunikationsprozedur für den Zugriff auf Web-Seiten.) Die meisten modernen Browser brauchen das Präfix `http://` nicht mehr, im Assistenten muß es jedoch angegeben werden.

Es ist ganz einfach, eine Test-Applikation anzulegen, die auf das Internet zugreift. Gehen Sie dazu wie folgt vor:

1. Legen Sie ein neues Projekt an, und doppelklicken Sie auf das Icon **VB-Anwendungs-Assistent**.
2. Klicken Sie auf **Weiter**, um das erste Dialogfeld zu schließen.
3. Wählen Sie **SDI** (Single Document Interface), um eine ganz einfache Applikation anzulegen.
4. Klicken Sie auf **Weiter**, um die nächsten vier Dialogfelder zu durchlaufen, und übernehmen Sie dabei die vorgegebenen Menüoptionen.
5. Im Dialogfeld **Internet-Verbindung** klicken Sie auf **Ja**. In diesem Beispiel behalten Sie die URL für die Web-Site von Microsoft bei.
6. Klicken Sie auf **Fertig**, um die Applikation anzulegen.



Ein Meldungsfeld weist Sie darauf hin, daß die Applikation erstellt wurde. Wenn Sie die Internet-Steuerelemente zum ersten Mal seit der Installation von Visual Basic verwenden, erscheint ein Dialogfeld, das darauf hinweist, daß einige Steuerelemente auf die Verwendung vorbereitet werden müssen. Klicken Sie auf OK, um diese Steuerelemente zu entpacken und sie in die Visual-Basic-Umgebung zu laden.



Abbildung 19.2:

Visual Basic teilt Ihnen in einem Meldungsfeld mit, daß die Applikation erstellt wurde.

Wenn Sie in die Entwicklungsumgebung von Visual Basic zurückkehren, sehen Sie, daß die Werkzeugsammlung erweitert wird. Abbildung 19.3 zeigt die Werkzeuge, die der Assistent hinzugefügt hat. Einige dieser Steuerelemente kennen Sie schon: das Standarddialogfeld, die Symbolleiste und die Abbildungsliste.

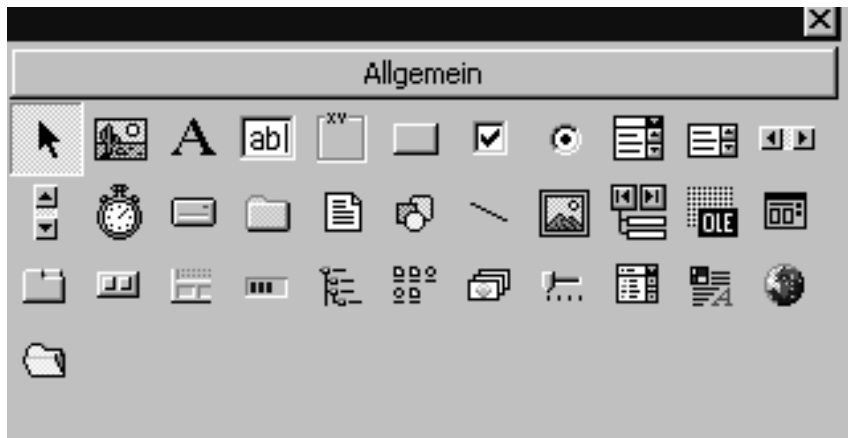


Abbildung 19.3:

Der Assistent hat Ihrer Werkzeugsammlung neue Werkzeuge hinzugefügt.

Diese zusätzlichen Werkzeuge ermöglichen das Web-Browsing in Ihrer Applikation. Offensichtlich geht es aber in diesem Kapitel hauptsächlich um das Web-Browser- Steuerelement.

Um ein Gefühl für die Applikation zu erhalten, drücken Sie (F5), um sie zu starten. Der Bildschirm, den Sie sehen, unterscheidet sich kaum von den Bildschirmen für andere vom Assistenten erzeugte Applikationen. Das Formular enthält ein Standardmenü und eine Symbolleiste. Sobald Sie jedoch im **Ansicht**-Menü den Eintrag **Web Browser** auswählen, erscheint die Internet-Funktion. In der Bildschirmmitte wird ein Dialogfeld für das Web-Browsing angezeigt. Dort melden Sie sich an, wie Sie es von Ihrem ISP gewohnt sind (es sei denn, Sie sind bereits angemeldet oder Sie werden automatisch angemeldet). Nachdem Sie Ihren Namen und Ihr Kennwort eingegeben haben, erscheint ein Fenster in der Art des Internet Explorer und zeigt die Web-Site von Microsoft an (siehe Abbildung 19.4).



Bei dem Internet-Explorer-Fenster, das Sie sehen, handelt es sich eigentlich um eine kleine Applikation, die um ein riesiges ActiveX-Steuerelement herum angelegt wurde. Der Web-Browser, den der Assistent eingefügt hat, ist ein Beispiel für ein solches ActiveX-Steuerelement. Der Web-Browser der Applikation ist einfacher als der vollfunktionale Internet-Explorer (weniger Schaltflächen und keine Menüs), aber der eingebettete Browser stellt alle gebräuchlichen Browser-Funktionen bereit, beispielsweise die Schaltflächen **Zurück**, **Vor**, **Home** usw. Wenn Sie auf die Schaltfläche **Suchen** klicken, verwendet der Internet-Explorer die Such-Funktion von Microsoft für die Abfrage.

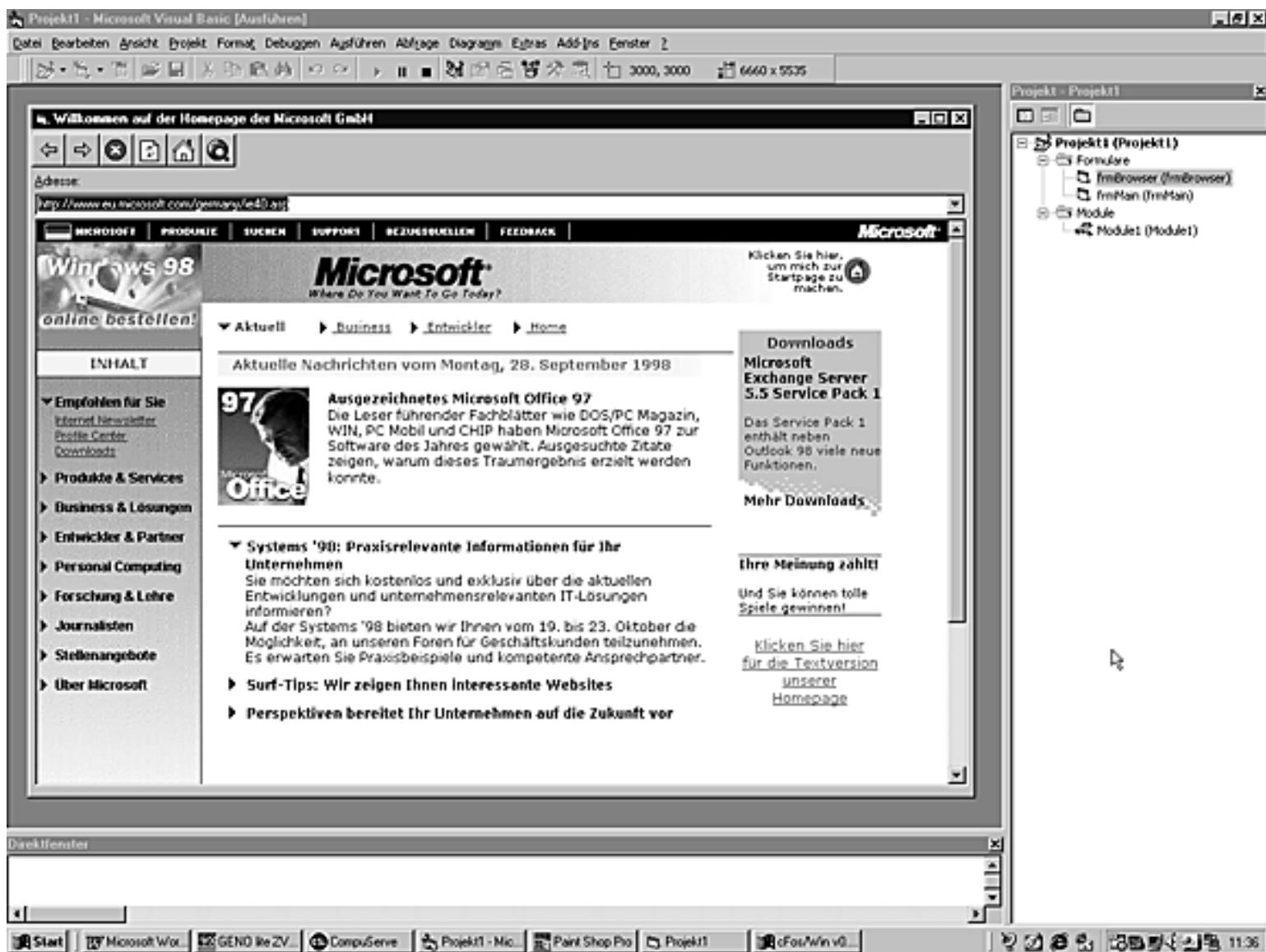


Abbildung 19.4:
Die vom Assistenten angelegte Applikation ist jetzt mit der Web-Site von Microsoft verbunden.

Wenn Sie sich vom Internet abmelden möchten, schließen Sie den Web-Browser, doppelklicken auf das Icon, das für Ihren ISP in der Taskleiste angezeigt wird, und melden sich ab. Der Web-Browser bietet keine Funktion, um sich vom Web abzumelden, aber Sie könnten sie durch Code realisieren.

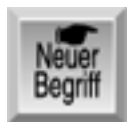
Internet-Steuerelemente

Wenn Sie die Professional- oder Enterprise-Edition von Visual Basic einsetzen, können Sie verschiedene komplexe ActiveX-Steuerelemente für den Internet-Zugriff in Ihren Applikationen einsetzen. Das Beispiel im vorigen Abschnitt hat die Leistungsfähigkeit eines einzigen Steuerelements demonstriert - das WebBrowser-Steuerelement. Dieser Abschnitt stellt weitere Steuerelemente vor.

Im Dialogfeld **Komponenten**, das Sie über das **Projekt**-Menü öffnen, finden Sie mehrere Internet-Steuerelemente. Dieser Abschnitt stellt diese Steuerelemente vor und erklärt, wie und wo Sie sie verwenden können, um in Ihren Projekten auf das Internet zuzugreifen.

Die Steuerelemente für die Kapselung

Der Begriff *Kapselung* bezieht sich auf unterschiedliche Dinge, abhängig davon, ob Sie Daten, Code oder beides kapseln. Im allgemeinen kann man die Kapselung jedoch immer als *Verpackung* betrachten. Visual Basic beinhaltet einige Internet-Steuerelemente, die Ihre existierende Applikation und den Code in Internet-fähige Applikationen kapseln oder *verpacken*. Diese Steuerelemente kapseln Ihre Applikationen, so daß sie mit der Internet-Technologie arbeiten können.



Kapselung bezeichnet das Verpacken von Komponenten, wie es auch bei Visual-Basic-Objekten der Fall ist, die Eigenschaften, Methoden und Ereignisse unterstützen.

Hier eine Liste der Kapselungs-Steuerelemente:

- Internet-Transfer - Kapselt die drei gebräuchlichsten Internet-Protokolle: HTTP, FTP (File Transfer Protocol) und Gopher (ein Suchprotokoll, das Ihnen hilft, Informationen im Internet zu finden). Mit FTP können Sie Dateien direkt innerhalb Ihrer Visual-Basic-Applikation herunterladen.
- Web Browser - Kapselt einen Web-Browser direkt in Ihrer Applikation.
- WinSock - Bietet ein allgemeines Verbindungs- und Datenaustausch-Steuerelement, das zwei Protokolle unterstützt: UDP (User Datagram Protocol) und TCP (Transmission Control Protocol).

Internet-Explorer-Steuerelemente

Visual Basic beinhaltet mehrere Steuerelemente, die Sie einem Projekt hinzufügen, um die Internet-Technologie zu nutzen und damit auf das Web zuzugreifen. Diese Steuerelemente haben im Dialogfeld **Komponenten** das Präfix *IE*.

Tabelle 19.1 hilft Ihnen, die in diesem Abschnitt beschriebenen Steuerelemente zu finden. Häufig beschreiben die Namen dieser Steuerelemente nicht ihre eigentliche Aufgabe. Tabelle 19.1 bietet genauere Informationen über die Steuerelemente, die Sie im Dialogfeld **Komponenten** auswählen können.

Tabelle 19.1: Mehrere Steuerelemente beinhalten Funktionen des Internet Explorer sowie andere Internet-Steuerelemente.

Komponente	Beschreibung
IE Animated Button	Eine animierte Schaltfläche, die die vom Internet Explorer eingerichtete Verbindung repräsentiert.
IE Popup Menu	Ein Menü-Steuerelement, das auf der Web-Seite erscheint.
IE Preloader	Lädt eine ganze Seite von einer Site, bevor der sichtbare Internet-Zugriff erfolgt.
IE Super Label	Label einer Web-Seite.
IE Timer	Stellt Timing-Operationen für Internet-Dienste bereit.
Microsoft Internet Controls	Web-Browser-Steuerelement.

Microsoft Internet Transfer Control 6.0	Das Steuerelement für das Übertragungsprotokoll.
Microsoft Winsock Control 6.0	Die Windows-Verbindung zu gebräuchlichen Internet-Protokollen.

	Wenn Sie den Online-Dienst Microsoft Network nutzen, können Sie unter Visual Basic 6 einige praktische Steuerelemente nutzen, die entsprechende Dienste für Ihre Applikationen bereitstellen, beispielsweise das MSN-Mail- Steuerelement. Diese Steuerelemente haben im Dialogfeld Komponenten das Präfix <i>MSN</i> .

Vorschau auf komplexe Aspekte

Wenn Sie Visual Basic nutzen möchten, um mit dem Internet zu kommunizieren, dann haben Sie sich vielleicht schon darüber informiert, worum es dabei geht. Am einfachsten ist es, die Internet-Funktionen mit Hilfe des Anwendungs-Assistenten einzufügen, wie in diesem Kapitel bereits gezeigt wurde. Wenn Sie andere Methoden nutzen möchten, habe Sie einen höheren Lernaufwand vor sich.

Dieser Abschnitt stellt einige der Begriffe und Konzepte vor, die Sie für die Arbeit mit dem Internet unter Visual Basic brauchen.

ActiveX-Dokumente

Wenn Sie eine Visual-Basic-Applikation entwickeln möchten, die nur mit dem Internet arbeitet, können Sie für den Anfang *ActiveX-Dokumente* verwenden. Ein ActiveX- Dokument verhält sich auf einem Formularfenster wie eine ganz normale Visual-Basic- Applikation, sendet aber ActiveX-Steuerelemente an den Computer des Benutzers, falls diese dort noch nicht enthalten sind. Das Dokument sieht für den Benutzer aus wie eine normale HTML-basierte Web-Seite. Beispielsweise kann das ActiveX-Dokument Hypertext-Links enthalten (ActiveX-Steuerelemente, die heruntergeladen oder einfach ausgeführt werden, abhängig davon, was auf dem Computer des Benutzers bereits enthalten ist). Außerdem können die Menüs des ActiveX-Dokuments automatisch mit denen der übergeordneten Applikation kombiniert werden.

	<i>HTML (Hypertext Markup Language)</i> ist die wichtigste Sprache für die Formatierung von Web-Seiten. Ein Web-Browser wertet alle HTML-Codes aus, um eine Web-Seite zu formatieren und anzuzeigen. Der letzte Abschnitt dieses Kapitels zeigt ein Beispiel für HTML-Code. Eine HTML-Datei ist einfach nur eine Textdatei, die HTML-Codes enthält. Sie können eine HTML-Datei in einem Texteditor erstellen, aber es gibt viele visuelle Werkzeuge, die Ihnen ermöglichen, Web-Seiten anzulegen, ohne dazu HTML- Code eingeben zu müssen. Sogar Visual Basic unterstützt Sie beim Entwurf von Web-Seiten. Dazu verwenden Sie den DHTML-Anwendungs-Assistenten . Alle HTML-Dateien erhalten die Dateinamenerweiterung <i>.HTM</i> .

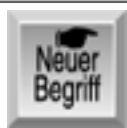
Die Professional- und die Enterprise-Edition von Visual Basic 6 unterstützen *DHTML (Dynamic Hypertext Markup Language)*, das auf die Aktionen auf einer Web-Seite reagiert. Das ActiveX-Dokument stellt eine Verbindung zu einer HTML-Seite her, die Sie angelegt haben oder nutzen. Wenn der Benutzer auf den Link zu Ihrem ActiveX- Dokument klickt, wird Ihr

ActiveX-Dokument aktiviert, die Steuerelemente werden auf den Computer des Benutzers geschickt, und der Code für das ActiveX-Dokument der Web-Seite wird ausgeführt, während der Benutzer die Seite besucht. Der Inhalt ist dynamisch, d.h. die Textfarbe und der Stil variieren abhängig von den Einstellungen auf der Maschine des Benutzers.

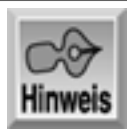


Das ActiveX-Dokument ist nicht statisch. Es handelt sich dabei in jeder Hinsicht um eine laufende Applikation. Durch das Dokument-Konzept können die Programmierer sehen, wie die Web-Seiten das eingebettete ActiveX- Dokument nutzen.

Der vielleicht wichtigste Grund dafür, warum man ActiveX-Dokumente erzeugt, ist, daß der Internet-Explorer sie ausführen kann, als handelte es sich dabei um ein Programm. Die Menüs des ActiveX-Dokuments werden mit denen des Internet-Explorers kombiniert (und überschreiben gegebenenfalls eine Funktionalität), und Sie müssen keine neue Sprache erlernen, wie beispielsweise Java, um die Web-Seiten zu aktivieren.



Java ist eine Internet-Programmiersprache, die auf C++ basiert und die genutzt wird, um Web-Seiten zu aktivieren und mit den Benutzern zu kommunizieren, indem ihnen zusammen mit der Web-Seite kleine Java-Programme, sogenannte *Applets*, geschickt werden.



Das Dialogfeld **Neues Projekt** enthält zwei Icons - **ActiveX Dokument EXE** und **ActiveX Dokument DLL** -, die ActiveX-Dokument-Shells erzeugen (siehe Abbildung 19.5). Nachdem Sie begonnen haben, das ActiveX- Dokument zu erzeugen, können Sie dem Formularfenster beliebige Funktionen hinzufügen, so wie es bei normalen Applikationen auch der Fall ist.

Der ActiveX-Dokument-Assistent

Die einfachste Methode, Ihre Applikationen für das Internet vorzubereiten, ist, Visual Basic die Arbeit zu überlassen. Wenn Sie der Menüoption **Add-In-Manager** den ActiveX-Dokument-Assistenten hinzufügen, habe Sie ein leistungsfähiges Werkzeug an der Hand, das fast jede Visual-Basic-Applikation in eine Internet-fähige Applikation umwandelt. Sie können ihre umgewandelten ActiveX-Applikationen auf einem Web- Server ablegen, und Ihre Benutzer kommunizieren über einen Web-Browser damit, so wie sie mit Web-Seiten arbeiten, die HTML, VBScript oder Java-Code enthalten.

Um zu sehen, wie einfach es ist, eine Applikation in eine Internet-fähige Applikation umzuwandeln, nutzen Sie in den folgenden Schritten den ActiveX-Dokument-Assistenten:

1. Wählen Sie **Add-Ins**, **Add-In-Manager**, und doppelklicken Sie auf den Eintrag **VB 6 ActiveX-Dokument-Assistent**, um den Assistenten der Liste Ihrer Add- ins hinzuzufügen.
2. Klicken Sie auf **OK**, um das Dialogfeld **Add-In-Manager** zu schließen.
3. Öffnen Sie das Beispielprojekt **Calc.vpb**, das in Visual Basic enthalten ist. Dazu gehen Sie in den Samples-Ordner der Installations-CD oder auf die Festplatte, wenn Sie die Beispiele bei der Installation von Visual Basic berücksichtigt haben.
4. Drücken Sie (F5), um die Applikation zu starten. Die Applikation simuliert einen Taschenrechner

auf Ihrem Bildschirm (siehe Abbildung 19.6).

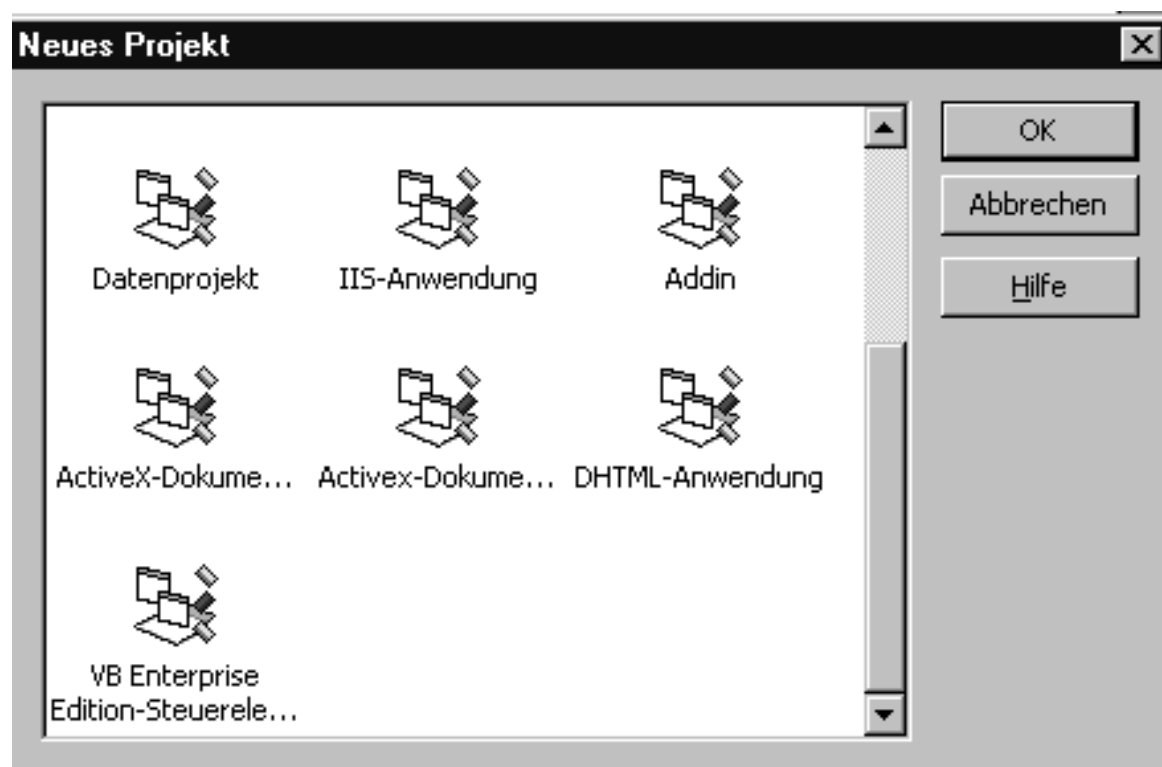


Abbildung 19.5:
Diese Applikation bildet die Funktionen eines Taschenrechners nach.

5. Beenden Sie die Programmausführung.
6. Wählen Sie **Add-Ins, ActiveX-Dokument-Assistent**, um den Assistenten zu starten.
7. Klicken Sie auf die Schaltfläche **Weiter**, um das erste Fenster zu überspringen.
8. Das zweite Fenster erscheint (siehe Abbildung 19.7), wo alle Formulare des aktuellen Projekts aufgelistet werden. Das einfache Projekt `Calc.vbp` enthält nur ein einziges Formular, deshalb wird hier auch nur ein einziger Eintrag gezeigt. Klicken Sie auf den Eintrag, um ihn zu selektieren.



Wenn Ihr Projekt mehrere Formulare enthält, können Sie festlegen, nur einige davon an Web-kompatible ActiveX-Dokumente zu senden. Jedes Formular wird zu einem eigenständigen ActiveX-Dokument.

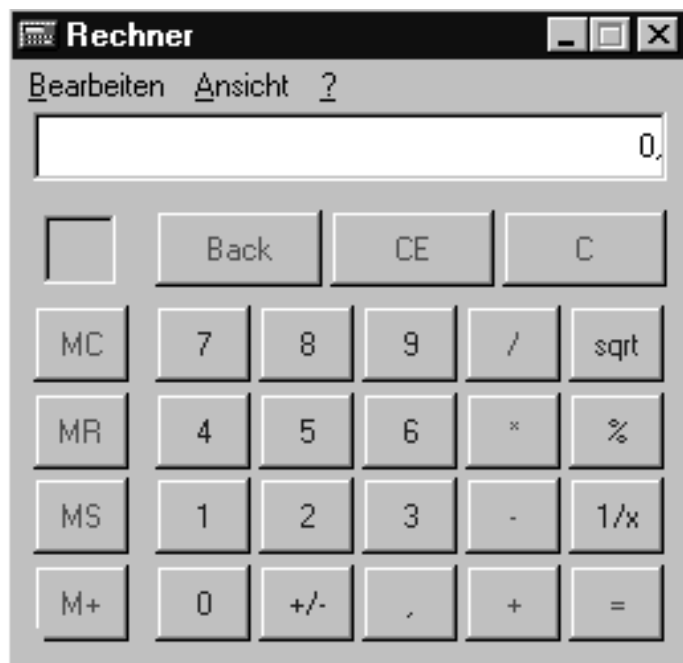


Abbildung 19.6:

Wählen Sie das Formular der Applikation aus, das in einem ActiveX-Dokument auf der Web-Seite erscheint.

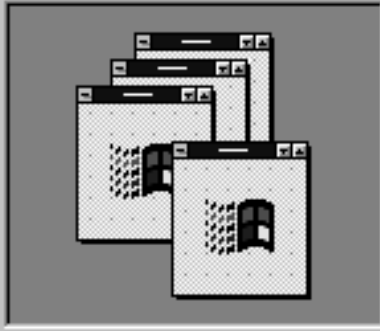
9. Das **Optionen**-Fenster, das Sie in Abbildung 19.8 sehen, legt fest, wie der Assistent die Elemente behandelt, die er nicht umwandeln kann. Es gibt Code, der in einem ActiveX-Dokument nicht ausgeführt werden kann, aber ein Großteil des Visual-Basic-Codes macht keine Probleme. Wenn Sie die erste Option auswählen, fügt Visual Basic Kommentarzeichen vor dem Code ein, den der Assistent nicht umwandeln kann. Die resultierende Applikation ist dann möglicherweise nicht vollständig, aber Sie können nach den Kommentaren suchen und den Code korrigieren oder ihn entfernen, wenn er für die Applikation nicht lebenswichtig ist. Die Applikation `Calc.vbp` enthält keinen ungültigen Code, deshalb brauchen Sie diese Option hier nicht auszuwählen.

Darüber hinaus könnte es sein, daß Sie die Formulare entfernen, die Sie aus dem Projekt konvertiert haben, weil diese sich nach Ausführung des Assistenten in einem ActiveX-Dokument befinden. Lassen Sie die Option **In ActiveX-EXE konvertieren** markiert, so daß der Assistent ein ausführbares Modul erstellt, im Gegensatz zu einer DLL.



Eine *DLL*, *Dynamic Link Library*, ist eine kompilierte Routine, die mehrere kompilierte Applikationen gemeinsam nutzen können.

ActiveX-Dokument-Assistent - Formularauswahl



Wählen Sie die Projektformulare aus, die in ActiveX-Dokumente konvertiert werden sollen.

☐ Calculator

Hilfe

Abbrechen

< Zurück

Weiter >

Fertigstellen

Abbildung 19.7:
Hier setzen Sie die Optionen für das ActiveX-Dokument.

In diesem Beispiel behalten Sie die Standardeinstellungen im **Optionen**-Dialogfeld bei und klicken auf **Weiter**.

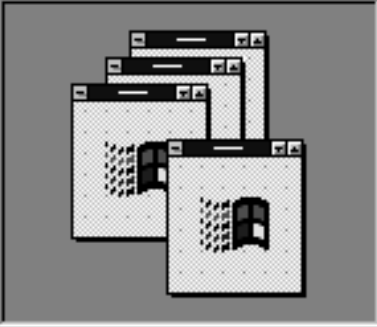
10. Behalten Sie alle Standardeinstellungen bei, und klicken Sie auf die Schaltfläche **Fertig**, um die Migration der `Calc.vbp`-Applikation zu einem ActiveX-Dokument zu starten. Nachdem die Konvertierung abgeschlossen ist, erscheint ein letztes Dialogfeld.

11. Klicken Sie auf **OK**, um das letzte Dialogfeld zu schließen.



Nach der Konvertierung erscheint ein Fenster mit einem zusammenfassenden Bericht, wie in Abbildung 19.9 gezeigt. Dieses Fenster ist wichtig, weil Sie dort erfahren, was Sie tun, um die Konvertierung zu testen. Nachdem Sie diesen Bericht gelesen haben, klicken Sie auf die Schaltfläche **Speichern**, um den Text zu speichern, und schließen das Fenster. Für dieses Beispiel können Sie das Fenster schließen, ohne den Bericht zu speichern.

ActiveX-Dokument-Assistent - Optionen



Wählen Sie die gewünschten Optionen aus.

☐ Ungültigen Code auskommentieren?

☐ Ursprüngliche Formulare nach der Konvertierung entfernen?

Ihr Projekttyp ist ungültig

☒ In ActiveX-EXE konvertieren?

☐ In ActiveX-DLL konvertieren?

Hilfe Abbrechen < Zurück Weiter > Fertigstellen

Abbildung 19.8:

Das Fenster mit dem zusammenfassenden Bericht beschreibt, welche Schritte Sie ausführen sollten, um die Konvertierung zu vervollständigen.

Nachdem Sie die Konvertierung abgeschlossen haben, führen Sie sie noch einmal in der Visual-Basic-Umgebung aus, um ein ausführbares ActiveX-Dokument-Objekt daraus zu machen. Visual Basic kompiliert die Applikation gewissermaßen, aber anders als bei der typischen Kompilierung (siehe Kapitel 21) erzeugt die konvertierte Applikation ein ActiveX-Dokument mit der Dateinamenerweiterung .VBD (für Visual Basic Document). (Sie müssen das ActiveX-Dokument in sein EXE-Format kompilieren, wenn Sie es außerhalb der Visual-Basic-Umgebung ausführen möchten.)

Sobald Sie die Applikation ausführen, zeigt Visual Basic das Dialogfeld **Projekteigenschaften** an (siehe Abbildung 19.10). Dort legen Sie fest, wie sich das ActiveX-Dokument bei der Programmausführung verhält. Im Projektfenster sehen Sie jetzt zwei Komponenten, das übliche **Calc**-Formular und ein zweites Objekt namens **docCalculator**. Das **docCalculator**-Objekt ist die ActiveX-Dokument-Komponente, die Sie erzeugt haben, als Sie die Applikation mit (F5) ausgeführt haben.

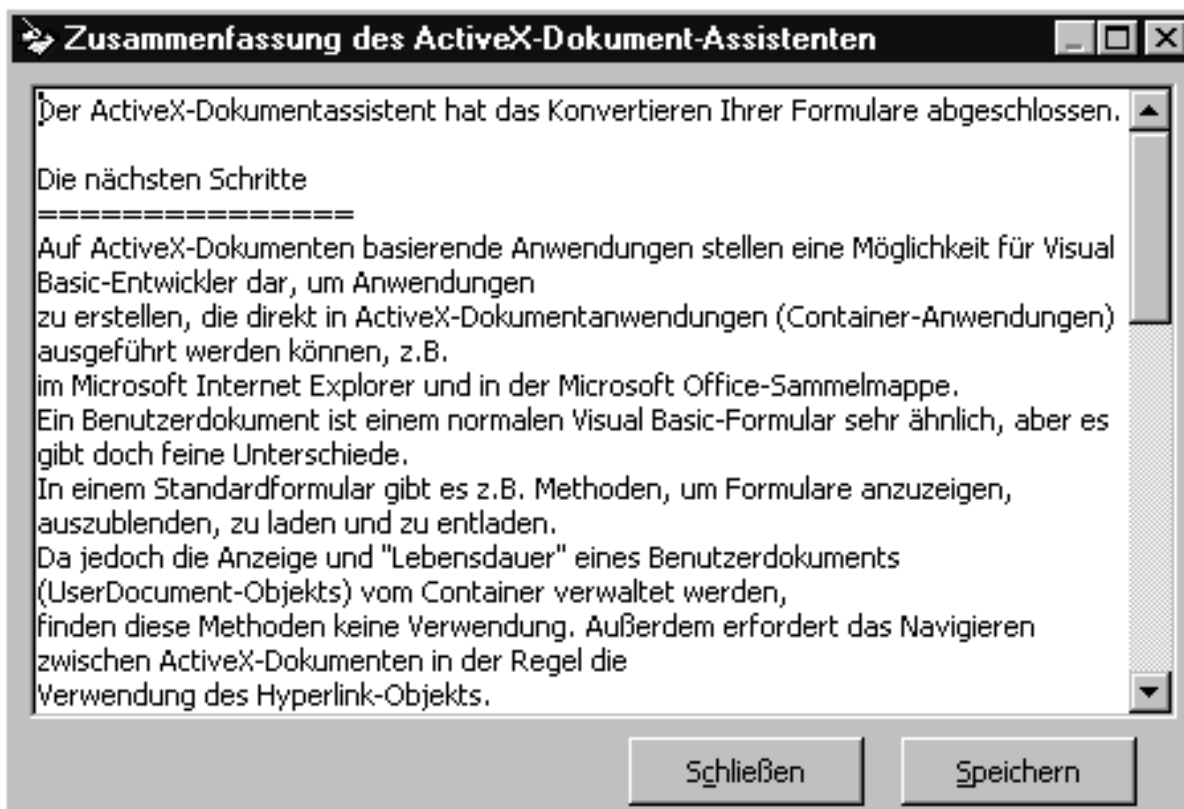


Abbildung 19.9:
Hier werden die Projekteigenschaften für das ActiveX-Dokument angegeben.

Wenn sie auf **OK** klicken, startet Visual Basic das ActiveX-Dokument in Ihrem Internet-Browser. Abbildung 19.11 zeigt das Ergebnis. Beachten Sie, was passiert ist:

- Der Internet-Browser selbst führt Ihre Visual-Basic-Applikation aus.
- Sie haben keinen HTML-Code geschrieben, um eine Web-Seite zu erzeugen.
- Die Calculator-Applikation wurde erfolgreich zu einer Web-Applikation gemacht.
- Wenn Sie das Projekt schließen, können Sie es jederzeit im Internet-Browser wieder starten, ohne daß Visual Basic dazu benötigt wird.
- Wenn Sie die Applikation auf Ihrem Web-Server ablegen (vorausgesetzt, Sie haben Zugriff auf einen solchen), kann jeder Internet-Nutzer der Welt, der einen ActiveX-komptiblen Web-Browser besitzt, Ihre Applikation ausführen. Die Applikation wird auf der Maschine des Benutzers ausgeführt, nachdem sie zusammen mit der Web-Seite heruntergeladen wurde. (Damit wird die Verhaltensweise eines Java-Applets nachgebildet.)



Abbildung 19.10:
Der Internet-Explorer ist die Plattform, auf der die Calculator-Applikation jetzt ausgeführt wird.

Nachdem Sie Ihren Internet-Browser geschlossen haben, wählen Sie Ausführen, Ende , um die Ausführung von Visual Basic zu beenden.

HTML und VBScript

Sie brauchen zwar nur die Programmiersprache Visual Basic zu kennen, um auf alle ihre Funktionen zur Internet-Verbindung zuzugreifen, aber es gibt zwei weitere Hilfs- sprachen, die Sie kennen sollten, weil sie das Ganze zusammenhalten. HTML ist die Formatierungssprache, die hinter den Web-Seiten steht. Die HTML-Sprache soll die folgenden Aufgaben erfüllen:

- Sie formatiert Web-Seiten in Spalten, mit Grafik und entsprechenden Titeln.
- Sie erlaubt die Integration zusätzlicher Internet-Dienstprogramme, beispielsweise VB-ActiveX-Dokumente und Java (eine kleine Programmiersprache, die Web-Seiten aktiviert)

HTML wird auch als *Skriptsprache* bezeichnet. Die Sprache wird nicht kompiliert. Statt dessen formatiert sie Web-Seiten, bestimmt, wo Grafiken und Trennlinien erscheinen, und erlaubt, eingebettete Applikationen zu aktivieren, beispielsweise ActiveX-Dokumente und Java-Programme.

VBScript ist, wie der Name schon sagt, ebenfalls eine Skriptsprache, aber Microsoft hat sie so entwickelt, daß sie auf der Programmiersprache Visual Basic aufsetzt. Sie werden sich in VBScript also wie Zuhause fühlen. VBScript ist praktisch, wenn Sie wichtige VB-Funktionen auf einer Web-Seite verwenden möchten, beispielsweise Popup-Meldungen, Eingabefelder, Berechnungen in Schleifen usw. VBScript ersetzt die ActiveX-Dokumente von Visual Basic nicht, sondern lädt diese zur Ausführung in eine HTML-Seite. VBScript ist das Medium, über das HTML-Dokumente die

ActiveX- Dokument-Applikationen ausführen.

Listing 19.1 zeigt ein Beispiel für die ersten Zeilen des HTML-Codes für die Web-Site von Microsoft.

Listing 19.1: Einige Zeilen HTML-Code zeigen, wie die Formatierung von Web-Seiten erfolgt.

```
<HTML>
<HEAD>
<TITLE>MSN.COM</TITLE>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<META http-equiv="PICS-Label" content=
'(PICS-1.0 "http://www.rsac.org/ratingsv01.html"
l comment "RSACi North America Server" by
"Microsoft Network"')>
</HEAD>
<FRAMESET rows="20,*" frameborder="0"
framespacing="0" border="0">
<FRAME src="/pilot.htm" name="pilot"
NORESIZE scrolling="no" marginwidth="0"
marginheight="0" frameborder="0" framespacing="0">
</FRAMESET>
</html>
```

Melden Sie sich am Internet an, und zeigen Sie die Homepage von Microsoft in Ihrem Web-Browser an (<http://www.microsoft.com>). Die Seite unterscheidet sich möglicherweise ein bißchen von der in Listing 19.1 beschriebenen Seite, aber sie sieht nicht wie Listing 19.1 aus! HTML ist eine Formatsprache. Die Befehle in Listing 19.1 teilen Ihrem Web-Browser mit, wie der Text und die Grafik der Seite auf Ihrem Bildschirm angezeigt werden sollen.

Listing 19.2 zeigt einen Ausschnitt aus einem VBScript-Beispiel. Beachten Sie, wie viel Sie durch Ihre Visual-Basic-Kenntnisse davon schon verstehen.

Listing 19.2: Ein Beispiel für ein VBScript-Listing demonstriert, wie ähnlich VBScript der Programmiersprache Visual Basic ist.

```
<SCRIPT Language="VBScript">
Call PrintWelcome
Call ModMessage

Sub PrintWelcome
If Date() = "2/2/98" Then
document.write ". . . .Kathy's Birthday!"
End If
If Date() = "2/5/98" Then
document.write ". . . .Eric's Birthday!"
End If
If Date() = "5/17/98" Then
document.write ". . . .Michael's Birthday!"
End If
If Date() = "7/25/98" Then
document.write ". . . .My Birthday!"
```

```
End If
End Sub
Sub ModMessage
Document.Write "<BR>This page was last modified:
"+Document.lastModified +"</FONT><BR>"
End Sub
</SCRIPT>
```

Zusammenfassung

In diesem Kapitel haben Sie erfahren, welche Rolle Visual Basic für den Zugriff auf das Internet spielt. Hier konnte natürlich nur ein kleiner Ausschnitt dessen gezeigt werden, was erforderlich ist, um Visual Basic zu einem echten Internet-Programmierungswerkzeug zu machen. Sie brauchen immenses Hintergrundwissen zu fast jedem Aspekt der Internet-Technologie, damit Sie Visual Basic für die Entwicklung von Internet-Applikationen einsetzen können. Es gibt mehrere gute Bücher über dieses Thema, am besten aber lesen Sie die Online-Dokumentationen von Visual Basic 6. Dort finden Sie schrittweise Beschreibungen, die Ihnen den Weg zum Internet-Programmierer zeigen.

Im nächsten Kapitel erfahren Sie, wie Sie Hilfe-Seiten für Ihre Visual-Basic-Applikationen anlegen. Außerdem lernen Sie, wie HTML arbeitet, wenn Sie mit HTML-ähnlichen Seiten ein interaktives Hilfesystem entwickeln.

Fragen und Antworten

Frage:

Ich habe eine Applikation mit drei Formularen. Kompiliert der ActiveX-Dokument-Assistent diese drei Formulare zu einem einzigen ActiveX-Dokument?

Antwort:

Der ActiveX-Dokument-Assistent wandelt Formulare, nicht aber vollständige Applikationen in ActiveX-Dokumente um. Mit anderen Worten, wenn Ihre Applikation vier Formulare enthält, wird jedes dieser Formulare zu einem separaten ActiveX-Dokument (vorausgesetzt, die Formulare enthalten keinen Code, der dagegen spricht). Es ist möglich, Dokumente im HTML-Code durch Hypertext-Links zu verknüpfen, aber jedes Formular wird zu einem separaten ActiveX-Dokument.

Frage:

Welche Applikationen konvertiert der ActiveX-Dokument-Assistent nicht?

Antwort:

Der ActiveX-Dokument-Assistent konvertiert die meisten Visual-Basic-Applikationen, außer solche mit eingebetteten OLE-Objekten, weil OLE eine ältere Technologie ist, die von Internet-Browsern nicht unterstützt wird. Darüber hinaus kann es sein, daß bestimmte komplexe Befehle und Steuerelemente nicht konvertiert werden können oder im resultierenden ActiveX-Dokument nicht wie erwartet funktionieren. Für die meisten Visual-Basic-Applikationen sollte es jedoch keine Probleme geben.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollen, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was macht die Web-Browsing-Applikation, die Sie mit dem Anwendungs-Assistenten von Visual Basic erzeugen, mit der URL, die Sie übergeben?
2. Richtig/Falsch. Die Benutzer Ihrer Applikation müssen den Web-Browser des Internet Explorer aufrufen, damit Ihr Web-Browser-Steuerelement funktioniert.
3. Richtig/Falsch. Sie müssen den Web-Browser des Internet Explorer starten, damit Ihr Web-Browser-Steuerelement funktioniert.
4. Was ist *Kapselung*?
5. Welchen Online-Dienst unterstützen einige Visual-Basic-Steuerelemente?
6. Was ist der Unterschied zwischen einem *Intranet* und dem *Internet*?
7. Was ist der Unterschied zwischen einem ActiveX-Dokument und einer normalen Visual-Basic-Applikation?
8. Welche Aufgabe hat Java?
9. Welche Skriptingsprache verwendet HTML, um ActiveX-Dokumente zu laden und auszuführen?
10. Wie konvertieren Sie existierende Applikationen in ein ActiveX-Dokument?

Übungen

1. Wenn Sie Microsoft Office Professional besitzen, können Sie den Office Binder nutzen, um ActiveX-Dokumente aufzunehmen! Probieren Sie das mit dem ActiveX-Dokument aus, das Sie in diesem Kapitel angelegt haben.
2. Wandeln Sie eine Applikation mit mehreren Formularen, beispielsweise Controls, in ein ActiveX-Dokument um.

Woche 3

Tag 20

Ein Hilfe- System erstellen

Dieses Kapitel erklärt, wie Sie Ihren Benutzern Hilfe zur Verfügung stellen. Wenn Sie es durchgearbeitet haben, wissen Sie, wie man einer Applikation ein Hilfesystem hinzufügt, das den Benutzern eine Online-Dokumentation präsentiert. Das Online-Hilfesystem ist »online«, weil die Information ständig zur Verfügung steht, wenn der Benutzer sie braucht - und nicht, weil Sie die Benutzer ins Internet schicken. Die Hilfe, die Sie bereitstellen werden, bildet die Online-Hilfe nach, die Sie aus vielen anderen Windows-Programmen kennen.

Als Vorbereitung auf die Online-Hilfe lernen Sie hier zunächst, wie man Steuerelementen ToolTips hinzufügt. Nachdem Sie erfahren haben, wie man Hilfedateien anlegt, können Sie auch die wichtige Direkthilfe erstellen.

Heute lernen Sie die folgenden Dinge kennen:

- ToolTips und Direkthilfe
- HTML-Hilfedateien
- HTML-Hilfe oder die traditionelle WinHelp-Hilfe?
- Text in Hilfedateien anlegen und formatieren
- RTF-Dateiformat
- Hilfe-Meldungen für Steuerelemente
- Standarddialoge für Hilfe-Bildschirme
- Hilfe-Kontext-IDs für spezielle Hilfethemen



Es gibt mehrere Möglichkeiten, Visual-Basic-Applikationen eine Online-Hilfe hinzuzufügen. Heute lernen Sie die beiden gebräuchlichsten Methoden kennen. Dabei wird beschrieben, wie man die Online-Hilfe anlegt, die man aus vielen Windows-Applikationen kennt, *WinHelp*. Seit Visual Basic Version 6 können Sie Ihren Applikationen auch HTML-basierte Hilfe hinzufügen. Ein Teil dieses Kapitels erklärt, wie Sie Ihren Applikationen HTML- Hilfe hinzufügen. Nicht alle Benutzer haben Web-Browser, die die HTML- basierte Hilfe unterstützen; Sie sollten also nicht davon ausgehen, daß Ihre Applikation mit solchen Systemen kompatibel sind - es sei denn, Sie verwenden die heute beschriebene Standard-Online-Hilfe.

ToolTip und Direkthilfe

Es ist ganz einfach, Objekten *ToolTips* hinzuzufügen. Die spezielle *Direkthilfe* basiert hauptsächlich auf Ihrer Hilfedatei. Ein Steuerelement erhält seinen ToolTip unmittelbar beim Anlegen auf dem Formular. Das Eigenschaftenfenster enthält die `ToolTip`-Eigenschaft, deren Text Sie beliebig setzen. Wenn ein Benutzer den Mauszeiger über ein Steuerelement schiebt, dessen `ToolTip`-Eigenschaft gesetzt ist, wird nach ein paar Sekunden der ToolTip-Text angezeigt. Und auch die Direkthilfe ist ganz einfach zu realisieren, wenn Sie eine detaillierte Hilfedatei erstellt haben.



ToolTips sind die Popup-Kurzbeschreibungen, die erscheinen, wenn Sie den Mauszeiger über ein Steuerelement schieben.



Die *Direkthilfe* ist eine kontextsensitive Hilfe (d.h. ihr Inhalt ist abhängig davon, von wo aus der Benutzer die Hilfe anfordert), die Steuerelemente auf dem Bildschirm beschreibt. Der Benutzer klickt auf die Schaltfläche für die Direkthilfe. Der Mauszeiger wird zu einem Zeiger mit Fragezeichen. Abhängig davon, wo der Benutzer mit der Maus klickt, werden Informationen über das betreffende Steuerelement angezeigt.



Bieten Sie Ihren Benutzern eine Direkthilfe an, wenn der Bildschirm viele Elemente enthält, die neue Benutzer möglicherweise verwirren.

Neben den ToolTips können Sie Ihren Benutzern in der Direkthilfe zusätzliche Informationen über Ihre Applikation bereitstellen. Die Direkthilfe ermöglicht den Benutzern, Hilfetext für beliebige Objekte anzuzeigen, die einen geeigneten Eintrag besitzen. Abbildung 20.1 zeigt ein Beispiel für die Anzeige der Direkthilfe. (Der Zeiger mit dem Fragezeichen wird in einen normalen Zeiger umgewandelt, sobald das Hilfefeld angezeigt wird.)

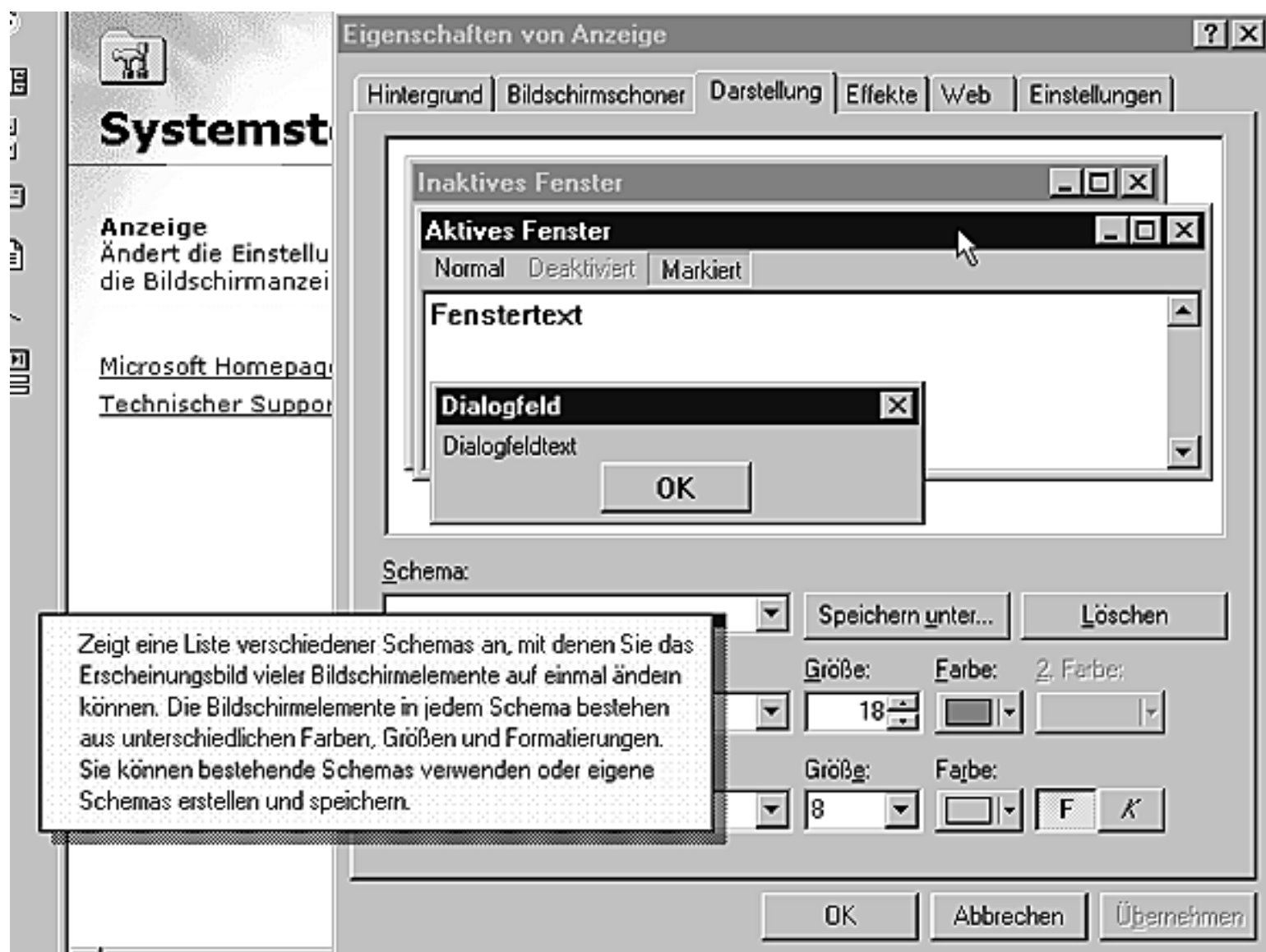


Abbildung 20.1:
Der Benutzer fordert die Direkthilfe an.

Einer der Gründe dafür, warum Sie ToolTips zusammen mit der Direkthilfe kennenlernen, ist ihre Ähnlichkeit aus der Perspektive des Benutzers. Der Benutzer schiebt entweder die Maus über ein Steuerelement, um einen Tooltip zu lesen, oder er klickt auf die Direkthilfe, um Hilfe zu einem Steuerelement zu erhalten. Obwohl die beiden Hilfestrategien dem Benutzer so ähnlich erscheinen, werden Sie überrascht sein, wie viel schwieriger es ist, einer Applikation eine Direkthilfe hinzuzufügen, im Vergleich zu einem Tooltip (was extrem einfach ist). Der restliche Abschnitt beschreibt also, wie man ToolTips einfügt, und das ganze restliche Kapitel beschäftigt sich mit der Direkthilfe.

Es ist ganz einfach, einem Visual Basic auf dem Formular eine Tooltip-Hilfe hinzuzufügen. Sie brauchen den gewünschten Text nur in die `ToolTipText`-Eigenschaft des Objekts einzugeben. Alles andere erledigt Visual Basic. Suchen Sie im Sample-Ordner nach der MDI-Notepad-Applikation, `Mdinode`. Als die Microsoft-Programmierer diese Applikation entwickelten, fügten sie fast jedem Steuerelement auf jedem Formular ToolTips hinzu. Sie können also jedes Steuerelement auswählen und seine `ToolTipText`-Eigenschaft lesen, um zu sehen, was erscheint, wenn Sie den Mauszeiger über dieses Steuerelement schieben.

Was Sie tun sollten

Gewöhnen Sie sich an, ToolTips einzufügen, wenn Sie in Ihrer Applikation Steuerelemente anlegen - es ist ganz einfach. Es ist sehr unwahrscheinlich, daß Sie die ToolTips später ersetzen; deshalb sollten Sie sie schreiben, wenn Sie die Aufgabe des Steuerelements noch gut im Gedächtnis haben.

Hilfe in einer Applikation

Die Entwicklung eines Hilfesystems kann müßig sein - aber Sie können sich das Windows-Hilfesystem für Ihre Applikationen borgen. Sie brauchen nur noch den Text zu schreiben, der dem Benutzer zur Hilfe angezeigt werden soll, und diesen dann unter Verwendung des Windows-Hilfesystems zum richtigen Zeitpunkt und am richtigen Ort anzuzeigen. Ihre Benutzer brauchen überhaupt nichts zu lernen, weil sie das Windows- Hilfesystem bereits kennen.



Warten Sie nicht, bis Ihre Applikation ganz fertig ist, bis Sie das Hilfesystem entwickeln. Am besten fangen Sie sofort damit an, während Sie an der Applikation arbeiten. Zur Entwurfszeit wissen Sie einfach noch mehr über die Applikation als zu jedem späteren Zeitpunkt, und Sie verstehen besser, welche Hilfe der Benutzer möglicherweise braucht.

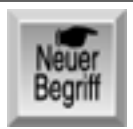
HTML-basierte Hilfesysteme

Wenn Sie das Hilfesystem von Visual Basic verwenden, sehen Sie HTML-Hilfedateien. Mit anderen Worten, die Hilfe erscheint in einem Browser-ähnlichen Format, und es werden Hyperlinks angezeigt. Mit der Schaltfläche **Zurück** gelangen Sie zum zuvor angezeigten Hilfebildschirm. Im linken Feld erscheint ein Themenüberblick. Wenn Sie auf eines dieser Themen doppelklicken, werden im rechten Fenster detaillierte Informationen dazu angezeigt.



HTML steht für *HyperText Markup Language* und ist die Sprache, mit der Web-Seiten formatiert werden, damit sie so aussehen, wie sie aussehen.

Sie können auch in Ihre eigenen Visual-Baisc-Applikationen solche HTML-basierte Hilfedateien einfügen. Diese Hilfe ist relativ neu. Bevor es die Web-Browser gab, verwendeten alle Windows-Applikationen dieselbe Hilfe-Engine, *WinHelp*. Sie zeigt Online-Hilfe an und verwendet ebenfalls Hyperlinks, aber nicht im Browser-Format (siehe Abbildung 20.2). WinHelp wurde mehrere Jahre lang in Windows-Applikationen als Hilfesystem eingesetzt. Die Programmierer sind davon überzeugt, daß die HTML-Hilfe innerhalb kurzer Zeit zur Standard-Online-Hilfe für Windows-Applikationen werden wird, aber auch WinHelp wird noch häufig eingesetzt.



WinHelp ist das Online-Hilfesystem, das von den meisten Windows-Applikationen eingesetzt wird und das möglicherweise irgendwann durch das browser-basierte HTML-Hilfesystem ersetzt wird.



Trotz des modernen Aussehens der HTML-basierten Hilfe können die Benutzer Ihrer Applikationen diese nicht nutzen, wenn sie keinen Web-Browser auf ihrem System installiert haben oder wenn sie einen Web-Browser einsetzen, der älter als der Internet Explorer 4.0 ist. Wenn Sie alle Ihre Benutzer ansprechen möchten, sollten Sie also bei der Standardhilfe bleiben, die im nächsten Abschnitt beschrieben wird. Dieser Abschnitt beschreibt, wie die browser-basierte PC-Oberfläche aussieht.

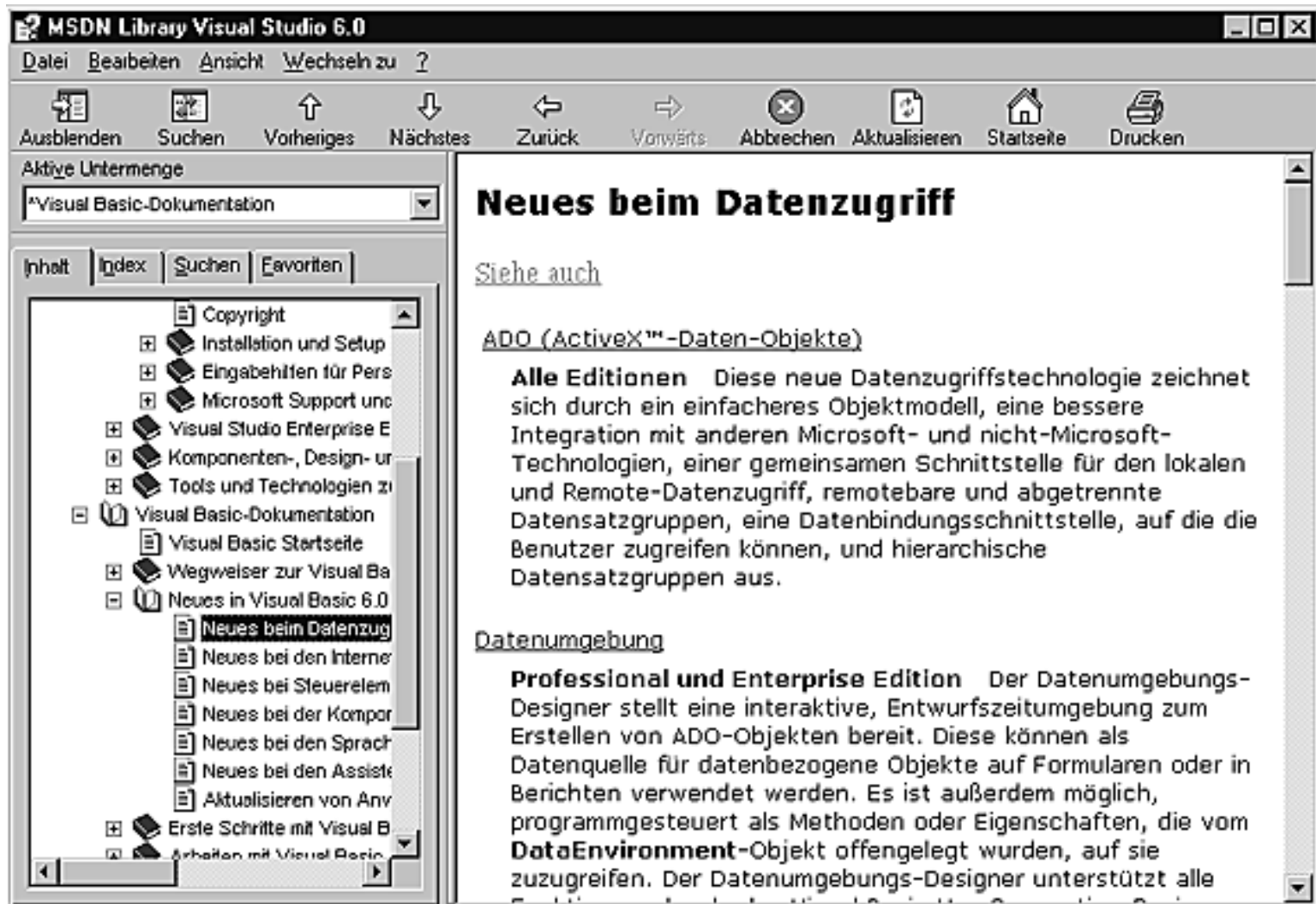


Abbildung 20.2:
WinHelp verwendet ein Explorer-ähnliches Fenster mit zwei Feldern.

Die HTML-Hilfdatei wird genau wie die WinHelp-Hilfdatei in die Applikation eingefügt: Sie spezifizieren den HTML-Hilfdateinamen in der `HelpFile`-Eigenschaft. Darüber hinaus können Sie die `HelpContextID`-Eigenschaft für jedes Objekt auf dem Formular setzen, damit eine kontextabhängige Hilfe angezeigt werden kann, wenn das Objekt auf dem Bildschirm selektiert ist und der Benutzer Hilfe anfordert.

Die eigentliche Arbeit dabei ist, die Hilfdatei zu erstellen. Im nächsten Abschnitt werden Sie ein recht kompliziertes Beispiel kennenlernen, für das eine WinHelp-Hilfdatei angelegt wird. Obwohl diese Datei relativ umfangreich ist, ist sie immer noch unvollständig. Selbst für einfache Applikationen können sehr viele Hilfebildschirme erforderlich sein, und je mehr vollständige Hyperlinks Sie einfügen, desto komplizierter wird die Arbeit.



HTML-Hilfdateien haben die Dateinamenerweiterung .CHM, WinHelp- Hilfdateien haben die Dateinamenerweiterung .HLP.



Häufig ist es nicht sinnvoll, eine komplizierte Hilfdatei anzulegen, egal ob HTML-Datei oder WinHelp-Datei. Dennoch sollten Sie es in den nächsten Abschnitten üben, weil Sie dann besser verstehen, wie das Hilfesystem mit der Applikation zusammenarbeitet. Wenn Sie dieses Kapitel bearbeitet haben, sollten Sie sich selbst einen Gefallen tun: Gehen Sie zu Ihrem Softwarehändler, und suchen Sie nach einem Programm für die Automatisierung von Hilfesystemen. Es gibt zahlreiche Programme, die Ihnen die Arbeit leichter machen.

Es gibt mehrere Werkzeuge - neben den Programmen zur automatischen Erstellung von Hilfesystemen -, mit denen Sie HTML-Seiten für die Hilfdateien anlegen können. Viele der heute verwendeten Textverarbeitungen, wie beispielsweise Microsoft Word 97, speichern und lesen HTML-Dateien. Statt einer Textverarbeitung können Sie jedoch auch ein Entwurfsprogramm für Web-Seiten verwenden, beispielsweise *FrontPage Express* im Internet Explorer 4. FrontPage Express kann von der Microsoft-Site heruntergeladen werden.

RTF-Hilfe

Die folgenden Abschnitte erklären, wie Sie mit der traditionellen WinHelp-Methode Hilfdateien anlegen und verknüpfen. Sie müssen nicht nur die Hilfdatei schreiben, sondern auch ein Hilfethema anlegen, das Sie dann in ein Hilfesystem kompilieren, welches mit der Applikation verknüpft wird. Der Text, der in Ihrer Hilfethemendatei steht, muß in dem speziellen RTF-Format (Rich Text Format) vorliegen, in dem die Applikation Hypertext-Sprünge von Thema zu Thema einbetten kann.

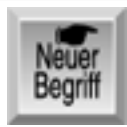


Ein *Hypertext-Sprung* ist eine Verknüpfung zu einem anderen Thema im Hilfesystem.



Sie brauchen eine Textverarbeitung oder einen Texteditor, womit Sie RTF- Dateien erzeugen können. Darüber hinaus muß die Textverarbeitung Fußnoten unterstützen. Microsoft Word ist die vielleicht gebräuchlichste Textverarbeitung, die für Hilfdateien verwendet wird. Wenn Sie nicht Word einsetzen, überprüfen Sie, ob Ihre Textverarbeitung RTF-Dateien und Fußnoten unterstützt.

Es gibt einen Grund, WinHelp-Hilfdateien zu verwenden: Jeder Benutzer kann auf diese Dateien zugreifen. Windows beinhaltet ein spezielles System zum Anlegen von Hilfdateien, den *Windows Help Viewer*, der die von Ihnen kompilierten RTF-Quelldateien verwaltet. (Der Windows Help Viewer kann keine HTML-Hilfdateien anzeigen; dafür brauchen Ihre Benutzer einen Web-Browser.)



Der *Windows Help Viewer* ist ein Windows-Werkzeug, das allen Benutzern zur Verfügung steht. Er zeigt WinHelp-Hilfdateien an.

Die Kompilierung für Hilfdateien hat nichts mit der Kompilierung Ihrer eigentlichen Applikation zu

tun. Im nächsten Kapitel erfahren Sie, wie Sie Ihre Applikation kompilieren, um sie an Ihre Benutzer weiterzugeben. Für Ihre Applikation - egal ob kompiliert oder nicht - muß eine Hilfedatei kompiliert werden, damit sie die Standarddateinamenerweiterung erhält, .HLP.

Vorbereitung der Themendatei

Wenn Sie Ihre Hilfedatei mit Word anlegen, sollten Sie die Formatierkennzeichen anzeigen. Die meisten Leute schreiben ihre Dokumente, ohne diese verborgenen Zeichen anzuzeigen, aber für Hypertext-Sprünge ist verborgener Text erforderlich, den Sie sehen müssen, auch wenn er später nicht mehr angezeigt wird. Klicken Sie auf die Schaltfläche **Einblenden/Ausblenden**, um verborgene Zeichen anzuzeigen.

Hypertext-Sprünge

Ein Großteil Ihrer Hilfedatei besteht aus normalem Text, der das Hilfethema beschreibt. Dafür ist keine spezielle Formatierung erforderlich, aber Sie können die Schriftgröße und den Schriftstil beliebig einsetzen. Für die Hypertext-Sprünge ist jedoch eine Formatierung nötig, so daß das Hilfesystem die Schlüsselwörter erkennt und weiß, wo sich die Themen befinden, zu denen eine Verknüpfung hergestellt werden soll.

Je mehr Hypertext-Sprünge Sie in Ihren Hilfethemen anbieten, desto praktischer ist das Hilfesystem für Ihre Benutzer. Wenn Sie Hypertext-Sprünge verwenden, brauchen Ihre Benutzer kein Menü, um relevante Themen auszuwählen, sonder »springen« einfach zu dem Thema, das sie lesen möchten.

Für Hypertext-Sprünge gelten die folgenden Voraussetzungen:

- Alle Hypertext-Sprungmöglichkeiten werden doppelt unterstrichen. Hypertext-Sprungmöglichkeiten erscheinen auf dem Bildschirm des Benutzers grün. In Word führen Sie eine doppelte Unterstreichung für ein Wort ein, indem Sie es selektieren, im **Format**-Menü den Eintrag **Zeichen** auswählen und aus dem Dropdown-Listenfeld **Unterstreichung** den Eintrag **Doppelt** auswählen. Sie können auch (Strg)+(°)+(D) drücken, um selektierten Text doppelt zu unterstreichen, oder einen Shortcut in Word dafür anlegen.
- Dem Hypertext-Sprung folgt ein eindeutiges Tag, der *Kontextstring*, der das Sprungziel enthält und als verborgener Text formatiert wird. Fügen Sie kein Leerzeichen zwischen dem Hypertext-Sprung und dem Kontextstring ein. Formatieren Sie nur den Kontextstring als verborgen, nichts anderes (auch keine Interpunktionszeichen oder Absatzmarkierungen). Text wird verborgen mit **Format, Absatz, Verborgenen**. Sie können auch (Strg)+(°)+(H) drücken, um selektierten Text zu verbergen, oder einen Shortcut in Word dafür anlegen.
- Die Themenseite mit dem Hypertext-Sprung wird durch einen Seitenumbruch von der Seite für das Sprungziel getrennt. Ein Seitenumbruch wird mit **Einfügen, Manueller Wechsel, Seitenwechsel** erzeugt (oder mit (Strg)+(¶)).
- Der Text für den Hypertext-Sprung wird mit mindestens einem der drei folgenden Fußnotensymbole mit der Sprungseite verbunden:

Symbol	Beschreibung
#	Für die Verbindung zur Sprung-Seite über den Kontextstring.
\$	Plaziert den Titel der Sprung-Seite im Textfeld Suchen und verbindet den Hypertext mit dem Titel der Sprung-Seite.
K	Verbindet ein Suchthema mit einem oder mehreren Schlüsselwörtern.

- Viele Hilfethemen richten mit allen drei Fußnotensymbolen eine Verknüpfung zu ihren Sprung-Seiten ein. Mit anderen Worten, die Benutzer können von Thema zu Thema springen, der Titel des Themas erscheint im **Suchen**-Textfeld, und die Benutzer finden Themen, indem sie mit mehreren Schlüsselwörtern suchen.
- Popup-Hilfebeschreibungen und Definitionen werden mit einfacher Unterstreichung gekennzeichnet. In Word unterstreichen Sie ein Wort, indem Sie es selektieren, im **Format**-Menü den Eintrag **Zeichen** auswählen und aus dem Dropdown-Listefeld **Unterstreichung** den Eintrag **Einfach** auswählen. Sie können auch (Strg)+(a)+(U) drücken, um selektierten Text doppelt zu unterstreichen, oder in der Symbolleiste auf die entsprechende Schaltfläche klicken.

Eine Hilfedatei anlegen

Sie haben in diesem Kapitel bereits gesehen, wie ToolTips funktionieren. Dieser Abschnitt erzeugt ein zusätzliches Hilfesystem für das MDI-Beispielprogramm. Die Applikation verwendet MDI-Formulare für die Verwaltung eines kleinen Texteditors, in dem mehrere Fenster geöffnet werden können. Der Texteditor ist relativ vollständig und viel besser als Wordpad von Windows (weil er mehrere Fenster unterstützt), aber es gibt neben den ToolTips keine Online-Hilfe.



In diesem Beispiel wird die Hilfedatei mit Microsoft Word angelegt. Abhängig von Ihren Systemvoraussetzungen können Sie aber auch eine andere Textverarbeitung verwenden.

Abbildung 20.3 zeigt ein Beispiel für einen Hilfebildschirm in Word. Beachten Sie, daß die doppelt unterstrichenen Passagen die Hypertext-Sprünge sind, die auf dem Benutzerbildschirm grün dargestellt werden. Der gesamte verborgene Text wird angezeigt; der punktiert unterstrichene Text stellt den verborgenen Text dar, der die Kontextstrings enthält.

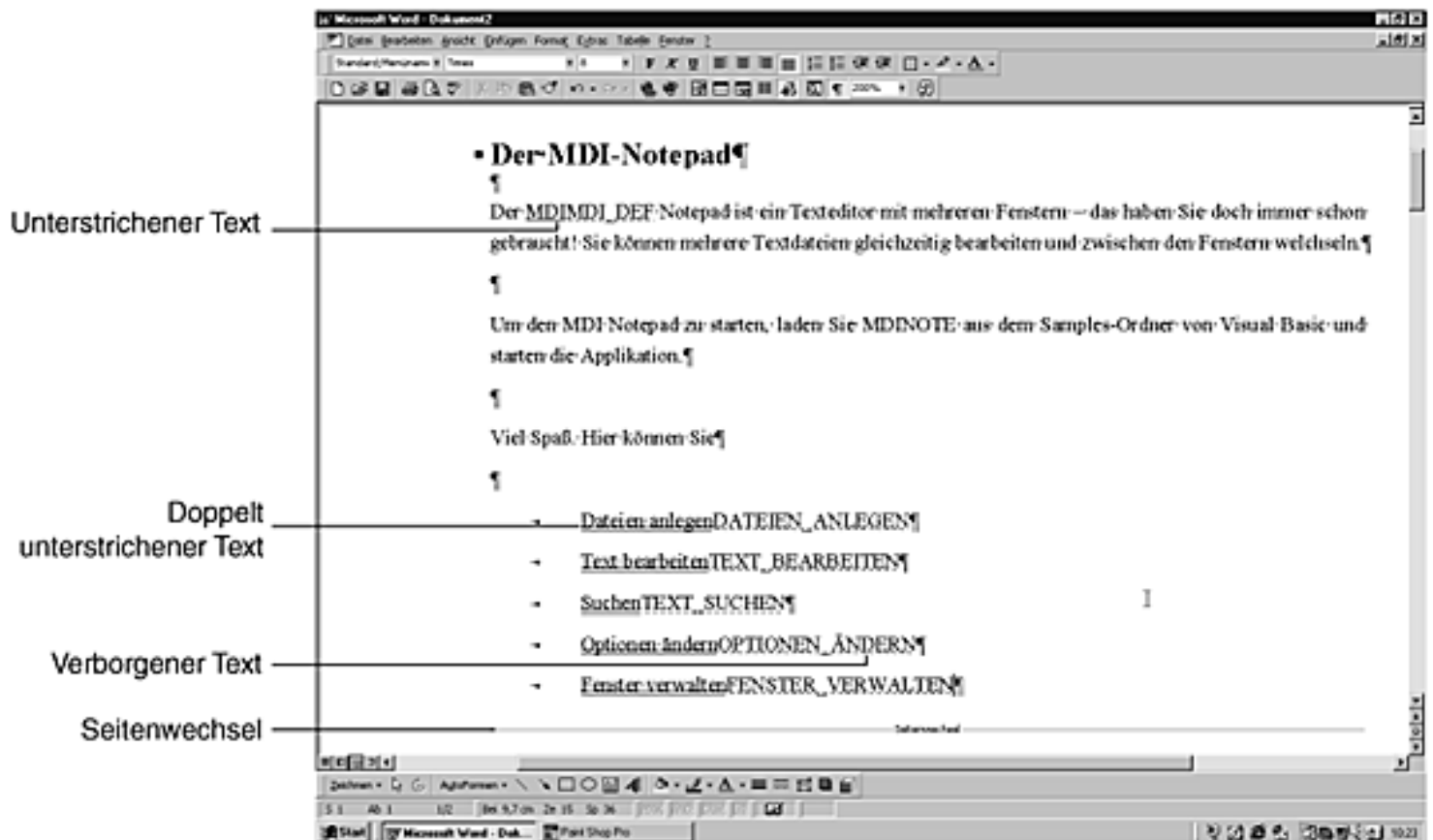


Abbildung 20.3:
Der Hilfebildschirm von MDI Notepad enthält Hypertext-Sprünge.

Abbildung 20.3 zeigt sechs Sprung-Kontextstrings: MDI_DEF, DATEIEN_ANLEGEN, TEXT_BEARBEITEN, TEXT_SUCHEN, OPTIONEN_ÄNDERN und FENSTER_VERWALTEN. Es muß also mindestens sechs weitere Word-Seiten unterhalb des Eröffnungsbildschirms geben. Sie verbinden diese Seiten mit Hilfe der speziellen Fußnotenkennzeichner mit den Hypertext-Sprüngen. Der erste Sprung, MDI_DEF, zeigt ein Popup-Definitions-feld für den Begriff *MDI* an.

Die gesamte Hilfedatei braucht eine Hilfekontext-ID, damit die zugrundeliegende Applikation auf diesen Hilfebildschirm zugreifen kann. Abbildung 20.4 zeigt zwei Fußnoten, die für den Eröffnungsbildschirm angelegt wurden. Um eine Fußnote einzufügen, setzen Sie den Textcursor vor das erste Zeichen im Titel, wählen im **Einfügen**-Menü den Eintrag **Fussnote** und geben das benutzerdefinierte Symbol # ein, das die Seitenposition für den Hpyertext-Sprung darstellt. Wiederholen Sie diese Schritte, um die Fußnoten \$ einzugeben. Die beiden Fußnotensymbole erscheinen links vom Text sowie neben dem entsprechenden Fußnotentext im unteren Fenster. Die Applikation kann diese Hilfekontext-ID nutzen, um auf den Hilfebildschirm zuzugreifen. Die Suchwerkzeuge der Help-Engine können den Titel anzeigen, der neben dem Fußnotensymbol \$ erscheint.



Verwenden Sie das Fußnotensymbol K nicht für den Eröffnungsbildschirm. K ist für Popup-Fenster reserviert, die angezeigt werden, wenn der Benutzer auf ein unterstrichenes Hilfethema klickt. Die Fußnote K zusammen mit # und \$ bewirkt, daß jedes Thema in der Themenliste des Hilfe-Dialogfelds erscheint.

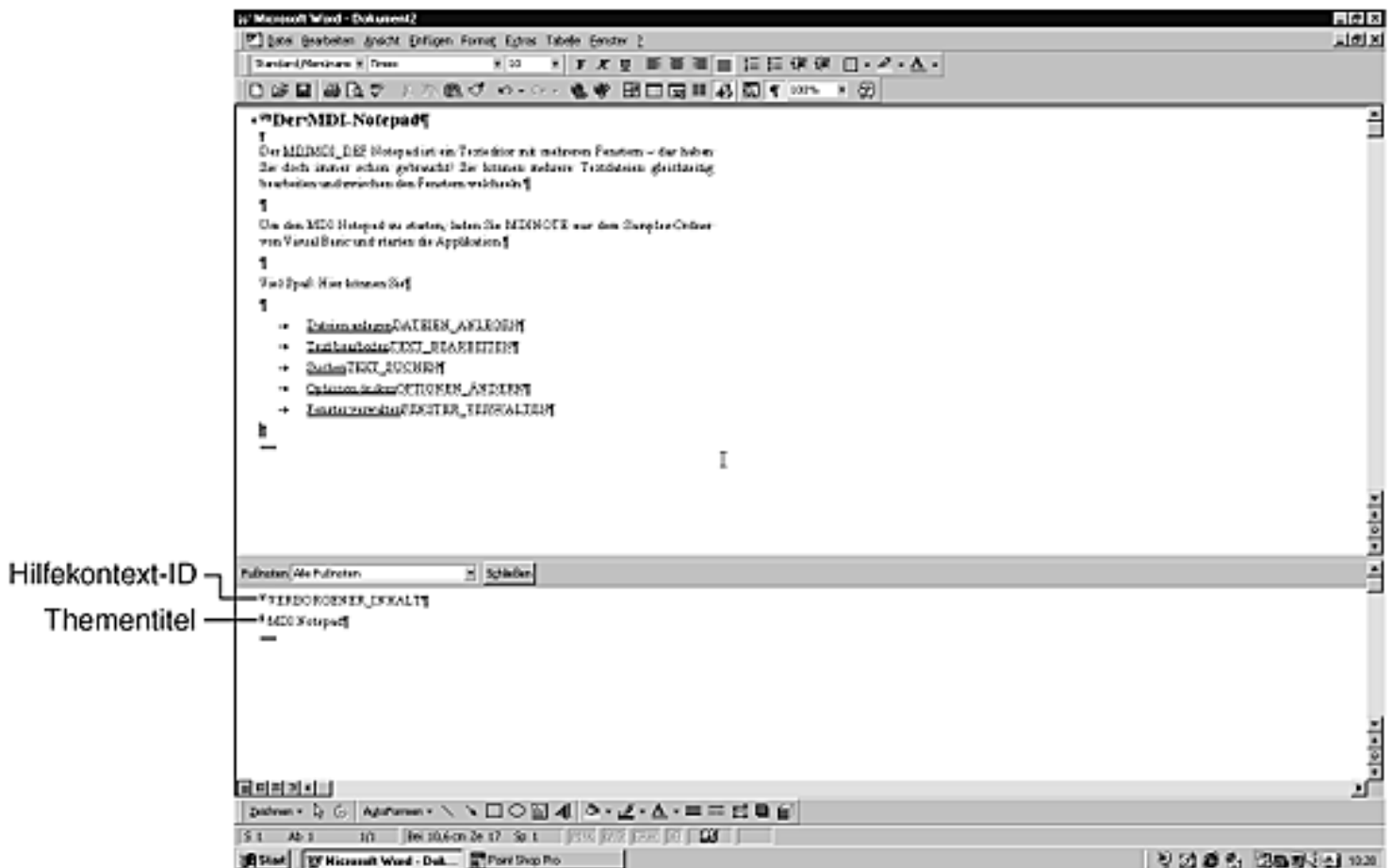


Abbildung 20.4:
Der Eröffnungsbildschirm erscheint, wenn nach seiner Kontext-ID oder seinem Titel gesucht wird.

In den nächsten Abschnitten werden Sie erfahren, daß das RTF-Dokument eine andere Hilfekontext-ID verwendet, als Visual Basic verwenden möchte. Sie müssen also den Text für die Kontext-ID-Werte in numerische Werte umwandeln, damit ein Applikation die kontextabhängige Hilfe verwenden kann.

Jetzt brauchen Sie noch Hilfeseiten für die restlichen Hypertext-Sprünge sowie Fußnoten, die die Seiten mit dem Eröffnungsbildschirm verknüpfen. Als erstes wird eine Popup-Definition für MDI angelegt. Die Seite unterhalb des Eröffnungsbildschirms muß die folgenden Informationen enthalten:

- Den MDI-Titel in der ersten Zeile
- Eine trennende Leerzeile
- Die Definition von MDI

Die Fußnote in Abbildung 20.5 fügt der Definition eine Hilfekontext-ID hinzu und realisiert so die Verbindung zwischen dieser Seite und der MDI-Position auf der Eröffnungsseite. Darüber hinaus zeigt das Fußnotensymbol K an, daß die verbundene Seite eine Popup-Definition darstellt, keinen Hypertext-Sprung.



Die Hilfedatei in diesem Kapitel verwendet dieselbe Hilfekontext-ID (in Großbuchstaben) wie der Thementitel, mit dem die Verknüpfung eingerichtet werden; diese beiden Werte müssen jedoch nicht gleich sein.

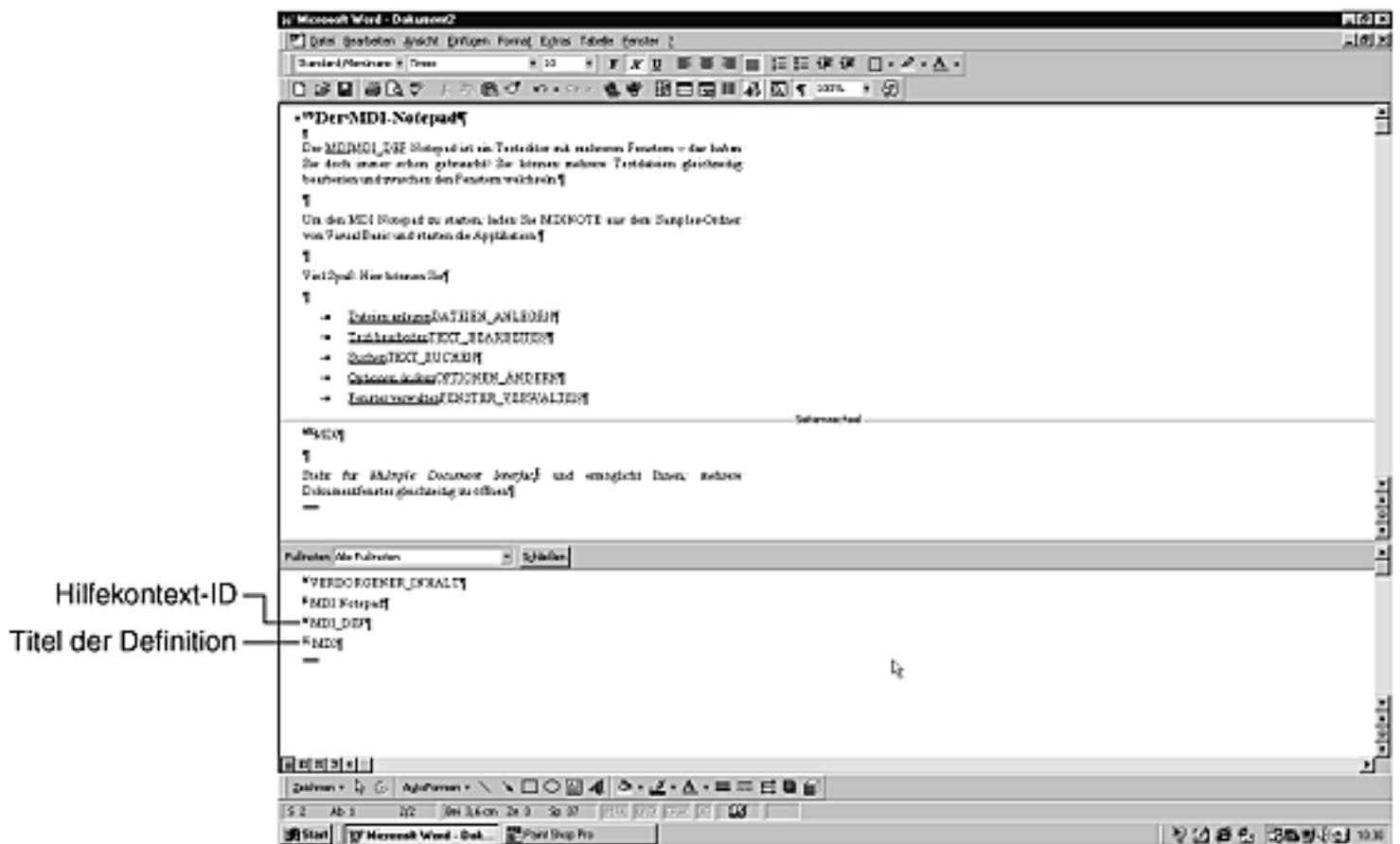


Abbildung 20.5:
Die Definition wird mit Hilfe des Fußnotensymbols K angezeigt.

Abbildung 20.6 zeigt den ersten Teil der restlichen Hilfethemen. Das Fußnotensym- bol # verbindet den Hypertext-Sprung für ein Thema mit dem entsprechenden Sprungziel.



Alle diese untergeordneten Sprungziele können wiederum Links zu anderen Seiten und Popup-Definitionen enthalten.

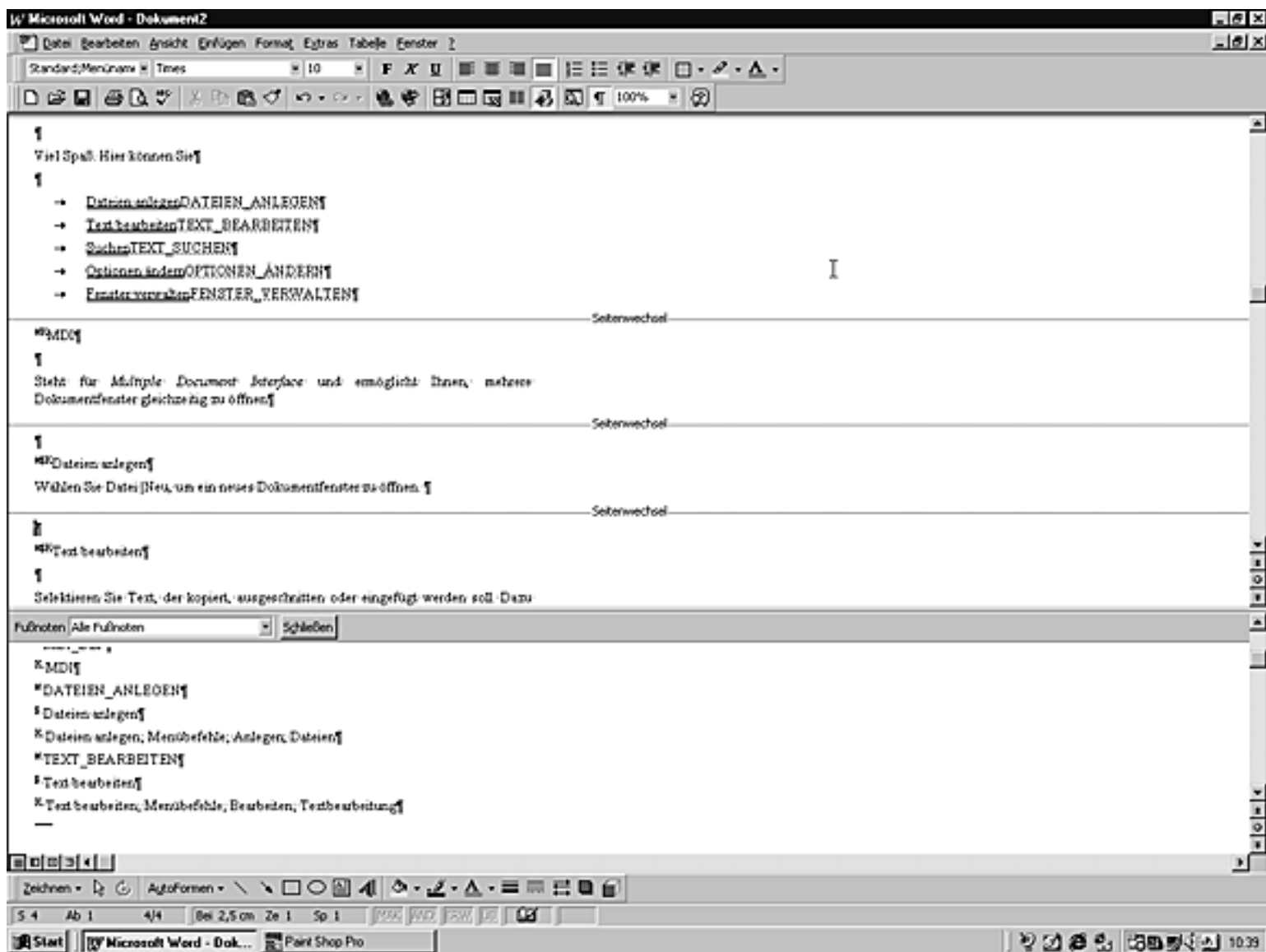


Abbildung 20.6:
Jetzt werden der Eröffnungsseite die weiteren Sprungziele hinzugefügt.

Nachdem Sie Ihre RTF-Hilfedatei fertiggeschrieben haben, speichern Sie das Dokument. Wählen Sie dazu den Dateityp **Rich Text Format** aus. Jetzt legen Sie die Hilfeprojektdatei mit einem weiteren Dateityp an - als ASCII-Textdatei. Word ist in der Lage, ASCII-Textdateien zu speichern. Die folgende Projektdatei wurde für die oben beschriebene Hilfedatei verwendet:

```

[OPTIONS]
contents=HID_CONTENTS
title=MDI Notepad Help
  
```

```

[FILES]
MDINote.rtf
  
```

Der [OPTIONS]-Abschnitt enthält die Kontext-ID der Eröffnungsseite und den Text in der Titelleiste. Der [FILES]-Abschnitt enthält die Hilfedatei, die kompiliert werden soll (möglicherweise müssen Sie einen Pfadnamen angeben, wenn sich die Datei an einer speziellen Position befindet). Geben Sie den Namen der zuvor erzeugten RTF-Hilfedatei ein. Im Hilfe-Compiler können Sie auch noch weitere Optionen für Ihre Hilfedatei setzen.

Speichern Sie die Projektdatei unter einem beliebigen Namen (am besten unter dem Namen der Applikationsdatei). Verwenden Sie jedoch jetzt die Dateinamenerweiterung **.HPJ**.

Führen Sie den Hilfe-Compiler von Ihrer Installations-CD aus, weil er bei der normalen Installation nicht berücksichtigt wird. Führen Sie das Hilfesystem wie folgt aus:

1. Legen Sie die Installations-CD von Visual Basic in Ihr CD-ROM-Laufwerk ein.
2. Wählen Sie im **Start**-Menü den Eintrag **Ausführen**.
3. Führen Sie das Programm HCW.EXE aus, das sich im Ordner \Common\Tools auf der CD befindet. Abbildung 20.7 zeigt das Programmfenster, das erscheint, zusammen mit einem praktischen Tip.

Nachdem Sie den Hilfe-Workshop geöffnet haben, laden Sie Ihre Hilfe-Projektdatei. Klicken Sie in der Symbolleiste auf die Schaltfläche **Compile**, und übernehmen Sie die Standardwerte. Die Kompilierung beginnt. Lesen Sie die Warnungen und Fehler, die der Compiler möglicherweise anzeigt. Häufig treten Warnungen auf, die Ihr Hilfesystem gar nicht betreffen, aber Sie sollten jedenfalls versuchen, eine Hilfedatei anzulegen, für die es keine Warnungen gibt. Falls Fehler auftreten, kann der Compiler die Datei nicht kompilieren.

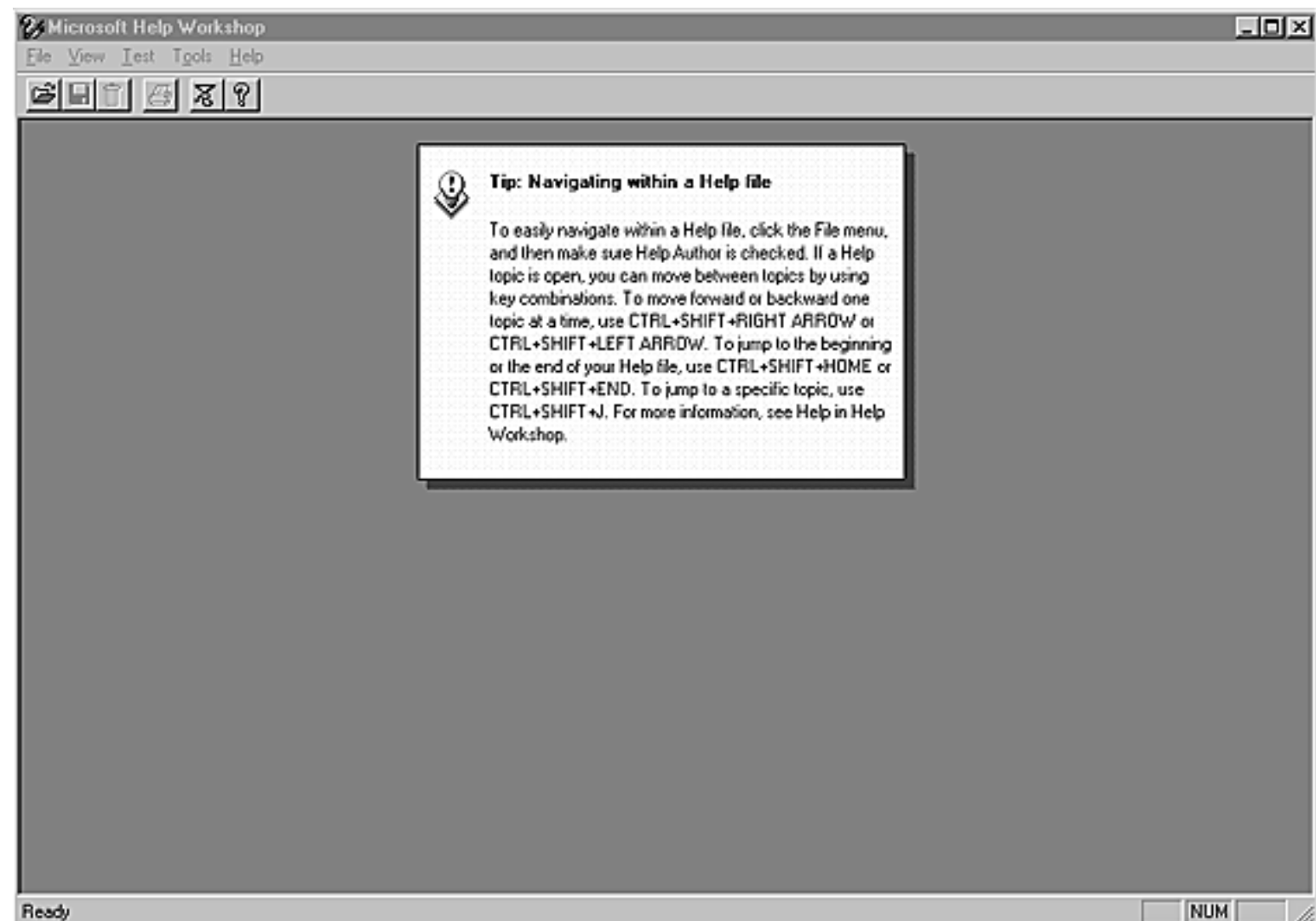


Abbildung 20.7:
Der Hilfe-Compiler zeigt beim Starten einen Tip an.

Nachdem Sie das Hilfesystem kompiliert haben, führen Sie es aus, um die Hilfeeinträge zu testen. Erst der nächste Abschnitt beschreibt, wie Sie Ihr Hilfesystem mit Ihrer Applikation verbinden. Sie können die Hilfedatei jedoch bereits testen:

1. Starten Sie den Windows-Explorer.

2. Gehen Sie in den Ordner, in dem Sie die Hilfedatei abgelegt haben. Für die Datei wird ein Hilfe-Icon angezeigt.

3. Klicken Sie mit der rechten Maustaste auf die Hilfedatei, und wählen Sie **Öffnen**. Die Windows-Onlinehilfe wird gestartet, und Sie können Ihre Hilfedatei ausprobieren.

Das erste Hilfefenster zeigt die Eröffnungsseite an. Die Indexseite zeigt eine vollständige Auflistung aller K-Fußnotenwerte an, die auf Hilfethemen verweisen.

Die Hilfedatei anzeigen

Nachdem Sie die Hilfedatei angelegt haben, verbinden Sie sie mit Ihrer Applikation. Die Hilfekontext-ID ordnet den Steuerelementen und anderen Teilen der Applikation Hilfethemen zu. Fügen Sie eine Hilfe-Option in Ihr Menü ein, damit die Benutzer jederzeit Hilfe aufrufen können.

Die Anzahl der Hilfethemen variiert von Applikation zu Applikation ganz drastisch. Anhand der Tiefe der Hilfedatei und der Komplexität der Applikation können Sie vorhersagen, wieviel Hilfe die Benutzer brauchen. Der restliche Abschnitt erklärt, wie Sie der Applikation eine Hilfedatei hinzufügen. (Visual Basic unterstützt mehrere Methoden, einer Applikation eine Hilfedatei hinzuzufügen; hier werden die gebräuchlichsten beschrieben.)

Das Dialogfeld Projekteigenschaften

Das Dialogfeld **Projekteigenschaften**, das Sie in Abbildung 20.8 sehen, stellt die wichtigste Verknüpfung zwischen Ihrem Projekt und der Hilfedatei dar. Es stellt sicher, daß die Hilfedatei aufgerufen wird, wenn der Benutzer die (F1)-Taste drückt.

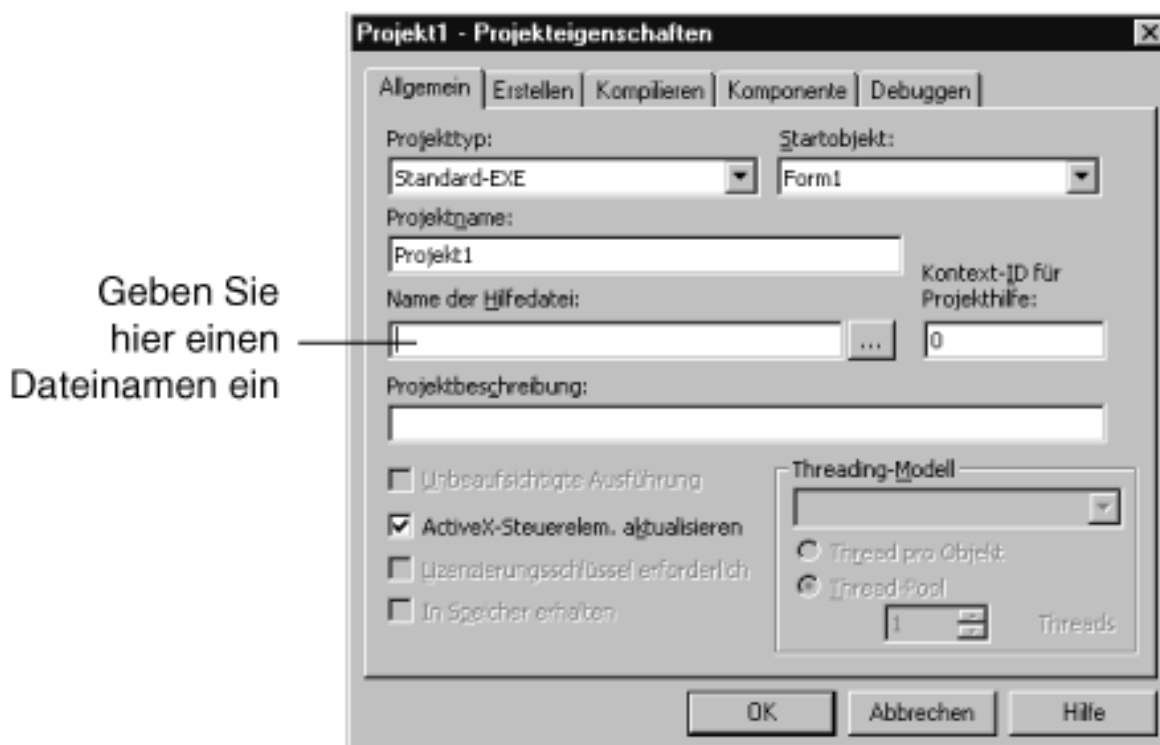


Abbildung 20.8:
Hier verbinden Sie die Hilfedatei mit Ihrem Projekt.

Das Standarddialogfeld

Die Verbindung des Hilfesystems mit der (F1)-Taste der Applikation ist ganz einfach. Das Fenster zeigt Ihre Hilfedatei an, weil Ihre Projekteigenschaften darauf verweisen. Wenn Sie Ihrer Applikation kontextabhängige Hilfe hinzufügen möchten, so daß die Benutzer (F1) drücken, um Hilfe zu erhalten, wenn ein Steuerelement den Fokus hat oder ein Menü ausgewählt ist, müssen Sie ein bißchen weiter gehen.

In Kapitel 9 haben Sie erfahren, wie man mit dem Standarddialog-Steuerelement verschiedene Dialogfelder erzeugt, beispielsweise zum Öffnen von Dateien oder zum Drucken. Nachdem Sie eine Hilfedatei angelegt haben, können Sie den Standarddialog nutzen, um interaktive Hilfebildschirme anzuzeigen.

Wenn Sie das Standarddialogfeld auf einem Formular anlegen, setzen Sie die `HelpFile`-Eigenschaft und wenden dann die `ShowHelp`-Methode an. Visual Basic führt die Hilfe-Engine von Windows aus, die Ihre Hilfedatei interpretiert und die üblichen Seiten für **Inhalt**, **Index** und **Suchen** bereitstellt, die auch andere Windows-Applikationen unterstützen.

Wenn Sie kontextabhängige Hilfe anbieten, müssen Sie der Hilfe-Engine genau mitteilen, welche Hilfefenster jeweils angezeigt werden sollen. Dazu bearbeiten Sie die Hilfeprojektdatei und ordnen den Kontext-IDs, die dort in Textform angegeben sind, eindeutige numerische Kontext-IDs zu. Das hört sich komplizierter an, als es eigentlich ist.



Wenn Sie nicht für alle Steuerelemente kontextabhängige Hilfe bereitstellen können, zeigt Visual Basic die Inhaltsseite der Hilfedatei an (den Eröffnungsbildschirm), falls der Benutzer versucht, die kontextabhängige Hilfe für ein Steuerelement aufzurufen, für das keine solche Hilfe bereitgestellt wird.

Jetzt ändern Sie die Projektdatei ab und ersetzen die Kontext-IDs, die in Textform vorliegen, durch numerische Kontext-IDs. Dazu fügen Sie der Projektdatei einen [MAP]-Abschnitt hinzu:

```
[OPTIONS]
contents=VERBORGENER_INHALT
title=MDI Notepad Help
```

```
[FILES]
MDINote.rtf
```

```
[MAP]
VERBORGENER_INHALT 1
MDI_DEF 2
DATEIEN_ANLEGEN 3
TEXT_BEARBEITEN 4
TEXT_SUCHEN 5
OPTIONEN_ÄNDERN 6
FENSTER_VERWALTEN 7
```

Achten Sie darauf, daß nie zwei gleiche Kontext-IDs verwendet werden. Die Zuordnung kann mit 1 beginnen, aber viele Visual-Basic-Programmierer reservieren bestimmte Zahlenbereiche, um verschiedene Hilfethemen darzustellen. Beispielsweise könnten alle Hilfethemen zu Schaltflächen in dem Bereich zwischen 1000 und 1050 abgelegt werden. Die Nummern müssen also nicht sequentiell sein. Kompilieren Sie die Projektdatei, damit die neuen Zuordnungen in der Hilfedatei in Kraft treten.

Jetzt beginnt die eigentliche Arbeit. Sie müssen jedes Steuerelement und jedes Formular ermitteln, für das Sie kontextabhängige Hilfe bereitstellen möchten. Im MDI Notepad könnten Sie beispielsweise das Formular `frmFind` (für die Menüoption **Suchen**) anzeigen und die `HelpContextID`-Eigenschaft dafür auf 5 setzen. Wenn das die einzige Änderung ist, die Sie vornehmen, wird dem Benutzer der Standard-Eröffnungsbildschirm angezeigt, wenn er die Applikation ausführt und (F1) drückt - außer wenn die **Suchen**-Schaltfläche den Fokus hat; in diesem Fall sieht der Benutzer das Hilfethema für die **Suchen**-Seite.

Listing 20.1 zeigt Code, der das Hilfethema mit einer bestimmten Kontext-ID anzeigt. Sie können solchen Code für eine Schaltfläche oder ein Hilfemenü verwenden, wenn Sie bestimmte Hilfethemen für Ihre Hilfedatei bereitstellen möchten.

Listing 20.1: Die Anzeige kontextabhängiger Hilfe.

```
cdbHelp.HelpFile = "MDINote.hlp" ' Auf die Hilfedatei zeigen
'
' Jetzt können Sie spezifische Hilfe für ein bestimmtes Thema
' anzeigen, indem Sie auf die Nummer im [MAP]-Abschnitt der HPJ
zeigen
' Ihre Text-Kontext-IDs)
cdbHelp.HelpContext = 3 ' Auf den Abschnitt verweisen
cdbHelp.HelpCommand = cdlHelpContex ' Kontextabhängige Hilfe
anfordern
cdbHelp.ShowHelp ' Hilfe anzeigen
```

Direkthilfe

Nachdem Sie nun wissen, wie man eine Hilfedatei anlegt, können Sie auch Direkthilfe zur Verfügung stellen. Für jede Direkthilfe, die Sie unterstützen möchten, brauchen Sie eine Hilfeseite. Nachdem Sie diese Hilfeseiten eingefügt und sie unter Verwendung der bereits vorgestellten Fußnotensymbole mit den anderen Hilfeseiten im System verknüpft haben, ordnen Sie den Seiten die numerischen Kontext-IDs zu. Die Direkthilfe-Engine verwendet diese Kontext-IDs, um die richtige Hilfedatei anzuzeigen, wenn der Benutzer an irgendeiner Stelle die Direkthilfe ausgewählt hat.

Für die Direkthilfe führen Sie zwei Schritte aus:

- Stellen Sie sicher, daß das Formular die Direkthilfe unterstützt, indem Sie die `WhatsThisButton`-Eigenschaft und die `WhatsThisHelp`-Eigenschaft auf `True` setzen.
- Geben Sie die Kontext-ID für die Hilfeseite in die `WhatsThisHelpID`-Eigenschaft des Objekts ein.

Um der in diesem Kapitel vorgestellten Beispielapplikation eine Direkthilfe hinzuzufügen, gehen Sie wie folgt vor:

1. Öffnen Sie das Projektfenster, und doppelklicken Sie auf das Formular `frmFind`, um das Formular für das **Suchen**-Dialogfeld anzuzeigen.
2. Weisen Sie den Eigenschaften `WhatsThisButton` und `WhatsThisHelp` den Wert `True` zu. Wenn Sie die Applikation ausführen, sehen Sie das Icon für die Direkthilfe im Formular.
3. Wiederholen Sie diese Zuweisung für die beiden anderen Formulare der Applikation.
4. Suchen Sie im [MAP]-Abschnitt der Hilfeprojektdatei nach den Kontext-IDs, und weisen Sie diese

den verschiedenen Menüoptionen (im Menü-Editor) und Formularen zu. Diese Hilfedatei ist jedoch keineswegs vollständig, aber mehrere der Hilfeseiten funktionieren schon ganz gut für die Objekte, insbesondere für die Einträge der Menüleiste.

Zusammenfassung

Dieses Kapitel hat Ihnen gezeigt, wie Sie Ihren Visual-Basic-Applikationen Hilfe hinzufügen. Den Visual-Basic-Entwicklern stehen unterschiedliche Hilfeformen zur Verfügung. Mit der HTML-Hilfe oder der Windows-Hilfe WinHelp können Sie ein vollständiges Hypertext-System mit Hilfeseiten und Popup-Definitionen anlegen. Ihr Benutzer erhält kontextabhängige Hilfe, wenn er auf (F1) drückt. Das Standarddialog-Steuerelement ermöglicht ebenfalls die Bereitstellung von Hilfe, die der Benutzer erhält, wenn er auf eine Schaltfläche klickt.

Mit der kontextabhängigen Hilfe erhalten Ihre Benutzer genau die Hilfe, die sie brauchen. Dabei kann man verschiedenen Objekten kontextabhängige Hilfe zuordnen, die erscheint, wenn der Benutzer ein Objekt selektiert und (F1) drückt. Damit bleibt es Ihren Benutzern erspart, den gesamten Index zu durchsuchen, wenn er Hilfe zu einem bestimmten Objekt benötigt.

Die ToolTips und die Direkthilfe sind zwei Hilfefunktionen, die Sie Ihren Applikationen einfach und schnell hinzufügen können. ToolTips sind extrem einfach. Sie brauchen dafür nur den ToolTips-Text im Eigenschaftenfenster zu setzen. Damit Sie eine Direkthilfe zuweisen können, brauchen Sie eine umfassende Hilfedatei, und Sie müssen Ihren Objekten numerische Kontext-IDs zuweisen.

Im nächsten Kapitel erfahren Sie, wie Sie Ihre Applikationen testen, debuggen und an Ihre Benutzer weitergeben.

Fragen und Antworten

Frage:

Warum kann ich die Hilfedatei nicht erst anlegen, nachdem ich die ganze Applikation fertig habe?

Antwort:

Sie können. Und das haben Sie in diesem Kapitel auch getan. Sie verstehen jedoch die Details einer Applikation besser, während Sie sie gerade entwickeln. Die Hilfe ist besser, wenn Sie sie während der Entwicklung anlegen. Halten Sie einfach Ihre Textverarbeitung in einem zweiten Fenster geöffnet, und tragen Sie immer wieder Hilfethemen ein.

Frage:

Warum ist es so müßig, Hilfe in Hilfedateien bereitzustellen?

Antwort:

Müßig ist, sicherzustellen, daß alle Hypertext-Sprünge zu den betreffenden Themen führen, daß alle Themen berücksichtigt werden, und daß die RTF-Datei richtig formatiert wird. Der Hilfe-Compiler erkennt fehlerhaft formatierte Hilfedateien, so daß Sie sie beim Kompilieren debuggen können. Wenn Sie Ihre Applikation kompilieren und testen, sollten Sie auch die Hilfedatei kompilieren, so daß sie synchron mit dem aktuellen Projekt bleibt. Die inkrementelle Erstellung einer Hilfedatei macht die Aufgabe insgesamt weniger müßig.

Es gibt viele Werkzeuge, die Sie dabei unterstützen, Hilfedateien für Ihre Visual-Basic-Programme zu

erzeugen, ohne RTF- oder HTML-Dateien anlegen zu müssen. Verwenden Sie keine HTML-basierte Hilfe, wenn Sie nicht genau wissen, daß Ihr Benutzer einen Internet-Browser einsetzt, der die HTML-Dateien anzeigen kann.

Mit den visuellen Programmen für die Bereitstellung von Hilfe ist es ganz einfach, Hilfedateien anzulegen. Darüber hinaus müssen Sie auch nicht so viel tippen, weil natürlich viele Hilfedateifehler auftreten, wenn Sie den RTF-Ansatz verwenden. Darüber hinaus ermöglichen Ihnen diese Programme, Hilfedialoge anzulegen, so daß Sie visuell darstellen können, wie die Hypertext-Sprünge verbunden sind. Damit ist noch weniger RTF-Codierung erforderlich, die Sie brauchen, um eine Hilfedatei anzulegen.

Frage:

Warum kann ich nicht einfach diese Werkzeuge einsetzen und den Stoff dieses Kapitels völlig ignorieren?

Antwort:

Visual Basic unterstützt die heute beschriebene Online-Hilfe ebenso wie andere Arten der Hilfe - beispielsweise HTML-basierte Hilfe. Sie können Werkzeuge von Drittanbietern einsetzen, um es sich sehr viel einfacher zu machen, Hilfe bereitzustellen. Wenn Sie sehr viele Applikationen entwickeln, sollten Sie diese Werkzeuge jedenfalls einsetzen. Sie wollen aber Visual Basic kennenlernen - und es enthält alle Werkzeuge, bis auf einen RTF-Editor, mit denen Sie die heute beschriebene Hilfe erzeugen können. Darüber hinaus haben nicht alle Ihre Benutzer Web-Browser, um HTML-Hilfe anzuzeigen. Dies ist also kein universeller Ansatz, es sei denn, Sie fordern, daß ein Web-Browser installiert wird, damit Ihre Applikation ausgeführt werden kann. Mit dieser »altmodischen« Methode, Hilfedateien anzulegen, wie Sie sie heute kennengelernt haben, erfahren Sie mehr über die Arbeitsweise von Hilfesystemen, und Sie verstehen, wie komplex ein Hypertext-System sein kann.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollen, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Welches Dateiformat verwenden Sie für eine Online-Hilfedatei?
2. Wie verbessern Hypertext-Sprünge ein Hilfesystem?
3. Welches benutzerdefinierte Fußnotensymbol erzeugt Popup-Hilfedefinitionen für den Text, der in der Hilfeseite doppelt unterstrichen dargestellt wird?
4. Nennen Sie einige Eigenschaften der Hilfe-Projektdatei.
5. Wie können Sie eine kompilierte Hilfedatei in eine Applikation einfügen, damit die Hilfe erscheint, wenn der Benutzer (F1) drückt?
6. Wie legen Sie kontextabhängige Hilfe für die Hilfethemen an?
7. Richtig oder Falsch? Die kontextabhängige Hilfe verwendet die Kontext-IDs, die in Ihrer Hilfedatei in Textform vorliegen.

8. Wie unterscheiden sich ToolTips und Direkthilfe?
9. Wie legen Sie die Direkthilfe auf einem Formular an?
10. Richtig oder Falsch? Sie können sowohl für Formulare als auch für Objekte Direkthilfe anbieten. (*Hinweis:* Sehen Sie sich die Eigenschaften des Formulars an.)

Übung

1. Fügen Sie jedem Objekt im MDI Notepad eine Direkthilfe hinzu. Diese Aufgabe erscheint etwas mühsam (und ist es auch irgendwie), aber Sie werden dadurch den Umgang mit der Direkthilfe, Kontext-IDs und Hilfeseiten schneller verstehen.

Woche 3

Tag 21

Applikationen veröffentlichen

Dieses Kapitel zeigt Ihnen, wie Sie Ihre Visual-Basic-Applikationen testen, debuggen und veröffentlichen. Keine Applikation ist je wirklich fertig. Sie könnten immer wieder eine Funktion einfügen, und selbst wenn Sie glauben, auch den letzten Fehler gefunden zu haben, taucht immer wieder ein Problem auf. Die Langzeitwartung ist Teil des Programmierprozesses. Sie können es sich jedoch leichter machen. In den ersten 20 Kapiteln dieses Buchs fanden Sie immer wieder Tips, wie Sie Ihren Code besser dokumentieren können und Wartungsprobleme mindern.

Am besten reduzieren Sie Probleme bei der Wartung, indem Sie Ihre Applikation sorgfältig testen und debuggen, bevor Sie sie an Ihre Benutzer weitergeben. Dieses Kapitel beschreibt einige der Debugging-Werkzeuge von Visual Basic, ebenso wie die Testprozeduren, die Sie durchführen könnten, bevor Sie Ihre Applikation veröffentlichen.

Sie haben das letzte Kapitel erreicht. Im Laufe dieses Buches sind Sie zum Profi für Visual Basic geworden. Jetzt sollten Sie so viel wie möglich mit Visual Basic arbeiten, um Ihre Fertigkeiten zu festigen.

Heute lernen Sie die folgenden Dinge kennen:

- Programmfehler
- Fehler während der Codierung erkennen
- Die Debugger-Fenster
- Einzelschrittmodus
- Mehrfache Haltepunkte
- Installationsroutinen
- Der Paket- und Weitergabe-Assistent

Debuggen und Testen

Alle Applikationen müssen getestet werden. Während der Programmierung schleichen sich zu viele Fehler ein. Beim Testen führen Sie die Applikation in verschiedenen Situationen aus. Dabei geben Sie für alle Steuerelemente, in denen der Benutzer Eingaben vornehmen kann, Zufalls- und Extremwerte ein, um sicherzustellen, daß die Applikation auch Werte verarbeiten kann, die nicht im Normalbereich liegen.

Das Debugging erfolgt in drei Phasen:

1. Fehler und ihre Position ermitteln
2. Fehler korrigieren
3. Testen, ob die Fehler wirklich eliminiert wurden

Es gibt weniger schlimme Fehler, wie beispielsweise Schreibfehler oder eine falsche Textausrichtung, und ernsthaftere, bei denen die Applikation die gesamte Windows- Sitzung abstürzen läßt und die zu einem Datenverlust führen. Für Ihre Benutzer ist ein Fehler alles, was verhindert, daß das Programm die erwarteten Ergebnisse erzeugt oder was die Programmausführung unmöglich macht.

Programmierer haben bei der Fehlersuche häufig Debugging-Probleme. Sie müssen sicherstellen, so viele Fehler wie möglich zu finden, und dann dafür sorgen, daß sie auch wirklich behoben sind und nicht mehr wieder auftauchen. Eine sorgfältige Planung vor, während und nach dem Codierprozeß hilft Ihnen, die Debugging-Phase für Ihre Applikation zu verkürzen.



Sie sollten Ihre Applikation in der Visual-Basic-Entwicklungsumgebung testen. Dort finden Sie Debugging-Werkzeuge, die Ihnen helfen, Fehler zu erkennen und zu finden. Erst nachdem Sie mit Ihren Testergebnissen zufrieden sind, sollten Sie Ihre Applikation kompilieren und an Ihre Benutzer weitergeben.

Windows und die leistungsfähige Visual-Basic-Entwicklungsumgebung kann Ihnen bei der Fehlersuche helfen. Wenn Sie eine Visual-Basic-Applikation ausführen, findet Visual Basic während der Kompilierung oder der Vorbereitung auf die Programmausführung manche Fehler (beispielsweise falsch geschriebene Schlüsselwörter) und zeigt Fehlermeldungen dafür an, wie in Abbildung 21.1 gezeigt.

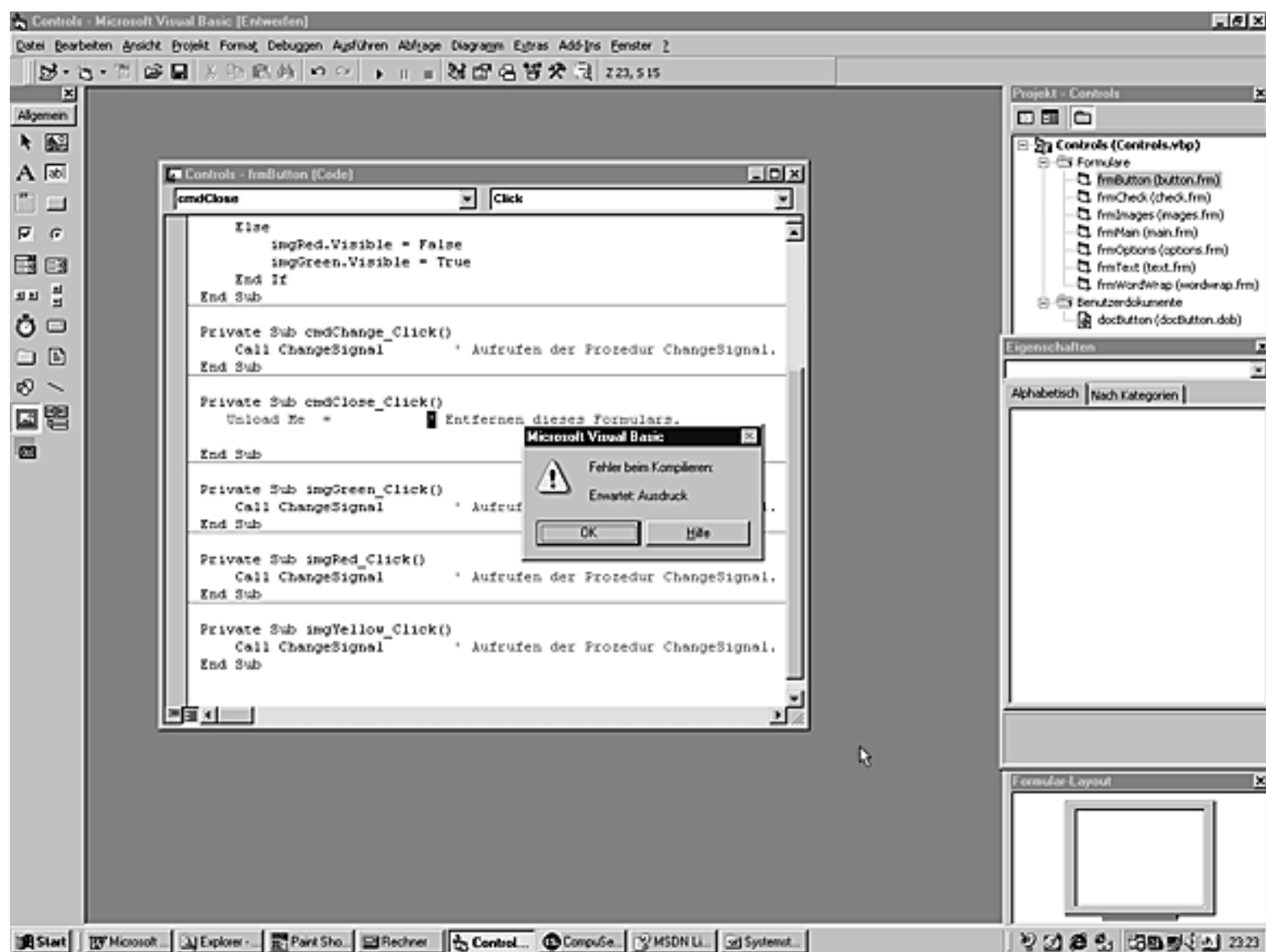


Abbildung 21.1:
Visual Basic hilft Ihnen, Fehler zu finden.

Wenn Sie eine Applikation ausführen und eine solche Fehlermeldung sehen, bevor das erste Formular auf dem Bildschirm angezeigt wird, haben Sie vermutlich einen Syntaxfehler im Code. Bei dem in Abbildung 21.1 gezeigten Fehler handelt es sich um einen Syntaxfehler. Die Fehlermeldung enthält zwar selten den Begriff »Syntaxfehler«, aber bei Problemen, die durch eine fehlerhafte Schreibweise oder eine falsch verwendete Grammatik entstehen, ist die Ursache in der Regel ein Syntaxfehler.

	Ein <i>Syntaxfehler</i> ist ein Fehler, der durch eine falsche Anwendung der Grammatik oder durch Schreibfehler entsteht.

Beachten Sie, daß Visual Basic in Abbildung 21.1 nicht nur auf einen Fehler hingewiesen hat, sondern diesen auch im Codefenster anzeigt. Auch wenn das Codefenster geschlossen ist, wenn Sie versuchen, ein Programm auszuführen, markiert Visual Basic den Fehler. Das Problem dabei war ein überflüssiges Gleichheitszeichen. Nachdem Sie den Syntaxfehler korrigiert haben, klicken Sie in der Symbolleiste auf die Schaltfläche **Starten** und starten das Programm erneut.

Wenn Sie auf der Registerkarte **Editor** im Dialogfeld **Extras, Optionen** das Kontrollkästchen **Automatische Syntaxüberprüfung** markiert haben, prüft Visual Basic, ob während der Codeeingabe Syntaxfehler auftreten. Einige Programmierer bevorzugen, beim Programmwurf mehr Freiheiten zu

haben und zum Teil unvollständige Anweisungen zu schreiben, die sie später ergänzen. Dieser unvollständige Code kann jedoch leicht zu Fehlern führen, wenn Sie nicht sorgfältig genug vorgehen und möglicherweise vergessen, vielleicht, eine Anweisung zu korrigieren. Nichtsdestotrotz gibt es Situationen, wo Sie Code später ergänzen möchten, beispielsweise nachdem Sie bestimmte Entwurfsentscheidungen mit dem Benutzer besprochen haben.

Für diesen Fall können Sie die automatische Syntaxüberprüfung deaktivieren. Wenn die Option deaktiviert ist, prüft Visual Basic nicht, ob bei der Eingabe Codierfehler auftreten, wie beispielsweise fehlende Klammern, bis Sie das Programm ausführen oder kompilieren. Irgendwann wird Visual Basic diese Fehler für Sie finden und die entsprechenden Meldungsfelder anzeigen (siehe Abbildung 21.1), aber mit Hilfe der Option **Automatische Syntaxüberprüfung** können Sie entscheiden, wann die Fehler angezeigt werden sollen.

Kompliziertere Fehler treten zur Laufzeit einer Applikation auf. Ein Syntaxfehler ist leicht zu erkennen, weil Visual Basic ihn für Sie findet. Ein Laufzeitfehler ist schwieriger zu finden und zu korrigieren. Betrachten Sie den in Abbildung 21.2 gezeigten Fehler. Er betrifft die Programmlogik. Offensichtlich wurde ein Adreßfeld geladen, wo ein Namensfeld erscheinen soll. Visual Basic erkennt nicht, daß das nicht stimmen kann, weil es einfach den Anweisungen des Programmierers folgt, auch wenn diese einen logischen Fehler darstellen.



Abbildung 21.2:
Visual Basic erkennt keine logischen Fehler.

Beim Auftreten von Logikfehlern müssen Sie das Programm beenden. (Visual Basic erkennt den Fehler nicht und beendet deshalb auch nicht das Programm, wie es bei Syntaxfehlern der Fall ist.) Anschließend müssen Sie versuchen, das Problem zu finden.

Dazu durchsuchen Sie den Programmcode nach Spuren, wo sich der Logikfehler befinden könnte, und versuchen dann, ihn zu beheben. Wenn das Problem das Erscheinungsbild eines Formulars oder eines Steuerelements auf dem Bildschirm beeinflußt, müssen Sie alle Verweise auf diesen Teil des Objekts überprüfen. Häufig kann Ihnen der Objektkatalog helfen, diesen Code für ein Objekt zu finden, aber nicht immer.

Visual Basic kann *ein paar* Logikfehler erkennen, wenn diese etwas fordern, was gar nicht möglich ist. Abbildung 21.3 zeigt, was passiert, wenn ein Programm Visual Basic auffordert, eine Zahl durch Null zu dividieren. Die Division durch Null ist mathematisch nicht definiert. Visual Basic kann also diese

Berechnung nicht durchführen, auch wenn in der Rechenvorschrift selbst kein Fehler vorliegt. Visual Basic unterbricht die Programmausführung und beschreibt den Fehler in einem Meldungsfeld.

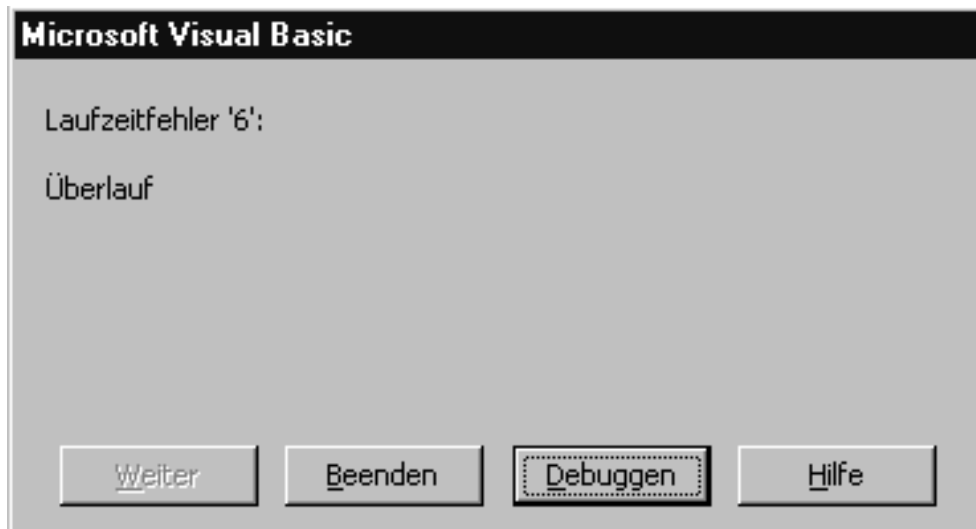


Abbildung 21.3:
Einige Logikfehler fordern von Visual Basic, etwas Unmögliches zu tun.

Bei der Ausführung der Applikation wird das Codefenster nicht angezeigt. Sobald Visual Basic erkennt, daß das Programm etwas Unmögliches fordert, zeigt es das Codefenster an und geht im Code an die Position, wo die Division durch Null versucht wird. Klicken Sie auf die **Hilfe**-Schaltfläche in dem Dialogfeld, um weitere Informationen über den Fehler zu erhalten. Mit **Ende** beenden Sie die Programmausführung, und mit **Debuggen** gehen Sie in den Debuggingmodus.

Der Debugger

Die Entwicklungsumgebung von Visual Basic enthält ein Debugging-Werkzeug, das Teil der Entwicklungsumgebung wird, wenn Sie Debugging-Hilfe anfordern. Dieses Werkzeug ermöglicht Ihnen, die folgenden Aufgaben auszuführen:

- Variableninhalte zur Laufzeit analysieren.
- Das Programm an einer beliebigen Anweisung anhalten und später fortsetzen.
- *Haltepunkte* im Code setzen, die die Programmierung automatisch unterbrechen, sobald sie erreicht sind.
- Änderung der Variableninhalte während der Programmausführung.
- Überwachungsausdrücke setzen, die die Programmausführung anhalten, wenn sie einen bestimmten Wert annehmen.
- Anweisungen überspringen, die während eines Tests nicht ausgeführt werden sollen.
- Im Ausgabefenster für das **Debug**-Objekt während der Programmausführung Werte ausgeben. Das Debug-Fenster ermöglicht Ihnen, Ausgaben aufzuzeichnen, beispielsweise Variablenwerte, ohne das normale Formularfenster dafür nutzen zu müssen.

So wechseln Sie in den Debuggingmodus und nutzen alle Funktionen des Debuggers:

- Drücken Sie (Strg)+(Untbr), um die Programmausführung zu unterbrechen.
- Beim Auftreten eines Laufzeitfehlers.
- Wenn die Applikation einen gesetzten Haltepunkt erreicht.
- Durch Anklicken einer Anweisung im Programm und Auswahl von **Ausführen bis Cursor**, so

daß das Programm wie üblich ausgeführt wird. Visual Basic unterbricht das Programm und geht in den Debugging-Modus, sobald die Ausführung den Cursor erreicht hat.

Haltepunkte setzen

Am einfachsten ist der Haltepunkt **Ausführen bis Cursor** zu setzen. Um Haltepunkte zu testen, laden sie die Applikation **Controls** aus Ihrem Samples-Ordner. Das Formular **Schaltflächen-Beispiel**, **frmButton**, ändert eine Ampel, wenn der Benutzer auf bestimmte Schaltflächen klickt. Angenommen, der Code in der Applikation ändert die Ampel nicht richtig. Sie könnten auf die erste ausführbare Anweisung in der Datenbank **ChangeSignal()** im Standardmodul anklicken und im **Debuggen**-Menü den Eintrag **Ausführen bis Cursor** wählen. (Verwenden Sie keinen Kommentar, weil dadurch kein Haltepunkt gesetzt wird.) Das Programm wird gestartet wie gewohnt, hält jedoch an dem Haltepunkt an und markiert die entsprechende Zeile.

Das Programm ist damit nicht beendet. Alle Variablen sind initialisiert, der Code wurde ausgeführt, und die Aktionen waren dieselben wie bei einer normalen Programmausführung. Wenn eine Ausgabe erfolgt, bevor die Cursorposition erreicht ist (wie es hier der Fall ist, wenn das erste Formular angezeigt wird), sehen Sie diese Ausgabe wie gewohnt auf dem Bildschirm. Die Titelleiste des Programms zeigt an, daß es unterbrochen ist. Die gelbe Markierung kennzeichnet die Zeile, wo der Cursor stand, als sie im **Debuggen**-Menü den Eintrag **Ausführen bis Cursor** ausgewählt haben.

Listing 21.1 zeigt die Prozedur, in der die Ausführung in unserem Beispiel unterbrochen wurde.

Listing 21.1: An einem Haltepunkt können Sie einzelne Prozeduren analysieren

```
1: Private Sub ChangeSignal()  
2: ' Prüfen, welche Farbe die Ampel hat, und sie dann auf die  
3: ' nächste Farbe setzen. Die Reihenfolge ist Grün, Gelb,  
4: ' Rot.  
5: If imgGreen.Visible = True Then  
6: imgGreen.Visible = False  
7: imgYellow.Visible = True  
8: ElseIf imgYellow.Visible = True Then  
9: imgYellow.Visible = False  
10: imgRed.Visible = True  
11: Else  
12: imgRed.Visible = False  
13: imgGreen.Visible = True  
14: End If  
15: End Sub
```

Vielleicht funktioniert die Ampel nicht, weil mehrere Farben gleichzeitig sichtbar sind. Sie können den aktuellen Wert der **Visible**-Eigenschaften für die drei möglichen Signale (**imgGreen**, **imgYellow** und **imgRed**) betrachten, um zu prüfen, ob nur eine davon den Wert **True** hat, wenn die Prozedurausführung beginnt.

Es war nie einfacher, den Inhalt eines Steuerelements (oder einer Variablen) einzusehen. Wie Abbildung 21.4 zeigt, müssen Sie nur den Mauszeiger über ein Steuerelement setzen, um den Wert einer Variablen oder den Eigenschaftswert eines Objekts anzuzeigen.

Angenommen, Sie überprüfen alle drei Werte und stellen fest, daß nur einer davon **True** enthält, was richtig ist. Damit haben Sie festgestellt, daß das Problem nicht in der Prozedur **ChangeSignal()**

liegt. Sie suchen also weiter im Code, wo die Signale durcheinandergeraten. Abhängig davon, wie das Problem sich während der Programmausführung präsentiert, könnten Sie das Programm ein Stück weiter ausführen. Durch eine Unterbrechung der Ausführung und die Analyse von Steuerelementen und Variablen können Sie feststellen, ob die Werte so gesetzt sind, wie sie zur Laufzeit aussehen sollten.

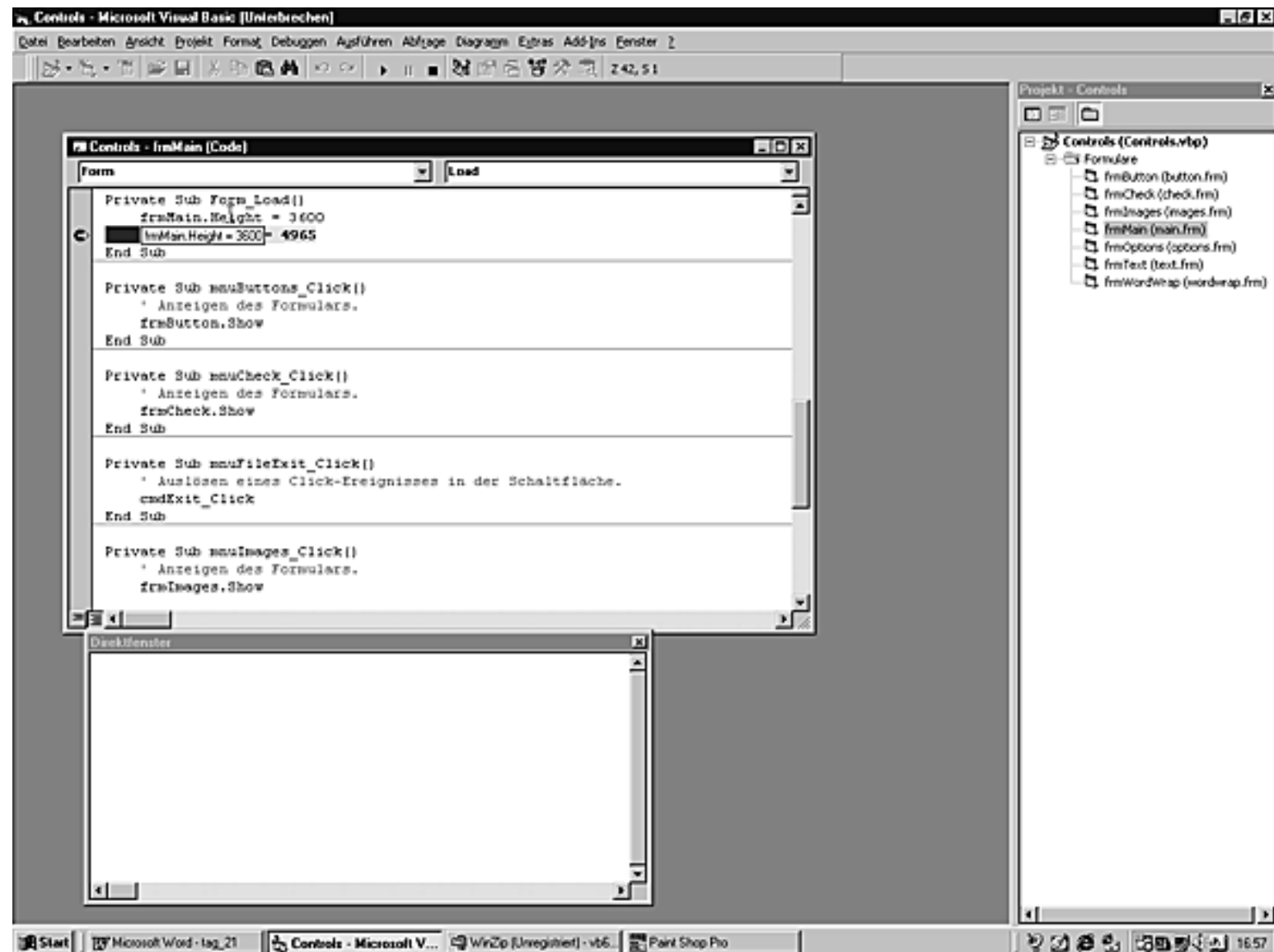
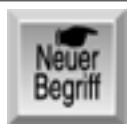


Abbildung 21.4:
Visual Basic zeigt alle Werte für Steuerelemente und Variablen an, wenn das Programm unter Verwendung des Debuggers angehalten wurde.

Verfolgen der einzelnen Schritte

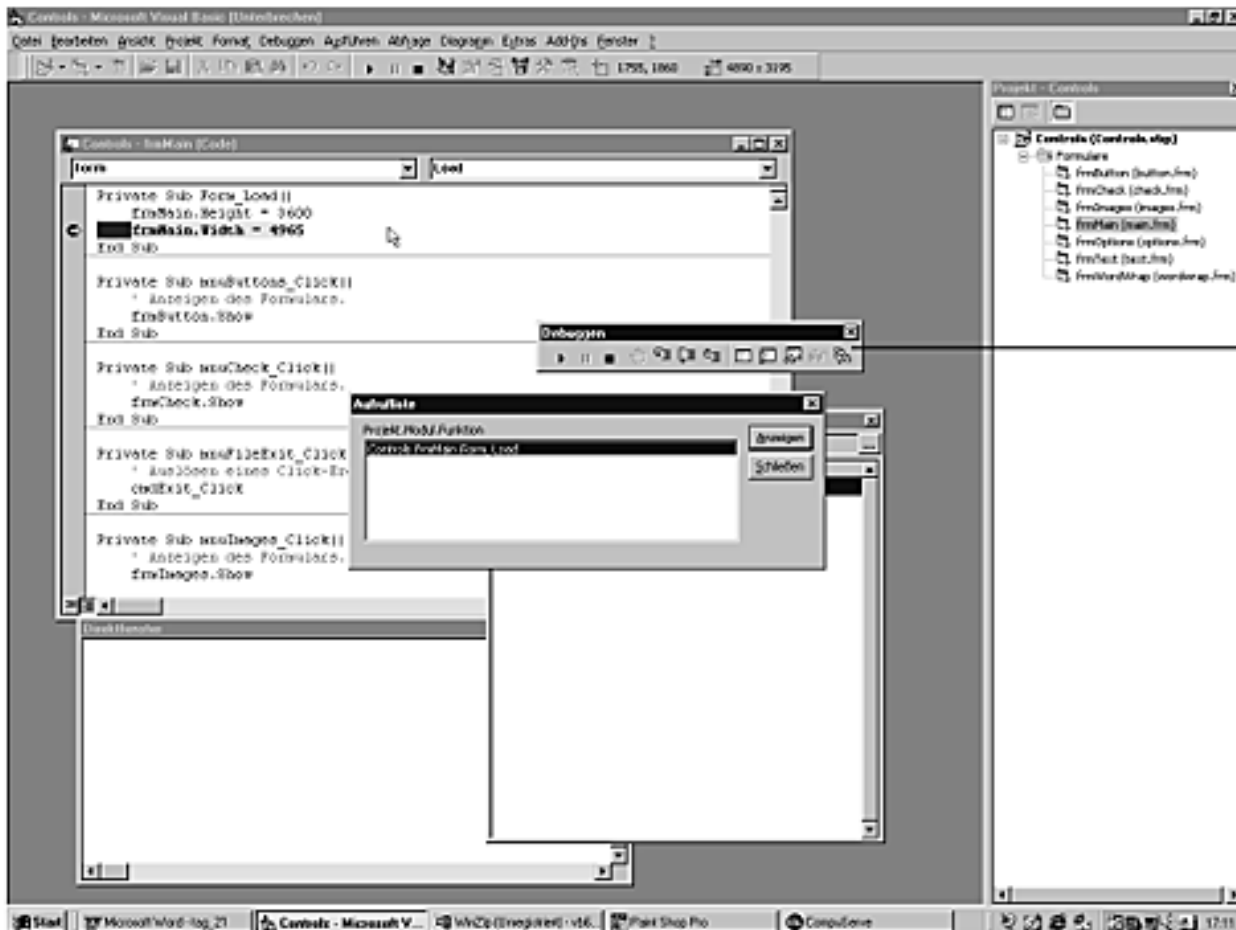
Bevor Sie Ihre Arbeit fortsetzen, zeigen Sie die **Debuggen**-Symbolleiste an. (Während des Debuggens kann die Symbolleiste an einer beliebigen Stelle angezeigt werden, Sie können sie jedoch auch oben im Fenster befestigen.) Wenn Sie sehen möchten, was die Programmausführung bis zum Erreichen eines bestimmten Haltepunkts gemacht hat, verwenden Sie eine der praktischsten Funktionen zum Debugging - die **Aufrufliste**. Klicken Sie in der Symbolleiste auf die Schaltfläche **Aufrufliste**. Wie in Abbildung 21.5 gezeigt, erscheint das Dialogfeld Aufrufliste und zeigt an, was das Programm bisher gemacht hat.



Das *Windows-API* (für *Application Programming Interface*) ist eine Menge interner Windows-Prozeduren, die Sie in Sprachen wie Visual Basic oder Visual C++ aufrufen, wenn Sie eine Routine oder eine Funktion von Windows benötigen. Der Debugger hat nicht die Möglichkeit und auch nicht die Berechtigung, Betriebssystemprozeduren zurückzuverfolgen.



Der Eintrag [*<Non-Basic Code>*] in der Aufrufliste weist darauf hin, daß der Aufruf von einer anderen Quelle aus erfolgt ist, wie es beispielsweise der Fall ist, wenn der Code das *Windows-API* aufruft.



Debuggen-Symbolleiste

Abbildung 21.5:
Im Dialogfeld Aufrufliste sehen Sie, welche Prozeduren in Ihrem Programm bisher ausgeführt wurden.

Wenn Sie eine der Prozeduren anzeigen möchten, die in der Aufrufliste angezeigt werden, doppelklicken Sie auf den entsprechenden Eintrag. Damit sehen Sie nicht nur den Code, sondern erhalten gleichzeitig die darin enthaltenen Werte. Beachten Sie, daß die Applikation immer noch an einem Haltepunkt steht; deshalb können Sie die Werte aller Steuerelemente, Variablen und Konstanten innerhalb der vorhergehenden Prozedur betrachten.

Einzelschritt

An einem Haltepunkt können Sie in der Symbolleiste auf die Schaltfläche **Einzelschritt** klicken, um die nächste Programmanweisung auszuführen (auch wenn es sich dabei um den Aufruf einer anderen Prozedur handelt).

Die **Debuggen**-Symbolleiste enthält drei Schaltflächen für das schrittweise Durchlaufen von Code. Tabelle 21.1 beschreibt ihre Funktion. Möglicherweise wollen Sie nicht *jede* Anweisung in der Applikation im Einzelschritt durchlaufen; diese Schaltflächen bieten Ihnen eine gewisse Flexibilität bei der weiteren Ausführung Ihrer Applikation.

Tabelle 21.1: Die Schrittmodi bestimmen, wie Ihre Applikation fortgesetzt wird.

Schrittmodus	Beschreibung
Einzelschritt	Führt nur die nächste Anweisung aus. Auch wenn die Anweisung in einer anderen Prozedur liegt (oder zu einer zuvor ausgeführten Prozedur zurückkehrt), wird die nächste Anweisung ausgeführt und der Cursor wird in diese Anweisung gestellt. Durch ständiges Drücken von (F8) können Sie eine ganze Applikation im Einzelschritt ausführen.
Prozedurschritt	Führt die nächste Anweisung aus, es sei denn, dort wird eine andere Prozedur aufgerufen. Die neue Prozedur wird als Ganzes ausgeführt und die Ausführung wird in der Anweisung hinter dem Prozeduraufruf wieder angehalten.
Prozedur abschließen	Führt die aktuelle Prozedur aus und unterbricht die Ausführung an der nächsten Anweisung außerhalb der Prozedur.

	Natürlich können Sie an jedem Haltepunkt auf die Starten -Schaltfläche klicken, um die Applikation ganz normal fortzusetzen. Wenn danach weitere Haltepunkte gesetzt sind, wird die Applikation dort wieder angehalten. Andernfalls verhält sich das Programm ganz normal, so, als ob Sie es nie unterbrochen hätten.

	Sie können den Debuggingmodus jederzeit beenden, indem Sie in der Symbolleiste von Visual Basic auf Beenden klicken oder indem Sie im Ausführen -Menü den Eintrag Beenden auswählen.

Mehrfache Haltepunkte

In einem Programm können mehrere Haltepunkte gesetzt werden. Sie unterbrechen die Programmausführung, so daß Sie den Inhalt von Variablen und Steuerelementen während der Ausführung einsehen können. Wenn Sie beispielsweise Laufzeitprobleme haben, die Sie bei der nächsten Programmausführung analysieren möchten, können Sie einen Haltepunkt setzen, indem Sie die betreffende Anweisung markieren und dafür in der Debuggen-Symbolleiste auf die Schaltfläche **Haltepunkt ein/aus** klicken. Auf diese Weise können Sie in Ihrem Code mehrere Haltepunkte setzen. Wenn Sie einen Haltepunkt erreichen (gekennzeichnet durch ein rotes Licht), den Sie in einer früheren Sitzung gesetzt haben, den Sie jetzt aber nicht mehr brauchen, klicken Sie ebenfalls auf **Haltepunkt ein/aus**, um den Haltepunkt zu entfernen. Sie können auch links von einer Anweisung klicken, um

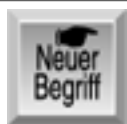
einen Haltepunkt zu setzen oder zu entfernen.



Haltepunkte können nur für Zeilen gesetzt werden, die ausgeführt werden. Es ist nicht möglich, Haltepunkte für Zeilen zu setzen, in denen Datentypen deklariert werden oder die nur Kommentare enthalten.

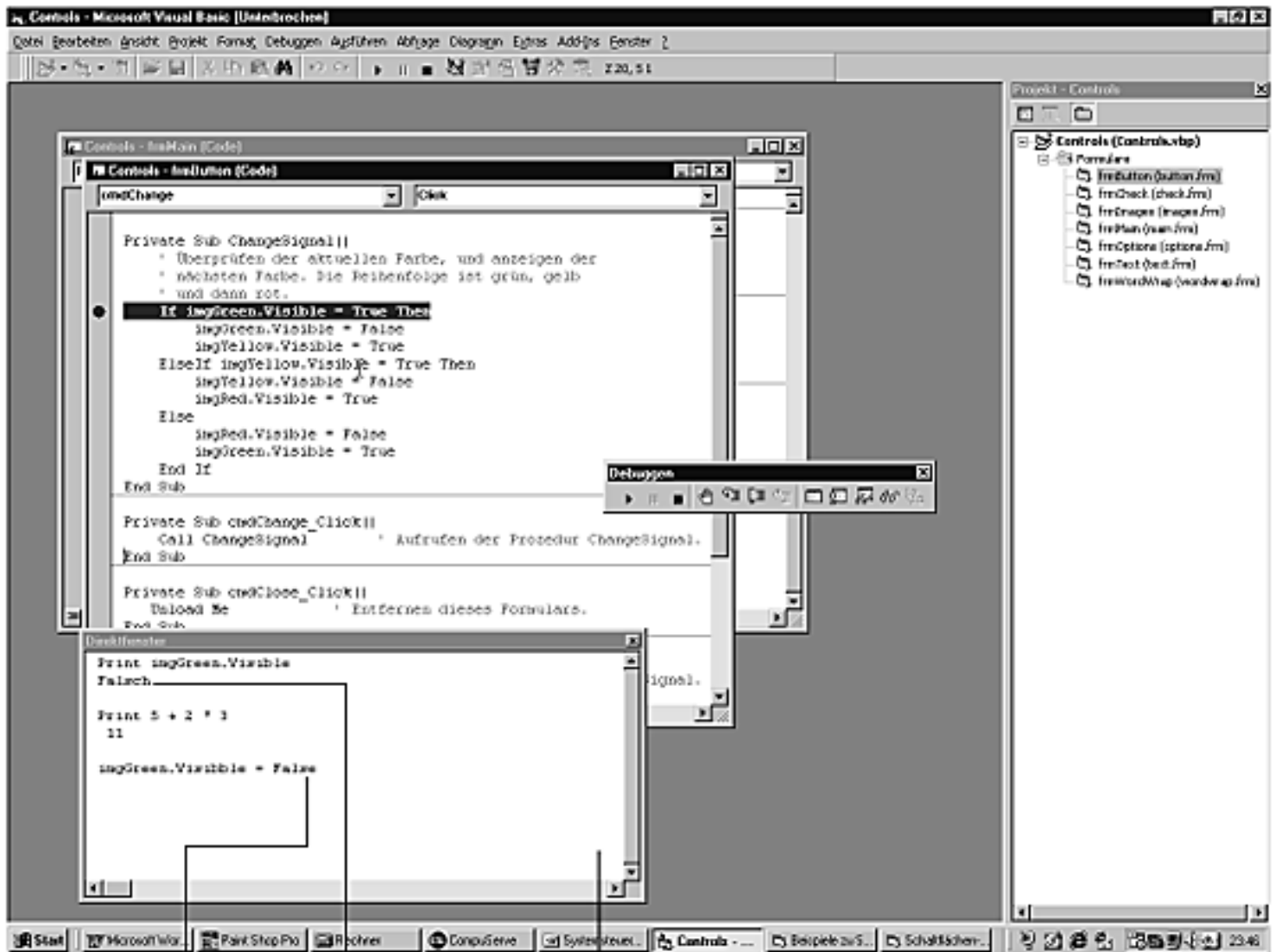
Das Debug-Fenster

An jedem Haltepunkt kann das Debug-Fenster angezeigt werden, so daß außerhalb der Programmumgebung gearbeitet werden kann. Das Debug-Fenster wird häufig auch als *Direktfenster* bezeichnet. Wenn Sie in der Debuggen-Symbolleiste auf die Schaltfläche Direktfenster klicken, wird unten im Codefenster das Direktfenster geöffnet.



Das *Direktfenster* (ein anderer Name für Debug-Fenster) ist ein Fenster innerhalb der Visual-Basic-Umgebung, in dem Sie Programmwerte anzeigen und Meldungen während der Programmausführung ausgeben können. Mit Hilfe der `Debug.Print`-Methode können Sie während der Programmausführung Fortschrittsmeldungen in das Direktfenster schicken. Diese Meldungen stören die normalen Ausgaben des Programms nicht.

In das Direktfenster können beliebige Visual-Basic-Anweisungen eingegeben werden. Das Ergebnis wird unmittelbar angezeigt. Eine der wichtigsten Methoden im Direktfenster ist `Print`, die Variablenwerte und Steuerelementeigenschaften ausgibt. `Print` sendet Ausgaben an verschiedene Objekte (nicht nur an den Drucker oder ein Formular), auch an das Direktfenster. Abbildung 21.6 zeigt einen Objektwert an, der direkt aus der Applikation stammt, die bis zum Haltepunkt ausgeführt wurde. Sie können außerdem die Ergebnisse beliebiger Ausdrücke ausgeben und Variablenwerte gegebenenfalls während der Ausführung ändern. Das Programm arbeitet mit dem neuen Wert weiter. Auf diese Weise sehen Sie, welchen Einfluß der Wert einer Variablen auf das Ergebnis hat.



Steuererelementwert Direktfenster
Änderung eines Steuererelementwerts

Abbildung 21.6:
Im Direktfenster können Sie Werte ausgeben und Ergebnisse ändern.

Das Lokal-Fenster

Wenn Sie in der Debug-Symbolleiste auf die Schaltfläche **Lokal-Fenster** klicken, zeigt Visual Basic das Lokal-Fenster an (siehe Abbildung 21.7). Das Lokal-Fenster zeigt den aktuellen Wert aller für die aktuelle Prozedur (das ist die Prozedur, in der sich der Haltepunkt befindet) lokalen Variablen an, ebenso wie globale Konstanten und Variablen. Der vielleicht wichtigste Vorteil des Lokal-Fensters ist die Anzeige aller Werte für die Steuerelemente des Formulars. Vergrößern Sie dieses Fenster, wenn Sie mehr Details sehen möchten.



Wenn Sie im Direktfenster eine lokale Variable verändern, wird ihr Wert auch im Lokal-Fenster geändert.

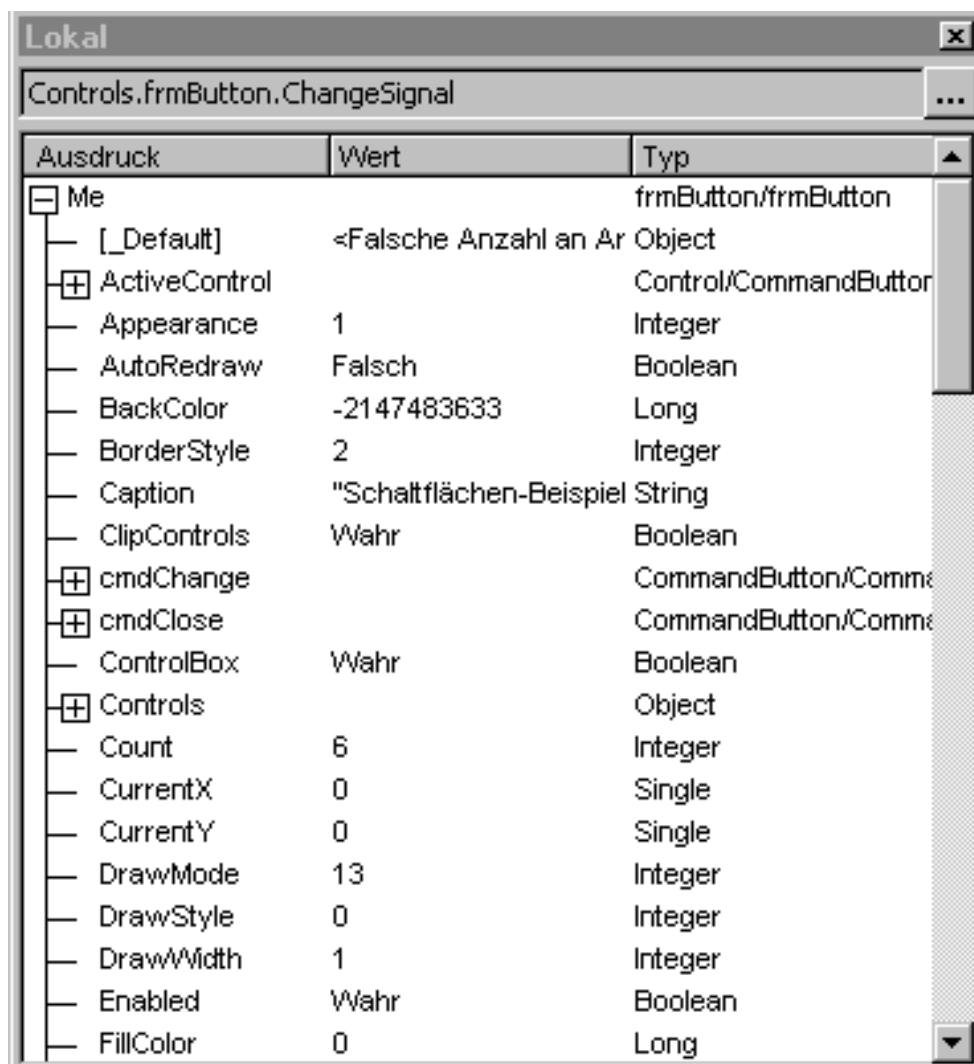


Abbildung 21.7:
Das Lokal-Fenster zeigt alle Variablen an, die lokal für die aktuelle Prozedur sind, ebenso wie alle globalen Werte.

Neben dem Variablennamen und dem Wert zeigt das Lokal-Fenster auch den Datentyp der Variablen oder Steuerelemente an. Klicken Sie auf die Schaltfläche mit den drei Punkten, die sich neben dem aktuellen Prozedurnamen im Lokal-Fenster befindet, um eine Aufrufliste anzuzeigen. Wenn Sie auf einen der Namen in der Aufrufliste klicken, wird das Lokal-Fenster aktualisiert und zeigt die lokalen Variablen dieser Prozedur an.

Das Überwachungsfenster

Während des Debuggings stellen Sie möglicherweise fest, daß ein bestimmter Fehler nur dann auftritt, wenn eine Variable einen bestimmten Wert hat. Manchmal können Probleme zu einer einzigen Anweisung zurückverfolgt werden. Sie müssen also die Variable oder den Ausdruck während einer gesamten Prozedur überwachen. Dafür stellt Visual Basic das Überwachungsfenster bereit, in das Sie Werte für Variablen oder Ausdrücke eingeben können. Rufen Sie das Überwachungsfenster auf, indem Sie in der Debuggen-Symbolleiste auf die Schaltfläche Überwachungsfenster klicken. Es gibt zwei Möglichkeiten, Werte in das Überwachungsfenster einzutragen:

- Wählen Sie im **Debuggen**-Menü den Eintrag **Überwachung hinzufügen**, um das Dialogfeld **Überwachung hinzufügen** (siehe Abbildung 21.8) anzuzeigen.
- Klicken Sie mit der rechten Maustaste in das Überwachungsfenster, und wählen Sie den Eintrag **Überwachung hinzufügen**, um das Fenster **Überwachung hinzufügen** anzuzeigen.

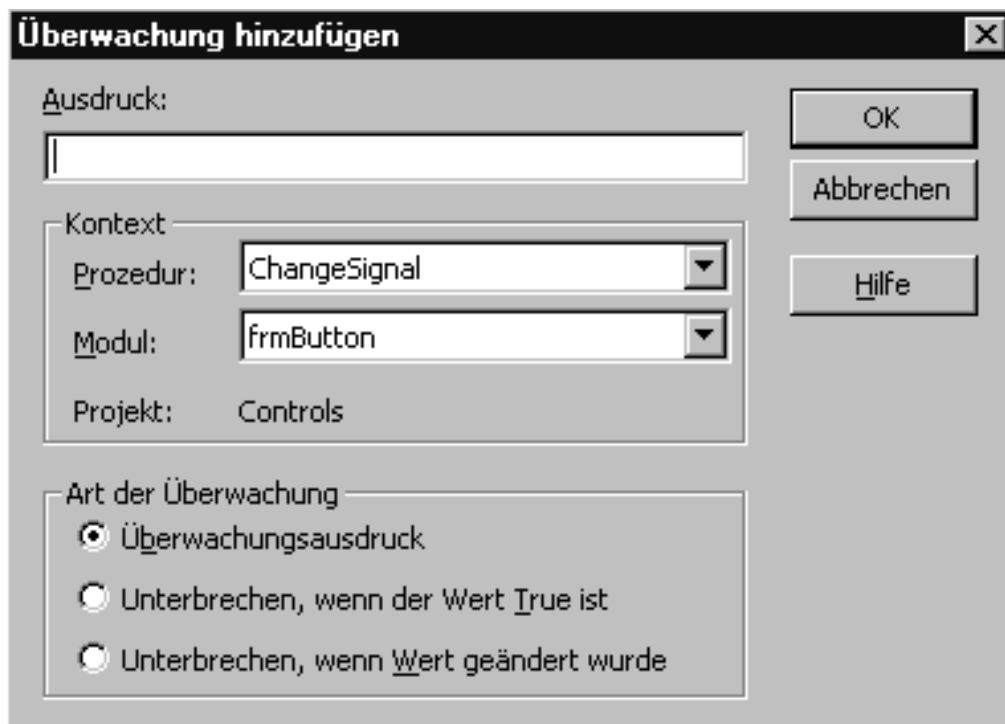


Abbildung 21.8:

Im Fenster Überwachung hinzufügen können Sie neue Ausdrücke eintragen, die überwacht werden sollen.

Wenn Sie einen Überwachungseintrag anlegen, teilt der **Kontext** Visual Basic den Gültigkeitsbereich für die Überwachung mit (also wo es den Ausdruck überwachen soll). Visual Basic kann eine einzelne Prozedur, ein Formular, ein Modul oder ein ganzes Projekt überwachen.

Im Bereich **Art der Überwachung** geben Sie an, wie Visual Basic auf den Überwachungsausdruck reagieren soll. Visual Basic kann die Daten im Ausdruck anzeigen, wenn sich der Wert zur Laufzeit ändern die Ausführung unterbrechen, wenn der Wert erreicht ist, oder das Programm bei jedem Erreichen des Werts unterbrechen. Bei der Ausführung aktualisiert Visual Basic das Überwachungsfenster gemäß den von Ihnen überwachten Werten.

Veröffentlichung Ihrer Applikationen

Sie haben Ihre Applikation entwickelt, getestet und ein Debugging dafür durchgeführt. Im letzten Schritt verpacken Sie sie so, daß Sie sie an andere weitergeben können. Wenn Sie die Applikation für sich selbst geschrieben haben, kompilieren Sie sie einfach und kopieren sie dann zusammen mit den unterstützenden Dateien in den Ordner, von dem aus Sie sie ausführen möchten. Verwenden Sie im **Start**-Menü die Option **Einstellungen, Taskleiste**, um die Applikation in Ihr **Start**-Menü aufzunehmen.

Wenn Sie Ihre Applikation an andere weitergeben möchten, müssen Sie die Installation automatisieren, so daß alle Projektdateien an die richtige Position kopiert werden und das Start-Menü von Windows einen entsprechenden Eintrag erhält. Dieser Abschnitt erklärt, wie Sie Ihre Applikation am besten veröffentlichen.

Kompilieren

Es ist ganz einfach, eine Applikation unter Visual Basic zu kompilieren. Die kompilierte Datei ist eine Programmdatei mit der Dateinamenerweiterung .EXE. Sie setzt sich aus allen beteiligten Modulen und Formularen zusammen. Möglicherweise sind auch Hilfsdateien erforderlich - beispielsweise eine Microsoft-Access-Datenbankdatei -, aber die meisten Dateien für Ihr Projekt werden in der ausführbaren Programmdatei zusammengefaßt, um die Veröffentlichung so einfach wie möglich zu machen.



Die kompilierte Applikation ist sicherer als ein Quellprojekt. Wenn Sie den Quellcode (das Projekt und seine Dateien) weitergeben, kann jeder, der Visual Basic einsetzt, Ihre Arbeit verändern. Die meisten Benutzer könnten den Quellcode für Ihr Programm nicht einmal ausführen, weil sie Visual Basic nicht installiert haben. Damit alle Benutzer Ihre Applikation ausführen können, muß eine Datei also kompiliert werden.

Ihre kompilierte Applikation läuft viel schneller als die Applikation in der Entwicklungsumgebung von Visual Basic. Ihre Applikation soll beim Benutzer so schnell und so gut wie möglich laufen. Die kompilierte Programmdatei macht die Ausführung einfach.

Bevor Sie Ihre Applikation kompilieren, sollten Sie sie so gut wie möglich testen und debuggen. Kompilierte Applikationen können mit dem Debugger nicht bearbeitet werden, weil sie nicht innerhalb der Entwicklungsumgebung ausgeführt werden können.

Wenn Sie sich davon überzeugt haben, daß Ihr Programm so stabil wie möglich läuft, wählen Sie **Datei, EXE erstellen**. Visual Basic zeigt das Dialogfeld **Projekt erstellen** an (siehe Abbildung 21.9). Wählen Sie den Ordner aus, in dem die kompilierte Applikation ausgeführt werden soll. (Als Projektnamen verwendet Visual Basic den Standard-Programmnamen.)



Abbildung 21.9:

Im Dialogfeld Projekt erstellen kompilieren Sie die Applikation.

Bevor Sie auf **OK** klicken und damit die Kompilierung beginnen, klicken Sie auf die Schaltfläche **Optionen**, um das Dialogfeld **Projekteigenschaften** anzuzeigen (siehe Abbildung 21.10). (Sie können auch über den Eintrag **Eigenschaften** im **Projekt**- Menü auf dieses Dialogfeld zugreifen.) Wenn Sie mehrere Versionen Ihrer Software planen, können Sie anhand der Versionsnummern die Reihenfolge festlegen. Geben Sie die Versionsinformationen in das Dialogfeld **Projekteigenschaften** ein, damit Sie sie beim Kompilieren nicht mehr bereitzustellen brauchen. Die Versionsnummern und beschreibende Informationen werden im Quellcode des Projekts abgelegt.

The screenshot shows the 'Controls - Projekteigenschaften' dialog box. The 'Erstellen' tab is selected. The 'Versionsnummer' section has 'Haupt: 6', 'Neben: 0', and 'Revision: 0'. The 'Anwendung' section has 'Name: CONTROLS' and 'Symbol: frmMain'. The 'Versionsinformationen' section has 'Typ: Copyright' and an empty 'Wert' field. The 'Befehlszeilenargumente' and 'Argumente für bedingte Kompilierung' fields are empty. The checkbox 'Informationen zu nicht verwendeten ActiveX-Steuerelementen entfernen' is checked. The 'OK', 'Abbrechen', and 'Hilfe' buttons are at the bottom right.

Abbildung 21.10:

Im Dialogfeld Projekteigenschaften setzen Sie die Optionen für das zu kompilierende Projekt.

Der Eintrag **Symbol** bestimmt das Icon, das im Start-Menü für Windows oder in der Taskleiste erscheint. Im allgemeinen geben Sie den Namen für das Hauptformular im Symbolfeld an. Das Eigenschaftenfenster für das Formular enthält einen Icon-Eintrag, den Sie als Symbol für das Formular und damit auch für die kompilierte Applikation verwenden können.

Klicken Sie auf die Registerkarte **Kompilieren** (siehe Abbildung 21.11). Um das kompilierte Projekt zu optimieren, damit es so schnell wie möglich ausgeführt wird, wählen Sie die Option **Kompilieren zu System-Code (Native Code)** aus. (Wenn Sie **Kompilieren zu P-Code** auswählen, brauchen Ihre Benutzer eine Visual-Basic-basierte Laufzeit-DLL in ihrem System-Ordner. Nativer Code läuft schneller und benötigt weniger Dateien, aber eine DLL-Datei ist immer noch erforderlich.)



Wenn Sie eine der Optionen auswählen, die erscheinen, wenn Sie auf die Schaltfläche **Weitere Optimierungen** klicken, werden einige der Laufzeitfehler nicht überprüft, aber die Ausführung wird schneller.

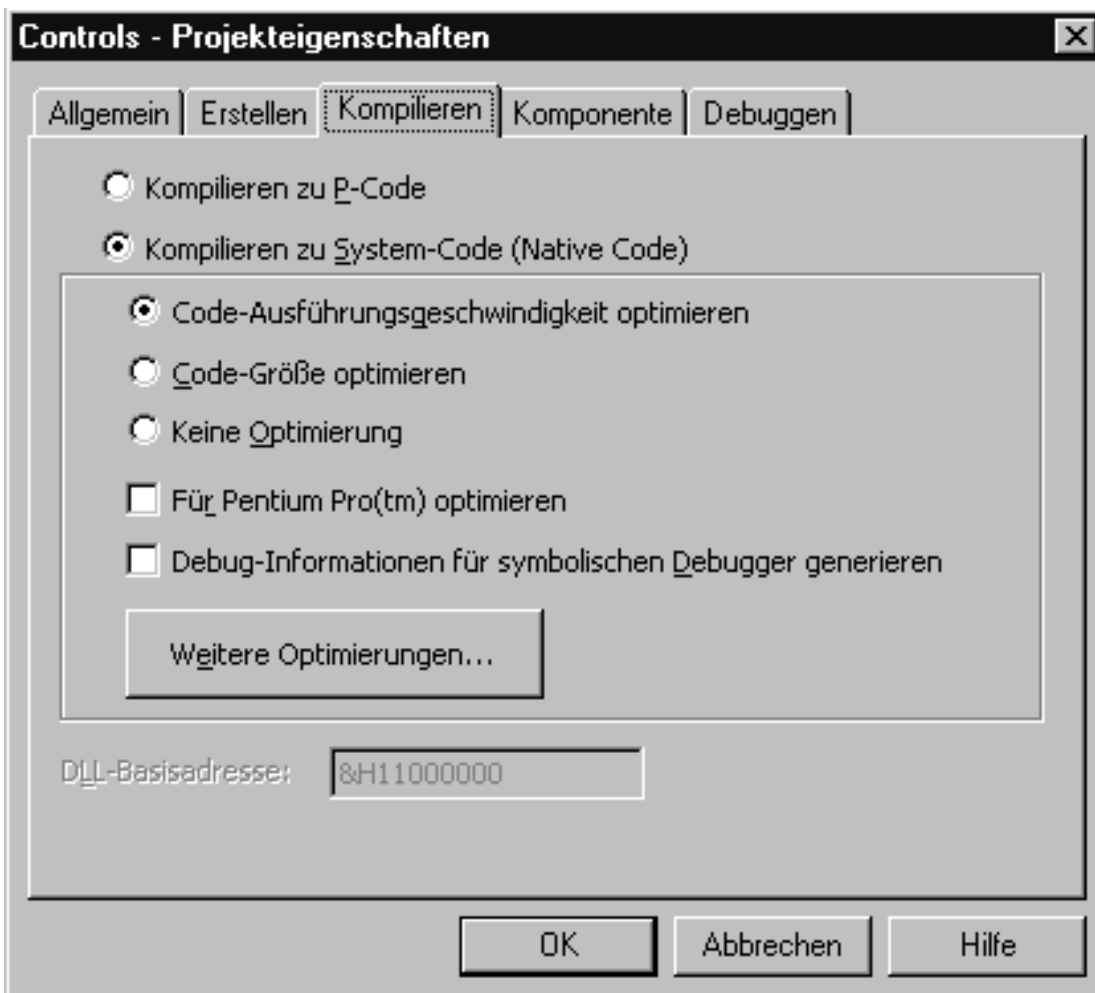


Abbildung 21.11:
Die Registerkarte Kompilieren zeigt die Optionen für die Kompilierung des Projekts an.

Schließen Sie das Dialogfeld **Projekteigenschaften**, und klicken Sie auf OK. Visual Basic kompiliert Ihren Code. Vorausgesetzt, bei der Kompilierung sind keine Fehler aufgetreten, erzeugt Visual Basic die EXE-Datei (rechts oben wird der Kompilierungsstatus angezeigt). Verlassen Sie Visual Basic, und führen Sie die Applikation aus, indem Sie im Start-Menü die Option Ausführen wählen und die EXE-Datei dafür angeben. Das Icon, das Sie für die Applikation ausgewählt haben, erscheint bei der Programmausführung in der Taskleiste.

Der Paket- und Weitergabe-Assistent

Der Paket- und Weitergabe-Assistent nimmt Ihnen viel Arbeit ab, unter anderem folgendes:

- Er kompiliert die Applikation und komprimiert die Dateien.
- Er erzeugt ein Setup-Programm, das für die Installation der Applikation genutzt werden kann.
- Er stellt fest, wie die Applikation am besten auf die Disketten kopiert wird, legt verschiedene Setup-Disketten an und verteilt sehr große Dateien über mehrere Disketten. Der Paket- und Weitergabe-Assistent teilt Ihnen im voraus mit, wie viele Setup-Disketten erforderlich sind.
- Er kopiert Ihre kompilierte Applikation auf eine Festplatte, so daß Sie die Applikation über ein Netzwerk installieren können.
- Er richtet Ihre Applikation für den verteilten Zugriff über das Internet ein (für Benutzer des Internet Explorer).

Der Paket- und Weitergabe-Assistent erzeugt eine Liste mehrerer Dateien, die man für den Setup braucht. Dabei wird eine einzige Setup.exe-Datei erzeugt. Häufig sind für eine Visual-Basic-Applikation DLL- und OCX-Dateien erforderlich, und diese Dateien müssen sich ebenfalls in dem Setup-Bereich (Disketten oder eine Festplatte) befinden, wo sich das kompilierte Programm und das Setup.exe-Programm befinden.

Damit Sie den Paket- und Weitergabe-Assistenten nutzen können, müssen Sie das Projekt Ihrer Applikation laden. Nachdem Sie das Projekt kompiliert haben, können Sie das Installationsmodul anlegen. Der Paket- und Weitergabe-Assistent kompiliert Ihren Code ein letztes Mal, falls Sie seit der letzten Kompilierung eine Änderung vorgenommen haben.

Der Paket- und Weitergabe-Assistent ist nicht Teil der Entwicklungsumgebung von Visual Basic. Gehen Sie in das Menü **Add-Ins**, und wählen Sie den **Add-In-Manager**. Laden Sie den Paket- und Anwendungs-Assistenten in das **Add-Ins**-Menü. Dort können Sie ihn starten. Abbildung 21.12 zeigt den Eröffnungsbildschirm für den Paket- und Weitergabe-Assistenten.



Abbildung 21.12:
Der Paket- und Weitergabe-Assistent bereitet Ihre Applikationen auf die Weitergabe vor.

Die erste Option im Paket- und Weitergabe-Assistent (die Sie wahrscheinlich am häufigsten verwenden) legt die Standardroutine Setup.exe an, die Ihre Benutzer für die Installation ausführen. Der Paket- und Weitergabe-Assistent kann diese Installationsroutine auf einer Festplatte, Disketten, einem CD-ROM-Brenner oder in speziellen CAB-Dateien zur Online-Weitergabe über das Internet vorbereiten. Die zweite Option sendet die Installationsroutine an einen Internet-Server, der die Applikation `remote` installiert. Während die Installationsroutine erzeugt wird, erzeugt der Paket- und Weitergabe-Assistent eine Skriptdatei, die die Setup-Routine beschreibt. In nachfolgenden Sitzungen können Sie ein bereits existierendes Setup-Skript abändern, oder Sie erzeugen das Setup aus dem

Originalprojekt. Die dritte Option im Eröffnungsbildschirm des Paket- und Weitergabe-Assistenten ermöglicht Ihnen die Verwaltung Ihrer Installationsscripts.

Die erste Option erzeugt allgemeine Installationsroutinen, die für die meisten Applikationen verwendet werden. Nachdem Sie auf die erste Option geklickt haben, sehen Sie das in Abbildung 21.13 gezeigte Fenster. Wenn Ihre Applikation keine zusätzlichen ActiveX-Steuerelemente oder Datenbankdateien benötigt, verwenden Sie diese erste Option.



Wenn Sie ein ActiveX-Steuerelement einrichten möchten, wählen Sie die Option **Abhängigkeitsdatei**, damit der Assistent alle Dateien aufnimmt, und zwar in der Reihenfolge, in der die Applikation sie benötigt.



Abbildung 21.13:
Hier legen Sie fest, welches Setup-Paket erstellt werden soll.

Klicken Sie auf **Weiter**, um das Dialogfeld **Paketordner** aufzurufen. Der Paket- und Weitergabe-Assistent muß wissen, wo die Setup-Applikation abgelegt werden soll. Wählen Sie ein leeres Verzeichnis dafür aus. Wenn der Assistent seine Arbeit abgeschlossen hat, wissen Sie, daß alle Dateien in diesem Verzeichnis vom Assistenten angelegt wurden. Wenn Sie auf Weiter klicken, durchsucht der Assistent Ihre Projektdatei, um festzustellen, welche Programmdateien Ihre Applikation benötigt. Der Assistent kann nicht erkennen, welche Datenbanktreiber benötigt werden, wenn Ihre Applikation Daten-Steuerelemente enthält. Falls in Ihrer Applikation ein Datenbankzugriff erfolgt, sehen Sie das in Abbildung 21.14 gezeigte Dialogfeld. Kopieren Sie die Datenbanktreiber für die Applikation in das rechte Feld.



Ihr Dialogfeld sieht möglicherweise anders aus - abhängig von dem Datenzugriff (ADO, DAO usw.), den Ihre Applikation verwendet.



Abbildung 21.14:
Wenn Sie in Ihrer Applikation Datenbankzugriff brauchen, müssen Sie die Treiberdateien manuell auswählen.

Nachdem Sie auf **Weiter** geklickt haben, sehen Sie das Dialogfeld **DAO ODBC Zugriff**, falls für Ihre Applikation ODBC-Unterstützung erforderlich ist. In diesem Fall geben Sie an, welche Art ODBC benötigt wird. Wenn Sie keine Datenbank mit ODBC-Zugriff verwenden, wird dieses Dialogfeld nicht angezeigt.

Wenn Sie jetzt wieder auf **Weiter** klicken, sammelt der Assistent alle Applikationsdateien, die Sie angegeben haben, und ruft das in Abbildung 21.15 gezeigte Dialogfeld auf. Sehen Sie nach, ob alle Dateien aufgelistet sind, die Sie für Ihre Applikation brauchen. Möglicherweise müssen Sie weitere Dateien hinzufügen (indem Sie auf die Schaltfläche **Hinzufügen** klicken), beispielsweise Readme.txt oder eine Datenbankdatei. Wenn zusätzliche Datenbankdateien benötigt werden, fügen Sie diese in die Dateiliste ein, damit die Installationsroutine sie zusammen mit den Installationsdateien im Setup-Paket ablegt.



Stellen Sie sicher, daß Sie Zugriff auf alle ActiveX-Steuerelemente haben, die Sie mit Ihrer Applikation weitergeben. Möglicherweise haben Sie nicht die erforderliche Lizenz, es sei denn, sie haben diese Steuerelemente selbst entwickelt. Erkundigen Sie sich beim Hersteller der verwendeten ActiveX- Steuerelemente, ob Sie sie an Ihre Benutzer weitergeben dürfen.

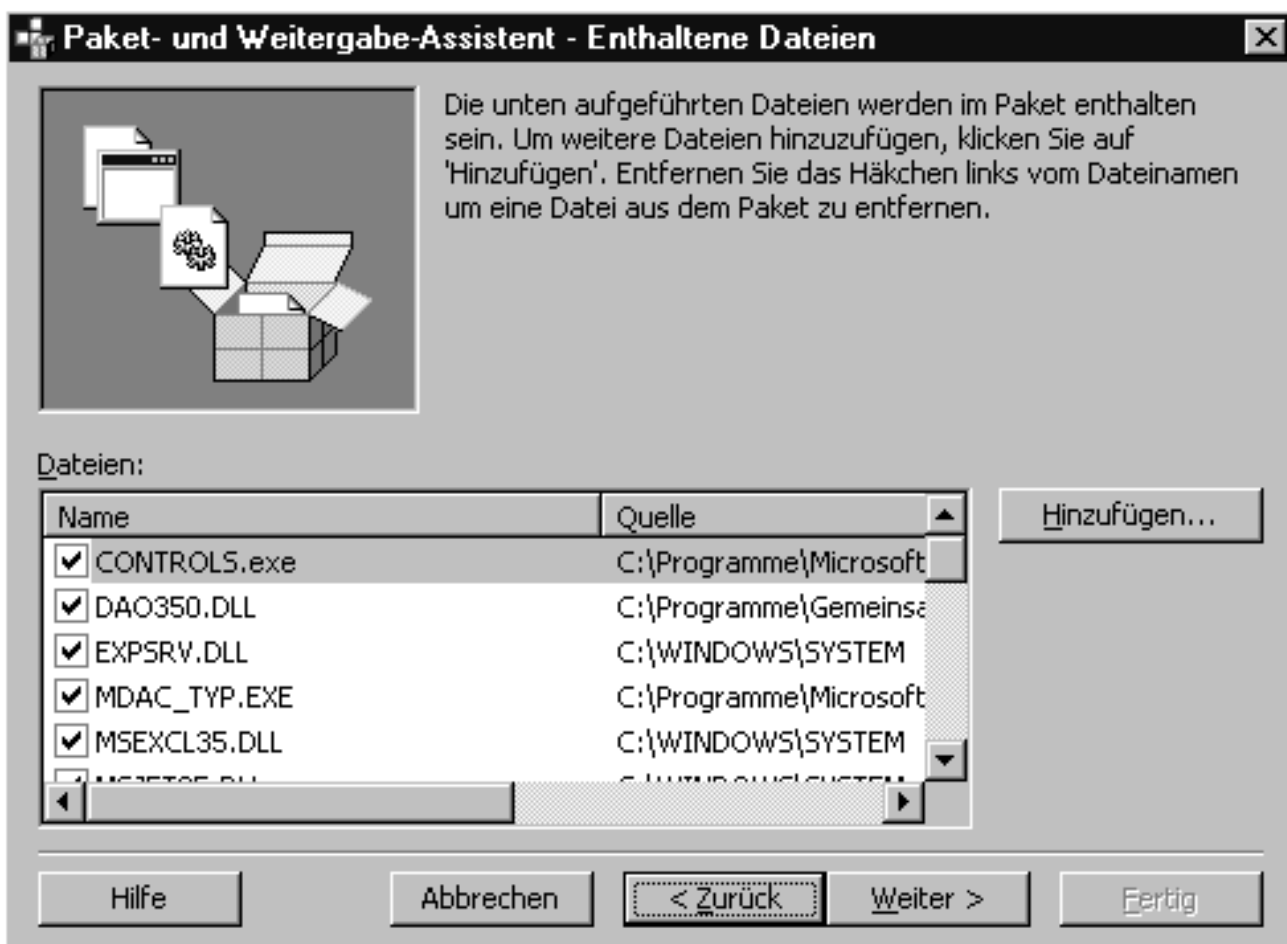


Abbildung 21.15:
Sehen Sie nach, ob der Paket- und Weitergabe-Assistent alle Dateien gesammelt hat, die für Ihre Applikation erforderlich sind.

Das nächste Dialogfeld im Paket- und Weitergabe-Assistent fragt Informationen für die Weitergabe ab. Sie können eine einzige Weitergabedatei anlegen, die Setup-Routine aber auch über mehrere Disketten oder andere Datenträger verteilen. Nachdem Sie festgelegt haben, wie der Paket- und Weitergabe-Assistent Ihre Installationsroutine zerlegt, klicken Sie auf **Weiter**, um den Bildschirm anzuzeigen, in dem Sie den Titel für die Applikation festlegen. Hier geben Sie den Namen Ihres Installationsprojekts ein und klicken dann auf Weiter zum Dialogfeld **Symbole**.

Im Dialogfeld **Symbole** legen Sie das Untermenü an, das im Start-Menü des PC erscheint. Klicken Sie auf die Schaltfläche **Neu**, und zeigen Sie das Dialogfeld **Startmenüelemente** an (siehe Abbildung 21.16). Die Untermenü-Elemente könnten beispielsweise eine Readme-Datei oder ein Hilfsprogramm (etwa ein System-Utility) beinhalten.

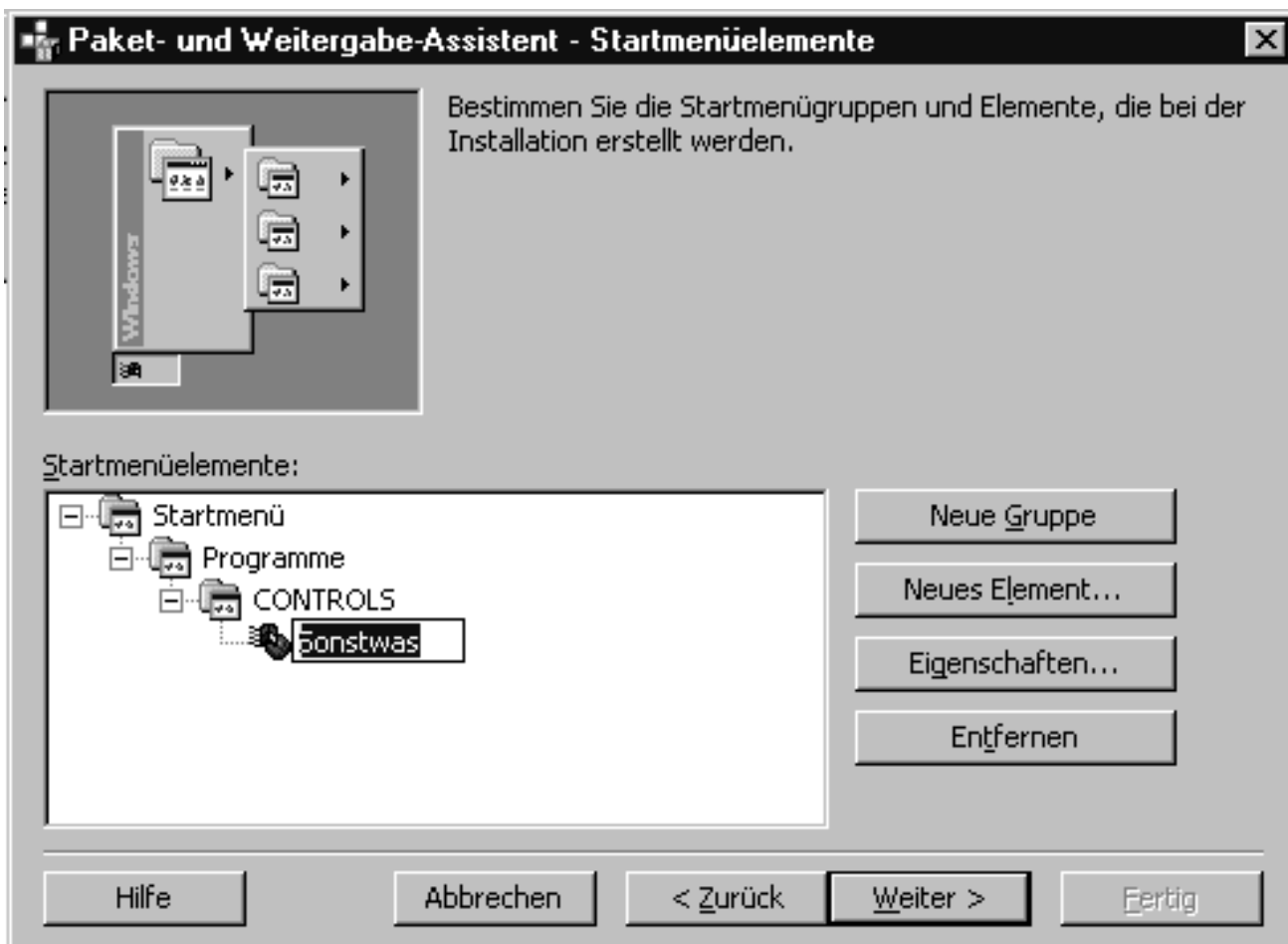


Abbildung 21.16:
Hier bestimmen Sie, wie die Applikation im Start-Menü des Benutzers erscheint.

Im nächsten Fenster legen Sie die Position der zu installierenden Dateien an. Die meisten dieser Dateien werden in dem Ordner angelegt, den der Benutzer bei der Installation auswählt, und wie in der Systemvariablen `AppPath` festgehalten, aber Sie können im Paket- und Weitergabe-Assistenten auch einzelne Dateien auswählen, die in einem anderen Ordner abgelegt werden sollen.

	Wie Sie sehen, sind im Paket- und Weitergabe-Assistenten zahlreiche Entscheidungen erforderlich. Mit diesen Entscheidungen haben Sie jedoch die volle Kontrolle darüber, wie und wo Ihre Applikation auf dem System des Benutzers angelegt wird.

Klicken Sie auf **Weiter**, um die Dateien auszuwählen, die Sie als gemeinsam genutzte Dateien verstanden haben wissen. Eine Datei kann nicht nur von den Benutzern genutzt werden (wie beispielsweise eine Datenbankdatei, auf die die Applikation zugreift), sondern auch von anderen Programmen auf dem Computer, wie etwa die ActiveX-Steuerelemente, die Ihr Projekt enthält. Markieren Sie die Dateien, die Sie für die gemeinsame Nutzung freigeben möchten.

Nachdem Sie auf **Weiter** geklickt haben, sehen Sie den letzten Bildschirm des Paket- und Weitergabe-Assistenten. Dort werden Sie gefragt, welchen Namen die Skriptdatei für die Installation erhalten soll. Wenn Sie eine Skriptdatei anlegen, brauchen Sie viele Fragen, die Sie während der Arbeit mit dem Assistenten beantworten mußten, nicht mehr beantworten, wenn Sie die Installationsroutine zum nächsten Mal ausführen. Darüber hinaus können Sie das Skript abändern, ohne die Installationsbildschirme noch einmal bearbeiten zu müssen, falls sich etwas im Installationsprozeß ändert.

Wenn Sie auf **Fertig** klicken, erzeugt der Paket- und Weitergabe-Assistent das Installationsskript,

erstellt die Installationsroutine und legt sie entsprechend den von Ihnen ausgewählten Optionen ab. Anschließend befinden sich die Setup-Dateien auf Ihrem PC. Sie können diese Setup-Dateien reproduzieren und damit Ihre Applikation auf anderen Computern neu erzeugen.

Nach dem Setup

Nachdem der Assistent die Setup-Routine angelegt hat, sollten Sie sie ausprobieren. Dazu führen Sie das Setup-Programm aus, das Sie eben erzeugt haben, um sicherzustellen, daß keine Fehler auftreten und die Applikation problemlos auf dem Computer ausgeführt werden kann.



Wenn Sie die Setup-Routine wirklich testen möchten, führen Sie das Setup- Programm auf einem Computer aus, der Ihre Applikation noch nie gesehen hat. Am besten verwenden Sie einen Computer, auf dem nicht einmal Visual Basic installiert ist. Wenn Sie Ihre Applikation auf einer solchen »sauberen« Maschine testen, können Sie sicherstellen, daß Ihre Applikation auch auf anderen Computern korrekt ausgeführt wird.

Der Paket- und Weitergabe-Assistent erzeugt die Setup-Dateien dort, wo Sie es im Setup-Prozeß festgelegt haben. Wenn Sie die Setup-Datei in einem einzelnen Test- Verzeichnis angelegt haben, finden Sie dort die Datei Setup.exe, die Datei Setup.1st (die eine Liste aller zu installierenden Dateien enthält) sowie möglicherweise mehrere andere Dateien, deren Dateinamenerweiterung mit einem Unterstrich (_) enden. Die Dateien mit der abgekürzten Erweiterungen sind komprimiert; die Routine Setup.exe expandiert diese komprimierten Dateien auf der Zielmaschine.

Die einfachste Methode, die Setup-Routine zu testen, ist, im Start-Menü den Eintrag Ausführen auszuwählen, und die Datei Setup.exe auszuführen. Klicken Sie auf die Schaltfläche Ausführen, um das Setup für die Applikation auszuführen. Eine typische Setup-Installation wird aufgerufen. Das Setup-Programm analysiert den Ziel- Computer, um sicherzustellen, daß keine Programme ausgeführt werden, die einen Konflikt mit einer der zu installierenden Dateien erzeugen. Abbildung 21.17 zeigt das Dialogfeld, das den Benutzer warnt, wenn er die Setup-Routine ausführt, während andere Programme laufen, die eine oder mehrere der gemeinsam genutzten Dateien der Routine verwenden.



Wenn Sie das Setup-Programm abbrechen, bevor es abgeschlossen ist, entfernt es alle bisher kopierten Dateien vom System.

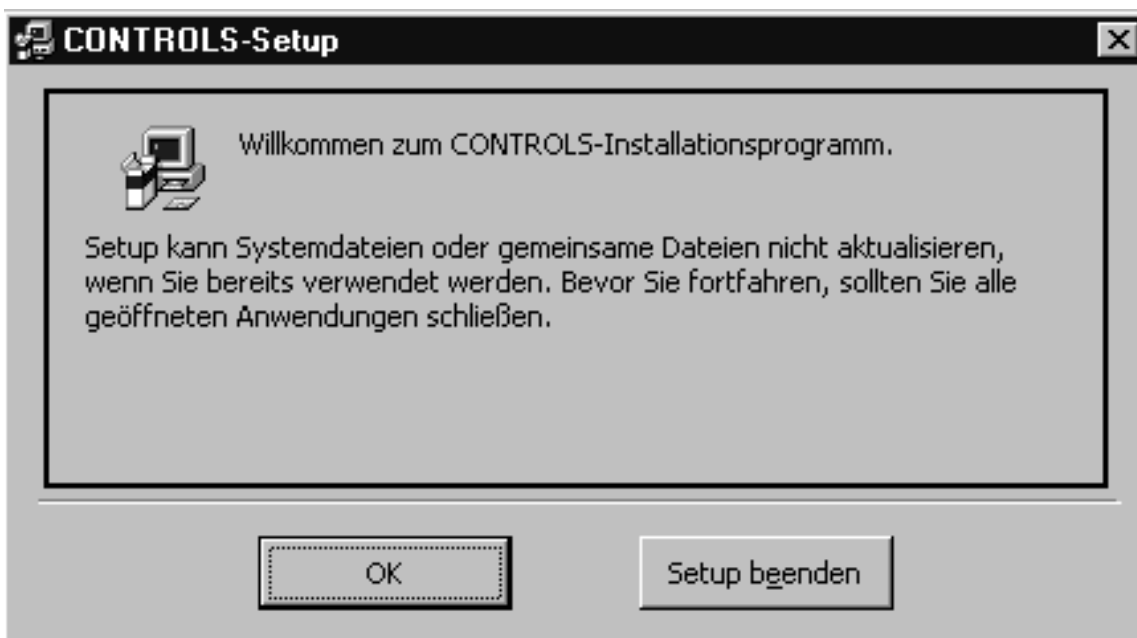


Abbildung 21.17:
Das Setup beginnt den Installationsprozeß.

Deinstallation der Applikation

Der Paket- und Weitergabe-Assistent erzeugt nicht nur die installierbare Setup-Routine, sondern auch eine Routine, die die Applikation deinstalliert. Damit wird es den Benutzern ermöglicht, alle Dateien der Applikation in einem Schritt zu entfernen. Der Paket- und Weitergabe-Assistent verwendet das Applet **Software Hinzufügen/Entfernen** der Systemsteuerung. Wenn ein Benutzer die Applikation vom System entfernen möchte, geht er wie folgt vor:

1. Wählen Sie im **Start**-Menü den Eintrag **Einstellungen** und dann **Systemsteuerung** .
2. Doppelklicken Sie auf **Hinzufügen/Entfernen**.
3. Wählen Sie die Applikation aus der Liste der installierten Applikationen aus. Nachdem bestätigt wurde, daß der Benutzer die Applikation entfernen möchte, entfernt die Uninstall-Routine das Programm und alle dafür angelegten Dateien vom Computer des Benutzers.

Der Paket- und Weitergabe-Assistent speichert die Informationen für die Deinstallation im selben Verzeichnis wie die Applikation. Die Datei mit den Anweisungen für die Deinstallation hat den Namen ST6UNST.LOG. Sie enthält alle Details, die Software Hinzufügen/Entfernen braucht, um die Deinstallation vorzunehmen. Nicht alle Dateien sollten entfernt werden, insbesondere die Systemdateien, die von anderen Programmen verwendet werden. Bevor Sie solche Dateien entfernen, die möglicherweise noch gebraucht werden, zeigt das Utility für die Deinstallation ein Dialogfeld an, in dem Sie darauf hingewiesen werden, daß die Dateien möglicherweise noch gebraucht werden.

Zusammenfassung

Dieses Kapitel hat Ihnen die leistungsfähigen Debugging-Werkzeuge von Visual Basic vorgestellt. Ihre Kenntnisse über das Debugging zahlen sich spätestens aus, wenn Sie auf die Fehlersuche gehen. Die Debugging-Werkzeuge von Visual Basic finden zwar keine logischen Fehler, aber mit Hilfe des Debuggers können Sie sie einfacher verfolgen. Sie können die Ausführung eines Programms beobachten, Haltepunkte setzen und zurückverfolgen, wie die Programmausführung stattgefunden hat.

Eine der leistungsfähigsten Eigenschaften des Debuggers ist die Interaktion mit dem Programm während einer Haltepunkt-Sitzung. Wenn Ihr Programm einen Haltepunkt erreicht, werden alle bisher initialisierten und berechneten Werte beibehalten. Sie können also die Variablen während der Programmausführung auswerten und feststellen, ob die Zwischenergebnisse Ihren Vorstellungen entsprechen. Außerdem können Sie während der Programmausführung die Werte von Variablen und Steuerelementen ändern und beobachten, wie die weitere Ausführung auf diese Änderung reagiert.

Nachdem Sie ein Debugging für Ihre Applikationen ausgeführt haben, können Sie sie an Benutzer weitergeben. Dazu ist mehr nötig, als die Applikation einfach zu kompilieren. Sie müssen eine Installationsroutine erstellen und sicherstellen, daß Ihren Benutzern alle Dateien zur Verfügung stehen, die sie für die Ausführung der Applikation brauchen.

Fragen und Antworten

Frage:
Kann ich alle Fehler meiner Applikation erkennen, indem ich das Kontrollkästchen Automatische Syntaxüberprüfung markiere?

Antwort:
Während Sie das Programm eingeben, kann diese Option natürlich sehr viele Tippfehler eliminieren. Das Problem dabei ist, daß die Option nur Syntaxfehler auffängt. Es können jedoch immer noch Laufzeitfehler auftreten, beispielsweise eine Division durch Null. Solche Fehler kann das Codefenster nicht erkennen.

Frage:
Welche Fehler sind am einfachsten zu finden?

Antwort:
*Bedauerlicherweise sind die Syntaxfehler am einfachsten zu erkennen. (Das ist bedauerlich, weil sie nicht nur einfach zu erkennen sind, sondern weil Visual Basic sie für Sie findet.) Wenn Sie die Option **Automatische Syntaxüberprüfung** markiert haben (oder auch nicht), findet Visual Basic alle Syntaxfehler, während Sie Ihren Code eingeben, oder spätestens, wenn Sie versuchen, das Programm auszuführen. Die logischen Fehler sind schwieriger zu erkennen, weil Visual Basic gar nicht erkennt, daß irgend etwas nicht stimmt. Selbst Sie werden zunächst nicht merken, daß ein Problem vorliegt. Sie sollten Ihre Programme ausführlich testen, um sicherzustellen, daß keine logischen Fehler vorliegen.*

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollen, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Was ist ein Syntaxfehler?
2. Was ist der Unterschied zwischen einem Syntaxfehler und einem logischen Fehler?
3. Richtig/Falsch. Einige Laufzeitfehler können bewirken, daß die Programmausführung unterbrochen

wird.

4. Richtig/Falsch. Einige Syntaxfehler können bewirken, daß die Programmausführung unterbrochen wird.
5. Wie analysieren Sie Variablen während der Programmausführung?
6. Wie unterstützt Sie der Einzelschrittmodus beim Debugging eines Programms?
7. Richtig/Falsch. Es ist möglich, die Werte von Variablen und Steuerelementen *während* der Programmausführung zu ändern.
8. Was ist schneller: ein kompiliertes Programm oder ein Visual-Basic-Programm in der Visual-Basic-Umgebung?
9. Welche Werkzeuge stellt Visual Basic bereit, um eine Installationsroutine für Ihre Applikationen zu erzeugen?
10. Was passiert, wenn Ihre Installationsroutine größer ist, als auf einer Diskette untergebracht werden kann, Sie aber die Applikation auf Disketten bereitstellen müssen?

Übungen

1. Welchen Fehler enthält die folgende Anweisung?

```
If (a < b) Therefore  
  lblTitle.Caption = "Zu klein!"  
End If
```

2. Richtig/Falsch. Der folgende Satz enthält zweierlei Fehler (denken Sie ruhig ein bißchen nach):
Dieser Satz enthält zwei Fehlr.

Woche 3

Tag 22

Mehrdimensionale Arrays

Kapitel 10 hat beschrieben, wie Arrays deklariert und verwendet werden. Arrays enthalten Listen mit Datenwerten. Wenn Sie mehrere Elemente mehrmals verwalten und verarbeiten müssen, haben Sie die Möglichkeit, diese Elemente in separaten Variablen mit verschiedenen Namen abzulegen - oder sie in einem einzigen Array mit einem einzigen Namen zu speichern. Der Vorteil eines Arrays ist, daß Sie alle Variablen mit Hilfe einer Schleife durchlaufen können, beispielsweise mit einer **For**-Schleife. Der Schleifenzähler wird so inkrementiert, daß er die Indizes für das Array nachbildet.

Visual Basic unterstützt verschiedene Arten von Arrays. Sie können Arrays für verschiedene Datentypen deklarieren, die Variablenlisten aufnehmen, Sie können aber auch Steuerelementfelder deklarieren, die sich wie Variablen-Arrays verhalten, aber den Vorteil bieten, daß sie mehrere Formularobjekte unter einem Namen und mit ähnlichen Eigenschaftswerten zusammenfassen. (Nicht jedes Steuerelement in dem Array muß dieselben Eigenschaftswerte haben.) Sie durchlaufen dieses Steuerelementfeld ähnlich wie ein Variablen-Array, um mit den Steuerelementen zu arbeiten.

Dieses Bonuskapitel führt das Konzept der Variablen noch einen Schritt weiter. Die Arrays, die Sie bisher kennengelernt haben, werden auch als *eindimensionale Arrays* bezeichnet, weil sie nur einen Index verwenden. In diesem Kapitel lernen Sie, wie Arrays so erweitert werden, daß sie mehrere Dimensionen darstellen, auch als mehrdimensionale Arrays bezeichnet. Diese mehrdimensionalen Arrays werden auch als *Tabellen* bezeichnet. Sie ermöglichen Ihnen, Werte in Zeilen/Spalten-Form abzulegen. Außerdem ist es möglich, mehr als zwei Dimensionen zu verwenden, um somit praktische Speicherbereiche für Daten zu schaffen.

Neben den mehrdimensionalen Arrays lernen Sie in diesem Bonuskapitel auch das Tabellen-Steuerelement kennen. Das Tabellen-Steuerelement ermöglicht, zweidimensionale Daten effizient darzustellen. Ihre mehrdimensionalen Tabellen sind in der Regel zweidimensional - das Tabellen-Steuerelement wird Ihnen also sehr gelegen kommen.

Heute lernen Sie die folgenden Dinge:

- Mehrdimensionale Arrays
- Deklaration mehrdimensionaler Arrays
- Initialisierung mehrdimensionaler Arrays
- Schleifen zur Verarbeitung mehrdimensionaler Arrays
- Grenzen für mehrdimensionale Arrays
- Das Tabellen-Steuerelement
- Eigenschaften des Tabellen-Steuerelements

- Methoden zum Auffüllen von Tabellen
- Die `FormatString`-Eigenschaft
- Bilder in Tabellenzellen

Mehrdimensionale Arrays - Grundlagen

Es gibt Daten, die am besten in eindimensionalen Arrays abgelegt werden (wie Sie in den ersten 21 Kapiteln dieses Buchs schon erfahren haben). Andere Daten werden besser im Tabellenformat abgelegt. Die Arrays, die Sie bisher kennengelernt haben, waren eindimensional und hatten nur einen einzigen Index. Eindimensionale Arrays stellen Wertelisten dar. Abbildung 22.1 zeigt, daß ein Array mit einer einzigen Dimension eine Länge und eine Richtung haben kann, so wie eine Linie.

 Beachten Sie, daß die Arrays im Speicher nicht so abgelegt werden, wie in Abbildung 22.1 gezeigt. Das Konzept einer einzigen Richtung hilft jedoch, die lineare Natur der Elemente in einem eindimensionalen Array zu verdeutlichen. Wenn Sie eine neue Dimension einführen, kommt auch eine neue Richtung hinzu, wie Sie gleich sehen werden.

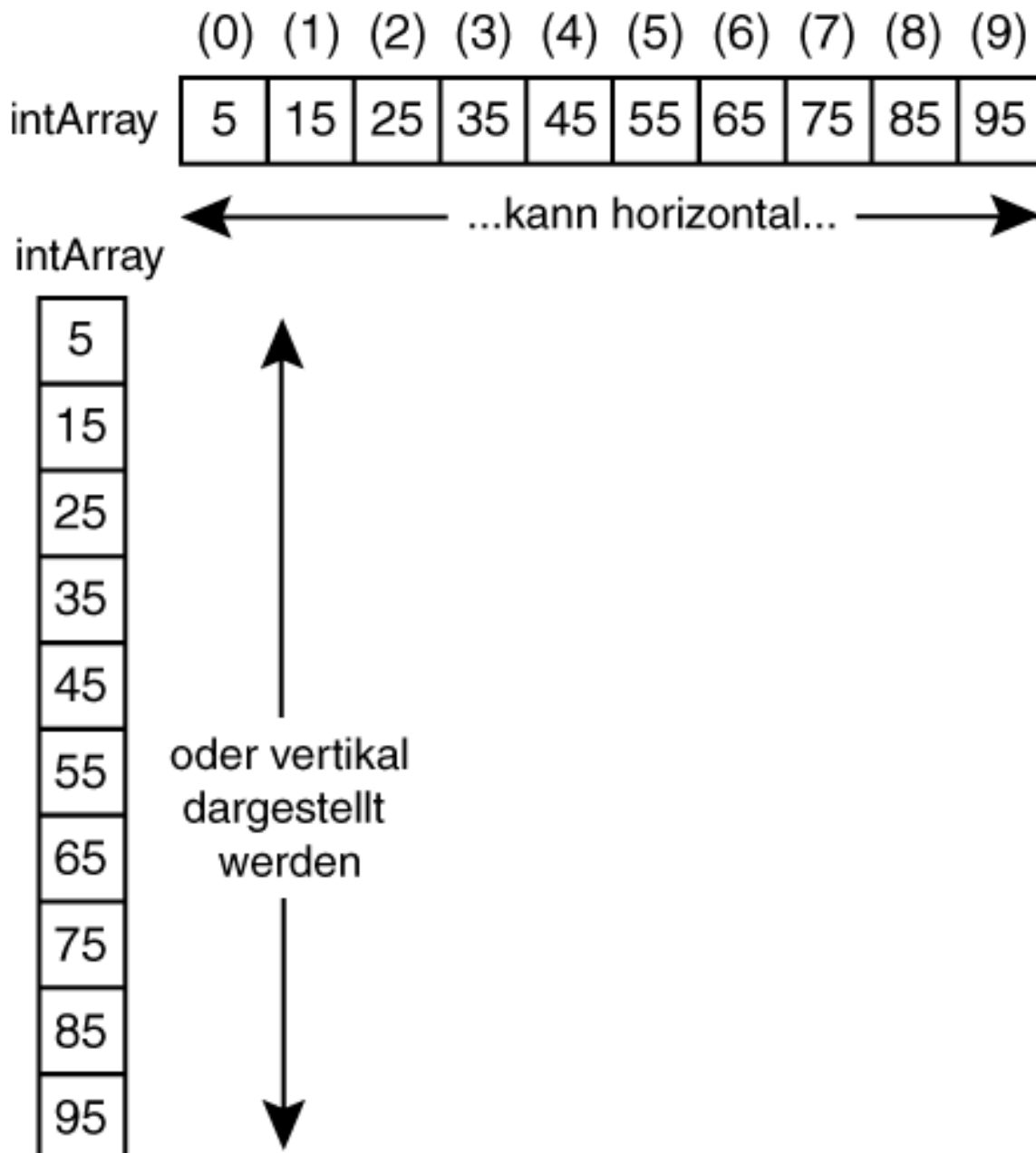


Abbildung 22.1:
Ein eindimensionales Array besitzt eine Länge und eine Richtung.

Das weitere Kapitel erklärt die Verwendung von Arrays mit mehreren Dimensionen, also *mehrdimensionaler Arrays*. Mehrdimensionale Arrays werden auch als *Tabellen* oder *Matrizen* bezeichnet, die sich aus Zeilen und Spalten zusammensetzen.



Ein *mehrdimensionales Array* verwendet mehrere Indizes und hat demzufolge mehrere Dimensionen. Die Anzahl der Dimensionen bestimmt die Anzahl der Richtungen einer Tabelle. Ein eindimensionales Array hat eine einzige Richtung, während ein dreidimensionales Array drei Richtungen hat, so wie ein Objekt im dreidimensionalen Raum Breite, Höhe und Tiefe aufweist.

Angenommen, ein Softball-Team möchte die Treffer seiner Spieler aufzeichnen. Das Team hat acht Spiele bestritten und besteht aus 10 Spielern. Tabelle 22.1 zeigt die erzielten Treffer.

Tabelle 22.1: Eine Tabelle eignet sich ausgezeichnet für die Aufzeichnung der Treffer eines Softball-Teams.

Spieler	Spiel 1	Spiel 2	Spiel 3	Spiel 4	Spiel 5	Spiel 6	Spiel 7	Spiel 8
Adams	2	1	0	0	2	3	3	1
Berryhill	1	0	3	2	5	1	2	2
Edwards	0	3	6	4	6	4	5	3
Grady	1	3	2	0	1	5	2	1
Howard	3	1	1	1	2	0	1	0
Powers	2	2	3	1	0	2	1	3
Smith	1	1	2	1	3	4	1	0
Townsend	0	0	0	0	0	0	1	0
Ulmer	2	2	1	1	2	1	1	2
Williams	2	3	1	0	1	2	1	1

Erkennen Sie die zweidimensionale Darstellung der Softball-Tabelle? Sie hat Zeilen (die erste Dimension) und Spalten (die zweite Dimension). Es handelt sich also um eine zweidimensionale Tabelle mit 10 Zeilen und acht Spalten. (Im allgemeinen wird zuerst die Anzahl der Zeilen angegeben.) Diese Tabelle erstreckt sich also in zwei Richtungen - horizontal und vertikal. Das eindimensionale Array erstreckt sich immer nur in eine einzige Richtung. Wie eindimensionale Arrays werden auch mehrdimensionale Arrays im Speicher nicht als Tabellen abgelegt, aber Visual Basic erlaubt Ihnen, die Daten so zu manipulieren, als wäre das der Fall.



Eine *Matrix* hat wie ein eindimensionales Array nur einen Namen. Darüber hinaus kann sie nur einen einzigen Datentyp aufnehmen, nämlich den, mit dem sie deklariert wurde. Wenn Sie die Matrix für den Datentyp `Variant` deklarieren, können die Zellen alle Daten enthalten, die durch den Datentyp `Variant` dargestellt werden können.



Eine *Zelle* ist ein Element eines Arrays beliebiger Dimension. In einem eindimensionalen Array ist eine Zelle ein Element der Arrayliste, in einem zweidimensionalen Array setzt sich eine Zelle aus der Schnittmenge zwischen Zeile und Spalte zusammen.

In Tabelle 22.1 enthält jede Zeile einen Spielernamen und jede Spalte stellt ein Spiel dar. Diese Überschriften sind aber nicht Teil der Daten. Die Daten umfassen nur 80 Werte (10 Zeilen mal acht Spalten). In diesem Fall ist jeder Wert ein Integer. Wenn die Tabelle Namen enthält, handelt es sich um eine Stringtabelle usw.

Die Anzahl der Dimensionen - in diesem Fall zwei - entspricht den Dimensionen der realen Welt. Die erste Dimension stellt eine Linie dar, ein eindimensionales Array ist eine »Wertelinie«: eine Liste. Zwei Dimensionen repräsentieren Länge und Breite. Auf einem Stück Papier können zwei Dimensionen dargestellt werden - eine flache Oberfläche. Drei Dimensionen repräsentieren Breite, Länge und Tiefe. Vielleicht haben Sie schon einmal dreidimensionale Filme gesehen; die Bilder haben nicht nur Höhe und Breite, sondern anscheinend auch Tiefe.



Wir gehen hier so detailliert auf die Dimensionen ein, weil Ihre Arbeit als Programmierer einfacher ist, wenn Sie sich mehrdimensionale Arrays genau vorstellen können.

Visual Basic erlaubt die Verwendung von bis zu 60 Dimensionen, aber es ist sehr schwierig, sich mehr als drei Dimensionen vorzustellen. Sie können sich jedoch jede Dimension höher drei wie eine weitere Instanz vorstellen. Mit anderen Worten, Sie können die Liste der Treffer eines Spielers in einer Saison als Array ablegen. Die Treffer des gesamten Teams werden in einem zweidimensionalen Array abgelegt (siehe Tabelle 22.1). Die Liga enthält die Trefferlisten mehrere Teams und könnte als dreidimensionale Tabelle dargestellt werden. Für jedes Team (die Tabellentiefe) gibt es Zeilen und Spalten für die Trefferdaten. Wenn es mehrere Ligen gibt, könnten Sie dafür noch eine Dimension einführen.



Obwohl Visual Basic bis zu 60 Dimensionen unterstützt, ist es sehr unwahrscheinlich, daß Sie Programme verwenden werden, die mehr als drei oder maximal vier Dimensionen verwenden. Größtenteils arbeiten Sie mit eindimensionalen oder zweidimensionalen Arrays.

Deklaration mehrdimensionaler Arrays

Wie für eindimensionale Arrays reservieren Sie auch für mehrdimensionale Arrays Speicher mit Hilfe der Anweisungen `Dim` oder `Public`. Statt eines einzigen Werts geben Sie in den Klammern für jede Dimension einen Wert an. Grundsätzlich sieht die Speicherreservierung für mehrdimensionale Arrays wie folgt aus:

```
Public taName(intSub) [As dataType][, taName(intSub) [As dataType]]...  
Dim taName (intSub) [As dataType][, taName (intSub) [As dataType]]...
```

Die `intSub`-Werte für die Tabelle können das folgende allgemeine Format annehmen:

```
[intLow To] intHighRow[, [intLow To] intHighColumn][, [intLow To]  
intHighDepth][, ...]
```

Wie für einzelne Dimensionen ist es auch für mehrere Dimensionen ganz einfach, Speicher zu reservieren, auch wenn das Format etwas kompliziert aussieht. Um beispielsweise die Teamdaten für Tabelle 22.1 zu deklarieren, würden Sie die folgende `Dim`-Anweisung verwenden:

```
Dim intTeams(1 To 10, 1 To 8) As Integer
```

Diese Symbolleiste reserviert im Speicher Platz für eine zweidimensionale Tabelle mit 80 Elementen. Die Indizes sind in Abbildung 22.2 gezeigt.

intTeams(1, 1)	intTeams(1, 2)	intTeams(1, 3)	...	intTeams(1, 7)	intTeams(1, 8)
intTeams(2, 1)	intTeams(2, 2)	intTeams(2, 3)	...	intTeams(2, 7)	intTeams(2, 8)
intTeams(3, 1)	intTeams(3, 2)	intTeams(3, 3)	...	intTeams(3, 7)	intTeams(3, 8)
:	:	:		:	:
intTeams(9, 1)	intTeams(9, 2)	intTeams(9, 3)	...	intTeams(9, 7)	intTeams(9, 8)
intTeams(10, 1)	intTeams(10, 2)	intTeams(10, 3)	...	intTeams(10, 7)	intTeams(10, 8)

Abbildung 22.2:
Für die Tabelle für das Softball-Team werden zwei Indizes verwendet.

Wenn Sie eine Liga mit 15 Teams verwalten möchten, führen Sie einen weiteren Index ein:
`Dim intTeams(1 To 15, 1 To 10, 1 To 8) As Integer`

Der erste Index steht für die einzelnen Teams in der Liga. Der zweite Index beschreibt die Anzahl der Spieler in jedem Team. Der dritte Index gibt an, wie viele Spiele jeder Spieler gespielt hat.



Sie können sich eine dreidimensionale Tabelle wie ein dreidimensionales Schachbrett vorstellen, wobei mehrere Bretter übereinandergestapelt sind. Eine vierdimensionale Darstellung würde mehrere solcher Schachbrettstapel beschreiben. Die vierte Dimension ist die Nummer, die jedem dieser Stapel entspricht.

Woher kennen Sie die Reihenfolge der Indizes, wie etwa in einer dreidimensionalen Tabelle? Woher wissen Sie, daß der Index ganz rechts die Spalten darstellt? Überhaupt nicht. Die Indizes stellen das dar, was Sie möchten. Standardmäßig wird bei der Umwandlung einer zweidimensionalen Tabelle in eine dreidimensionale Tabelle der zusätzliche Index als erster Wert angegeben, um sicherzustellen, daß die beiden anderen die Zeilen und die Spalten darstellen. Damit halten Sie die Indizes übersichtlich.

Die folgende Anweisung reserviert genügend Speicherelemente für die Shows, die ein Fernsehsender in einer Woche ausstrahlt:

```
Dim strShows(1 To 7, 1 To 48) As String
```

Diese Anweisung reserviert sieben Tage (die Zeilen) für 30-Minuten-Shows (weil ein Tag 24 Stunden hat, kann diese Tabelle bis zu 48 30-Minuten-Shows aufnehmen).

Wie Sie wissen, muß jedes Element der Tabelle denselben Datentyp haben. In diesem Fall handelt es sich dabei um Stringvariablen. Die folgenden Anweisungen initialisieren einige der Elemente:

```
strShows(3, 12) = "As the Hospital Turns"  
strShows(1, 5) = "Guessing-Game Show"  
strShows(7, 20) = "Raspberry Iced Tea Infomercial"
```

Bei der Reservierung mehrerer mehrdimensionaler Arrays wird sehr schnell sehr viel Speicher

verbraucht. Die folgenden Anweisungen sparen sehr viel Speicher:

```
Public ara1(10, 20) As Single  
Dim ara2(4, 5, 5) As Double  
Public ara3(6, 10, 20, 30) As Integer
```

`ara1` verbraucht 200 Speicherpositionen für einfache Genauigkeit. `ara2` verbraucht 100 Speicherpositionen für doppelte Genauigkeit. `ara3` verbraucht 36000 Speicherpositionen. Wie Sie sehen, summiert sich die Anzahl der Elemente sehr schnell. Achten Sie also darauf, nicht zu viele Arrayelemente zu reservieren und damit den ganzen Speicher zu füllen.

Wenn Sie die Tabellendaten in mehrdimensionale Arrays einlesen und dort damit arbeiten, beschleunigen Sie die Ausführung Ihres Programms. Alles, was Sie im Speicher erledigen können, ist schneller, als bei jedem Zugriff auf die Werte von der Platte zu lesen oder dorthin zu schreiben. Sie haben jedoch sehr viel mehr Plattenspeicher als Hauptspeicher. Wenn Sie mit sehr großen Dateien arbeiten, sollten Sie Speichereffizienz gegen Plattenkapazität abwägen.

Tabellen und For-Schleifen

Wie Sie in einigen der nächsten Programmbeispiele sehen werden, sind verschachtelte For-Schleifen gut dafür geeignet, die einzelnen Elemente mehrdimensionaler Tabellen zu durchlaufen. Beispielsweise gibt Listing 22.1 alle sechs möglichen Indexwerte für ein mehrdimensionales Array in aufeinanderfolgenden Meldungsfeldern aus.

Listing 22.1: Verschachtelte Schleifen ermöglichen Ihnen, Tabellen schnell zu durchlaufen

```
1: For intRow = 1 To 2  
2: For intCol = 1 To 3  
3: MsgBox("Zeile: " & intRow & ", Spalte: " & intCol)  
4: Next intCol  
5: Next intRow
```

Wenn Sie den Code in Listing 22.1 ausführen, sehen Sie die folgende Ausgabe in den Meldungsfeldern:

```
Zeile: 1, Spalte: 1  
Zeile: 1, Spalte: 2  
Zeile: 1, Spalte: 3  
Zeile: 2, Spalte: 1  
Zeile: 2, Spalte: 2  
Zeile: 2, Spalte: 3
```

Statt der Meldungsfelder könnten Sie im Code auch `Print`-Methoden verwenden, um direkte Ausgaben auf das Formular vorzunehmen. `Print` ist wahrscheinlich die beste Methode, wenn Sie die Indizes eines mehrdimensionalen Arrays ausgeben wollen, weil Sie auf einem Formular sehr viele Ausgaben unterbringen können. Listing 22.2 verwendet die `Print`-Methode, um direkt auf ein Formular zu drucken. Abbildung 22.3 zeigt das Formular.

Listing 22.2: Verschachtelte Schleifen ermöglichen Ihnen, die gesamte Tabelle zu durchlaufen

```
1: For intRow = 1 To 2  
2: For intCol = 1 To 3  
3: Form1.Print "Zeile: " & intRow & ", Spalte: " & intCol  
4: Next intCol  
5: Form1.Print  
6: Next intRow
```



Betrachten Sie Abbildung 22.3. In Listing 22.2 sollte demonstriert werden, wie eine For-Schleife Werte bereitstellt, mit denen die Zeilen und Spalten einer Tabelle durchlaufen werden können. Die in Abbildung 22.3 ausgedruckten Werte sind *keine* Array-Werte, sondern die Indizes eines Arrays, das mit zwei Zeilen und drei Spalten deklariert ist.

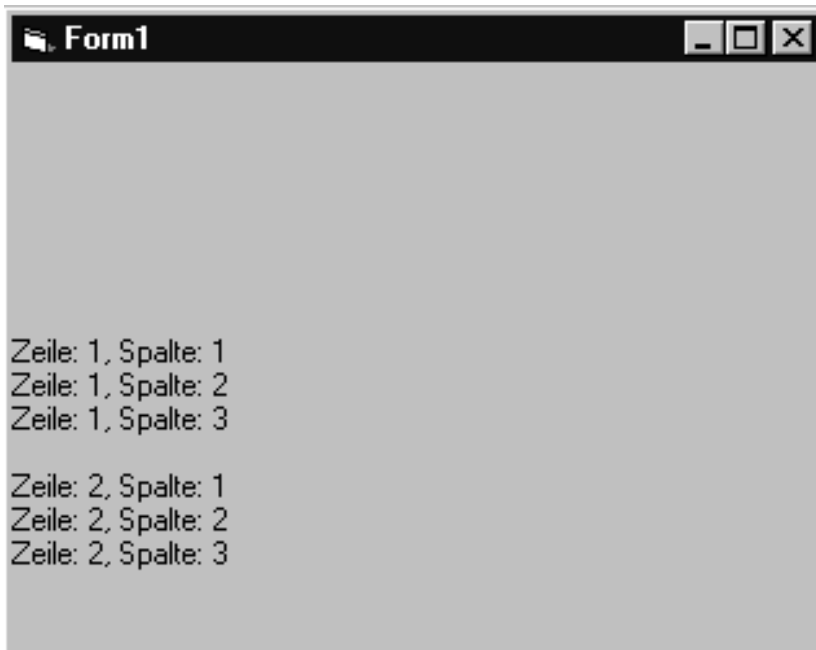


Abbildung 22.3:

Sie können die Tabellenwerte auch mit `Print` ausgeben, um alle Ausgaben auf einem einzigen Formular anzuzeigen.

Wenn Sie die Indizes für eine Tabelle mit zwei Zeilen und drei Spalten ausgeben, die mit der folgenden `Dim`-Anweisung deklariert wurde, sehen Sie die in Listing 22.2 ausgegebenen Indexnummern.

```
Dim intTable(1 To 2, 1 To 3)
```

Beachten Sie, daß es im Code so viele `For . . Next`-Anweisungen gibt, wie es in der `Dim`-Anweisung Indizes gibt (zwei). Die äußere Schleife stellt den ersten Index dar (die Zeilen), die innere Schleife den zweiten (die Spalten). Die verschachtelte Schleife ist vielleicht die gebräuchlichste Methode, eine Tabelle zu durchlaufen. Für die effiziente Arbeit mit mehrdimensionalen Arrays sollten Sie also wissen, wie man verschachtelte Schleifen anlegt.

Arrays initialisieren

Die Elemente eines mehrdimensionalen Arrays können auf verschiedenen Arten initialisiert werden. Hier nur ein paar davon:

- Wertzuweisungen an die Tabellenelemente
- Mit `InputBox` die Elemente nacheinander mit den aus einem Eingabefeld ermittelten Werten füllen
- Werte nacheinander von einer Platte oder einer Datenbankdatei lesen
- Werte aus anderen Werten berechnen

Daran erkennen Sie, daß Tabellen und alle anderen mehrdimensionalen Arrays wie alle anderen Variablen initialisiert werden. Diese Methode ermöglicht Ihnen jedoch, sich Daten in tabellarischer Form vorzustellen, was die Programmierung und die Programmwartung beschleunigt.

Die meisten Daten in mehrdimensionalen Arrays stammen aus Formularen oder - noch häufiger - aus Dateien. Unabhängig davon, wie Sie Werte in mehrdimensionalen Arrays speichern, stellen verschachtelte For-Schleifen die beste Methode dar, die Indizes zu durchlaufen. Das folgende Beispiel erklärt genauer, wie verschachtelte For-Schleifen für mehrdimensionale Arrays eingesetzt werden.

Angenommen, eine Computerfirma verkauft Disketten in zwei Größen: 3½ Zoll und 5¼ Zoll. Jede Diskette kann eine von vier Datenkapazitäten aufweisen: single-sided, low-density; double-sided, low-density; single-sided, high-density; und double-sided, high-density. Die Inventarliste wird als zweidimensionale Tabelle verwaltet. Die Disketten haben die folgenden Verkaufspreise:

	Single-Sided Low-Density	Double-Sided Low-Density	Single-Sided High-Density	Double-Sided High-Density
3½"	\$2.30	2.75	3.20	3.50
5¼"	\$1.75	2.10	2.60	2.95

Die Prozedur in Listing 22.3 speichert den Preis der einzelnen Disketten in einer Tabelle und gibt die Werte mit Hilfe einer verschachtelten For-Schleife auf das Formular aus. Diese Prozedur kann in einem Standardmodul oder einer Ereignisprozedur abgelegt werden, wo die Ausführung ausgelöst wird.

Listing 22.3: Inventarlisten werden häufig als Tabellen dargestellt

```
1: Private Sub disks ()
2: ' Weist Diskettenpreise zu und druckt sie aus
3: Dim curDisks(1 To 2, 1 To 4) As Currency
4: Dim intRow As Integer, intCol As Integer
5: ' Den Elementen Preise zuordnen
6: curDisks(1, 1) = 2.30 ' Row 1, Column 1
7: curDisks(1, 2) = 2.75 ' Row 1, Column 2
8: curDisks(1, 3) = 3.20 ' Zeile 1, Spalte 3
9: curDisks(1, 4) = 3.50 ' Zeile 1, Spalte 4
10: curDisks(2, 1) = 1.75 ' Zeile 2, Spalte 1
11: curDisks(2, 2) = 2.10 ' Zeile 2, Spalte 2
12: curDisks(2, 3) = 2.60 ' Zeile 2, Spalte 3
13: curDisks(2, 4) = 2.95 ' Zeile 2, Spalte 4
14: ' Preise im Tabellenformat ausgeben
15: Form1.Print
16: Form1.Print Tab(12); "Single-sided, Double-sided, ";
17: Form1.Print "Single-sided, Double-sided"
18: Form1.Print Tab(12); "Low-density Low-density ";
19: Form1.Print "High-density High-density"
20: For intRow = 1 To 2
21: If (intRow = 1) Then
22: Form1.Print "3-1/2 inch"; Tab(15);
23: Else
24: Form1.Print "5-1/4 inch"; Tab(15);
25: End If
26: For intCol = 1 To 4
27: Form1.Print curDisks(intRow, intCol); Spc(8);
28:
29: Next intCol
```

```

30: Form1.Print ' Cursor in die nächste Zeile setzen
31: Next intRow
32: End Sub

```

Diese Prozedur erzeugt die in Abbildung 22.4 gezeigte Ausgabe, nachdem Sie das Fenster so vergrößert haben, daß die gesamte Tabelle angezeigt wird. Die Tabelle ist zwar klein, aber das 2x4-mehrdimensionale Array demonstriert, welche Daten für die Tabellendarstellung geeignet sind. Die beiden Zeilen und die vier Spalten der Preistabelle sind ausgezeichnet für die Verwendung eines mehrdimensionalen Arrays geeignet. Der Code erscheint relativ umfangreich für eine Tabelle mit nur 8 Einträgen. Beachten Sie jedoch, daß Sie Tabellen selten so initialisieren, wie hier in den Zeilen 6 bis 13 geschehen. Für dieses kleine Beispiel und als Einführung in die Arbeit mit Tabellen sind die Zuweisungen völlig in Ordnung.



In Kapitel 10 haben Sie die `Array()`-Funktion kennengelernt, die einem vollständigen Array in einer einzigen Anweisung Werte zuweist. Verwenden Sie `Array()` nicht für mehrdimensionale Arrays. `Array()` darf nur für eindimensionale Arrays eingesetzt werden. Es ist nicht möglich, mehrdimensionale Arrays mit `Array()` zu initialisieren.

	Single-sided, Low-density	Double-sided, Low-density	Single-sided, High-density	Double-sided, High-density
3-1/2 inch	2,3	2,75	3,2	3,5
5-1/4 inch	1,75	2,1	2,6	2,95

Abbildung 22.4:
Die Diskettenpreise werden in Tabellenform dargestellt.

Das Tabellen-Steuerelement

Das gebräuchlichste zweidimensionale Array, die Tabelle, wird den Benutzern am besten in Zeilen/Spalten-Form präsentiert. Das Tabellen-Steuerelement stellt eine bequeme Methode dar, Ihren Benutzern Tabellendaten anzuzeigen. Die Benutzer bewegen sich mit Hilfe von Bildlaufleisten in der Tabelle. Das Tabellen-Steuerelement muß also nicht so groß sein wie die Tabelle, weil es automatisch Bildlaufleisten anzeigt.

Installation des Tabellen-Steuerelements

Das Tabellen-Steuerelement ist nicht Teil der Standard-Werkzeugsammlung. Gehen Sie wie folgt vor, um es zu installieren:

1. Drücken Sie (Strg)+(T), um das Dialogfeld **Komponenten** zu öffnen.
2. Wählen Sie **Microsoft FlexGrid Control 6.0** aus.
3. Klicken Sie auf **OK**. Das FlexGrid-Steuerelement (auch einfach nur als Tabellen- Steuerelement bezeichnet) erscheint in Ihrer Werkzeugsammlung, wie in Abbildung 22.5 gezeigt.

	Visual Basic beinhaltet mehrere Tabellen-Steuerelemente. Sie finden sie alle im Dialogfeld Komponenten. Bei einigen davon handelt es sich um gebundene Steuerelemente, die Sie in Kapitel 18 kennengelernt haben. Diese Steuerelemente können also Datenbankwerte aufnehmen.



Abbildung 22.5:
Das Tabellen-Steuerelement ermöglicht Ihnen,
tabellenförmige Daten anzuzeigen.

Die Verwendung des Tabellen-Steuerelements

Wenn Sie das Tabellen-Steuerelement auf einem Formular anlegen, müssen Sie seine Größe anpassen, damit es wie eine Tabelle aussieht. Das Steuerelement sieht auch nach der Vergrößerung noch nicht wie eine Tabelle aus (wie Sie in Abbildung 22.6 sehen). Das Problem dabei ist, daß die Anzahl der Zeilen und Spalten standardmäßig auf 2 gesetzt wird. Wenn Sie Zeilen und Spalten einfügen, sieht das Steuerelement schon eher wie eine Tabelle aus.

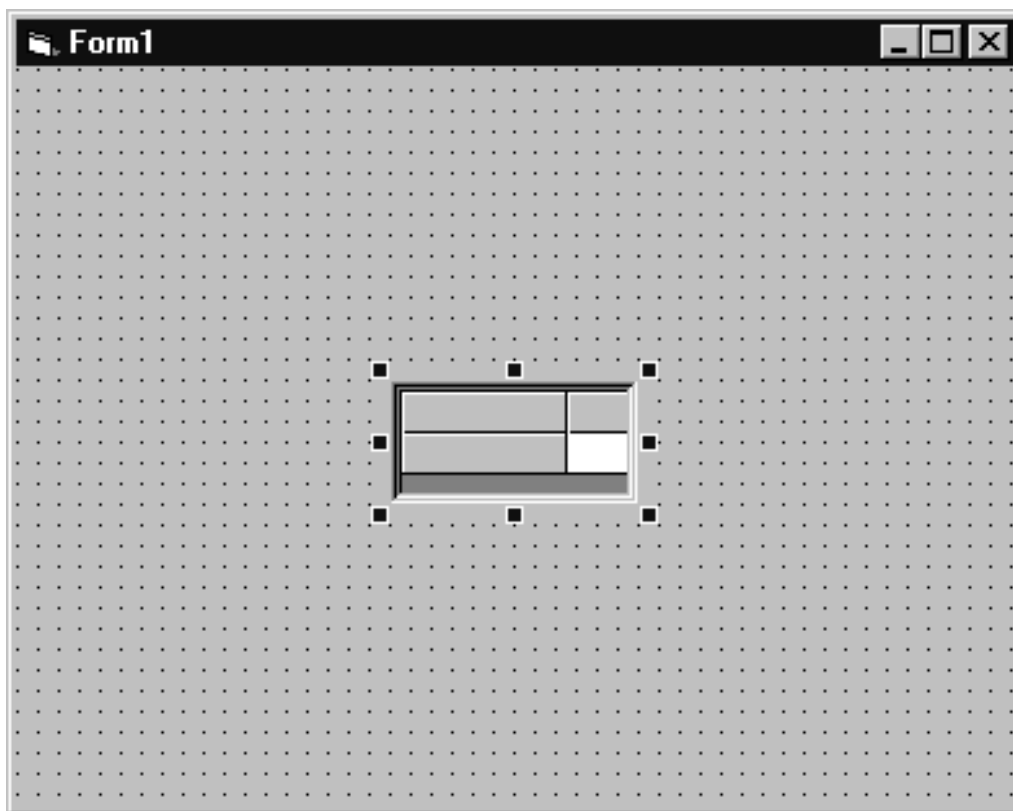


Abbildung 22.6:
Das Tabellen-Steuerelement sieht nicht wie eine Tabelle aus, wenn es auf einem Formular angelegt wird.

Wie Sie sehen, müssen Sie als erstes die Anzahl der Zeilen und Spalten anpassen, bis die Tabelle die richtige Größe hat. Beachten Sie, daß das Tabellen-Steuerelement nicht dieselbe Anzahl an Zeilen und Spalten beinhalten muß wie Ihr mehrdimensionales Array, weil das Steuerelement Bildlaufleisten anzeigt, mit denen sich der Benutzer durch die Daten bewegt.



Das Tabellen-Steuerelement unterstützt eine feste Anzahl von Zeilen und Spalten, wie in Abbildung 22.7 gezeigt. Sie können diese Anzahl und das Erscheinungsbild von Zeilen und Spalten bestimmen. Wenn Sie die Anzahl der festen Zeilen und Spalten erhöhen, werden auch die grau unterlegten Bereiche entsprechend angepaßt.



Feste Zeilen und Spalten sind die Zeilen und Spalten in einer Tabelle, die nicht verschoben werden, wenn der Benutzer in die Bildlaufleisten klickt. Die festen Zeilen und Spalten zeigen Beschriftungen an, die die in der Tabelle ausgegebenen Daten beschreiben, ähnlich wie in Tabellenkalkulationen.

Die festen Zeilen und Spalten werden häufig auch als *Zeilen- und Spaltentitel* bezeichnet.

AllowUserResizing	Bestimmt, wieviel Kontrolle Sie Ihren Benutzern an die Hand geben, die Breite und die Höhe der Spalten anzupassen. Ist diese Eigenschaft gleich <code>flexResizeNone</code> , darf der Benutzer die Höhe und Breite von Spalten und Zeilen nicht ändern. Ist sie <code>flexResizeColumns</code> , darf der Benutzer die Spaltenbreite ändern. Ist sie <code>flexResizeRows</code> , darf der Benutzer die Zeilenhöhe ändern. Ist sie <code>flexResizeBoth</code> , darf der Benutzer die Breite und die Höhe von Spalten und Zeilen ändern. Abhängig von den Daten braucht der Benutzer diese Möglichkeiten, um die Daten im Tabellen-Steuerelement zu lesen.
Cols	Gibt die Anzahl der Spalten im Tabellen-Steuerelement an.
Custom	Öffnet das Dialogfeld Eigenschaftenseiten für das Steuerelement (siehe Abbildung 22.8).
FillStyle	Gibt an, wie eine Zelle oder ein Zellbereich formatiert wird. Ist diese Eigenschaft gleich <code>flexFillSingle</code> , wird die aktuelle Zelle formatiert. Ist sie <code>flexFillRepeat</code> , werden alle selektierten Zellen formatiert.
FixedCols	Gibt die Anzahl der festen (nicht bewegbaren) Spalten für die Überschriften an.
FixedRows	Gibt die Anzahl der festen (nicht bewegbaren) Zeilen für die Überschriften an.
FocusRect	Bestimmt, wie die vom Benutzer selektierte Zelle auf dem Bildschirm hervorgehoben dargestellt werden soll. Für den Wert <code>flexFocusNone</code> wird die Zelle nicht gesondert hervorgehoben. Ist er <code>flexFocusLight</code> , erscheint eine helle Umrißlinie um die aktuelle Zelle. Ist er <code>flexFocusHeavy</code> , wird die aktuelle Zelle blau unterlegt.
FormatString	Enthält einen String, der angibt, wie eine Zelle oder mehrere selektierte Zellen formatiert werden sollen
GridLines	Gibt an, wie die nicht festen Linien aussehen sollen. Ist diese Eigenschaft auf <code>flexGridNone</code> gesetzt, werden die Zellen nicht durch Linien voneinander getrennt. Ist sie auf <code>flexGridFlat</code> gesetzt, werden die Zellen durch graue Linien voneinander getrennt. Ist sie auf <code>flexGridRaised</code> gesetzt, erscheint die Zelle im 3D-Stil.
GridLinesFixed	Gibt an, wie die festen Linien für die Tabelle aussehen. Ist diese Eigenschaft auf <code>flexGridNone</code> gesetzt, werden die Zellen nicht durch Linien voneinander getrennt. Ist sie auf <code>flexGridFlat</code> gesetzt, werden die Zellen durch graue Linien voneinander getrennt. Ist sie auf <code>flexGridRaised</code> gesetzt, erscheint die Zelle im 3D-Stil.
HighLight	Gibt an, wie die selektierten Zellen aussehen sollen. Ist diese Eigenschaft auf <code>flexHighlightNever</code> gesetzt, werden die selektierten Zellen nicht gekennzeichnet. Ist sie auf <code>flexHighlightAlways</code> gesetzt, erscheinen die selektierten Zellen markiert (mit dunklem Hintergrund). Ist sie auf <code>flexHighlightWithFocus</code> gesetzt, erscheinen die markierten Zellen nur, wenn das Tabellen-Steuerelement den Fokus besitzt.
RowHeigthMin	Gibt an, wie viele Twips eine Zeile mindestens hoch sein muß.

Rows	Gibt die Anzahl der Zeilen an, die im Tabellen-Steuerelement angezeigt werden.
SelectionMode	Gibt an, wie Zellen selektiert werden. Ist diese Eigenschaft gleich <code>flexSelectionFree</code> , kann der Benutzer beliebige rechteckige Zellbereiche auswählen. Ist sie auf <code>flexSelectionByRow</code> gesetzt, erstrecken sich die selektierten Zellen immer auf eine ganze Zeile. Ist sie auf <code>flexSelectionByColumn</code> gesetzt, erstrecken sich die selektierten Zellen immer auf eine ganze Spalte.
WordWrap	Gibt an, ob ein Umbruch in einer Zelle erfolgt, wenn der Inhalt die Höhe oder Breite überschreitet.



Abbildung 22.8:

Das Dialogfeld Eigenschaftenseiten ermöglicht Ihnen, auf einfache Weise gebräuchliche Eigenschaften für das Tabellen-Steuerelement zu setzen.

Das Tabellen-Steuerelement unterstützt zahlreiche Laufzeiteigenschaften, die gesetzt werden müssen. Andernfalls ist das Steuerelement nicht funktional. Tabelle 22.3 enthält viele Laufzeiteigenschaften, die von Ihrem Code so gesetzt werden müssen, daß das Steuerelement korrekt arbeitet.

	Einige der Eigenschaften für die Formatierung, wie beispielsweise <code>CellAlignment</code> , ermöglichen Ihnen, das Steuerelement zur Laufzeit zu formatieren. Um eine Zelle zur Entwurfszeit zu formatieren, verwenden Sie die Eigenschaft <code>FormatString</code> (siehe Tabelle 22.2).

Tabelle 22.3: Ihr Visual-Basic-Code muß mehrere Laufzeit-Eigenschaften setzen.

Eigenschaft	Beschreibung

CellAlignment	Bestimmt die Ausrichtung von Werten in einer Zelle. Ist diese Eigenschaft auf <code>flexAlignLeftTop(0)</code> gesetzt, wird der Zellinhalt an der linken oberen Ecke ausgerichtet. Ist sie auf <code>flexAlignLeftCenter(1)</code> gesetzt, wird der Zellinhalt links ausgerichtet und zentriert. Auf ähnliche Weise richten die anderen Konstantenwerte den Zellinhalt aus: <code>flexAlignLeftBottom (2)</code> , <code>flexAlignCenterTop (3)</code> , <code>flexAlignCenterCenter (4)</code> , <code>flexAlignCenterBottom (5)</code> , <code>flexAlignRightTop (6)</code> , <code>flexAlignRightCenter (7)</code> , <code>flexAlignRightBottom (8)</code> , und <code>flexAlignGeneral (9)</code> .
Col	Setzt die Spaltennummer der Zelle, deren Wert Sie ändern möchten.
ColAlignment	Bestimmt die Ausrichtung der Werte innerhalb einer bestimmten Spalte. Dazu werden dieselben Werte wie für <code>CellAlignment</code> verwendet.
ColWidth	Gibt die Spaltenbreite in Twips an.
Row	Setzt die Zeilennummer der Spalte, die Sie ändern möchten.
SelEndCol	Gibt die Spaltennummer der äußersten rechten Spalte des selektierten Bereichs an.
SelEndRow	Gibt die Zeilennummer der untersten Zeile des selektierten Bereichs an.
SelStartCol	Gibt die Spaltennummer der äußersten linken Spalte des selektierten Bereichs an.
Text	Gibt den Inhalt der betreffenden Zelle an. Sie können der aktuellen Zelle (der Schnittpunkt zwischen <code>Row</code> und <code>Col</code>) einen neuen Wert zuweisen, indem Sie der <code>Text</code> -Eigenschaft diesen Wert zuweisen.

	Es gibt noch viele andere Eigenschaften für Entwurfszeit und Laufzeit; die Tabellen 22.2 und 22.3 listen die wichtigsten davon auf.

Der Zellinhalt wird mit Hilfe der `Text`-Eigenschaft gelesen oder gesetzt, aber die Daten selbst müssen keine Stringdaten sein. Sie können der `Text`-Eigenschaft numerische Werte zuweisen. Visual Basic wandelt die Zahlen in einen String um, bevor es die Zuweisung ausführt. Auf ähnliche Weise können Sie einer Zelle eine numerische Variable zuweisen, die Visual Basic umwandelt, bevor es die Zuweisung vornimmt. Wenn Visual Basic den Wert nicht in eine Zahl umwandeln kann, tritt ein Laufzeitfehler auf, wenn Sie den Wert einer numerischen Variablen zuweisen.

Das Tabellen-Steuerelement in einer Applikation

Am besten verstehen Sie die Arbeitsweise des Tabellen-Steuerelements anhand einer Beispiel-Applikation. Lesen Sie also die nächsten Abschnitte, in denen eine Applikation entwickelt wird, die sowohl mehrdimensionale Arrays als auch ein Tabellen-Steuerelement verwendet. Die Applikation braucht das Tabellen-Steuerelement, um selektierte Zellen innerhalb des Steuerelements zu verwalten, damit Sie sehen, wie man mit Zellbereichen arbeitet.

Das Projekt

Abbildung 22.9 zeigt den Bildschirm für die Applikation an, die Sie hier entwickeln werden. Das Tabellen-Steuerelement zeigt die Verkaufszahlen für bestimmte Produkte und bestimmte Verkäufer an. Die Namen der Verkäufer erscheinen in der Zeilenüberschrift, die Produktnamen links von der Tabelle.

Alle Daten werden in einer zweidimensionalen Tabelle abgelegt und in das Tabellen- Steuerelement übertragen. Diese Applikation lädt die Tabelle nicht von der Platte und speichert sie auch nicht dort. Der Quellcode enthält deshalb zahlreiche Zuweisungen. In einer echten Applikation sollten Sie die Tabelle auf einer Platte ablegen, am besten jedoch in einer Datenbank. In diesem Projekt wollen wir uns jedoch auf Tabellen und das Tabellen-Steuerelement konzentrieren und die I/O-Funktionen nicht berücksichtigen.

Das Tabellen-Steuerelement und Tabellen

Verkaufzahlendiagramm

	Smith	Johnson	Lake	West
#1 Widget	2,00	3,00	4,00	5,00
#2 Widget	1,50	2,00	2,50	3,00
Long tube	1,33	1,67	2,00	2,33
Short tube	1,25	1,50	1,75	2,00
Wood liner	1,20	1,40	1,60	1,80
Plastic liner	1,17	1,33	1,50	1,67
Rubber liner	1,14	1,29	1,43	1,57
Basket	1,13	1,25	1,38	1,50
Bolt 3C	1,11	1,22	1,33	1,44
Bolt 5A	1,10	1,20	1,30	1,40

1% 15% 1% 15%

Steigerung um 15%

Verringerung um 15%

Beenden

Abbildung 22.9:
Das Tabellen-Steuerelement ermöglicht Ihnen, Verkaufszahlen zu verwalten.

Nachdem Sie diese Applikation entwickelt und ausgeführt haben, haben Sie ein besseres Gefühl für die Verwendung des Tabellen-Steuerelements. Außerdem können Sie dieses einfache Beispiel für eigene Applikationen einsetzen.

Legen Sie ein neues Projekt an und fügen Sie die in Tabelle 22.4 beschriebenen Steuerelemente hinzu.



Fügen Sie Ihrer Werkzeugsammlung das Tabellen-Steuerelement (Microsoft FlexGrid Control 6.0) hinzu, bevor Sie das Formular bearbeiten.

Tabelle 22.4: Legen Sie diese Steuerelemente auf dem Formular für die Verwaltung der Verkaufszahlen an.

Eigenschaft	Wert
Formular Name	frmSales
Formular Caption	Verwendung des Tabellen-Steuerelements
Formular Height	7920
Formular Width	9180
Bezeichnungsfeld #1 Name	lblSales
Bezeichnungsfeld #1 Alignment	Center
Bezeichnungsfeld #1 Caption	Diagramm Verkaufszahlen
Bezeichnungsfeld #1 Font size	18
Bezeichnungsfeld #1 Font style	Bold
Bezeichnungsfeld #1 Height	495
Bezeichnungsfeld #1 Left	1200
Bezeichnungsfeld #1 Top	1200
Bezeichnungsfeld #1 Width	6015
Bezeichnungsfeld #2 Name	lblInMin
Bezeichnungsfeld #2 Alignment	Center
Bezeichnungsfeld #2 Caption	1%
Bezeichnungsfeld #2 Font Size	8
Bezeichnungsfeld #2 Height	255
Bezeichnungsfeld #2 Left	960
Bezeichnungsfeld #2 Top	5640
Bezeichnungsfeld #2 Width	375
Bezeichnungsfeld #3 Name	lblInMax
Bezeichnungsfeld #3 Alignment	Center
Bezeichnungsfeld #3 Caption	15%
Bezeichnungsfeld #3 Font Size	8
Bezeichnungsfeld #3 Height	255
Bezeichnungsfeld #3 Left	2880
Bezeichnungsfeld #3 Top	5640
Bezeichnungsfeld #3 Width	375
Bezeichnungsfeld #4 Name	lblDeMin
Bezeichnungsfeld #4 Alignment	Center

Bezeichnungsfeld #4 Caption	1%
Bezeichnungsfeld #4 Font Size	8
Bezeichnungsfeld #4 Height	255
Bezeichnungsfeld #4 Left	5160
Bezeichnungsfeld #4 Top	5640
Bezeichnungsfeld #4 Width	375
Bezeichnungsfeld #5 Name	lblDeMax
Bezeichnungsfeld #5 Alignment	Center
Bezeichnungsfeld #5 Caption	15%
Bezeichnungsfeld #5 Font Size	8
Bezeichnungsfeld #5 Height	255
Bezeichnungsfeld #5 Left	7080
Bezeichnungsfeld #5 Top	5640
Bezeichnungsfeld #5 Width	375
Tabellen-Steuerelement Name	grdSales
Tabellen-Steuerelement Cols	8
Tabellen-Steuerelement FocusRect	flexFocusNone
Tabellen-Steuerelement Height	3015
Tabellen-Steuerelement Left	1320
Tabellen-Steuerelement Rows	20
Tabellen-Steuerelement Top	1800
Tabellen-Steuerelement Width	5895
Horizontale Bildlaufleiste #1 Name	hscIncrease
Horizontale Bildlaufleiste #1 Left	1080
Horizontale Bildlaufleiste #1 Max	15
Horizontale Bildlaufleiste #1 Min	1
Horizontale Bildlaufleiste #1 Top	5280
Horizontale Bildlaufleiste #1 Width	2055
Horizontale Bildlaufleiste #2 Name	hscDecrease
Horizontale Bildlaufleiste #2 Left	5280
Horizontale Bildlaufleiste #2 Max	15
Horizontale Bildlaufleiste #2 Min	1
Horizontale Bildlaufleiste #2 Top	5280

Horizontale Bildlaufleiste #2 Width	2055
Befehlsschaltfläche #1 Name	cmdIncrease
Befehlsschaltfläche #1 Caption	Um 15% &verringern
Befehlsschaltfläche #1 Height	495
Befehlsschaltfläche #1 Left	1320
Befehlsschaltfläche #1 Top	6000
Befehlsschaltfläche #1 Width	1575
Befehlsschaltfläche #2 Name	cmdDecrease
Befehlsschaltfläche #2 Caption	&Um 15% erhöhen
Befehlsschaltfläche #2 Height	495
Befehlsschaltfläche #2 Left	5520
Befehlsschaltfläche #2 Top	6000
Befehlsschaltfläche #2 Width	1575
Befehlsschaltfläche #3 Name	cmdExit
Befehlsschaltfläche #3 Caption	&Beenden
Befehlsschaltfläche #3 Height	495
Befehlsschaltfläche #3 Left	3720
Befehlsschaltfläche #3 Top	6720
Befehlsschaltfläche #3 Width	1215

Der Code

Listing 22.4 enthält den Code für die Initialisierung aller Programmwerte. Der Code ist eine Steuerprozedur, weil beim Laden des Formulars mehrere `Call`-Anweisungen andere Sub-Prozeduren aufrufen. Beachten Sie, daß ein Großteil dieser Prozedur für die Initialisierung der Tabellenüberschriften zuständig ist und dort die Namen der Verkäufer bzw. die Produkte anzeigt.



Listing 22.4 zeigt ein gutes Beispiel für *modulare Programmierung*. Wenn Sie später eine Änderung an dem Programm vornehmen möchten, brauchen Sie nicht mehrere Seiten Code für eine einzige Prozedur zu durchsuchen. Statt dessen haben alle aufgerufenen Prozeduren jeweils nur eine Aufgabe. Die Prozeduren können abgeändert werden, ohne den Code anderer Prozeduren zu stören.



Die *modulare Programmierung* legt Code, der für eine bestimmte Aufgabe zuständig ist, in einer allgemeinen Sub-Prozedur ab und ruft diesen Code dann von einer allgemeinen, steuernden Prozedur aus auf.

Listing 22.4: Die Prozedur `Form_Load()` initialisiert mehrere Werte in Sub-Prozeduren

```

1: Private Sub Form_Load()
2: ' Definition der Zellausrichtung und Zuweisung der
3: ' Zellentitel für die festen Zeilen und Spalten.
4: ' Darüber hinaus wird die Tabelle mit Werten initialisiert,
5: ' und dann an das Tabellen-Steuerelement geschickt.
6: '
7: Call InitScrolls ' Bildlaufleisten initialisieren
8: Call CenterCells ' Zellen zentrieren
9: Call SizeCells ' Zellenbreite
10: Call Titles ' Spalten- und Zeilentitel
11: Call FillCells ' Zellen mit Werten füllen
12: End Sub

```

In den Zeilen 7 bis 11 erkennen Sie auf den ersten Blick, welche Prozeduren `Form_Load()` aufruft. würde sich der aufgerufene Code nicht in separaten Prozeduren befinden, sondern in `Form_Load()` eingebettet sein, wäre eine spätere Abänderung des Codes sehr viel komplizierter.

Listing 22.5: Die ersten drei Prozeduren, die `Form_Load()` aufruft, richten die Tabelle ein

```

1: Private Sub InitScrolls()
2: ' Beide Bildlaufleisten auf maximale Werte setzen.
3: ' Sie können diese Werte im Eigenschaftenfenster setzen,
4: ' aber diese Prozedur ermöglicht Ihnen, die Maximalwerte
5: ' für die Bildlaufleisten einfacher anzupassen, wenn
6: ' sich die Programmanforderungen ändern.
7: '
8: hscIncrease.Value = 15
9: hscDecrease.Value = 15
10: End Sub
11:
12: Private Sub CenterCells()
13: ' Zellen zentriert ausrichten. Dabei sollen sowohl
14: ' die Zeilen als auch die Spalten zentriert
15: ' werden.
16: '
17: Dim Column As Integer
18: '
19: ' Überschriftszellen zentrieren
20: For Column = 0 To 7
21:   grdSales.Col = Column ' Aktuelle Spalte setzen
22:   ' Feste Zellen in dieser Spalte zentrieren
23:   grdSales.ColAlignment(Column) = flexAlignCenterCenter
24: Next Column
25: End Sub
26:
27: Private Sub SizeCells()
28: ' Breite der Zellen
29: Dim Column As Integer
30: For Column = 0 To 7
31:   grdSales.ColWidth(Column) = 1100 ' In twips
32: Next Column
33: End Sub

```

Die Prozedur `InitScrolls()` ist nicht zwingend erforderlich, weil die maximalen Positionen für die Bildlaufleisten auch im Eigenschaftfenster gesetzt werden können. Spätere Änderungen werden jedoch wesentlich einfacher durch die Bereitstellung einer separaten Prozedur.

Die Prozedur `CenterCells()` zentriert alle Zellenwerte in der Tabelle. Die Eigenschaft `ColAlignment` benötigt eine Spaltennummer als Index. Zeile 23 stellt diesen Index bereit. Die Prozedur `SizeCells()` in Zeile 27 setzt die Zellenbreiten auf 1100 Twips. Dazu wird die Tabelle durchlaufen und dabei auf jede Spalte die `ColWidth`-Eigenschaft angewendet (Zeile 31).

Listing 22.6 zeigt den Code für die Initialisierung der Tabelle. Dabei werden die Zelltitel gesetzt und die Werte in die Tabelle eingetragen. Die zweidimensionale Tabelle ist ein Zwischenspeicher, der in dieser Applikation eigentlich nicht notwendig ist. Der Code soll jedoch demonstrieren, wie einfach es ist, Daten, die in einem mehrdimensionalen Array abgelegt sind, in einem Tabellen-Steurelement anzuzeigen.

Listing 22.6: Initialisierung der Tabelle

```
1: Private Sub Titles()  
2: ' Spaltentitel  
3: ' Normalerweise kommen solche Daten aus einer Datenbank  
4: grdSales.Row = 0 ' Alle Verkäufernamen in Zeile 0  
5: grdSales.Col = 1  
6: grdSales.Text = "Smith"  
7: grdSales.Col = 2  
8: grdSales.Text = "Johnson"  
9: grdSales.Col = 3  
10: grdSales.Text = "Lake"  
11: grdSales.Col = 4  
12: grdSales.Text = "West"  
13: grdSales.Col = 5  
14: grdSales.Text = "Gates"  
15: grdSales.Col = 6  
16: grdSales.Text = "Kirk"  
17: grdSales.Col = 7  
18: grdSales.Text = "Taylor"  
19: ' Now fill products  
20: grdSales.Col = 0 ' Alle Produktnamen in Spalte 0  
21: grdSales.Row = 1  
22: grdSales.Text = "#1 Widget"  
23: grdSales.Row = 2  
24: grdSales.Text = "#2 Widget"  
25: grdSales.Row = 3  
26: grdSales.Text = "Long tube"  
27: grdSales.Row = 4  
28: grdSales.Text = "Short tube"  
29: grdSales.Row = 5  
30: grdSales.Text = "Metal liner"  
31: grdSales.Row = 5  
32: grdSales.Text = "Wood liner"  
33: grdSales.Row = 6  
34: grdSales.Text = "Plastic liner"  
35: grdSales.Row = 7  
36: grdSales.Text = "Rubber liner"
```



```

37: grdSales.Row = 8
38: grdSales.Text = "Basket"
39: grdSales.Row = 9
40: grdSales.Text = "Bolt 3C"
41: grdSales.Row = 10
42: grdSales.Text = "Bolt 5A"
43: grdSales.Row = 11
44: grdSales.Text = "Hex nut 3C"
45: grdSales.Row = 12
46: grdSales.Text = "Hex nut 5A"
47: grdSales.Row = 13
48: grdSales.Text = "#12 Nail"
49: grdSales.Row = 14
50: grdSales.Text = "#15 Nail"
51: grdSales.Row = 15
52: grdSales.Text = "#16 Nail"
53: grdSales.Row = 16
54: grdSales.Text = "Eye bolt #4"
55: grdSales.Row = 17
56: grdSales.Text = "Eye bolt #6"
57: grdSales.Row = 18
58: grdSales.Text = "Eye bolt #8"
59: grdSales.Row = 19
60: grdSales.Text = "Gasket"
61: End Sub
62:
63: Private Sub FillCells()
64: ' Alle 160 Zellen mit Werten füllen, die einfach
65: ' aus den Zeilen- und Spaltenwerten berechnet
66: ' werden. Diese Daten haben keine Bedeutung, sondern
67: ' sollen einfach nur Inhalt für die Tabelle darstellen
68: '
69: ' Normalerweise kommen solche Daten aus einer Datenbank
70: '
71: ' Deklaration einer 20x7-Tabelle, die mit dem Tabellen-
72: ' Steuerelement übereinstimmt. Die Indizes beginnen bei
73: ' 0, weil das Tabellen-Steuerelement sie so verarbeitet.
74: Dim curData(19, 7) As Currency
75: Dim Row As Integer
76: Dim Column As Integer
77: '
78: ' Tabelle mit Daten füllen
79: For Row = 1 To 19
80: For Column = 1 To 7
81: curData(Row, Column) = ((Row + Column) / Row)
82: Next Column
83: Next Row
84: ' Tabelleninhalt in das Tabellen-Steuerelement schreiben
85: For Row = 1 To 19
86: For Column = 1 To 7

```

```

87: grdSales.Row = Row
88: grdSales.Col = Column
89: grdSales.Text = Format(curData(Row, Column), "###.00")
90: Next Column
91: Next Row
92: End Sub

```

Die Zuweisungen in den Zeilen 4 bis 60 schreiben einfach nur die Verkäufernamen in die oberste Zeile und die Produktnamen in die linke Spalte. Bei der Arbeit mit einem Tabellen-Steuerelement müssen eigentlich nur die Überschriften initialisiert werden. Häufig stammt diese Information aus einer Datenbank. In diesem einfachen Beispiel haben wir Zuweisungen verwendet.

Die Zeilen 79 bis 83 füllen die Tabelle mit Daten. Die Werte werden aus den Zeilen- und Spaltennummern berechnet. Sie sollen einfach nur einen Tabelleninhalt darstellen. Die Tabelle dient als Zwischenspeicher für die Tabellenwerte und ist für diese Applikation nicht zwingend erforderlich. Anhand des Beispiels erkennen Sie jedoch, wie eine Tabelle in einer verschachtelten Schleife mit Werten gefüllt wird. Wie in diesem Bonuskapitel bereits erwähnt, sind verschachtelte Schleifen die beste Methode, mehrdimensionale Arrays zu durchlaufen.



Zeile 89 verwendet die interne Funktion `Format()`, um die Daten so zu formatieren, daß sie im Währungsformat erscheinen. Später werden Sie lernen, wie Sie mit einer Eigenschaft des Tabellen-Steuerelements das Format der Zellen festlegen.

Der restliche Code ist dafür zuständig, auf die Aktionen des Benutzers zu reagieren.

Listing 22.7: Die Tabellenwerte können vom Benutzer verändert werden

```

1: Private Sub hscDecrease_Change()
2: ' Titel der Schaltfläche ändern
3: cmdDecrease.Caption = "&Decrease by" & Str(hscDecrease.Value) & "%"
4: End Sub
5:
6: Private Sub hscIncrease_Change()
7: ' Titel der Schaltfläche ändern
8: cmdIncrease.Caption = "&Increase by" & Str(hscIncrease.Value) & "%"
9: End Sub
10:
11: Private Sub cmdIncrease_Click()
12: ' Werte der selektierten Zellen erhöhen,
13: ' wenn die Bildlaufleiste einen höheren Wert spezifiziert
14: Dim SelRows As Integer
15: Dim SelCols As Integer
16: Dim SelStartRow As Integer
17: Dim SelStartCol As Integer
18: Dim RowBeg As Integer
19: Dim ColBeg As Integer
20:
21: If (grdSales.HighLight) Then ' Falls selektiert...
22: ' Zellenwerte speichern
23: SelStartRow = grdSales.RowSel
24: SelStartCol = grdSales.ColSel
25: RowBeg = grdSales.Row

```

```

26: ColBeg = grdSales.Col
27: ' Alle selektierten Zellen durchlaufen
28: For SelRows = RowBeg To SelStartRow
29: For SelCols = ColBeg To SelStartCol
30: grdSales.Row = SelRows
31: grdSales.Col = SelCols
32: ' Zellwert entsprechend erhöhen
33: grdSales.Text = grdSales.Text + (hscIncrease.Value / 100 * _
grdSales.Text)
34: grdSales.Text = Format(grdSales.Text, "####.00")
35: Next SelCols
36: Next SelRows
37: ' Markierung entfernen
38: grdSales.Row = RowBeg
39: grdSales.Col = ColBeg
40: grdSales.RowSel = SelStartRow
41: grdSales.ColSel = SelStartCol
42: End If
43: End Sub
44:
45: Private Sub cmdDecrease_Click()
46: ' Selektierte Zellewerte entsprechend dem
47: ' Bildlaufleistenwert verringern
48: Dim SelRows As Integer
49: Dim SelCols As Integer
50: Dim SelStartRow As Integer
51: Dim SelStartCol As Integer
52: Dim RowBeg As Integer
53: Dim ColBeg As Integer
54:
55: If (grdSales.HighLight) Then ' Falls selektiert...
56: ' Selektierte Zellwerte speichern
57: SelStartRow = grdSales.RowSel
58: SelStartCol = grdSales.ColSel
59: RowBeg = grdSales.Row
60: ColBeg = grdSales.Col
61: ' Selektierte Zellen durchlaufen
62: For SelRows = RowBeg To SelStartRow
63: For SelCols = ColBeg To SelStartCol
64: grdSales.Row = SelRows
65: grdSales.Col = SelCols
66: ' Zellwert entsprechend verringern
67: grdSales.Text = grdSales.Text - (hscDecrease.Value / 100 * _
grdSales.Text)
68: grdSales.Text = Format(grdSales.Text, "####.00")
69: Next SelCols
70: Next SelRows
71: ' Markierung entfernen
72: grdSales.Row = RowBeg
73: grdSales.Col = ColBeg

```

```

74: grdSales.RowSel = SelStartRow
75: grdSales.ColSel = SelStartCol
76: End If
77: End Sub
78:
79: Private Sub cmdExit_Click()
80: ' Terminate application
81: End
82: End Sub

```

Die Funktionen `hscDecrease_Change()` und `hscIncrease_Change()` ändern einfach nur die `Caption`-Eigenschaft für die Schaltflächen, wenn der Benutzer die Bildlaufleiste anklickt. Die Bildlaufleisten bestimmen, wie ein Zellwert erhöht oder verringert werden soll.

Der restliche Code, beginnend in Zeile 11, ist verhältnismäßig umfangreich. Die beiden Prozeduren, `cmdIncreaseClick()` und `cmdDecreaseClick()`, sind jedoch grundsätzlich gleich, außer daß die eine die Zellenwerte erhöht, die andere sie verringert.



Wenn der Benutzer nur eine einzelne Zelle selektiert, bevor er auf eine Schaltfläche klickt, ändert sich bei Ausführung der `Click`-Prozedur nur der Wert dieser Zelle.

In der Analyse werden wir nur eine der beiden großen Prozeduren betrachten, `cmdIncrease_Click()`, weil die andere sehr ähnlich ist. Zeile 21 stellt sicher, daß der Benutzer mindestens eine Zelle selektiert hat, bevor die Zellwerte verändert werden. Ist keine Zelle selektiert, macht die Prozedur überhaupt nichts. Die Zeilen 23 bis 26 speichern den rechteckigen Bereich der selektierten Zellen. Dieser Bereich wird durch die Zeile und Spalte der aktuellen Zelle (das ist die Zelle in der oberen linken Ecke des ausgewählten Zellbereichs) und die letzte ausgewählte Zelle bestimmt.

Die Zeilen 28 und 29 setzen eine verschachtelte `For`-Schleife ein, um die Zellen zu durchlaufen. Die Zellen werden entsprechend dem mit der Bildlaufleiste festgelegten Wert erhöht. Die Zeilen 38 bis 41 heben die Markierung für die selektierten Zellen auf, denn wenn Sie die `Row`- und `Col`-Werte für die Tabelle ändern, wird jede Auswahl aufgehoben. Die Zeilen 40 und 41 setzen die Auswahl mit `RowSel` und `ColSel` wieder auf den Ausgangswert zurück.

Ausführung der Applikation

Nachdem Sie die Applikation fertiggestellt haben, führen Sie sie aus. Zunächst werden Sie die Bedeutung der Bildlaufleisten und ihre Verbindung mit den Schaltflächen nicht erkennen, aber nachdem Sie mit dem Programm gearbeitet haben, werden Sie schnell verstehen, wie es funktioniert:

1. Klicken Sie in die linke Bildlaufleiste, um die Position zu verringern (schieben Sie den Schieberegler nach links). Die `Caption`-Eigenschaft der darunterliegenden Schaltfläche wird geändert.
2. Klicken Sie in eine Zelle, um sie in der Tabelle zu markieren.
3. Klicken Sie auf die rechte Schaltfläche. Der Wert der Zelle wird um 15% verringert.
4. Wählen Sie einen Zellbereich aus, beispielsweise wie in Abbildung 22.10 gezeigt.

Verkaufzahlendiagramm

	Smith	Johnson	Lake	West	
#1 Widget	2,00	3,00	4,00	5,00	
#2 Widget	1,50	2,00	2,50	3,00	
Long tube	1,33	1,67	2,00	2,33	
Short tube	1,25	1,50	1,75	2,00	
Wood liner	1,20	1,40	1,60	1,80	
Plastic liner	1,17	1,33	1,50	1,67	
Rubber liner	1,14	1,29	1,43	1,57	
Basket	1,13	1,25	1,38	1,50	
Bolt 3C	1,11	1,22	1,33	1,44	
Bolt 5A	1,10	1,20	1,30	1,40	

◀ 1% 15% ▶

◀ 1% 15% ▶

Steigerung um 15%

Verringerung um 15%

Beenden

Abbildung 22.10:

Die Werte aller selektierten Zellen können erhöht oder verringert werden.

5. Klicken Sie auf die linke Schaltfläche, um die selektierten Zellen um den Wert zu ändern, der auf der Schaltfläche angezeigt ist. Beachten Sie, daß die Auswahl bestehen bleibt, falls Sie die Zellwerte erhöhen möchten.

6. Selektieren Sie einen anderen Bereich, um die Werte zu verringern.

7. Klicken Sie auf die Schaltfläche Beenden, um das Programm zu beenden.

Die FormatString-Eigenschaft

Statt die Daten bei der Plazierung in der Tabelle mit der internen Funktion `Format()` zu formatieren, können Sie auch einzelne Zellen oder Zellbereiche so formatieren, daß alle später darin abgelegten Daten ein bestimmtes Format aufweisen. Der `FormatString`-Eigenschaft des Tabellen-Steuerelement wird ein String-Wert zugewiesen, der festlegt, wie der Zellinhalt dargestellt werden soll. `FormatString` erscheint zunächst sehr kompliziert, aber nachdem Sie es verstanden haben, werden Sie damit Ihre Zellen sehr effektiv formatieren können.



Der `FormatString`-Wert kann auch im Dialogfeld **Benutzerdefiniert** eingegeben werden. Geben Sie die `FormatString`-Eigenschaft auf der Registerkarte **Stil** in das Textfeld **Format** ein. Darüber hinaus können Sie den `FormatString` -Wert auch direkt zuweisen, wie in diesem Abschnitt noch gezeigt wird.

Eine der besten Eigenschaften von `FormatString` ist, daß Sie damit die Zeilen- und Spaltentitel setzen können und keine weiteren Zuweisungen erforderlich sind.

Die `FormatString`-Eigenschaft kann den folgenden Inhalt aufnehmen:

- Das Pipe-Symbol (`|`) trennt die einzelnen Segmente in `FormatString`.
- Jedes Segment zwischen den Pipes definiert eine Spalte.
- Text in den Segmenten stellt die Überschriftswerte für Zeile 0 dar.
- Text in den Segmenten definiert die Breite einer Spalte.
- Das Zeichen `<` richtet den Text linksbündig aus, `^` zentriert den Text, und `>` richtet den Text rechtsbündig aus.
- Text, der einem Semikolon (`;`) folgt, bestimmt die Breite der Titel für Spalte 0.
- Die Anzahl der Segmente bestimmt die Mindestanzahl der Tabellenspalten.

Die `FormatString`-Eigenschaft definiert Zeilen und Spalten sowie ihre Breite und Titel, aber Sie müssen die Größe des Tabellen-Steuerelements auf Ihrem Formular festlegen, damit der Benutzer alle Zeilen und Spalten sieht. `FormatString` führt keinerlei Größenänderungen für die Tabelle aus.

Das folgende Beispiel erzeugt die in Registerkarte 22.11 gezeigte Tabelle:

```
Dim strCol As String
Dim strRow As String
' Spaltentitel der Tabelle setzen
' Das Symbol ^ bewirkt, daß alle folgenden Werte
' zentriert erscheinen. Das Pipe-Symbol definiert
' die einzelnen Spalten.
strCol = "^|Smith|Johnson|Lake|West|Gates|Kirk|Taylor"
' Zeilentitel der Tabelle setzen. Dafür wird ein String
' definiert. An den Semikolons erkennt Visual Basic, daß es
' sich um Zeilentitel handelt.
strRow = ";|#1 Widget|#2 Widget|Long tube|Short tube|"
strRow = strRow & "Metal liner|Wood liner|Plastic liner|"
strRow = strRow & "Rubber liner|Basket|Bolt3C|Bolt 5A|"
strRow = strRow & "Hex nut 3C|Hex nut 5A|#12 Nail|#15 Nail|"
strRow = strRow & "#16 Nail|Eye bolt #4|Eye bolt #6|"
strRow = strRow & "Eye bolt #8|Gasket"
' Tabelle durch Zuweisung der beiden Strings formatieren
grdSales.FormatString = strCol & strRow
```

Form1

	Smith	Johnson	Lake	West	Gates	Kirk	Taylor
#1 Widget	2,00	3,00	4,00	5,00	6,00	7,00	8,00
#2 Widget	1,50	2,00	2,50	3,00	3,50	4,00	4,50
Long tube	1,33	1,67	2,00	2,33	2,67	3,00	3,33
Short tube	1,25	1,50	1,75	2,00	2,25	2,50	2,75
Wood liner	1,20	1,40	1,60	1,80	2,00	2,20	2,40
Plastic liner	1,17	1,33	1,50	1,67	1,83	2,00	2,17
Rubber liner	1,14	1,29	1,43	1,57	1,71	1,86	2,00
Basket	1,13	1,25	1,38	1,50	1,63	1,75	1,88
Bolt 3C	1,11	1,22	1,33	1,44	1,56	1,67	1,78
Bolt 5A	1,10	1,20	1,30	1,40	1,50	1,60	1,70
Hex nut 3C	1,09	1,18	1,27	1,36	1,45	1,55	1,64
Hex nut 5A	1,08	1,17	1,25	1,33	1,42	1,50	1,58
#12 Nail	1,08	1,15	1,23	1,31	1,38	1,46	1,54
#15 Nail	1,07	1,14	1,21	1,29	1,36	1,43	1,50
#16 Nail	1,07	1,13	1,20	1,27	1,33	1,40	1,47
Eye bolt #4	1,06	1,13	1,19	1,25	1,31	1,38	1,44
Eye bolt #6	1,06	1,12	1,18	1,24	1,29	1,35	1,41
Eye bolt #8	1,06	1,11	1,17	1,22	1,28	1,33	1,39
Gasket	1,05	1,11	1,16	1,21	1,26	1,32	1,37



Abbildung 22.11:
Mit Hilfe der `FormatString`-Eigenschaft richten Sie Zeilen- und Spaltentitel ein.

Beachten Sie, daß diese Tabelle grundsätzlich so aussieht wie die, die Sie im vorigen Beispiel erzeugt haben. Die Tabelle in Abbildung 22.11 verwendet jedoch Spaltenbreiten, die den Spaltentiteln entsprechend.



Tip `FormatString` könnte den Code aus dem vorigen Beispiel um viele Zeilen verkürzen. Listing 22.6 enthält viele Zuweisungen für Spalten- und Zeilentitel. Die `FormatString`-Eigenschaft erledigt die Initialisierung dieser Titel effizienter.

Bilder im Tabellen-Steuerelement

Das Tabellen-Steuerelement ermöglicht Ihnen, Bitmaps und Icons in den Zellen abzulegen. Dazu verwendet es die Eigenschaft `CellPicture`. Um ein Bild in einer Zelle anzulegen, gehen Sie wie folgt vor:

1. Weisen Sie die `Row`- und `Col`-Eigenschaften der Zelle zu, in der das Bild erscheinen soll.
2. Weisen Sie den `ColWidth`- und `ColHeight`-Eigenschaften die Höhe des Bildes zu, wenn Sie

bestimmen möchten, wie groß es sein soll.

3. Weisen Sie der `CellPicture`-Eigenschaft das Bild zu.



Bilder können nicht zur Entwurfszeit zugewiesen werden.



Wenn Sie Ihr Bild in einem Anzeigefeld auf dem Formular anlegen, dessen `Visible`-Eigenschaft Sie auf `False` setzen, können Sie der Zelle auch das Anzeigefeld zuweisen, statt in `LoadPicture()` den genauen Dateinamen für das Zellenbild angeben zu müssen. Durch Verwendung des Anzeigefelds ist es einfacher, die Breite und die Höhe für das Bild zu setzen.

Angenommen, Sie möchten in der oberen linken Zelle des Tabellen-Steuerelements ein Bild anzeigen. Diese Zelle enthält normalerweise keinen Titeltext. Der folgende Code legt das Bild aus einem Anzeigefeld in der Zelle an.

```
' Zelle oben links zur aktuellen Zelle machen
grdSales.Row = 0
grdSales.Col = 0
' Breite und Höhe der Zelle an das Bild anpassen
grdSales.ColWidth(0) = Image1.Width
grdSales.RowHeight(0) = Image1.Height
' Der Zelle wird das Anzeigefeld zugewiesen
Set grdSales.CellPicture = Image1.Picture
```

Beachten Sie, daß das Bild der Zelle mit einer `Set`-Anweisung zugewiesen wird. Eine normale Zuweisung wäre hier nicht ausreichend, weil damit nur der Pfadname zugewiesen würde, nicht das eigentliche Bild. Abbildung 22.12 zeigt ein Beispiel für ein Bild in einem Tabellen-Steuerelement.

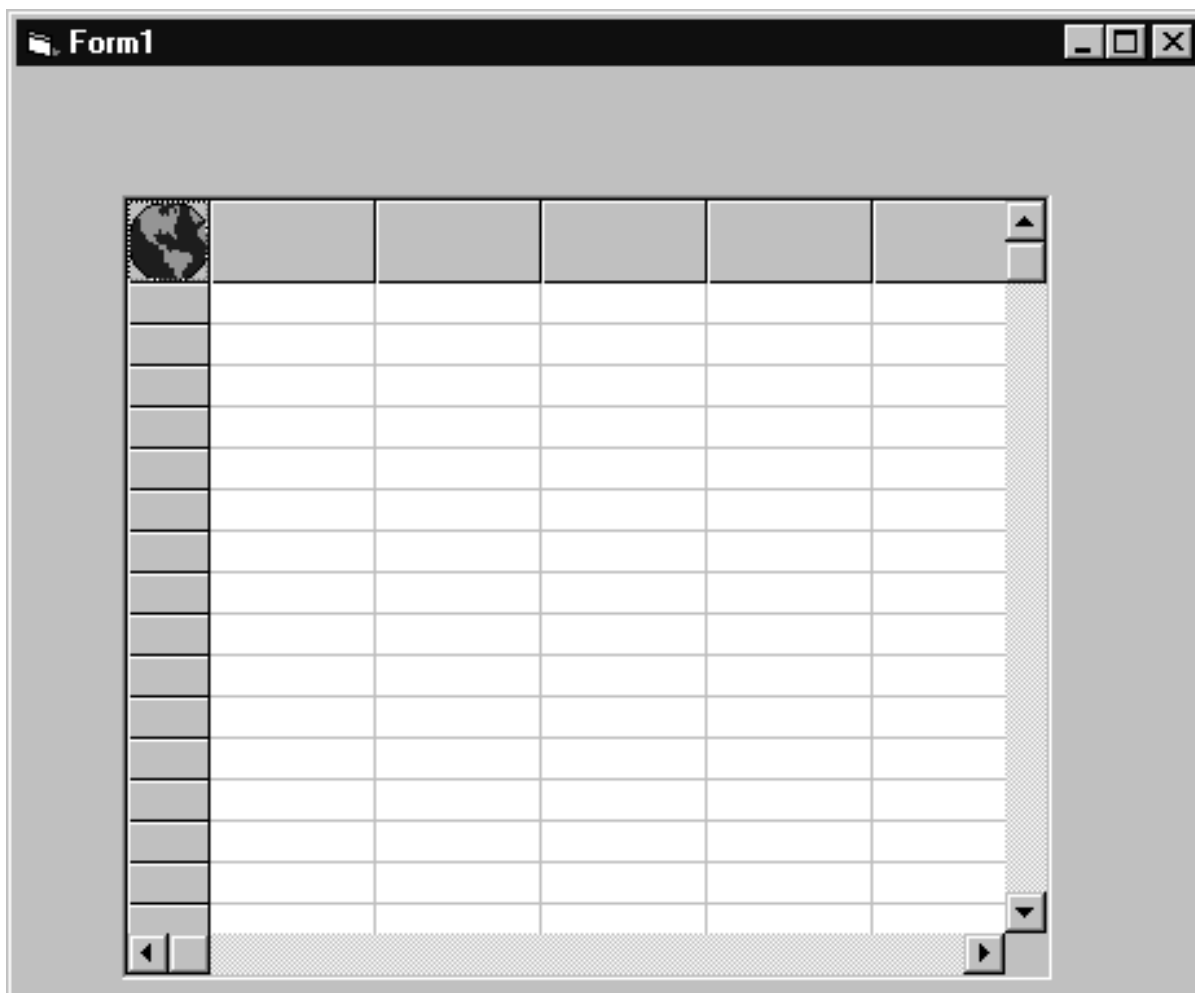


Abbildung 22.12:
Bilder verschönern ein Tabellen-Steuerelement.

Zusammenfassung

In diesem Bonuskapitel wurde erklärt, wie mehrdimensionale Arrays angelegt werden, um dort Daten zu verwalten. Das gebräuchlichste mehrdimensionale Array, die zweidimensionale Tabelle, wird in den verschiedensten Situationen eingesetzt, um Zeilen und Spalten mit Daten zu verarbeiten, beispielsweise eine Inventartabelle. Visual Basic erlaubt die Verwendung von bis zu 60 Dimensionen, aber in der Regel sind drei oder vier Dimensionen völlig ausreichend.

Eine der einfachsten Methoden, Tabellendaten anzuzeigen, stellt das Tabellen-Steuerelement dar. Es gibt unterschiedliche Tabellen-Steuerelemente, die Sie im Komponenten-Dialogfeld laden können. `FlexGrid` ist eines der gebräuchlichsten Beispiele für ein Tabellen-Steuerelement. In diesem Bonuskapitel wurde erklärt, wie es eingerichtet und verwaltet wird.

Fragen & Antworten

Frage:

Wie erkenne ich, wofür meine Indizes stehen, wenn ich ein mehrdimensionales Array mit vier oder fünf Dimensionen deklariere?

Antwort:

Jeder Index stellt das dar, wofür Sie ihn angelegt haben. In einer zweidimensionalen Tabelle beispielsweise muß der erste Index nicht unbedingt die Zeilen und der zweite die Spalten darstelle; Sie

können die beiden Indizes auch in umgekehrter Reihenfolge verwenden. Allerdings ist es sehr gebräuchlich, den rechten Index für die Spalten, den mittleren für die Zeilen und den ersten für die Tabellenschichten (wie beispielsweise bei einem Würfel aus einzelnen Blöcken) zu verwenden. Diesen Standard sollten Sie bei Ihrer Arbeit berücksichtigen. Wenn irgendwann jemand Ihren Code lesen soll, vereinfachen Sie ihm diese Aufgabe ganz wesentlich, wenn Sie die Tabellenwerte nach dem gängigen Standard setzen.

Frage:

Wie oft verwendet man mehrdimensionale Arrays mit mehr als drei oder vier Dimensionen?

Antwort:

So gut wie nie.

Frage:

Kann ich mit `FormatString` Daten in mein Tabellen-Steuerelement einfügen?

Antwort:

Nein. `FormatString` dient nur dazu, Ihrem Tabellen-Steuerelement Zeilen- und Spaltentitel hinzuzufügen. Die Datenwerte werden auf andere Weise in die Tabelle eingetragen.

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Richtig/Falsch. Alle Elemente in einer Zeile eines mehrdimensionalen Arrays müssen denselben Datentyp haben, aber die verschiedenen Zeilen können unterschiedliche Datentypen aufnehmen.

2. Welcher Index steht normalerweise für die Zeilen:

`Dim sngArray(8, 9, 10)`

3. Betrachten Sie die folgende Tabelle mit Integerwerten (`intAra`). Welche Werte repräsentieren die folgenden Elemente, vorausgesetzt, im Programm wird die Anweisung `Option Base 1` ausgeführt?

a. `intAra(1, 1)`

b. `intAra(3, 2)`

c. `intAra(2, 3)`

4 1 3 5 9

10 2 12 1 6

25 43 2 91 8

14 7 28 71 14

4. Richtig/Falsch. Visual Basic unterstützt bis zu 60 Dimensionen.

5. Richtig/Falsch. Mit der `Array()`-Funktion kann ein mehrdimensionales Array in einer einzigen Anweisung initialisiert werden.

6. Mit welchem Steuerelement werden Tabellendaten am besten dargestellt?

7. Wofür werden die festen Zeilen und Spalten in einem Tabellen-Steuerelement verwendet?

8. Wie weisen Sie den Zellen eines Tabellen-Steuerelements eine Tabelle zu?
9. Richtig/Falsch. Um einer Zelle ein Bild zuzuweisen, setzen Sie zur Entwurfszeit die `CellPicture`-Eigenschaft, so wie für alle anderen Steuerelemente.
10. Was ist für die Einrichtung von Tabellentiteln effizienter? Die Verwendung von `FormatString` oder die wiederholte Zuweisung?

Übungen

1. Berechnen Sie die Anzahl der Elemente, die durch die folgenden Anweisungen reserviert werden:

`Option Base 1`

`Dim intAra(4, 7) As Integer`

2. Wie viele Elemente werden mit der folgenden Anweisung reserviert, wenn Sie im Deklarationsmodul der Prozedur die Anweisung `Option Base` nicht angeben?

`Dim intAra(4, 7) As Integer`

3. Ändern Sie die in diesem Kapitel vorgestellte Verkäufer-Applikation so ab, daß die Tabellenwerte gespeichert werden, bevor das Programm beendet wird.

Woche 3

Tag 23

Das Windows-API

Dieses Bonuskapitel beschreibt den Zugriff auf interne Windows-Routinen. Visual Basic bietet zwar alles, was Sie brauchen, aber für einige Applikationen sind Funktionen erforderlich, die relativ kompliziert zu programmieren sind. Glücklicherweise stellt Windows diese Routinen für Ihre Visual-Basic-Applikation bereit - die Sie beispielsweise bereits in C oder C++ geschrieben und in DLLs abgelegt haben.

Mit Hilfe dieser Windows-Routinen erweitern Sie die Leistungsfähigkeit von Visual Basic und ermöglichen ihm die Ausführung von Funktionen, die normalerweise nur Windows ausführen darf. Wenn Ihre Applikation beispielsweise den Cursor in einem Fenster verwalten soll, gibt es dafür bereits eine interne Routine in Windows. Es ist viel einfacher, diese Routinen zu nutzen, statt sie in Ihrem Visual-Basic-Programm nachzubilden.

Heute lernen Sie die folgenden Dinge kennen

- Windows-API
- Wozu Ihre Applikationen Windows-Routinen brauchen, die es in Visual Basic nicht gibt
- DLLs (Dynamic Link Libraries)
- Die `Declare`-Anweisung
- Neue Prozeduren, die über API-Routinen zur Verfügung stehen
- Probleme bei der Verwendung von API-Routinen vermeiden
- Neue Argument-Datentypen für Windows-API-Routinen
- Hüllen für gebräuchliche API-Routinen anlegen, so daß Ihre Visual-Basic-Applikationen einfacher auf die Prozeduren zugreifen können

Das Windows-API

Das *Windows-API* stellt Routinen bereit, die Sie in Ihren eigenen Programmen nutzen können. Diese API-Routinen verhalten sich ähnlich wie die internen Funktionen von Visual Basic. Wenn Sie den Code einer API-Routine brauchen, ruft Ihr Visual-Basic- Programm diese Routine auf. Nachdem das Windows-API abgearbeitet ist, geht die Steuerung wieder an Ihr Programm zurück.

Es gibt so viele Windows-API-Routinen, daß Sie fast alles, was unter Windows möglich ist, auch von einer Visual-Basic-Applikation aus ausführen können. Beispielsweise können Sie ein Neubooten des Systems erzwingen, indem Sie die entsprechende Windows-API-Routine aufrufen.



Das *Windows-API*, oder *Application Programming Interface*, ist eine Sammlung interner Windows-Routinen, die Sie von Visual Basic aus aufrufen können.

Alle Windows-API-Routinen sind in speziellen Dateien abgelegt, sogenannten DLLs. Es stehen mehrere tausend solcher API-Routinen zur Verfügung. Sie sind in den Ordnern Windows und Windows\System abgelegt. Bei der Installation von Windows werden auch die DLL-Dateien installiert; Sie haben also automatisch Zugriff auf diese Bibliotheken.



Eine *DLL* oder *Dynamic Link Library* ist eine Sammlung API-basierter Routinen, die für Programmiersprachen bereitgestellt werden, die DLLs unterstützen.

Die meisten DLL-Dateien haben die Dateinamenerweiterungen .DLL oder .EXE. Jedes Programm, das Sie schreiben, hat Zugriff auf die Windows-DLLs. Diese DLLs gab es auch in älteren Windows-Versionen (vor Windows 95), aber sie trugen damals den Zusatz »32« nicht in ihren Namen, der sie als 32-Bit-kompatibel kennzeichnet. Windows-Versionen vor Windows 95 waren 16-Bit-kompatibel, d.h. die Daten durchliefen das System in 16-Bit-Gruppen (zwei Bytes). Die Programmierung in einer 32-Bit-Umgebung ist viel flexibler, schneller und effizienter als in der alten 16-Bit-Umgebung.

Hier die drei gebräuchlichsten DLLs:

- **USER32.DLL** - Enthält Funktionen zur Steuerung der Windows-Umgebung und der Benutzeroberfläche, beispielsweise für Cursor, Menüs oder Fenster.
- **GDI32.DLL** - Enthält Funktionen, die Ausgaben auf dem Bildschirm und anderen Geräten kontrollieren.
- **KERNEL32.DLL** - Enthält Funktionen, die die interne Soft- und Hardwareschnittstelle von Windows steuern. Fast alle Routinen für Speicher-, Datei- und Verzeichnisdienste sind in dieser DLL abgelegt.



Die GDI32.DLL-Bibliothek hat ihren Namen vom *Graphics Device Interface*.



Windows ist ein Betriebssystem, das in mehreren Schichten angelegt ist, beginnend mit der Schicht, die dem Benutzer angezeigt wird (die grafische Benutzeroberfläche), bis hin zu der Schicht unmittelbar oberhalb der Hardware, die den Datenfluß zwischen Programmen und Hardware steuert. Diese unterste Betriebssystemschicht wird auch als *Kernel* bezeichnet. Daher der Name KERNEL32.DLL für die DLL mit den Kernel-bezogenen Routinen.

Diese drei Dateien enthalten die meisten API-Routinen oder -Funktionen, die Sie von Ihren Visual-Basic-Applikationen aus aufrufen können. In den Ordnern Windows und Windows\System sehen Sie, daß es auch noch andere DLLs gibt, beispielsweise COMDLG.DLL, MAPI32.DLL, NETAPI32.DLL oder WINMM.DLL. Mit den neuen Funktionen für das Betriebssystem fügt Windows auch neue DLLs ein.

DLLs sind nicht nur in Windows enthalten. Wenn Sie Ihrem System neue Applikationen hinzufügen, bringen diese häufig ihre eigenen DLLs mit. Mit der Zeit werden auf Ihrem System also sehr viele DLLs bereitstehen.



DLLs bieten Ihnen viel mehr Kontrolle über Ihr System, als es unter Visual Basic normalerweise der Fall ist. Wenn Sie eine Windows-API-Funktion aufrufen, arbeiten Sie mit den Interna des Betriebssystems. Damit haben Sie natürlich auch eine gewisse Verantwortung. Die Umgebung und der Debugger von Visual Basic erkennen die normalen internen Funktionen von Visual Basic. API-Funktionen liegen jedoch weit außerhalb des Gültigkeitsbereichs von Visual Basic. Sie können bewirken, daß ein System abstürzt und Sie Ihre gesamten Daten verlieren, wenn Sie einer Windows-API-Funktion auch nur ein falsches Argument übergeben.



Speichern Sie Ihr Projekt häufig, wenn Sie mit API-Funktionen arbeiten. Auf diese Weise verlieren Sie nicht Ihre gesamte Arbeit, wenn durch den Aufruf einer API-Funktion ein Fehler verursacht wird.

Abbildung 23.1 zeigt, wie sich eine API-Routine Ihrer Visual-Basic-Applikation präsentiert. Beachten Sie, daß die API-Routinen aus dem Betriebssystem stammen und nichts mit Visual Basic zu tun haben.

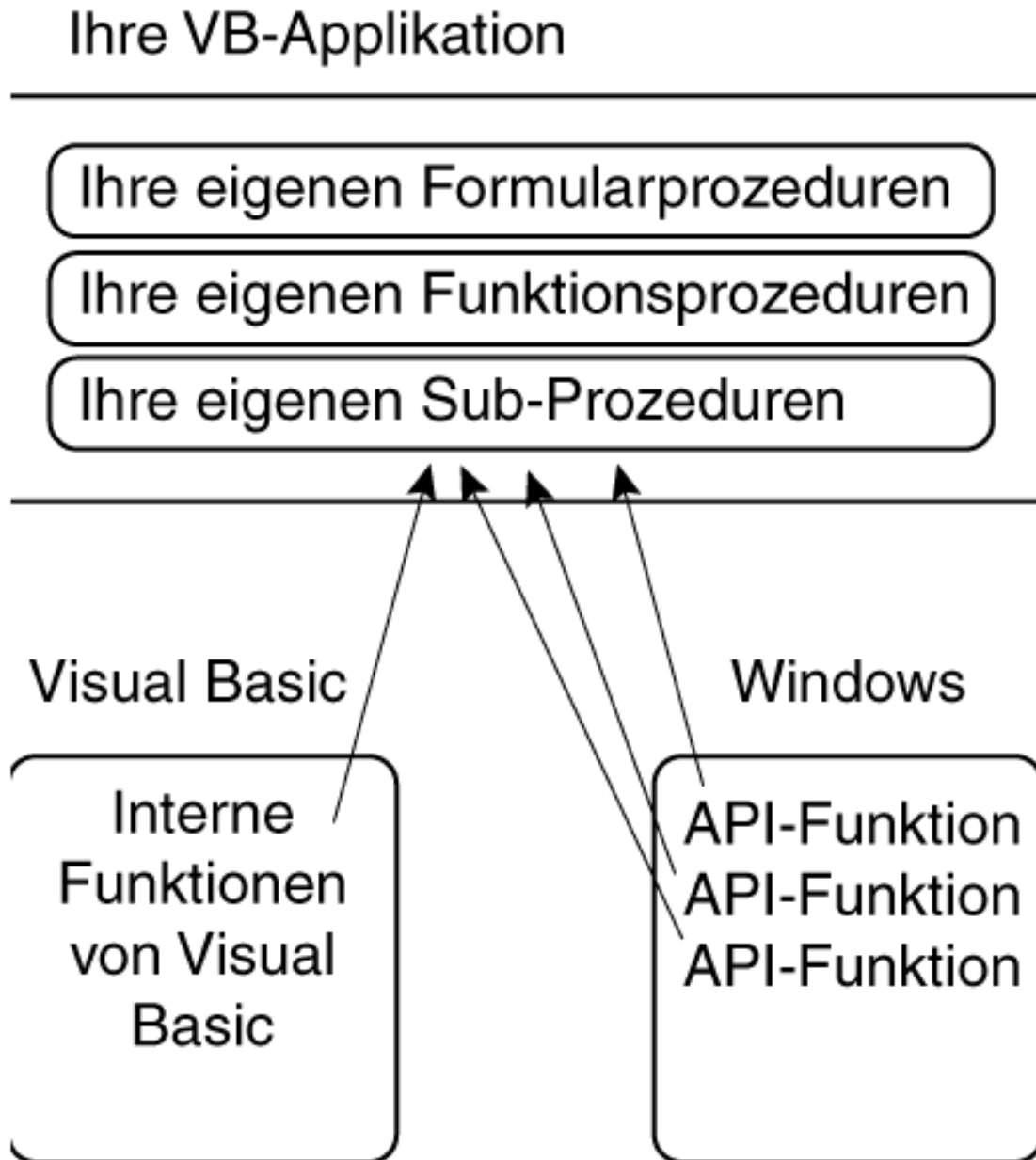


Abbildung 23.1:
API-Routinen befinden sich im Betriebssystem.

Die Natur der DLLs

Der Begriff »Dynamic Link« hat eine besondere Bedeutung für Programmierer. Wenn eine Routine dynamisch zu einem Programm gelinkt wird, dann wird sie erst nach der Programmkompilierung damit verbunden. Die Funktion steht nur zur Laufzeit zur Verfügung. Die Funktionen, die Sie im Codefenster schreiben, sind statisch gelinkt, d.h. sie werden bei der Kompilierung mit dem restlichen Programmcode verbunden. Die DLL-Dateien werden nicht in Ihr Programm aufgenommen. Ihr Programm hat zur Laufzeit Zugriff darauf, aber die EXE-Datei Ihres Programms enthält diese DLL-Routinen nicht.

Dieser Unterschied ist wesentlich, wenn Funktionen verwendet werden sollen, die in solchen DLLs abgelegt sind, weil weder die Bibliothek noch die Funktionen, die Ihre Applikation aufruft, je Teil Ihres Programms sind. Die API-Funktionen haben keinerlei Auswirkung auf die Größe Ihres Programms. Bei der Ausführung werden sie einfach geladen und ausgeführt, bis sie nicht mehr gebraucht werden. Danach gibt das Betriebssystem die dafür verwendeten Ressourcen wieder frei, die nun für neue Routinen genutzt werden können.

Der große Vorteil beim dynamischen Linken ist jedoch nicht die effiziente Verwendung von Ressourcen. Wenn sich Windows ändert, werden die DLLs durch neue DLLs ersetzt. Ihre Applikationen unterstützen also neue Windows-Funktionen, ohne daß Sie jede Applikation, die ein Windows-API verwendet, neu kompilieren müßten. Beispielsweise wurde in Windows 95 das Aussehen der Fenster verändert. Die Icons in der oberen rechten Ecke eines Fensters unterscheiden sich von den in Windows 3.11 verwendeten Icons. Ein Visual-Basic-Programm, das ein Windows-API aufruft, um ein Fenster anzuzeigen, kann in jeder Windows-Umgebung ausgeführt werden. Wenn es unter Windows 3.11 läuft, zeigt es die alten Icons an. Läuft es dagegen unter Windows 95, zeigt es die neuen Icons an, obwohl sich das Programm nicht geändert hat. Häufig müssen Programme, die das Windows-API verwenden, nicht geändert werden, wenn Sie auf eine neue Windows-Version umsteigen.



Windows ist nicht einfach nur ein einziges, riesiges Programm. Es setzt sich aus vielen Programmen zusammen, die sich zum Teil in DLL-Dateien befinden. Windows selbst ist vermutlich der wichtigste Nutzer von DLL-Dateien.



Der Vorteil bei der Verwendung von DLL-Routinen ist, daß mehrere Windows-Programme gleichzeitig darauf zugreifen können. Darüber hinaus liegen allen Ihren Benutzern Standard-DLLs vor. Für die Ausführung Ihres Visual-Basic-Programms ist Windows erforderlich, und damit stehen auch die benötigten DLLs zur Verfügung.

Die Declare-Anweisung

Für den Aufruf von Windows-API-Routinen ist eine spezielle Anweisung erforderlich, `Declare`. Normale interne Visual-Basic-Funktionen benötigen keine `Declare`-Anweisung, weil Visual Basic weiß, wie seine eigenen Funktionen arbeiten und welche Argumente dafür erforderlich sind. API-Routinen dagegen liegen außerhalb des Gültigkeitsbereichs von Visual Basic, deshalb müssen Sie `Declare` verwenden, um ihm Informationen über die aufgerufenen API-Funktionen zur Verfügung zu stellen.

Die `Declare`-Anweisung übernimmt die folgenden Aufgaben:

- Sie gibt an, wo sich die API-Funktion befindet.
- Sie gibt an, welche Argumente die API-Funktion benötigt (Anzahl und Datentyp)
- Sie gibt an, ob die API-Funktion einen Wert zurückgibt.

Die Position der `Declare`-Anweisung hat Einfluß darauf, wie Ihre Visual-Basic-Applikation die Funktion verwaltet. Dabei wird festgelegt, von wo aus Ihre Applikation die Windows-API-Funktion aufrufen darf. Die `Declare`-Anweisung beschreibt einen von zwei Windows-API-Funktionsbereichen, abhängig von den beiden folgenden Bedingungen:

- Wenn Sie die Windows-API-Routine im Formularmodul deklarieren, und zwar außerhalb des allgemeinen Deklarationsbereichs (beispielsweise innerhalb einer Ereignisprozedur), kann nur der Code im Formularmodul die API-Routine aufrufen. Die `Declare`-Anweisung muß die API-Routine mit dem Schlüsselwort `Private` als private Routine kennzeichnen.
- Wenn Sie die Windows-API-Routine im allgemeinen Deklarationsbereich eines Moduls oder Formulars anlegen, steht die API-Routine in der gesamten Applikation zur Verfügung. Dieser

öffentliche Gültigkeitsbereich wird mit dem Schlüsselwort **Public** gekennzeichnet.

Wie alle Prozeduren kann eine Windows-API-Routine entweder eine Sub-Prozedur oder eine Funktion sein, abhängig davon, ob sie einen Wert zurückgibt. Das folgende Format beschreibt die **Declare**-Anweisung als Sub-Prozedur.

```
Declare Sub procName Lib "libName" [Alias "alias"] ([([ByVal] var1 [As _  
dataType][,[ByVal] var2 [As dataType]] ... [, [ByVal] varN [As dataType]])]
```

Die **Declare**-Anweisung teilt Visual Basic den Typ der API-Prozedur mit (Sub-Prozedur oder Funktion), den Namen der Routine, den Namen der DLL-Bibliothek, in der die Routine abgelegt ist (beispielsweise **KERNEL32.DLL**), sowie die Argumente und ihre Datentypen. Wenn es sich bei der Routine um eine Funktion handelt, wird außerdem der Datentyp des Rückgabewerts angegeben.



Das Format der **Declare**-Anweisung sieht sehr kompliziert aus, ist aber viel einfacher, als es den Anschein hat. Sie sollten jedoch sorgfältig darauf achten, daß alle Argumente und ihre Werte übereinstimmen, damit die Routine korrekt ausgeführt werden kann.

Das folgende Format beschreibt die Funktions-Version der **Declare**-Anweisung. Beachten Sie, daß sich dieses Format nur durch das Schlüsselwort **Function** und den Rückgabedatentyp am Ende der Anweisung von der Sub-Prozedur-Version unterscheidet:

```
Declare Function procName Lib "libName" [Alias "alias"] ([([ByVal] var1 _  
[As dataType] _  
[, [ByVal] var2 [As dataType]] ... [, [ByVal] varN [As dataType]])] As _  
dataType
```

Die folgenden Anweisungen verdeutlichen die Verwendung von **Declare** (Sie finden diese Anweisungen im allgemeinen Modul **Module1** im Beispielprojekt **CallDlls.VBP**, das in Visual Basic enthalten ist):

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias  
"GetWindowsDirectoryA" _  
(ByVal lpBuffer As String, ByVal nSize As Long) As Long  
Declare Sub GetSystemInfo Lib "kernel32" (lpSystemInfo As SystemInfo)
```

Beachten Sie, daß einige DLL-Deklarationen sehr umfangreich sind, andere dagegen sehr kurz. Wie auch die internen Funktionen verwenden Windows-DLL-Deklarationen und Aufrufe eine unterschiedliche Anzahl von Argumenten.



Der Name der Windows-API-Routine muß so geschrieben werden, wie vorgegeben. Die Groß-Klein-Schreibung wird dabei berücksichtigt. Bei den API-Funktionen handelt es sich eigentlich um C-Routinen, und Visual Basic muß die korrekte Syntax von C verwenden, damit sie korrekt ausgeführt werden können.

API-Datentypen

Einer der Gründe, warum API-Routinen so schwierig aufzurufen sind, ist, daß Windows etwas andere Datentypen verwendet als Visual Basic. Das Windows-API verwendet zwar auch **String**- und **Long**-Datentypen, aber auch andere, wie beispielsweise **RECT** oder **MSG**. Es kann also kompliziert sein, das richtige Format anzugeben.



Ihre Argumente müssen nicht nur nach Anzahl und Typ mit der entsprechenden API-Argumenteliste übereinstimmen, sondern auch auf die richtige Weise übergeben werden, also als Wert oder als Referenz. (Kapitel 8 erklärt den Unterschied zwischen den beiden Methoden zur Argumentübergabe.) Verwenden Sie gegebenenfalls das Schlüsselwort `ByVal` - andernfalls wird implizit `ByRef` vorausgesetzt. Es ist möglich, daß innerhalb einer API-Routine unterschiedliche Übergabemethoden verwendet werden.

In der `Declare`-Anweisung wird das Schlüsselwort `Alias` genutzt, um einige String-Werte, die unerlaubte Zeichen verwenden, in legale Visual-Basic-Äquivalente umzuwandeln. Das Schlüsselwort `Alias` wandelt außerdem die Namen von Windows-API-Routinen, die nicht erlaubt sind, wie beispielsweise `_lopen` (ein gültiger Name für ein Windows-API, aber nicht für eine Visual-Basic-Prozedur), so um, daß sie dem Namensstandard von Visual Basic entsprechen.

Möglicherweise begegnen Ihnen seltsame Datentypen, die Sie nicht erkennen. Tabelle 23.1 beschreibt einige dieser Datentypen, die Sie für die Arbeit mit API-Routinen brauchen. Dabei werden nur die Datentypen aufgelistet, die sich von den in Visual Basic implementierten Datentypen unterscheiden.

Tabelle 23.1: Spezielle Datentypen in API-Routinen

Datentyp	Beschreibung
ANY	Eine Windows-API-Routine, die unterschiedliche Datentypen verarbeitet, gibt für diese Datentypen ANY an. Alle ANY-Argumente werden als Referenz übergeben, Sie brauchen also das Schlüsselwort <code>ByVal</code> nicht anzugeben.
ATOM	Integer-Daten. Diese Daten werden immer als Wert übergeben und sind in der Deklaration der API-Routine als <code>ByVal argument%</code> oder <code>ByVal argument As Integer</code> spezifiziert.
BOOL	Long-Integer-Daten. Diese Daten werden immer als Wert übergeben und sind in der Deklaration der API-Routine als <code>ByVal argument%</code> oder <code>ByVal argument As Long</code> spezifiziert.
CHAR	Byte-Daten. Diese Daten werden immer als Wert übergeben und sind in der Deklaration der API-Routine als <code>ByVal argument As Byte</code> spezifiziert.
COLOREF	Long-Integer-Daten für die Angabe von Farbwerten. Diese Daten werden immer als Wert übergeben und sind in der Deklaration der API-Routine als <code>ByVal argument%</code> oder <code>ByVal argument As Long</code> spezifiziert.
DWORD	Long-Integer-Daten. Diese Daten werden immer als Wert übergeben und sind in der Deklaration der API-Routine als <code>ByVal argument%</code> oder <code>ByVal argument As Long</code> spezifiziert.
NULL	Long-Integer-Datentypen für nicht initialisierte Werte. Diese Daten sind in der Deklaration der API-Routine als <code>ByVal argument&</code> und <code>ByVal argument As Long</code> spezifiziert.
LPSTR, LPCSTR	Entspricht dem Datentyp String. Diese Daten sind in der Deklaration der API-Routine als <code>ByVal argument\$</code> oder <code>ByVal argument As String</code> spezifiziert.
STRUCTURE	Manchmal begegnen Ihnen seltsame API-Datentypen wie beispielsweise <code>RECT</code> , <code>MSG</code> und <code>UDT</code> . Sie definieren komplexe Datentypen, die sich aus mehreren anderen Datentypen zusammensetzen. Jede STRUCTURE-basierte API-Routine verwendet eine ganz bestimmte Struktur. Sehen Sie in der Argumenteliste der API-Routine nach, wie diese Argumente formatiert sein sollen.

Beachten Sie, daß Tabelle 23.1 nur einige der Datentypen beschreibt, die in den `Declare` -Anweisungen für Windows-API-Routinen vorkommen können. Aber wie erkennen Sie, welche dieser Datentypen Sie für diese speziellen API-Routinen und ihre zahlreichen Argumente verwenden sollen? Im nächsten Abschnitt lernen Sie ein Werkzeug kennen, das in Visual Basic enthalten ist, und mit dessen Hilfe Sie API-Routinen verwalten können.



Wenn für die API-Routine ein String-Datentyp erforderlich ist, sollten Sie einen String übergeben, der mit einer festen Länge definiert ist, und viele Füllzeichen dafür einfügen. Beispielsweise könnten Sie die Länge des längsten zu erwartenden Strings verdoppeln und das String-Argument damit deklarieren, bevor Sie der API-Routine den String übergeben. (Sie brauchen sich keine Gedanken über die Stringlänge zu machen, wenn die Routine den String nicht verändert.)

Der API-Viewer

Windows enthält Tausende von API-Routinen, die Sie aufrufen können. Es ist fast unmöglich, sich das Format von jeder dieser Routinen zu merken. Visual Basic stellt dazu ein spezielles Werkzeug zur Verfügung, den API-Viewer, mit dem Sie das Format von API-Routinen anzeigen können.



Der *API-Viewer* zeigt API-Prozeduren an und gruppiert sie dem Thema nach, so daß Sie schnell die Routinen finden können, die Sie brauchen.

Der API-Viewer ermöglicht Ihnen, API-Routinen und Argumente zu finden und diese Information dann zu kopieren und in Ihr Codefenster einzufügen. Abhängig von Ihrer Visual-Basic-Installation starten Sie den API-Viewer auf eine von zwei Arten.



Die **Kopieren**-Schaltfläche des API-Viewers kopiert die ausgewählten Deklarationsinformationen in die Windows-Zwischenablage. Wenn Sie die Option **Public** oder **Private** markieren, bevor Sie auf **Kopieren** klicken, fügt der API-Viewer das entsprechende Schlüsselwort `Private` oder `Public` in die `Declare`-Anweisung ein, so daß Sie die Deklaration nicht manuell ändern müssen.

Einige Installationen plazieren den API-Viewer im **Start**-Menü. Wählen Sie **Start, Programme, Microsoft Visual Basic 6.0, Microsoft Visual Basic 6.0 Werkzeuge** . Wenn der API-Viewer dort nicht enthalten ist, können Sie ihn aus der Visual- Basic-Umgebung starten. Dazu wählen Sie **Add-Ins, Add-In-Manager**, um das Dialogfeld **Add-In-Manager** anzuzeigen. Doppelklicken Sie auf den Eintrag **API-Viewer**, um Ihrem **Add-Ins**-Menü den API-Viewer hinzuzufügen. Starten Sie das Programm, indem Sie **Add-Ins, API-Viewer** auswählen.

Abbildung 23.2 zeigt das Fenster für den API-Viewer.



Es ist möglich, daß Sie den API-Viewer nicht starten können. Möglicherweise ist er auf Ihrem System nicht installiert. Rufen Sie die Systemsteuerung auf und wählen Sie **Software hinzufügen**. Suchen Sie den Visual- Basic-Eintrag und installieren Sie den API-Viewer aus der Liste. Dazu brauchen Sie vielleicht Ihre Installations-CD.



Der API-Viewer ermittelt seine Informationen aus den Textdateien `Apiload.txt` , `Api32.txt` und `Win32api.txt`, die zusammen mit dem API-Viewer auf Ihrem System installiert wurden.

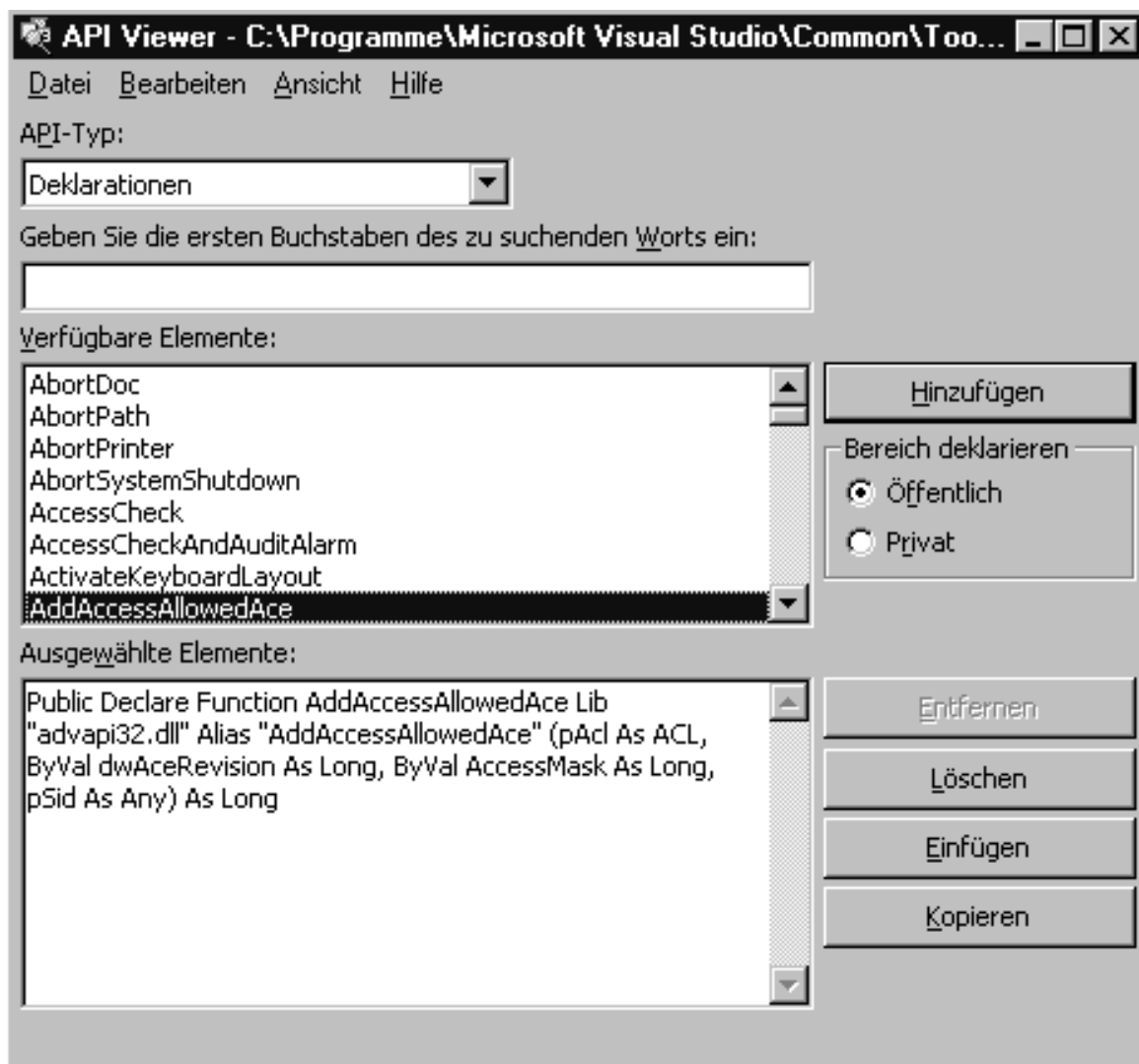


Abbildung 23.2:
Der API-Viewer zeigt Ihnen das Format von API-Routinen.

Die meisten API-Routinen, die Sie brauchen, befinden sich in der Datei Win32.api. Wählen Sie im API-Viewer **Datei, Textdatei laden**, und wählen Sie Win32 . api aus. Der API-Viewer wandelt die Textdatei in eine Access-Datenbank um (mit der Dateinamenerweiterung .MDB), wenn Sie im Meldungsfeld auf die Schaltfläche **Ja** geklickt und dann die Textdatei Win32api . txt ausgewählt haben (siehe Abbildung 23.3). Nachdem Sie die Textdatei in eine Datenbankdatei umgewandelt haben, wird die Menüoption zum Laden als **Datei, Datenbankdatei laden** angezeigt.

Beachten Sie, daß das oberste Listenfeld im Dialogfeld **API-Typ** heißt. Dieses Listenfeld enthält drei Einträge:

- **Konstanten** - Listet alle Konstantennamen auf, die die geladene Windows-API- Datei erkennt.
- **Deklarationen** - Listet alle Deklarationen auf, die in der geladenen API-Datei enthalten sind.
- **Typen** - Listet alle Datentypen auf, die die geladene Windows-API-Datei erkennt.

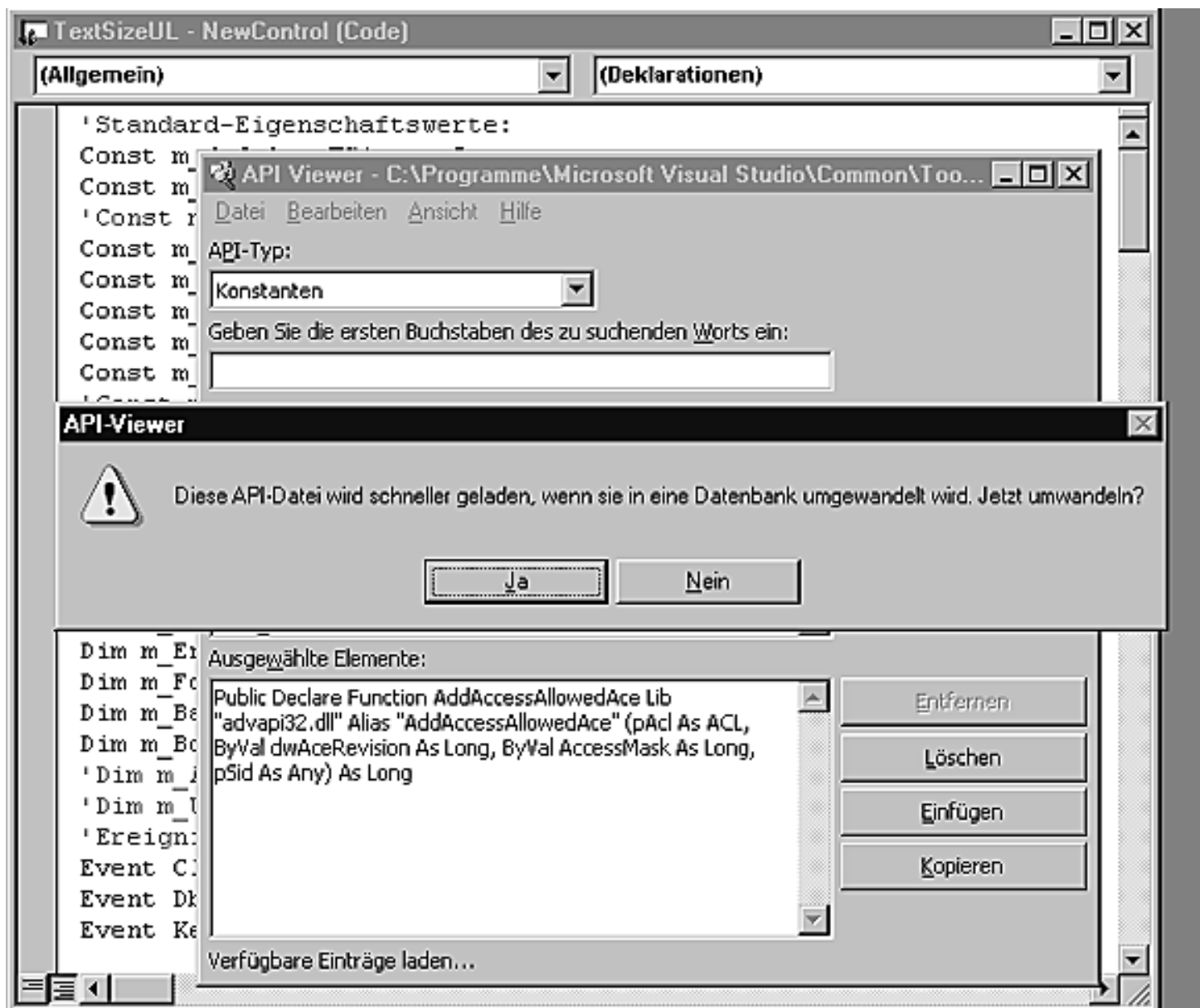


Abbildung 23.3:

Der API-Viewer kann die ihm zugrundeliegenden Daten in eine Datenbank umwandeln, so daß ein schnellerer Zugriff darauf möglich ist.

Im Listenfeld **Verfügbare Elemente** sind alle Windows-API-Routinen für die von Ihnen geladene Datei und die ausgewählten API-Typen aufgeführt. Wenn Sie beispielsweise die `Declare`-Anweisung für die API-Routine `GetWindowsDirectory` suchen, gehen Sie wie folgt vor:

1. Wählen Sie im Listenfeld **API-Typ** den Eintrag **Deklorationen** aus. Im Listenfeld **Verfügbare Elemente** erscheinen zahlreiche `Declare`-Anweisungen.
2. Sie finden eine bestimmte `Declare`-Anweisung, indem Sie die ersten paar Buchstaben in das Textfeld eingeben. Geben Sie `getw` ein. Alle Einträge, die mit diesen Buchstaben beginnen, werden im Listenfeld **Verfügbare Elemente** angezeigt.
3. Blättern Sie zum Eintrag `GetWindowsDirectory`.
4. Doppelklicken Sie auf den Eintrag `GetWindowsDirectory`, um die `Declare`-Anweisung für diese Funktion anzuzeigen (siehe Abbildung 23.4).

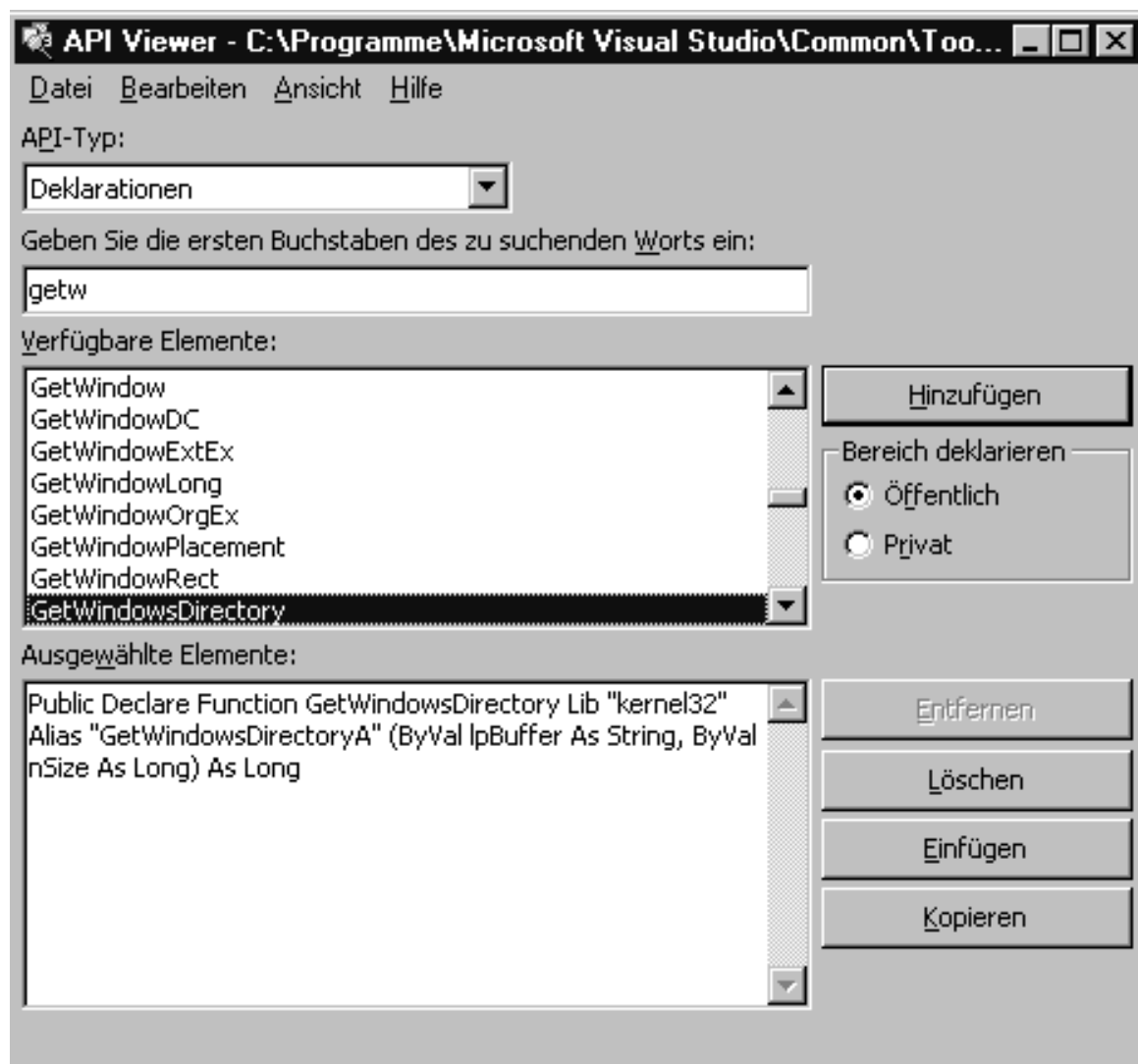


Abbildung 23.4:

Der API-Viewer zeigt die `Declare`-Anweisung an, die Sie für die ausgewählte Anweisung brauchen.

Kopieren Sie die `Declare`-Anweisung und fügen Sie sie in Ihr Codefenster ein.

Aufruf einer einfachen API-Routine

Bevor Sie mehr über das Windows-API erfahren, beobachten Sie die Arbeitsweise einer der API-Routinen. Eine der einfachsten API-Prozeduren ist die `MessageBeep`-Funktion. Die Funktion führt zweierlei Aufgaben aus:

- Wenn Sie der API-Funktion `MessageBeep()` ein positives Argument übergeben, wird über die *Soundkarte* des PC ein Ton ausgegeben.
- Wenn Sie der API-Funktion `MessageBeep()` ein negatives Argument übergeben, wird über den *Lautsprecher* des PCs ein Ton ausgegeben.



Es ist natürlich sehr viel einfacher, diesen Ton mit dem Visual-Basic-Befehl `Beep` auszugeben, aber diese kleine Applikation, die wir hier demonstrieren, soll Ihnen zeigen, wie API-Routinen aufgerufen werden. Viele Visual-Basic-Funktionen und Befehle entsprechen API-Aufrufen, weil Microsoft die erforderlichen Routinen intern aufruft, wenn Sie die entsprechende Funktion oder den betreffenden Befehl ausführen.

Legen Sie ein Projekt an, das in der Mitte des Formularfensters eine Schaltfläche enthält. Setzen Sie den Namen der Schaltfläche auf `cmdBeep` und geben Sie ihr den Titel `&Beep`. Doppelklicken Sie auf die

Schaltfläche, um die Ereignisprozedur `Click` zu öffnen.

Starten Sie Ihren API-Viewer. Laden Sie die Datei `Win32api.txt` (oder die Datenbankdatei, in die Sie sie umgewandelt haben) und setzen Sie den **API-Typ** auf Deklarationen. Im zweiten Listenfeld geben Sie `message` ein. Im Listenfeld **Verfügbare Elemente** erscheinen zahlreiche API-Prozeduren. Die erste Funktion ist `MessageBeep`. Doppelklicken Sie auf diesen Eintrag, um die Prozedurdeklaration im Textfeld **Ausgewählte Elemente** anzuzeigen.

Markieren Sie die gesamte Deklaration, um sie zu kopieren und in Ihr eigenes Codefenster einzufügen. Gehen Sie zurück zum Codefenster Ihrer Visual-Basic-Applikation. Fügen Sie vor der Ereignisprozedur `Click` eine Zeile ein, so daß Sie die Deklaration im allgemeinen Deklarationsbereich einfügen können.

Fügen Sie den Inhalt der Zwischenablage in Ihr Codefenster ein. Ändern Sie die `Public`-Anweisung in `Private`.



Sie ändern hier `Public` in `Private`, weil die Funktionsdeklaration lokal für das Formularmodul ist und nicht in einem allgemeinen Deklarationsbereich gespeichert wird.

Fügen Sie in die Ereignisprozedur `Click` der Schaltfläche den folgenden Code ein:

```
Dim Beeper As Variant  
Beeper = MessageBeep(1)
```

Listing 23.1 zeigt die ganze Prozedur.

Listing 23.1: Mit Hilfe des Windows-API senden Sie Töne an den Lautsprecher

```
1: Private Declare Function MessageBeep Lib "user32" _  
2: (ByVal wType As Long) As Long  
3:  
4: Private Sub cmdBeep_Click()  
5: Dim Beeper As Variant  
6: Beeper = MessageBeep(1)  
7: End Sub
```

Die `Declare`-Anweisung teilt Visual Basic genau mit, wie es die Funktion `MessageBeep()` findet. Wenn es sich dabei um eine interne Visual-Basic-Funktion handeln würde, beispielsweise `Abs()`, bräuhete Visual Basic keine `Declare`-Anweisung, weil es die Funktion problemlos finden würde. Die `MessageBeep()`-Funktion befindet sich jedoch außerhalb der Visual-Basic-Umgebung. Die `Declare`-Anweisung teilt Visual Basic mit, wie es die Funktion ausführen soll und welche Werte ihr übergeben werden. Darüber hinaus erfährt Visual Basic von der `Declare`-Anweisung, daß sich die Funktion in der Datei `USER32.DLL` befindet.

Beachten Sie, daß `MessageBeep()` eine Funktion ist, keine Sub-Prozedur. `MessageBeep()` gibt also einen Wert zurück, einen `Long`. Ihre Applikation braucht diesen Rückgabewert jedoch nicht zu verarbeiten. Die `Variant`-Variable `Beeper` speichert den Wert.

Bei der Programmausführung erscheint die Schaltfläche in der Formularmitte. Wenn Sie auf die Schaltfläche klicken, hören Sie einen Ton Ihrer Soundkarte. Falls Sie nichts hören, ändern Sie das Argument in `-1` ab, um den Ton von Ihrem PC-Lautsprecher zu hören. Auch wenn Sie keine Soundkarte haben, oder wenn die Lautsprecher ausgeschaltet sind, gibt Ihr interner PC-Lautsprecher den Ton aus.



Viele Visual-Basic-Anweisungen bilden Windows-API-Routinen nach. Das ist praktisch, weil Sie dann nicht so viele API-Routinen aufrufen müssen. Darüber hinaus gibt es Hunderte von Windows-API-Routinen, die Sie niemals brauchen werden, weil sie keine sinnvollen Aufgaben für Ihre Programme anbieten.

Aufruf eines anderen API

Jetzt entwickeln Sie eine weitere einfache Applikation, die den Aufruf einer Windows- API-Routine testet. Listing 23.2 zeigt Code, der Informationen über eine Festplatte in ein Bezeichnungsfeld ausgibt.

Listing 23.2: Mit Hilfe des Windows-API ermitteln Sie innerhalb Ihrer Applikation Informationen über die Festplatte

```
1: Private Declare Function GetDriveType Lib "kernel32.dll" Alias _  
  "GetDriveTypeA"(ByVal nDrive As String) As Long  
2:  
3: Private Sub cmdDrive_Click()  
4: Dim lngDriveType As Long  
5:  
6: ' Den betreffenden Laufwerksnamen an die Funktion  
7: ' GetDriveType() übergeben  
8: lngType = GetDriveType("c:\")  
9: '  
10: ' Rückgabewerte auswerten, um den Laufwerkstyp  
11: ' zu erkennen  
12: Select Case lngType  
13: Case 2  
14: txtDrive.Text = "Wechselplatte"  
15: Case 3  
16: txtDrive.Text = "Festplatte"  
17: Case 4  
18: txtDrive.Text = "Netzwerk-Laufwerk"  
19: Case Else  
20: txtDrive.Text = "Unbekannt"  
21: End Select  
22: End Sub
```

Wenn Sie den Code in Listing 23.2 eingeben und testen möchten, suchen Sie die Deklaration von `GetDriveType()` im API-Viewer und kopieren sie in Ihre Zwischenablage. Ersetzen Sie die `Declare`-Anweisung im Codefenster durch den kopierten Text und ändern Sie das Schlüsselwort `Public` in `Private`.

Als nächstes fügen Sie im Formularfenster die Schaltfläche `cmdDrive` ein, ebenso wie ein Textfeld, `txtDrive`. Die Darstellung dieser beiden Steuerelemente ist für dieses Beispiel nicht relevant. Geben Sie in Zeile 8 ein anderes Laufwerk ein, beispielsweise das Diskettenlaufwerk, wenn Sie einen anderen Laufwerkstyp überprüfen möchten. Führen Sie das Programm aus. Im Textfeld erscheint die Laufwerksbeschreibung, wie in Abbildung 23.5 gezeigt.



Abbildung 23.5:
Die API-Routine gibt Informationen über Ihr Laufwerk zurück.

Der Windows-Ordner

Manche Programme greifen auf Systemdateien zu, die im Windows-Verzeichnis abgelegt sind. Visual-Basic-Programmierer verwenden die Windows-API-Routinen, um den Windows-Ordner zu finden (das Verzeichnis, in dem Windows installiert ist). Andere Ordner, die Sie möglicherweise finden müssen, sind System und Temp. Es gibt Windows-API-Routinen, die Ihnen diese Informationen bereitstellen.

	Löschen Sie die Dateien, die während Ihrer Programmausführung im Temp-Ordner abgelegt wurden. Viele Benutzer löschen die Dateien im Temp-Verzeichnis, die länger als zwei oder drei Tage dort liegen, um Plattenspeicher freizugeben. Und Sie möchten sicher nicht, daß Ihre Applikation eines der Programme ist, das alte Dateien im Temp-Ordner ablegt und sie nicht mehr löscht. Außerdem gibt es auch Benutzer, die ihre Temp-Ordner nicht regelmäßig bereinigen - und die manchmal gar nicht wissen, was das ist. Sie sollten nicht einfach Dateien dort hinterlassen. Sie belegen damit Speicherplatz und Ihre Benutzer erkennen gar nicht, daß sie mit jeder Ausführung Ihrer Applikation weniger Speicher zur Verfügung haben.

Die folgenden Schritte beschreiben, wie Sie die kleine Applikation anlegen, in der Informationen über den Windows-Ordner ermittelt werden:

1. Legen Sie eine neue Applikation an.
2. Erstellen Sie die in Tabelle 23.2 beschriebenen Steuerelemente auf dem Formular. Abbildung 23.6 zeigt das Formularfenster.

Verzeichnis-Info

Windows-Verzeichnis-Informationen

Windows-Verzeichnis:

System-Verzeichnis:

Temporäres Verzeichnis:

Beenden

Abbildung 23.6:
Dieses Formularfenster zeigt Informationen über das Windows-Verzeichnis des Benutzers an.

3. Doppelklicken Sie auf das Formular, um die Ereignisprozedur `Form_Load()` zu erzeugen. Fügen Sie vor der Prozedur `Form_Load()` ein paar leere Zeilen ein, um Platz für die `Declare`-Anweisungen zu schaffen, die die Informationen über die Verzeichnisse ermitteln.
4. Starten Sie den API-Viewer. Öffnen Sie die Text- oder Datenbankdatei `Win32.api`.
5. Wählen Sie im Listenfeld **API-Typ** den Eintrag **Deklarationen** aus.
6. Suchen Sie im Listenfeld **Verfügbare Elemente** den Eintrag für `GetWindowsDirectory`. Doppelklicken Sie auf diesen Eintrag, um die Deklaration für die Routine `GetWindowsDirectory()` zu erhalten.
7. Kopieren Sie die Funktion `GetWindowsDirectory()` aus dem API-Viewer und fügen Sie sie oben in Ihr Codefenster ein.
8. Ändern Sie das Schlüsselwort `Public` in `Private`, weil sich die Deklaration in Ihrem Formularmodul befindet, nicht in einem allgemeinen Codemodul.
9. Wiederholen Sie die Schritte 6 bis 8 für die Funktionsdeklarationen `GetSystemDirectory()` und `GetTempPath()`.
10. Geben Sie den Code ein, wie in Listing 23.3 beschrieben.
11. Führen Sie die Applikation aus, um zu prüfen, ob die Verzeichnisinformationen gefunden werden. Abbildung 23.7 zeigt die Applikation bei der Ausführung.

Tabelle 23.2: Legen Sie diese Steuerelemente auf Ihrem Formular an.

Eigenschaft	Eigenschaftswert

Formular Name	frmFolder
Formular Caption	Directory Info
Formular Height	4500
Formular Width	5790
Bezeichnungsfeld Name	lblTitle
Bezeichnungsfeld #1 Alignment	2-Zentriert
Bezeichnungsfeld #1 BorderStyle	1-Fest einfach
Bezeichnungsfeld #1 Caption	Windows-Verzeichnis-Information
Bezeichnungsfeld #1 Font Size	14
Bezeichnungsfeld #1 Font Style	Bold
Bezeichnungsfeld #1 Height	855
Bezeichnungsfeld #1 Left	1080
Bezeichnungsfeld #1 Top	360
Bezeichnungsfeld #1 Width	3375
Bezeichnungsfeld #2 Name	lblWD
Bezeichnungsfeld #2 Alignment	1-Rechts
Bezeichnungsfeld #2 Caption	Windows-Verzeichnis:
Bezeichnungsfeld #2 Height	255
Bezeichnungsfeld #2 Left	720
Bezeichnungsfeld #2 Top	1680
Bezeichnungsfeld #2 Width	1455
Bezeichnungsfeld #3 Name	lblSD
Bezeichnungsfeld #3 Alignment	1-Rechts
Bezeichnungsfeld #3 Caption	System-Verzeichnis:
Bezeichnungsfeld #3 Height	255
Bezeichnungsfeld #3 Left	720
Bezeichnungsfeld #3 Top	2280
Bezeichnungsfeld #3 Width	1455
Bezeichnungsfeld #4 Name	lblTD
Bezeichnungsfeld #4 Alignment	1-Rechts
Bezeichnungsfeld #4 Caption	Temp-Verzeichnis:
Bezeichnungsfeld #4 Height	255
Bezeichnungsfeld #4 Left	720
Bezeichnungsfeld #4 Top	2880
Bezeichnungsfeld #4 Width	1455

Bezeichnungsfeld #5 Name	lblWinD
Bezeichnungsfeld #5 Alignment	0-Links
Bezeichnungsfeld #5 BorderStyle	1-Fest einfach
Bezeichnungsfeld #5 Height	255
Bezeichnungsfeld #5 Left	2400
Bezeichnungsfeld #5 Top	1680
Bezeichnungsfeld #5 Width	2295
Bezeichnungsfeld #6 Name	lblWinS
Bezeichnungsfeld #6 Alignment	0-Links
Bezeichnungsfeld #6 BorderStyle	1-Fest einfach
Bezeichnungsfeld #6 Height	255
Bezeichnungsfeld #6 Left	2400
Bezeichnungsfeld #6 Top	2280
Bezeichnungsfeld #6 Width	2295
Bezeichnungsfeld #7 Name	lblWinT
Bezeichnungsfeld #7 Alignment	0-Links
Bezeichnungsfeld #7 BorderStyle	1-Fest einfach
Bezeichnungsfeld #7 Height	255
Bezeichnungsfeld #7 Left	2400
Bezeichnungsfeld #7 Top	2880
Bezeichnungsfeld #7 Width	2295
Befehlsschaltfläche Name	cmdExit
Befehlsschaltfläche Caption	&Beenden
Befehlsschaltfläche Left	2040
Befehlsschaltfläche Top	3480
Befehlsschaltfläche Width	1215

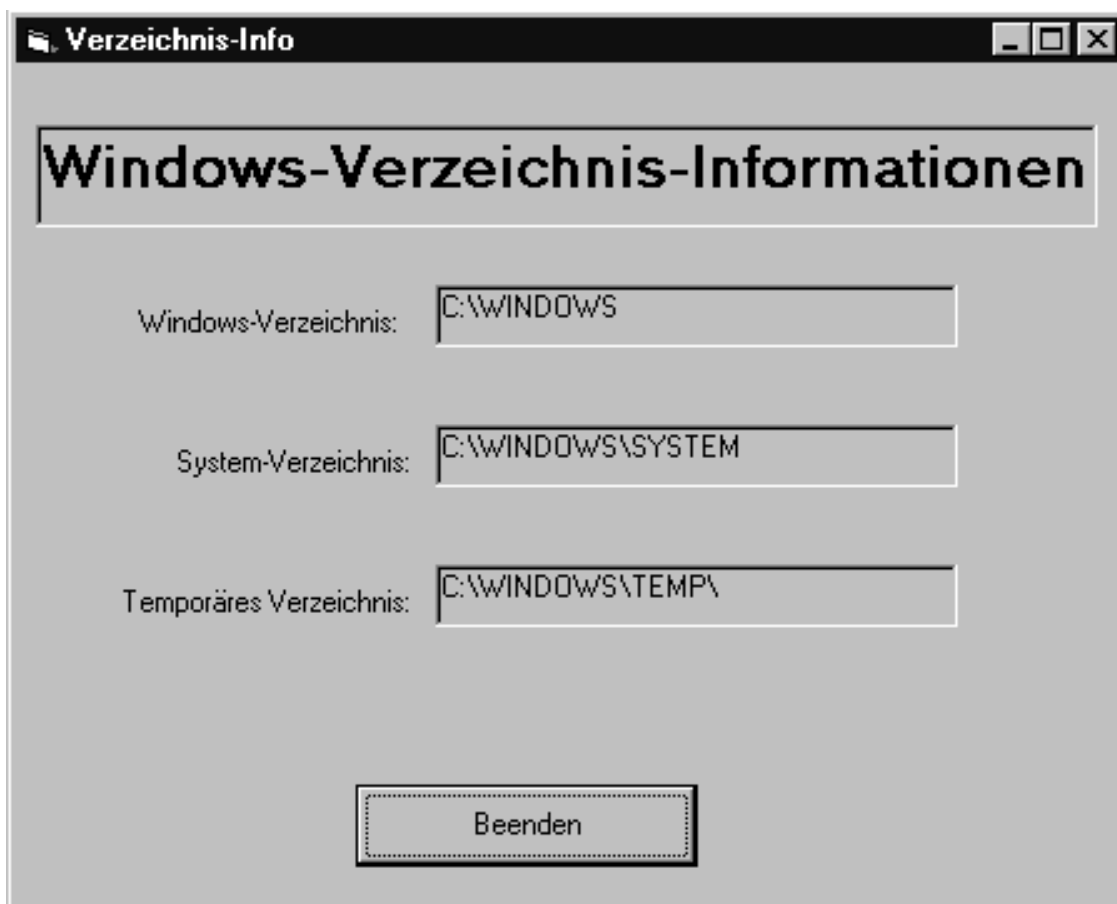


Abbildung 23.7:
Die Verzeichnisinformation ist von der vom Benutzer eingesetzten Maschine abhängig.

Listing 23.3: Die API-Funktionen, die Verzeichnisinformationen ermitteln, machen etwas mehr Aufwand erforderlich als die Routinen, die Sie bisher kennengelernt haben.

```

1: Private Declare Function GetWindowsDirectory Lib "kernel32" Alias _
  "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) _
  As Long
2: Private Declare Function GetSystemDirectory Lib "kernel32" Alias _
  "GetSystemDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) _
  As Long
3: Private Declare Function GetTempPath Lib "kernel32" Alias _
  "GetTempPathA" _
  (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
4: Private Sub Form_Load()
5: ' Bezeichnungsfelder für den System-Ordner initialisieren
6: ' String fester Länge für die ermittelte Information anlegen
7: Dim strFolder As String * 255
8: Dim intLength As Integer
9:
10: '
11: ' Information über das Windows-Verzeichnis ermitteln
12: intLength = GetWindowsDirectory(strFolder, 255)
13: lblWinD.Caption = Left(strFolder, intLength)
14: '
15: ' Information über das System-Verzeichnis ermitteln
16: intLength = GetSystemDirectory(strFolder, 255)
17: lblWinS.Caption = Left(strFolder, intLength)
18: '

```

```

19: ' Information über das Temp-Verzeichnis ermitteln
20: intLength = GetTempPath(255, strFolder)
21: lblWinT.Caption = Left(strFolder, intLength)
22: End Sub
23:
24: Private Sub cmdExit_Click()
25: End
26: End Sub

```

Der Code, mit dem Sie die Ordernamen ermitteln, ist interessant, weil er mehr Arbeit bedingt als der API-basierte Code, den Sie bisher kennengelernt haben. Diese Routinen ermitteln die erforderlichen Verzeichnisinformationen, Sie müssen diese jedoch aus einem langen String extrahieren, den sie zurückgeben.

Zeile 7 reserviert Speicher für einen String fester Länge, der 255 Zeichen umfaßt. Das System braucht nicht annähernd so viel Platz für diese Ordnerinformation, aber besser zu viel als zu wenig Platz. Zeile 8 deklariert eine Integer-Variable, die für den Rückgabewert der Funktionen verwendet wird. Der in `intLength` abgelegte Rückgabewert (Zeile 12) enthält die Anzahl der gültigen Zeichen in dem 255 Zeichen langen String mit den Ordnerinformationen. Verschiedene Computer geben kürzere oder längere Pfade zurück, abhängig davon, wo die Ordner gespeichert sind. Der 255 Zeichen lange String gibt Ihrer Applikation also genügend Platz, um die Pfade zu solchen Ordnern aufzunehmen. Es gibt nur äußerst wenig Fälle, wo ein Verzeichnispfad länger als 255 Zeichen ist.

Bei der Übergabe des Ordners in Zeile 12 benötigt die API-Funktion `GetWindowsDirectory()` sowohl den String mit dem Pfadnamen als auch die Länge dieses Strings. `GetWindowsDirectory()` versucht also nicht, mehr als 255 Zeichen im String `strFolder` abzulegen.

Zeile 13 ermittelt den linken Teil des Rückgabewerts, der Pfadnameninformation enthält. Der Rückgabewert der API-Funktion bestimmt, wie viele Zeichen des Strings den eigentlichen Pfad darstellen. Die Zeichen in dem String fester Länge (255 Zeichen), die rechts vom Pfad stehen, enthalten keine relevanten Informationen, die in den Bezeichnungsfeldern nicht angezeigt werden sollen.

Zeile 20 zeigt eine Anomalie, die Ihnen bei API-Routinen häufig begegnen wird: obwohl die Funktion `GetTempPath()` den Funktionen `GetWindowsDirectory()` und `GetSystemDirectory()` sehr ähnlich ist, sind hier die String- und Integer-Argumente in einer anderen Reihenfolge aufgelistet als in den beiden anderen Funktionen. Achten Sie bei der Arbeit mit System-API-Routinen auf die `Declare`-Anweisungen, damit Sie nicht versehentlich Argumente verwechseln.



Wie sich herausstellt, sind die drei in dieser Applikation vorgestellten Funktionen nicht ganz so einzigartig, wie es den Anschein hat. Für viele der API- Routinen sind solche Manipulationen vor und nach dem Funktionsaufruf erforderlich. Manchmal kehren ähnliche Funktionen ihre Argumente einfach um (wie es hier bei der Funktion `GetTempPath()` im Vergleich zu den beiden anderen aufgerufenen API-Funktionen der Fall war). Die Eleganz der eigenen internen Funktionen von Visual Basic liegt darin, daß sie zu einer einheitlicheren Darstellung führen, daß sie perfekt mit den vordefinierten Datentypen von Visual Basic übereinstimmen, und daß sie nicht die Gefahren in sich tragen, die eine fehlerhafte Verwendung des Windows-API bergen kann (beispielsweise ein unerwartetes Neubooten des Systems).

Zusammenfassung

Dieses Bonuskapitel hat Ihnen gezeigt, wie man die API-Routinen von Windows einsetzt. Windows selbst besteht aus zahlreichen DLLs, die Tausende von Routinen enthalten, auf die Sie von Ihrem Visual-Basic-Programm aus zugreifen können. Sie brauchen zwar nicht viele der verfügbaren

API-Prozeduren, aber viele davon sind ganz praktisch, wenn Sie Systeminformationen brauchen oder eine systembezogene Funktion ausführen möchten, beispielsweise, um die Registrierungsdatenbank des Benutzers zu lesen oder den PC neu zu starten.

Es dauert Zeit, bis Sie das Windows-API beherrschen. Viele Programmierer lernen nie alle API-Prozeduren kennen - viele davon sind für das Betriebssystem erforderlich, nicht jedoch für normale Anwenderprogramme. Durch diese API-Prozeduren haben Sie jedoch Zugriff auf eine reichhaltige Auswahl an Funktionen, um Informationen über das System des Benutzers zu ermitteln.

Die Verwendung des Windows-API ist etwas mühsam. Glücklicherweise bietet Visual Basic ein Werkzeug dafür an: den API-Viewer. Damit können Sie jede Windows-API- Routine, auf dem System anzeigen und die Deklaration kopieren, die Sie dann direkt in Ihren eigenen Code einfügen. Nachdem Sie eine Deklaration in Ihre Visual-Basic- Applikation eingefügt haben (mit einer Declare-Anweisung), können Sie das Windows-API von Ihrem Programm aus aufrufen. Zur Laufzeit stellt Windows Ihrem Programm die Prozeduren zur Verfügung.

Fragen & Antworten

Frage:

Warum scheinen die Deklarationen und Aufrufprozeduren von Windows-API-Routinen so kompliziert zu sein?

Antwort:

Diese Komplexität stammt aus der Beziehung zwischen Ihrer Visual-Basic-Applikation und der eigentlichen Routine. Sie wissen, daß die API-Routinen nicht unter Berücksichtigung von Visual Basic entwickelt wurden. Die Originalroutinen wurden unter C geschrieben, einer Sprache, die etwas andere Datentypen verwendet als Visual Basic und die die Routinen außerdem anders aufruft. Die Deklaration stellt die Möglichkeit für Visual Basic dar, Routinen zu verstehen und zu verwenden, die außerhalb seiner eigenen Umgebung liegen.

Frage:

Gibt es noch andere Beispiele für Windows-API-Routinen, die ich ansehen könnte?

Antwort:

Es gibt viel zu viele, als daß man sie hier aufzählen könnte. Die folgende Liste soll Ihnen jedoch einen Überblick zeigen, was Ihnen alles zur Verfügung steht:

- Ermittlung von Werten aus der Systemregistrierung
- Ermittlung freier und belegter Systemressourcen, beispielsweise Arbeits- und Plattenspeicher
- Zugriff auf die aktuelle Windows-Version
- Arbeit mit Fenstern
- Low-Level-Grafiken
- Verwaltung von Werten in INI-Dateien (INI-Dateien wurden vor Windows 95 verwendet, um Systemwerte aufzunehmen; einige Windows- Programme verwenden sie immer noch).

Manchmal muß man ein bißchen raten, um eine Routine zu finden. Im allgemeinen werden Sie jedoch für jede Aufgabenstellung eine API-Routine finden. Sehen Sie dazu auch in der Online-Hilfe von Visual Basic nach. Man kann aus dem Namen einer API-Routine nicht immer auf ihren Einsatzzweck schließen. Um beispielsweise einen Wert aus einer INI-Datei zu lesen oder ihn dorthin zu schreiben, verwenden Sie die Funktionen `GetSetting()` und `SaveSetting()` .

Workshop

Im Workshop finden Sie Quizfragen, die Ihnen helfen sollten, das Verständnis für die im Kapitel vorgestellten Themen zu vertiefen. Außerdem können Sie hier praktisch anwenden, was Sie gelernt haben. Sie sollten die Quizfragen beantworten und die Übungen nachvollziehen, bevor Sie das nächste Kapitel lesen. In Anhang A finden Sie die Antworten.

Quiz

1. Wofür steht *API*?
2. Warum sollte es je erforderlich sein, eine Windows-API-Routine aufzurufen - Visual Basic bietet doch so viele interne Funktionen?
3. Warum machen die in einer Applikation eingesetzten DLLs diese nicht größer?
4. Warum haben sich die Namen der Standard-DLLs mit der Weiterentwicklung von Windows geändert?
5. Mit welchem Werkzeug ermitteln Sie das Format von API-Routinen?
6. Welche Anweisung deklariert Windows-API-Routinen?
7. Richtig/Falsch. Die Windows-API-Routinen haben ein einheitliches Aussehen und einen einheitlichen Aufrufmechanismus.
8. Was bewirkt die `Declare`-Anweisung?
9. Welches Schlüsselwort verwenden Sie für Windows-API-Prozeduren in einem Modul: `Public` oder `Private`?
10. Welchen Aufgabe hat eine Hüllprozedur?

Übung

Welche Datei enthält die API-Funktion `GetSystemTimer`, `GDI32.DLL` oder `KERNEL32.DLL`?

Tag A

Anhang A: Lösungen

Tag 1

Quizantworten

1. Visual Basic basiert auf der Sprache BASIC.
2. Visual Basic ist visuell und verwendet eine einfache, BASIC-ähnliche Programmiersprache, bietet Ihnen aber dennoch die Möglichkeit, leistungsfähige Windows-Programme zu entwickeln.
3. Die visuelle Natur von Visual Basic ist in vielerlei Hinsicht wichtiger als seine Programmiersprache. Die visuelle Oberfläche gibt Ihrem Programm seinen ganz speziellen Charakter und realisiert die Kommunikation mit dem Benutzer. Die Programmiersprache arbeitet im Hintergrund und verbindet die visuellen Elemente miteinander.
4. Ein Formularfenster *kann* das Programmfenster sein, aber ein Programm kann auch mehrere Formularfenster beinhalten. Das Formularfenster ist das Fenster, das Ihr Benutzer bei der Programmausführung sieht.
5. Ein *Bug* ist ein Programmfehler. Das *Debuggen* ist der Prozeß, Fehler aus einem Programm zu entfernen.
6. Ein Programm in einer kompilierten Sprache ist in der Ausführung viel schneller als ein Programm in einer interpretierten Sprache.
7. Ein Programm in einer interpretierten Sprache ist einfacher zu debuggen als ein Programm in einer kompilierten Sprache.
8. Ein Eröffnungsbildschirm sieht immer gleich aus und erscheint bei jedem Aufruf des Programms. Ein Tip-des-Tages-Bildschirm zeigt bei jedem Programmaufruf einen anderen Tip an, und der Benutzer kann bestimmen, ob der Bildschirm beim nächsten Aufruf ebenfalls wieder erscheinen soll.
9. Ein Steuerelement ist ein Element, beispielsweise ein Bezeichnungsfeld, eine Schaltfläche, ein Optionsfeld oder ein Formular. Eine Steuerelementeigenschaft ermöglicht, Steuerelemente voneinander zu unterscheiden.
10. Falsch. Steuerelemente enthalten keinen Code. Die meisten Steuerelemente sind visuelle Objekte im Formularfenster, mit denen der Benutzer arbeitet. Der Code wird separat abgelegt (in einem sogenannten Modul), wie Sie in den nächsten 20 Kapiteln noch erfahren werden.

Übungen

Führen Sie den Anwendungs-Assistenten aus, und klicken Sie auf **Ja**, wenn Sie gefragt werden, ob die Applikation Internet-Zugriff erhalten soll. Wenn Sie einen Internet-Zugang haben, wählen Sie **Ansicht, Web-Browser**, so daß ein Web-Browser geöffnet wird, von dem aus Sie auf das Internet zugreifen können.

Tag 2

Quizantworten

1. Durch die Symbolleiste haben Sie schnellen Zugriff auf gebräuchliche Befehle und Menüoptionen. Bei der Werkzeugsammlung handelt es sich um einen »Behälter« für die Steuerelemente, die Sie auf Ihrem Formular plazieren können.
2. MSDN ist der abonnierbare Online-Dienst von Microsoft.
3. Falsch. Das Formularfenster kann so viele Formulare enthalten, wie für die Applikation benötigt werden. Es kann jedoch zu jedem Zeitpunkt jeweils nur ein Formular aktiv sein.
4. Der Mauszeiger wird zu einem Fadenkreuz, wenn Sie die Maus über das Formular schieben, damit Sie das Steuerelement auf dem Formularfenster aufspannen können.
5. Visual Basic plaziert das Steuerelement in der Mitte des Formularfensters. Sie können es an die vorgesehene Position verschieben und seine Größe entsprechend anpassen.
6. Falsch. Sie setzen die Steuerelementeigenschaften im Eigenschaftenfenster (oder im Code).
7. Das Eigenschaftenfenster zeigt die Eigenschaften für das aktive Objekt an.
8. Die Schaltfläche mit den drei Punkten für die **Font**-Eigenschaft weist darauf hin, daß zum Setzen des Eigenschaftswerts ein Dialogfeld angezeigt wird, in dem Sie mehrere Werte für diese Eigenschaft festlegen.
9. Die **Caption**-Eigenschaft für die Schaltfläche enthält den Titel, der auf der Schaltfläche erscheint.
10. Die Standardnamen sind wenig aussagekräftig. Geben Sie den Steuerelementen sprechende Namen. Insbesondere durch die Angabe des Präfixes für ein Steuerelement kann vom Namen besser auf die Aufgabe geschlossen werden. Das ist vor allem sinnvoll, wenn in Ihrem Projekt mehrere Steuerelemente verwendet werden.

Übungen

Um einen blauen Hintergrund darzustellen, klicken Sie auf die **BackColor**-Eigenschaft des Formulars. Ein Dropdown-Listenfeld erscheint. Wenn Sie das Listenfeld öffnen, wird ein Dialogfeld mit mehreren Registerkarten angezeigt. Die Registerkarte **Palette** bietet zahlreiche Farben für den Formularhintergrund. Auf der Registerkarte **System** finden sie konventionellere Windows-Farben. Gehen Sie auf die Registerkarte **Palette**, und wählen Sie eine blaue Farbe aus.

Fügen Sie irgendwo auf dem Formular eine Schaltfläche ein, vielleicht in der rechten unteren Ecke des Formularfensters. Setzen Sie die **Name**-Eigenschaft auf `cmdExit`. Setzen Sie die

Caption-Eigenschaft auf **Beenden**. Um die erforderliche Codezeile einzugeben, doppelklicken Sie auf die Schaltfläche und geben in die Prozedur den Befehl **End** ein. Bei der Programmausführung sehen Sie den blauen Hintergrund und die neue Schaltfläche. Nachdem Sie die Grafik angezeigt haben, klicken Sie auf die Schaltfläche **Beenden**, um das Programm zu beenden.

Tag 3

Quizantworten

1. Eine Beschleunigtaste ist eine Tastenkombination für eine Menüoption oder ein Steuerelement, beispielsweise (Alt)+(R).
2. Falsch. Steuerelemente unterstützen Ereignisse.
3. Das **Cancel**-Ereignis ermöglicht dem Benutzer, die Schaltfläche auszuwählen, indem er (Esc) drückt.
4. Das Steuerelement mit dem Fokus wird mit einem gestrichelten Rahmen gekennzeichnet.
5. Der Benutzer drückt (ÿ__) und (^)+(ÿ__) (und abhängig von der Applikation manchmal auch die Pfeiltaste), um den Fokus zwischen den Steuerelementen zu verschieben.
6. Die **TabIndex**-Eigenschaft bestimmt die Fokusreihenfolge.
7. Anhand der Klammern erkennen Sie, daß **LoadPicture()** eine Prozedur ist. **LoadPicture()** war die Prozedur, die in Tag 2 das Bild im Anzeigefeld angezeigt hat. **LoadPicture()** ist eine Funktion, und zwar eine in die Programmiersprache Visual Basic eingebaute Funktion. Sie brauchen den Code für diese Funktion also nicht zu schreiben, weil dieser in Visual Basic bereits vorhanden ist. Diese internen Funktionen helfen Ihnen, Zeit zu sparen, weil Sie den Code, der immer wieder benötigt wird, nicht schreiben zu brauchen. Mehr darüber erfahren Sie in Tag 5.
8. Falsch. Visual Basic erzeugt häufig eine **Click**-Ereignisprozedur, aber nicht immer. Visual Basic versucht, die gebräuchlichste Ereignisprozedur für das Steuerelement, auf das Sie doppelklicken, zu erraten. In einigen Fällen, wie beispielsweise bei einem Textfeld, erzeugt Visual Basic die erste und die letzte Zeile für die **Change**- Ereignisprozedur im Codefenster.
9. Richtig.
10. Die **PasswordChar**-Eigenschaft ermöglicht Ihnen, die Eingaben des Benutzers zu verbergen, falls es sich dabei um kritische Informationen handelt, die bei der Eingabe nicht von Zuschauern gelesen werden sollen.

Übungen

- 1.1: `Private Sub frmMyApp_Load ()`
2. In den Zeilen 1 und 5 erkennt man, daß es sich bei der Prozedur um eine Funktion handelt. Ereignisprozeduren müssen Sub-Prozeduren sein. Ersetzen Sie **Function** durch **Sub**, um das Problem zu beheben.
3. Die drei Textfelder müssen verschiedene **Style**-Eigenschaften für die verschiedenen Bildlaufleisten erhalten. Der Vorgabetext wird bei der Programmentwicklung in der

`Text`-Eigenschaft festgelegt. Es ist nicht möglich, in der `Text`-Eigenschaft mehrere Zeilen vorzugeben; wählen Sie für die Vorgabe ein einzelnes Wort oder einen kurzen Satz.

Setzen Sie die `Caption`-Eigenschaft der Schaltfläche auf `&Beenden`, um sicherzustellen, daß der Benutzer das Programm mit einer Tastenkombination beenden kann.

Tag 4

Quizantworten

1. Menüs werden mit Hilfe des Menü-Editors angelegt.
2. Richtig.
3. Menüoptionen erzeugen nur das `Click`-Ereignis.
4. Ein Shortcut ist ein Befehl, der eine Menüoption auslöst, ohne daß der Benutzer das Menü anzeigen muß.
5. Die Benutzer wählen die Menüoptionen schnell über Tastenkombinationen aus.
6. Shortcuts bilden Mausklicks im Menü nach, erzeugen also `Click`-Ereignisse.
7. Falsch. Der Menü-Editor unterstützt nur den Menüentwurf, nicht jedoch den zugehörigen Code.
8. Die Option `Checked` ermöglicht Ihnen, links von einer Menüoption ein Häkchen anzuzeigen.
9. Richtig. Es können mehrere Optionen markiert werden, was im Code gesteuert werden kann.
10. Bei den Zeilen handelt es sich um Kommentare, die beschreibende Informationen über das Programm anzeigen. Im Bonusprojekt 1 wird erklärt, worum es sich bei Kommentaren handelt und wie sie verwendet werden.

Übungen

1. Der Menü-Editor enthält mehrere Pfeil-Schaltflächen. Wenn Sie auf die Schaltfläche mit dem Pfeil nach rechts klicken, um drei Punkte zu erzeugen, legt der Menü- Editor die betreffende Option als Dropdown-Menüoption an.
2. Manuel sollte wissen, daß er dafür verantwortlich ist, die Markierung von Menüoptionen im Code aufzuheben. Markierte Menüoptionen müssen sich nicht wechselseitig ausschließen, wie es in diesem Kapitel der Fall war; deshalb muß der Programmierer genau festlegen, wie sich die Markierungen verhalten sollen.
3. Öffnen Sie das Codefenster, und klicken Sie auf alle Dropdown-Menüoptionen in der unteren Fensterhälfte. Wählen Sie einen im Menü-Editor bereitgestellten Shortcut aus. Bei der nächsten Programmausführung erscheinen die Shortcuts neben den entsprechenden Menüoptionen. Probieren Sie den Shortcut aus, indem Sie die betreffende Tastenkombination drücken, ohne das zugehörige Menü anzuzeigen.

Tag 5

Quizantworten

1. Der Code für die Datendeklaration befindet sich im Deklarationsbereich eines Programms.
2. Lokale Variablen können von zwei oder mehr Prozeduren gemeinsam genutzt werden. In den folgenden Kapiteln werden Sie mehr darüber erfahren.
3. Richtig.
4. Falsch. Eine Variable kann während der Programmausführung verschiedene Werte aufnehmen.
5. Ein Divisionsoperator führt eine normale Fließkommadivision durch, während der andere Operator eine ganzzahlige Division vornimmt.
6. Ein überladener Operator ist ein Operator, der abhängig vom Kontext zwei verschiedene Operationen ausführt.
7. Das Ampersand-Zeichen wird bevorzugt, weil das Pluszeichen allgemein als Additionsoperator verwendet wird.
8. Der Datentyp `Variant` kann beliebige andere Datentypen aufnehmen.
9. Falsch. Visual Basic bedingt keine Präfixe, Sie sollten sie aber zur Dokumentation Ihres Codes verwenden.
10. Sie platzieren die Anweisung `Option Explicit` am Anfang des Deklarationsbereichs oder markieren den entsprechenden Eintrag im Dialogfeld Optionen. Das Dialogfeld wird global verwendet, während die Anweisung `Option Explicit` nur sicherstellt, daß die Variablen in dem betreffenden Modul deklariert werden.

Übungen

1. Wenn Sie sagen, Visual Basic gibt einen Fehler für die mittlere Variable zurück, `abc`, dann wäre das gut geraten - ist aber falsch. Visual Basic nimmt an, daß alle deklarierten Variablen den Datentyp `Variant` haben, es sei denn, Sie geben etwas anderes an. Visual Basic deklariert `abc` also als die Variable mit dem Datentyp `Variant`.
2. Visual Basic berechnet die Division vor der Addition. Um ein korrektes Ergebnis zu erhalten, sollte Sally zuvor die Additionen addieren:

`sngAvg = (sngGrade1 + sngGrade2 + sngGrade3) / 3`

- a. 5
- b. 6
- c. 5
- d. 5
- e. 7

3. a. `a = (3 + 3) / (4 + 4)`

$$b. x = (a - b) * (a - 2) ^2$$

$$c. f = a ^{(1/2)} / b ^{(1/2)}$$

4. Das Ampersand-Zeichen konkateniert Stringkonstanten, so daß Visual Basic sie in der Zeilenfortsetzung wie eine große Konstante behandelt.

Tag 6

Quizantworten

1. Der Or-Operator gibt True zurück, wenn einer der Werte True ist.
2. Ein bedingter Operator vergleicht zwei Werte. Ein logischer Operator kombiniert zwei bedingte Ausdrücke.
3. Eine Schleife beinhaltet einen oder mehrere Ausdrücke, die mehrfach ausgeführt werden können.
4. Die Integer-Variable enthält nach der Zuweisung 10 weniger als vor der Zuweisung.
5. Die Schleife wird nie ausgeführt, weil `intN` oben in der Schleife gleich 0 ist.
6. Wenn `Exit For` ohne eine einführende `If`-Anweisung erscheint, wird die `For`-Schleife immer nur einmal ausgeführt, weil `Exit For` eine Schleife beim ersten Schleifendurchlauf beendet.
7. Falsch. Es wird immer nur ein Block einer `If . . . Else`-Anweisung ausgeführt. Der `If`-Block wird ausgeführt, wenn die Bedingung zutrifft, und `Else` wird ausgeführt, wenn die Bedingung falsch ist.
8. Richtig. Wenn der Endwert zum Anfang der `For`-Schleife bereits größer ist als der Startwert, wird die Schleife überhaupt nicht ausgeführt und das Programm wird in der dem `For` folgenden Anweisung ausgeführt.
9. Verschachtelte Schleifen werden verwendet, wenn Schleifen mehrfach ausgeführt werden sollen.
10. Eine Entscheidungsanweisung kann den Coderumpf abhängig von dem Ergebnis einmal, und nur einmal, ausführen. Eine Schleife kann den Coderumpf mehrfach ausführen.

Übungen

1. `If (a = b) And (b = c) Then ' Es folgt der If-Rumpf`
2. Larrys Schleife wird nie beendet, weil sich die Steuervariable im Schleifenrumpf nie ändert.
3. Die Uhr in einem Fußballspiel bildet eine verschachtelte `For`-Schleife nach. Betrachten Sie den folgenden Code:

```
1: For Qtr = 1 to 4
2: For Minutes = 15 to 0 Step -1
3: ' Ball im Spiel
4: Next Minutes
5: Next Qtr
```

Der folgende Code verwendet alle Case-Anweisungen innerhalb eines Select Case:

```
1: Select Case intHours
2: Case 1 To 40
3: curOverTime = 0.0
4: Case 41 To 49
5: curOverTime = (intHours - 40) * 1.5 * sngRate
6: Case Is >= 50
7: curOverTime = ((intHours - 50) * 2 + (10 * 1.5)) * sngRate
8: End Select
```

Tag 7

Quizantworten

1. Eine interne Funktion verwendet einen Argumentwert, den Sie ihr übergeben. Sie gibt basierend auf diesem übergebenen Argumentwert einen neuen Wert zurück.
2. Richtig.
3. Empty ist das Schlüsselwort, das äquivalent zu einem leeren String ist.
4. Falsch. Die Tabellen beschreiben alle ein Argument, nämlich das zweite Argument der Funktion `MsgBox()`, das im heutigen Kapitel mit dem Platzhalter `intStyle` bezeichnet wurde.
5. Visual Basic verwendet den Namen der Applikation als Titel, wenn Sie keinen Fenstertitel angeben.
6. Der Benutzer kann beliebig viele Kontrollkästchen markieren, aber nur ein Optionsfeld.
7. Richtig - Wenn Sie allen Werten beim Laden des Formulars `False` zuweisen (beispielsweise in der Ereignisprozedur `Form_Load()`).
8. Eine `Value`-Eigenschaft von 1 oder 0 zeigt an, ob ein Kontrollkästchen markiert oder nicht markiert ist.
9. Eine `Value`-Eigenschaft von `True` oder `False` zeigt an, ob ein Optionsfeld markiert oder nicht markiert ist.
10. Wenn Sie mehrere Gruppen mit Optionsfeldern auf einem Formular platzieren möchten, so daß der Benutzer aus jeder dieser Gruppe ein Optionsfeld markieren kann, legen Sie diese innerhalb von Rahmen an.

Übungen

1. Prüfen Sie, ob die `InputBox()`-Funktion einen leeren String zurückgibt. Vergleichen Sie dazu mit dem leeren String, »«. Wenn die Funktion etwas anderes zurückgibt, ist der Rückgabewert des Benutzers korrekt.

2. Erzeugen Sie das Meldungsfeld wie folgt:

```
strAns = MsgBox("Kein Papier", vbExclamation + _  
vbDefaultButton2 +
```

`vbAbortRetryIgnore, "Drucker")`

3. Die folgende Ereignisprozedur kombiniert Stadt und Staat des Benutzers und zeigt den verknüpften String in einem Meldungsfeld an:

```
1: Private Sub cmdGetLocation_Click()  
2: ' Stadt und Staat in zwei separaten Eingabefeldern abfragen  
3: ' Konkatenieren  
4: ' Verknüpften String anzeigen  
5: Dim strCity As String  
6: Dim strState As String  
7: Dim strBoth As String  
8: ' Placeholder for MsgBox() return  
9: Dim intAnswer As Integer  
10:  
11: ' Nach Stadt und Staat fragen  
12: strCity = InputBox("Stadt?", "Stadt ermitteln")  
13: strState = InputBox("Staat?", "Staat ermitteln")  
14:  
15: ' Strings verknüpfen  
16: strBoth = strCity & ", " & strState  
17:  
18: ' Verknüpften String anzeigen  
19: intAnswer = MsgBox("Sie leben in " & strBoth, " ")  
20: End Sub
```

4. Ihre Optionsfeld-Prozeduren sind `Click`-Ereignisprozeduren, weil ein `Click`-Ereignis ausgelöst wird, wenn der Benutzer auf das Optionsfeld klickt. Sie müssen sich nicht darum kümmern, ein Optionsfeld zu löschen, wenn der Benutzer ein anderes markiert - das übernimmt Visual Basic für Sie.

Tag 8

Quizantworten

1. Öffentliche Variablen haben den größten Gültigkeitsbereich.
2. Lokale Variablen haben den geringsten Gültigkeitsbereich.
3. Richtig. Alle Argumente werden als Referenz übergeben, es sei denn, Sie spezifizieren `ByVal`.
4. Sub-Prozeduren geben keine Rückgabewerte zurück.
5. `IIf( )` und `Choose( )` sind Abkürzungen für die `If`-Anweisung.
6. `Choose( )` gibt Null zurück, wenn ihr erstes Argument kleiner 1 ist.
7. `Abs( )` gibt den Absolutwert für sein Argument zurück.
8. a. 74135 -
b. 12 2 pm (abhängig von der Tageszeit, wann diese Anweisung ausgeführt wird)
c. 12345.67

9. `intN` enthält 192.

10. `Now` gibt Datum und Zeit zurück, während `Time` nur die Zeit zurückgibt.

Übungen

1. Der folgende Code enthält die Funktion `SalesTax()`.

```
1: Private Sub GetTotal()  
2: ' Diese Prozedur addiert Summen in einem Formular  
3: ' und sendet die Gesamtsumme und einen Rabattsatz  
4: ' an die Prozedur zur Steuerberechnung  
5: Dim curTotal As Currency  
6: Dim sngDisc As Single ' Special tax discount  
7: '  
8: ' Summe aus den Inhalten der Textfelder bilden  
9: curTotal = txtSale1.Text + txtSale2.Text + txtSale3.Text  
10: '  
11: ' Gesamtsumme an die Steuerprozedur übergeben  
12: intMsg = MsgBox("The sales tax is " & SalesTax(curTotal, _  
sngDisc))  
13: End Sub  
14:  
15: Public Function SalesTax(curTotal As Currency, sngRateDisc As _  
Single) As Currency  
16: ' Steuer berechnen und Rabatt berücksichtigen  
17: Dim curSalesTax As Currency  
18: '  
19: ' Dieser Code erscheint im Kapitel als Sub-Prozedur  
20: ' Hier wird eine Steuer berechnet, die auf 3%  
21: ' basiert, plus einem Aufschlag von 0.5 Prozent  
22: curSalesTax = (curTotal * .03) + (curTotal * .005)  
23: '  
24: ' Rabatt abziehen  
25: curSalesTax = curSalesTax - (sngRateDisc * curTotal)  
26: '  
27: ' Rückgabewert setzen  
28: SalesTax = curSalesTax  
29: '  
30: ' Prozeduren kehren nach der Abarbeitung automatisch zur  
31: ' aufrufenden Prozedur zurück  
32: End Function
```

2. Die folgende `IIf()`-Funktion erledigt dieselbe Aufgabe wie die `If`-Anweisung:

```
strTitle = IIf(intTotal > 1000, "Gut gemacht!", "Das war nix!")
```

3. Die folgende `Choose()`-Funktion erledigt dieselbe Aufgabe wie die `If`-Anweisung:

```
Choose(ID, intBonus = 50, intBonus = 75, intBonus = 100)
```

4. `intN` enthält -6, `intO` enthält -5 und `intP` enthält -6.

Tag 9

Quizantworten

1. Sie fügen Ihrer Werkzeugsammlung das Steuerelement über die Menüoption **Projekt , Komponenten** hinzu.
2. Das Standarddialogfeld-Steuerelement zeigt Dialogfelder **Farbe, Öffnen, Speichern , Schriftart, Drucken** und **Hilfe** an.
3. Das Standarddialogfeld-Steuerelement erzeugt eines von sechs Standarddialogfeldern.
4. Das Standarddialog-Steuerelement wird zur Laufzeit nicht angezeigt. Es erscheint nur, wenn der Benutzer eine seiner Methoden ausführt, die ein Dialogfeld auf dem Bildschirm anzeigen.
5. Richtig.
6. Die **Filter**-Eigenschaft legt fest, welche Filtertypen in dateibezogenen Dialogfeldern verwendet werden.
7. Die **Flags**-Eigenschaft richtet ein Standarddialogfeld ein, bevor Sie es mit einer Methode anzeigen können.
8. Richtig.
9. Falsch.
10. Falsch. Sie müssen die Methode weiter qualifizieren, beispielsweise **ShowFont** und **ShowPrinter**.

Übungen

1. Der folgende Code stattet das Dialogfeld Schriftart mit Code aus, der die Fehlerverarbeitung für das Anklicken der Abbrechen-Schaltfläche bereitstellt.
- ```
1: ' Assumes CancelError is True
2: On Error Goto dbErrHandler
3: ' Flags-Eigenschaft setzen
4: CdbFont.Flags = cdlCFBoth Or cdlCFEffects
5: CdbFont.ShowFont ' Display the Font DB
6: ' Eigenschaften eines Bezeichnungsfeldes auf die
7: ' vom Benutzer angegebene Schriftinformation setzen
8: LblMessage.Font.Name = CdbFont.FontName
9: LblMessage.Font.Size = CdbFont.FontSize
10: LblMessage.Font.Bold = CdbFont.FontBold
11: LblMessage.Font.Italic = CdbFont.FontItalic
12: LblMessage.Font.Underline = CdbFont.FontUnderline
13: LblMessage.Font.Strikethru = CdbFont.FontStrikethru
14: LblMessage.ForeColor = CdbFont.Color
15: Exit Sub ' Wichtig, sonst geht's mit der Fehlerbehandlung
weiter!
```

```
16:
17: dbErrHandler:
18: ' Der Benutzer hat auf die Abbrechen-Schaltfläche geklickt
19: Exit Sub ' Schrift nicht ändern
```

2. Der folgende Code erzeugt das in Abbildung 9.10 gezeigte Dialogfeld:

```
1: Private Sub mnuFileOpen_Click ()
2: ' CancelError ist True
3: On Error Goto dbErrHandler
4: ' Festlegen, was im Textfeld
5: ' Dateitypen erscheint
6: cdbFile.Filter = "Text Files (*.txt) | *.txt"
7: ' Standardfilter vorgeben
8: cdbFile.FilterIndex = 1
9: cdbFile.DialogTitle = "Open"
10: ' Dialogfeld Öffnen anzeigen
11: cdbFile.ShowOpen
12:
13: ' *****
14: ' Hier plazieren Sie Code oder *
15: ' rufen eine Prozedur auf, die die*
16: ' ausgewählte Datei öffnet *
17: ' *****
18: Exit Sub '
19:
20: dbErrHandler:
21: ' Benutzer hat auf Abbrechen geklickt
22: Exit Sub ' Keine Datei öffnen
23: End Sub
```

## Tag 10

### Quizantworten

1. Die Ereignisse `MouseDown` und `MouseUp` sind bestimmten Maustasten zugeordnet und ermöglichen Ihnen festzustellen, welche Taste der Benutzer geklickt hat. Dazu werden die Argumente ausgewertet, die ein Ereignis der Prozedur übergibt. Die Ereignisse `Click` und `DbClick` beinhalten keine Information darüber, mit welcher Taste sie erzeugt wurden.
2. Anhand des `intButton`-Arguments wird ermittelt, welche Taste geklickt wurde.
3. Der Pfad für das Icon wird in der `DragIcon`-Eigenschaft des Steuerelements gesetzt.
4. Ignorieren Sie mehrere Timer-Ereignisse, und reagieren Sie erst, nachdem eine bestimmte Zeit vergangen ist.
5. Sie können das Steuerelement zur Entwurfszeit initialisieren, indem Sie die Elemente in die `List`-Eigenschaft eintragen, oder zur Laufzeit (die üblichere Vorgehensweise), indem Sie sie mit der `AddItem`-Methode einfügen.

6. Die `Value`-Eigenschaft des Listenfelds enthält den Index des vom Benutzer ausgewählten Elements.
7. Sie können die Elemente einzeln mit `RemoveItem` entfernen oder alle mit `Clear`.
8. Wenn der Fokus weitergegeben wird, kann die ausgelöste Ereignisprozedur das neue Element in die Liste eintragen.
9. Setzen Sie die `Sorted`-Eigenschaft auf `True`.
10. Die Anweisung reserviert dreiundzwanzig Elemente.

## Übungen

1. Sie legen ein Formular mit einem Listefeld an, das mit dem folgenden Code in der Ereignisprozedur `Form_Load( )` initialisiert wird:

```
Private Sub Form_Load()
 lstFamily.AddItem ("Martha")
 lstFamily.AddItem ("William")
 lstFamily.AddItem ("Clyde")
 lstFamily.AddItem ("Larry")
 lstFamily.AddItem ("Sandy")
 lstFamily.AddItem ("Pauline")
 lstFamily.AddItem ("Paul")
 lstFamily.AddItem ("Eddie")
 lstFamily.AddItem ("Carlton")
 lstFamily.AddItem ("Charlie")
 lstFamily.AddItem ("Robert")
End Sub
```

2. Für diese Übung ist keine Antwort erforderlich.

## Tag 11

### Quizantworten

1. Wenn das `Resize`-Ereignis auftritt, können Sie Ihre Steuerelemente innerhalb der neuen Größe des Formulars zentrieren.
2. Die `ScaleMode`-Eigenschaft legt fest, welche Maßeinheit folgende `Print`-Methoden für die Koordinatenpositionen `CurrentX` und `CurrentY` verwenden.
3. Der erste Index ist 0.
4. Falsch. SDI-Applikationen *können* mehrere Formulare unterstützen, allerdings nicht, wenn jedes Formularfenster eine andere Datenmenge verarbeitet.
5. Eine MDI-Applikation kann mehrere Formularfenster mit verschiedenen Datenmengen enthalten.
6. Symbolleisten werden in der Regel oben im Formular angezeigt.
7. Die Abbildungsliste enthält die Icons für die Symbolleiste.

8. Die Abbildungsliste enthält nur die Icons für die Coolbar. Das Steuerelement erscheint zur Laufzeit nicht auf dem Formular; es ist also unerheblich, wo es angelegt wird.
9. `Spc( )` fügt eine feste Anzahl Leerzeichen in eine Ausgabezeile ein. `Tab( )` setzt den Textcursor in eine bestimmte Spalte.
10. Verwenden Sie die `Print`-Methode, geben Sie jedoch rechts davon nichts an, was gedruckt werden soll.

## Übungen

1. Für diese Übung ist keine Antwort erforderlich.
2. Die Ausgabe der beiden `Print`-Methoden erscheint nebeneinander, weil hinter der ersten `Print`-Methode ein Semikolon steht.  
`Zeile 1`  
`Zeile 2`
3. Für diese Übung ist keine Antwort erforderlich.
4. Verwenden Sie in der `Click`-Ereignisprozedur eine `For`-Anweisung, die etwa so aussieht:  

```
For intCtr = 1 To 100
Form1.Print intCtr; " ";
Next intCtr
```
5. Der folgende Code zählt die Steuerelemente in allen Formularen einer Applikation:  

```
' Zähler vor dem Zählen auf Null setzen
intCount = 0
' Step through each form
For intCtr = 1 to Forms.Count
' Steuerelemente in allen Formularen addieren
intCount = intCount + Forms(intCtr).Count
Next intCtr
```

## Tag 12

### Quizantworten

1. Die `Close`-Anweisung schließt alle geöffneten Dateien.
2. Die Funktion `FreeFile( )` gibt die nächste freie Dateinummer zurück.
3. Visual Basic überschreibt eine Datei, falls sie bereits existiert, die Sie für sequentielle Ausgaben öffnen.
4. Visual Basic fügt die Ausgabe an das Ende der sequentiellen Datei an.
5. Die Anweisung öffnet eine sequentielle Datei und ermöglicht folgenden Ausgabeanweisungen, am Dateiende weiterzuschreiben.
6. Dateien mit wahlfreiem Zugriff müssen in der Lage sein, die Position jedes beliebigen Datensatzes zu berechnen. Deshalb müssen die Datensätze alle dieselbe Länge haben.

7. Die Strings fester Länge ermöglichen Datensätze fester Länge, während Strings variabler Länge Datensätze verschiedener Längen erzeugen würden.
8. Die Type-Anweisung definiert neue Datentypen.
9. Falsch. Die Anweisung definiert einen neuen Datentyp, deklariert aber keine Variablen dieses Typs.
10. `Dir()` mit Argument gibt die erste Datei zurück, die mit den Wildcard-Zeichen übereinstimmt. `Dir()` ohne Argument gibt die nächste Datei im Ordner zurück, die mit Ihren Wildcard-Zeichen übereinstimmt.

## Übungen

1. Frannie versucht wahrscheinlich, Bills Ordner zu löschen, solange dieser noch Dateien enthält. Mit dem `Rmdir`-Befehl können keine Ordner entfernt werden, die noch Dateien enthalten.

2. Die folgende Prozedur füllt die Arrays in den Zeilen 10 bis 14 und schreibt sie in den Zeilen 21 bis 23 in eine geöffnete Datei:

```
1: Private Sub output ()
2: Dim strNames(5) As String
3: Dim intAges(5) As Integer
4: Dim strColors(5) As String
5:
6: Dim intCtr As Integer ' Schleifenzähler
7: Dim intFNum As Integer ' Dateinummer
8:
9: ' Array-Info ermitteln
10: For intCtr = 0 To 4
11: strNames(intCtr) = InputBox("Nächster Name?", "Namen_ermitteln")
12: intAges(intCtr) = InputBox("Nächstes Alter?", "Alter ermitteln")
13: strColors(intCtr) = InputBox("Nächste Farbe?", "Farben
ermitteln")
14: Next intCtr
15:
16: intFNum = FreeFile
17:
18: ' Ausgabe schreiben
19: ' Pfad gegebenenfalls ändern
20: Open "C:\Stuff.txt" For Output As #intFNum
21: For intCtr = 0 To 4
22: Write #intFNum, strNames(intCtr), intAges(intCtr), _
strColors(intCtr)
23: Next intCtr
24:
25: Close #intFNum
26: End Sub
```

3. Legen Sie ein Listenfeld ähnlich dem in Abbildung 12.1 gezeigten an. Sie brauchen für alle drei dateibezogenen Listenfelder eine Ereignisprozedur, die die beiden anderen Listenfelder aktualisiert.

Welches Ereignis die Ereignisprozedur auslöst, ist vom Steuerelement abhängig. Das Verzeichnislistenfeld unterstützt ein **Change**-Ereignis. In der **Change**-Ereignisprozedur dieses Steuerelements ändern Sie also die angezeigten Dateien, weil das Verzeichnis gewechselt wurde. Hier ein Beispiel:

```
Private Sub Direct_Change()
' Der Benutzer hat das Verzeichnislistenfeld geändert
filFile.Path = dirList.Path ' Datei-Steuerelement ändern
'
' Sicherstellen, daß nur eine Datei ausgewählt ist
If (filFile.ListCount > 0) Then
' Erste Datei in der Liste selektieren
filFile.ListIndex = 0
End If
End Sub
```

Wenn der Benutzer ein anderes Laufwerk auswählt, müssen Sie den Pfad zu diesem Laufwerk ändern (was wiederum die oben gezeigte Ereignisprozedur auslöst):

```
Private Sub Drive_Change()
' Setzt den Standardpfad auf das Laufwerk
dirList.Path = drvList.Drive
End Sub
```

## Tag 13

### Quizantworten

1. `Printers.Count - 1` gibt die Anzahl der im System installierten Drucker an.
2. Richtig.
3. Die `ScaleMode`-Eigenschaft bestimmt, welche Maßeinheit Visual Basic verwendet.
4. Ausgaben werden mit der Methode `NewPage` auf eine neue Seite geschickt.
5. Die Anweisung `Is TypeOf` überprüft, welchen Datentyp Objekte haben.
6. Richtig.
7. Falsch. `KillDoc` verwirft nur die Druckausgabe der Applikation.
8. `Me` bezeichnet das aktuelle Formular.
9. `PrintForm` verwendet die Bildschirmauflösung für die Ausgabe.
10. Setzen Sie die `AutoRedraw`-Eigenschaft auf `True`, bevor Sie Formulare ausdrucken.

### Übungen

1. Die folgende Zeile schreibt, beginnend in Spalte 32, *Printing Peter* auf den Drucker:  
`Printer.Print Tab(32); "Printing Peter"`

2. Patty sollte wissen, daß Windows verschiedene Schriftarten und Schriftgrößen unterstützt, so daß ihre Berechnungen für die Seitenlänge auch die Größe der verwendeten Schrift berücksichtigen sollten.

3. Der folgende Code vervollständigt Listing 13.2:

```
1: Public Function IsColor() As Boolean
2: Dim blnIsColor As Boolean
3: Dim prnPrntr As Printer
4: '
5: ' Angenommen, es wird kein Farbdrucker gefunden
6: blnIsColor = False
7: '
8: ' Alle Drucker durchsuchen
9: For Each prnPrntr In Printers
10: If prnPrntr.ColorMode = vbPRCMColor Then
11: ' Farbdrucker als Systemstandard verwenden
12: Set Printer = prnPrntr
13: blnIsColor = True
14: Exit For ' Nicht weitersuchen
15: End If
16: Next ' Ggf. alle durchlaufen
17: '
18: ' blnIsColor ist weiterhin false, wenn kein Farb-
19: ' drucker vorhanden ist, true, wenn es einen gibt
20: ' Rückgabewert der Funktion entsprechend setzen
21: IsColor = blnIsColor
22: End Function
```

## Tag 14

### Quizantworten

1. Das Figur-Steuerelement zeichnet Kreise.
2. Das Figur-Steuerelement zeichnet Quadrate (und Rechtecke).
3. Die Methoden `PSet` und `Line` zeichnen Rechtecke auf dem Formular.
4. Richtig.
5. Die Option `B` bewirkt, daß die Rechtecke ausgefüllt gezeichnet werden.
6. Nicht alle Schaltflächen werden von allen Multimedia-Geräten benötigt.
7. Gibt den Status des Multimedia-Geräts zurück.
8. Verringern Sie den Wert in der `UpdateInterval`-Eigenschaft, um die Aktualisierungsfrequenz des Multimedia-Geräts zu erhöhen.
9. Das Bildfeld nimmt die Videoausgabe auf.

10. Ihr Code muß den Windows-Gerätekontext auf das Bildfeld setzen, so daß das Multimedia-Steuerelement weiß, wohin es die Videoausgaben senden soll.

## Übungen

1. Das Codefenster für die Unterstützung zusätzlicher Listenfelder sollte Code enthalten, der etwa wie folgt arbeitet:

```
1: Private Sub Form_Load()
2: ' Listenfeld Shape initialisieren
3: lstShape.AddItem "0 - Rechteck"
4: lstShape.AddItem "1 - Quadrat"
5: lstShape.AddItem "2 - Oval"
6: lstShape.AddItem "3 - Kreis"
7: lstShape.AddItem "4 - Abgerundetes Rechteck"
8: lstShape.AddItem "5 - Abgerundetes Quadrat"
9:
10: ' Dropdown-Listenfeld Fill-Style initialisieren
11: lstPattern.AddItem "0 - Durchgängig"
12: lstPattern.AddItem "1 - Transparent"
13: lstPattern.AddItem "2 - Horizontale Linie"
14: lstPattern.AddItem "3 - Vertikale Linie"
15: lstPattern.AddItem "4 - Aufwärts Diagonal"
16: lstPattern.AddItem "5 - Abwärts Diagonal"
17: lstPattern.AddItem "6 - Kreuz"
18: lstPattern.AddItem "7 - Diagonal Kreuz"
19:
20: ' Dropdown-Listenfeld FillColor initialisieren
21: ' (Nicht alle Farben werden berücksichtigt)
22: lstFillColor.AddItem "Schwarz"
23: lstFillColor.AddItem "Weiß"
24: lstFillColor.AddItem "Blau"
25: lstFillColor.AddItem "Rot"
26: lstFillColor.AddItem "Grün"
27: lstFillColor.AddItem "Gelb"
28:
29: ' Dropdown-Listenfeld BorderColor initialisieren
30: lstBorderColor.AddItem "Schwarz "
31: lstBorderColor.AddItem "Weiß "
32: lstBorderColor.AddItem "Blau "
33: lstBorderColor.AddItem "Rot "
34: lstBorderColor.AddItem "Grün "
35: lstBorderColor.AddItem "Gelb "
36:
37: ' Den ersten Wert der Liste als Vorgabe verwenden
38: lstShape.SelectedIndex = 0
39: lstPattern.SelectedIndex = 0
40: lstFillColor.SelectedIndex = 0
41: lstBorderColor.SelectedIndex = 0
```



```
42: End Sub
43:
44: Private Sub lstPattern_Click()
45: ' Muster entsprechend ändern
46: shpSample.FillStyle = lstPattern.ListIndex
47: End Sub
48:
49: Private Sub lstShape_Click()
50: ' Umriß entsprechend ändern
51: shpSample.Shape = lstShape.ListIndex
52: End Sub
53:
54: Private Sub lstFillColor_Click()
55: ' Füllfarbe entsprechend ändern
56: Select Case lstFillColor.ListIndex
57: Case 0:
58: shpSample.FillColor = vbBlack
59: Case 1:
60: shpSample.FillColor = vbWhite
61: Case 2:
62: shpSample.FillColor = vbBlue
63: Case 3:
64: shpSample.FillColor = vbRed
65: Case 4:
66: shpSample.FillColor = vbGreen
67: Case 5:
68: shpSample.FillColor = vbYellow
69: End Select
70: End Sub
71:
72: Private Sub lstBorderColor_Click()
73: ' Rahmenfarbe entsprechend ändern
74: Select Case lstBorderColor.ListIndex
75: Case 0:
76: shpSample.BorderColor = vbBlack
77: Case 1:
78: shpSample.BorderColor = vbWhite
79: Case 2:
80: shpSample.BorderColor = vbBlue
81: Case 3:
82: shpSample.BorderColor = vbRed
83: Case 4:
84: shpSample.BorderColor = vbGreen
85: Case 5:
86: shpSample.BorderColor = vbYellow
87: End Select
88: End Sub
89:
90: Private Sub mnuFileExit_Click()
```

91: End  
92: End Sub

## Tag 15

### Quizantworten

1. Formularvorlagen ermöglichen Ihnen, Ihren Applikationen Standardformulare hinzuzufügen und Ihre Programmentwicklung zu beschleunigen.
2. Sie finden den Anwendungs-Assistenten im Dialogfeld **Projekt, Hinzufügen**. Oder Sie klicken mit der rechten Maustaste in das Projektfenster und wählen im Kontextmenü **Hinzufügen, Formular**.
3. Wenden Sie einfach die **Show**-Methode auf das Formular für das **Info**-Feld an, wenn der Benutzer **Hilfe, Info** aus dem Menü auswählt.
4. Falsch.
5. Ein Eröffnungsbildschirm sollte beim Starten des Programms eine bestimmte Zeit lang auf dem Bildschirm angezeigt werden.
6. Das Dialogfeld **Tips und Tricks** bleibt nur so lange auf dem Bildschirm, wie der Benutzer das möchte, während der Eröffnungsbildschirm nur kurz als Begrüßung erscheint. Darüber hinaus zeigt das Dialogfeld **Tips und Tricks** weitere Tips an, wenn der Benutzer auf die Schaltfläche **Nächster Tip** klickt. Der Benutzer kann den Bildschirm **Tips und Tricks** deaktivieren, so daß er beim nächsten Programmaufruf nicht mehr erscheint. Der Eröffnungsbildschirm dagegen wird bei jedem Programmaufruf angezeigt.
7. Der Eintrag **SaveSettings** gibt in Verbindung mit dem Dialogfeld **Tips und Tricks** an, ob der Benutzer beim Programmaufruf Tips sehen will.
8. *ODBC* steht für *Open Database Connectivity*. Es bietet eine standardisierte Vorgehensweise für den Zugriff auf Datenbanken auf anderen Computersystemen.
9. Machen Sie das Dialogfeld **Tips und Tricks** oder den Eröffnungsbildschirm zum Startobjekt.
10. Bei der Tip-Datei handelt es sich um eine Textdatei, die Sie mit einem Editor wie beispielsweise Notepad anlegen. Die Datei enthält für jeden Tip eine Zeile. Legen Sie die Tip-Datei im selben Ordner wie die Applikation ab, von der sie benutzt wird.

### Übungen

1. Die Lösung wird im Kapiteltext beschrieben.
2. Für diese Frage ist keine Antwort erforderlich.

## Tag 16

## Quizantworten

1. Wenn Sie ein OLE-Objekt linken, bleibt dieses Objekt innerhalb seiner eigenen Applikation. Wenn Sie ein OLE-Objekt einbetten, erhält Ihre Applikation eine Kopie des Objekts. Wenn ein eingebettetes Objekt innerhalb seiner Applikation geändert wird, erfährt Ihre Applikation nichts über diese Änderung.
2. Für das Einbetten ist am meisten Plattenspeicher erforderlich, weil innerhalb Ihrer Applikation eine Kopie des Objekts gehalten werden muß.
3. Falsch. Sie müssen den Code schreiben, der Änderungen am OLE-Objekt speichert.
4. Die `SaveToFile`-Methode speichert Objekte auf der Platte.
5. Die `ReadFromFile`-Methode lädt Objekte von der Platte.
6. Die Anweisung `If TypeOf` und die Funktion `TypeName` überprüfen die Objektklassen.
7. Falsch. Systemobjekte sind bereits global, Sie müssen sie also nicht zwischen Prozeduren übergeben.
8. In der Liste erscheinen häufig Funktionen, Konstanten, interne Funktionen, Prozeduren und Klassen.
9. Visual Basic gruppiert alle Mitglieder und Klassen nach ihrer Aufgabe und nicht alphabetisch.
10. Falsch. Eine der praktischsten Funktionen des Objektkatalogs ist die Möglichkeit, die Objekte Ihrer eigenen Applikationen zu finden.

## Übungen

1. Die `With`-Klausel erspart keinen Programmieraufwand, wenn nur zwei Eigenschaften gesetzt werden sollen.
2. Folgen Sie den Anweisungen in diesem Kapitel, um Ihrer Applikation ein Paint- Objekt hinzuzufügen (im Kapiteltext wurde diese Aufgabe anhand eines WordPad- Objekts demonstriert).

## Tag 17

### Quizantworten

1. Automatisierung ist der Prozeß, sich die Funktionalität einer anderen Applikation zu borgen und damit Datenobjekte dieser Applikation zu erzeugen.
2. Eine neue Word-Instanz wird gestartet, und die beiden redundanten Prozesse verbrauchen viele Systemressourcen.
3. Es ist nicht möglich, Applikationen in Variablen zu speichern; Sie müssen also mit Hilfe des `Set`-Befehls einen Verweis auf eine Applikation anlegen.
4. Wenn Sie mit der Anweisung `On Error` einen Fehler auffangen, können Sie anhand des Werts von `Err.Number` prüfen, welcher Fehler aufgetreten ist.

5. ActiveX-Steuerelemente werden durch Verwendung von Unterklassen bereits existierender Steuerelemente angelegt, mit Hilfe von Unterklassen mehrerer Steuerelemente oder völlig neu und benutzerdefiniert.
6. Am einfachsten ist es, ein ActiveX-Steuerelement anzulegen, indem man eine Unterklasse von einem anderen Steuerelement anlegt.
7. Richtig.
8. Aufzählungsblöcke definieren Aufzählungskonstanten.
9. Visual Basic verwendet die Dateinamenerweiterung .OCX für Ihre ActiveX-Steuerelemente.
10. ActiveX-Steuerelemente machen mindestens eine **Get**- und eine **Let**-Prozedur erforderlich, damit Sie Eigenschaftswerte zuweisen und lesen können.

## Tag 18

### Quizantworten

1. Der Visual Data Manager ermöglicht Ihnen die Analyse von Datenbanken.
2. Eine Tabelle ist eine Datendatei in einer Datenbank.
3. Falsch. Die Anzahl der Spalten wächst nicht, es sei denn, Sie ändern die Datenbankstruktur.
4. Richtig. Eine Tabelle ist eine Art Recordset.
5. Ein gebundenes Steuerelement ist ein Steuerelement, das einem Datenbank-Steuerelement zugeordnet ist, beispielsweise das Daten-Steuerelement, das Datensätze anzeigt, wenn der Benutzer die Datenbank durchläuft.
6. Ein Recordset besteht aus mehreren Datensätzen. Ein Dynaset besteht aus Datensätzen, die nicht die Standardreihenfolge aufweisen und beispielsweise bestimmte Kriterien erfüllen. Das Dynaset ändert sich, wenn sich die Datenbank ändert. Ein Snapshot ist ein Dynaset zu einem bestimmten Zeitpunkt. Mit anderen Worten, ein Snapshot ist ein Recordset, der sich nicht ändert, sondern nur Datensätze mit Datenbankwerten enthält, wie sie zum Zeitpunkt dieser Momentaufnahme ausgesehen haben.
7. ADO ist schneller, leistungsfähiger und flexibler als das Daten-Steuerelement.
8. EOF bezeichnet das Ende einer Tabelle, BOF den Anfang.
9. Eine Master-Ansicht ist ein Datensatz, eine Detail-Ansicht ist eine Datensatzmenge, die dem Master-Datensatz zuzuordnen sind.
10. Der Anwendungs-Assistent von Visual Basic erzeugt Formulare direkt aus Ihrer Datenbank-Tabellenstruktur.

# Übungen

1. Fügen Sie dem Formular keine zusätzlichen Daten-Steuerelemente hinzu. Sie brauchen nur weitere Textfelder, die Sie mit dem bereits vorhandenen Daten- Steuerelement verbinden. Setzen Sie die DataSource-Eigenschaft in jedem Textfeld auf das Daten-Steuerelement und die DataField-Eigenschaft für jedes Textfeld auf die entsprechende Spalte der Tabelle.
2. Um eine 1:n-Beziehung einzurichten, wählen Sie zwei Tabellen aus, die mindestens ein Feld gemeinsam haben. Wenn Sie Master/Detail auswählen, erhalten Sie im nächsten Fenster die Möglichkeit, die Tabelle auszuwählen, die als Master- Datensatzquelle verwendet werden soll. Als nächstes wird das Fenster für die Detail-Datensatzquelle angezeigt, wie in Abbildung A.1 gezeigt. Wählen Sie die entsprechenden Felder aus.

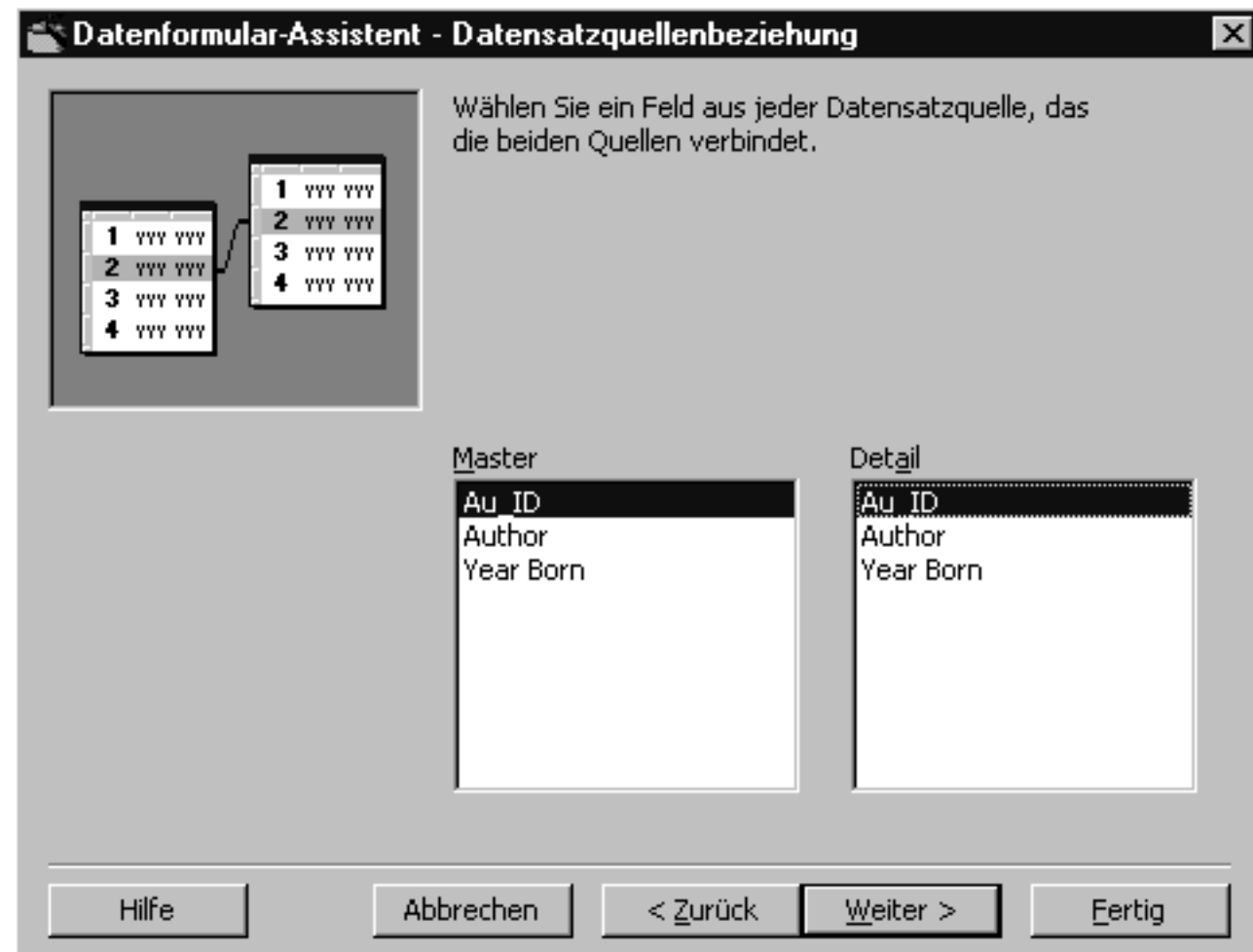


**Abbildung A.1:**  
Teilen Sie dem Assistenten mit, wie er die Felder verknüpfen soll.

## Tag 19

### Quizantworten

1. Der Web-Browser zeigt die Seite mit der URL an, die Sie für die Anmeldung im Internet vorgegeben haben.
2. Falsch. Ihre Applikationen enthalten einen eingebetteten Web-Browser, den der Benutzer verwenden kann, egal welche Web-Browser er auf seinem System installiert hat.

3. Richtig. Sie brauchen den Internet Explorer 4 auf Ihrem System, um die Web-Programmierung von Visual Basic nutzen zu können. Dieser Browser ist auch im Lieferumfang von Visual Basic enthalten. Sie müssen den Internet Explorer nicht zum Standard-Web-Browser machen, aber Visual Basic braucht ihn zur Entwicklung.
4. Die Kapselung beschreibt die Kombination von Code und Eigenschaften, so daß ein Objekt seine eigenen Verhaltensweisen und Beschreibungen mit sich bringt.
5. Einige der Steuerelemente von Visual Basic 6 bieten direkten Zugriff auf den Online-Dienst Microsoft Network. Wenn Ihre Benutzer jedoch keinen Zugang zu Microsoft Network besitzen, können sie sich auch über die Steuerelemente nicht dort anmelden.
6. Ein Intranet ist eine interne, Internet-ähnliche Verbindung zwischen vernetzten Computern.
7. Ein ActiveX-Dokument ist eine Web-Seite, die eine vollfunktionale Visual-Basic- Applikation enthält.
8. Java ist eine C++-ähnliche Programmiersprache, die den Entwicklern ermöglicht, kleine Programme, sogenannte Applets, zu schreiben, die auf den PC des Benutzers geladen werden, wenn dieser die betreffende Web-Seite besucht.
9. VBScript verwendet HTML, um ActiveX-Dokumente zu laden.
10. Mit dem ActiveX-Schnittstellen-Assistenten konvertieren Sie existierende Applikationen in ActiveX-Dokumente.

## Tag 20

### Quizantworten

1. Verwenden Sie RTF-Dateien (Rich Text Format) für Hilfe-Text und eine HPJ-Projektdatei, es sei denn, Sie verwenden HTML-Hilfe, dann erhält Ihre Hilfedatei die Dateinamenerweiterung .CHM.
2. Der Benutzer kann zwischen den Hilfethemen umherspringen, ohne zum Index zurückkehren zu müssen.
3. Die Fußnote K verbindet die Hilfe mit unterstrichenen Themen.
4. Die Hilfeprojektdatei identifiziert die Hilfedatei, die Kontext-IDs der Hilfeseite und den Text für die Titelleiste des Hilfefensters. Der Hilfe-Compiler verwendet diese Projektdatei zur Kompilierung der Hilfedatei.
5. Mit Hilfe des Dialogfelds **Projekteigenschaften** wird der (F1)-Taste Hilfe zugeordnet.
6. Mit Hilfe der Kontext-IDs werden kontextabhängigen Hilfe-Objekten Hilfe-Themen zugeordnet.
7. Falsch. Die kontextabhängigen IDs verwenden die numerischen Kontext-IDs.
8. ToolTips werden angezeigt, wenn der Benutzer den Mauszeiger über ein Element bewegt. Die Direkthilfe erscheint nur, wenn sie vom Benutzer explizit angefordert wird.
9. Setzen Sie die `WhatsThisButton`-Eigenschaft auf `True`, um die Schaltfläche für die Direkthilfe in der Symbolleiste anzulegen.

10. Richtig.

## Übungen

Erweitern Sie das Beispiel aus diesem Kapitel. Es ist ganz einfach, weitere Hilfethemen einzufügen. Gehen Sie dazu wie folgt vor:

1. Weisen Sie der `ToolTips`-Eigenschaft für jedes Objekt einen Wert zu.
2. Legen Sie für jedes Objekt auf dem Formular eine kontextabhängige Hilfe an.
3. Weisen Sie jedem Hilfethema der Projektdatei eine Kontext-ID zu.
4. Kompilieren Sie die Hilfedatei.

## Tag 21

### Quizantworten

1. Ein Syntaxfehler ist ein Schreibfehler oder eine fehlerhafte Verwendung von Sprachkonstrukten.
2. Der Computer erkennt Syntaxfehler. Die logischen Fehler müssen in der Regel vom Entwickler erkannt werden.
3. Richtig.
4. Falsch. Wenn eine Applikation einen Syntaxfehler enthält, führt Visual Basic sie nicht aus, bis der Fehler behoben ist.
5. Im Debugger ist es möglich, das Programm zu jedem beliebigen Zeitpunkt mit Hilfe von Haltepunkten anzuhalten. Wenn Sie den Mauszeiger über eine Variable schreiben, wird der Laufzeitwert angezeigt. Sie können diese Variable auch in das Überwachungsfenster einfügen.
6. Wenn Sie immer nur eine Programmzeile ausführen, können Sie die Variablen und Steuerelemente schrittweise auswerten und außerdem die Logik und den Programmfluß verfolgen.
7. Richtig. Im Direktfenster können Zuweisungen vorgenommen werden.
8. Ein kompiliertes Visual-Basic-Programm läuft viel schneller als ein Programm, das Sie in der Visual-Basic-Umgebung ausführen.
9. Der Paket- und Weitergabe-Assistent legt Installationsroutinen für Ihre Applikationen an.
10. Der Paket- und Weitergabe-Assistent erzeugt Installationsdateien, die auf verschiedenen Datenträgern bereitgestellt werden können.

## Übungen

1. Die Anweisung enthält einen Syntaxfehler, weil statt dem `Then` in der `If`-Anweisung ein `Therefore` verwendet wird.
2. Richtig. Der Satz enthält mit dem Schreibfehler nicht nur einen Syntaxfehler, sondern auch einen logischen Fehler: Er enthält nämlich nur einen Fehler. Damit ist auch die Logik des Satzes falsch.

# Bonustag 22

## Quizantworten

1. Falsch. Alle Elemente in jeder Zeile und jeder Spalte eines Arrays mit beliebig vielen Indizes müssen denselben Datentyp haben.
2. Der zweite Index (9) gibt in der Regel die Anzahl der Zeilen im mehrdimensionalen Array an.
3. a. 4  
b. 43  
c. 12
4. Richtig.
5. Falsch. `Array()` kann nur für eindimensionale Arrays eingesetzt werden.
6. Das Tabellen-Steuerelement zeigt die Daten einer Tabelle an.
7. Die festen Zeilen und Spalten sind die Titelzellen der Tabelle.
8. Verwenden Sie eine verschachtelte Schleife, um der Tabelle Werte zuzuweisen.
9. Falsch. Sie müssen die `CellPicture`-Eigenschaft zur Laufzeit mit der internen Funktion `LoadPicture()` setzen oder der `CellPicture`-Eigenschaft ein Anzeigefeld zuweisen.
10. `FormatString` ist einfacher zu verwenden und effizienter als die Zuweisungen für die Einrichtung der Titelzeilen einer Tabelle.

## Übungen

1. Es werden 28 Elemente reserviert.
2. Es werden 30 Elemente reserviert (beachten Sie den Index 0).
3. Am besten speichern Sie die Tabellenwerte innerhalb der `cmdExit_Click()`-Prozedur:

```
Private Sub cmdExit_Click()
' Applikation beenden und Werte speichern
Dim curData(19, 7) As Currency
' Tabelle füllen
For Row = 1 To 19
For Column = 1 To 7
grdSales.Row = Row
grdSales.Col = Column
curData(Row, Column) = grdSales(Row, Column)
Next Column
Next Row
' Programm beenden
End
End Sub
```



# Bonustag 23

## Quizantworten

1. API steht für Application Programming Interface.
2. Visual Basic beinhaltet viele Funktionen, kann aber nicht alles. Beispielsweise enthält es keine Funktion, die Ihr System neu bootet. Die Windows-API-Routinen bieten Funktionen auf Systemebene, auf die Sie von Ihrer Visual-Basic-Applikation aus zugreifen können.
3. DLLs werden zur Laufzeit dynamisch mit Ihrer Applikation verbunden, nicht beim Kompilieren. Das bedeutet, die Ressourcen für die DLLs werden vom Betriebssystem nicht für die gesamte Ausführungszeit Ihres Programms reserviert.
4. Als mit Windows 95 ein 32-Bit-Betriebssystem eingeführt wurde, änderten sich auch die Namen der DLL-Dateien. Um die neuen Dateien von den alten zu unterscheiden, fügte Microsoft »32« in die Namen ein (z.B. GDI32.DLL).
5. Der API-Viewer ist ein Werkzeug, das Sie Ihrer Visual-Basic-Umgebung hinzufügen (im **Add-Ins**-Menü), um API-Routinen und ihre Deklarationen anzuzeigen. Diese Deklarationen können in Ihre eigenen Applikationen übernommen werden.
6. Die `Declare`-Anweisung deklariert Windows-API-Routinen.
7. Falsch.
8. Die `Declare`-Anweisung informiert Visual Basic über die Position und die Argumenteliste der externen Windows-API-Routine. Visual Basic erkennt diese externen Routinen nicht, weil es eigene interne Funktionen besitzt, die sich von den Windows-Prozeduren unterscheiden. Die `Declare`-Anweisung ermöglicht Visual Basic, eine von Ihrer Applikation benötigte Windows-API-Routine zu finden und eine Verknüpfung dafür einzurichten.
9. Wenn Sie Windows-API-Routinen innerhalb eines Formularmoduls deklarieren, verwenden Sie das Schlüsselwort `Private`.
10. Der *Hüllcode* ist der Visual-Basic-Code, den sie um einen Windows-API-Aufruf herum schreiben.

## Übungen

1. Die Datei KERNEL32.DLL enthält die Funktion `GetSystemTime()`. Wählen Sie die Funktion im API-Viewer aus. Anhand des Arguments erkennen Sie die Datei, in der die Funktion enthalten ist.

# Tag B

## Anhang B: Operator- Prioritäten

Tabelle B.1 zeigt die Operator-Prioritäten. Die Operatoren werden dabei nach dem Operationstyp gruppiert.

**Tabelle B.1: Die Operatorreihenfolge in Visual Basic**

| Arithmetisch                       | Vergleich           | Logisch |
|------------------------------------|---------------------|---------|
| Exponentiation (^)                 | Gleich (=)          | Not     |
| Negation (-)                       | Ungleich (<>)       | And     |
| Multiplikation und Division (*, /) | Kleiner (<)         | Or      |
| Ganzzahlige Division (\)           | Größer (>)          | Xor     |
| Modulo (Mod)                       | Kleiner gleich (<=) | Eqv     |
| Addition und Subtraktion (+, -)    | Größer gleich (>=)  | Imp     |
| Stringkonkatenation (&)            | Like, Is            |         |

# Tag C

## Anhang C: ASCII-Tabelle

| Dezimal X10 | Hexadezimal X16 | Binär X2  | ASCII-Zeichen |
|-------------|-----------------|-----------|---------------|
| 000         | 00              | 0000 0000 | NULL          |
| 001         | 01              | 0000 0001 | !             |
| 002         | 02              | 0000 0010 | "             |
| 003         | 03              | 0000 0011 | #             |
| 004         | 04              | 0000 0100 | \$            |
| 005         | 05              | 0000 0101 | %             |
| 006         | 06              | 0000 0110 | &             |
| 007         | 07              | 0000 0111 | '             |
| 008         | 08              | 0000 1000 | (             |
| 009         | 09              | 0000 1001 | )             |
| 010         | 0A              | 0000 1010 | *             |
| 011         | 0B              | 0000 1011 | +             |
| 012         | 0C              | 0000 1100 | ,             |
| 013         | 0D              | 0000 1101 | -             |
| 014         | 0E              | 0000 1110 | .             |
| 015         | 0F              | 0000 1111 | /             |
| 016         | 10              | 0001 0000 | 0             |
| 017         | 11              | 0001 0001 | 1             |
| 018         | 12              | 0001 0010 | 2             |
| 019         | 13              | 0001 0011 | 3             |
| 020         | 14              | 0001 0100 | 4             |

|      |    |           |      |
|------|----|-----------|------|
| 021  | 15 | 0001 0101 | 5    |
| 022  | 16 | 0001 0110 | 6    |
| 023  | 17 | 0001 0111 | 7    |
| 024  | 18 | 0001 1000 | 8    |
| 025  | 19 | 0001 1001 | 9    |
| 026  | 1A | 0001 1010 | :    |
| 027  | 1B | 0001 1011 | ;    |
| 028  | 1C | 0001 1100 | <    |
| 029  | 1D | 0001 1101 | =    |
| 030  | 1E | 0001 1110 | >    |
| 031  | 1E | 0001 1111 | ?    |
| 032  | 20 | 0010 0000 | Leer |
| 033  | 21 | 0010 0001 | !    |
| 034  | 22 | 0010 0010 | "    |
| 0335 | 23 | 0010 0011 | #    |
| 036  | 24 | 0010 0100 | \$   |
| 037  | 25 | 0010 0101 | %    |
| 038  | 26 | 0010 0110 | &    |
| 039  | 27 | 0010 0111 | '    |
| 040  | 28 | 0010 1000 | (    |
| 041  | 29 | 0010 1001 | )    |
| 042  | 2A | 0010 1010 | *    |
| 043  | 2B | 0010 1011 | +    |
| 044  | 2C | 0010 1100 | '    |
| 045  | 2D | 0010 1101 | -    |
| 046  | 2E | 0010 1110 | .    |
| 047  | 2F | 0010 1111 | /    |
| 048  | 30 | 0011 0000 | 0    |
| 049  | 31 | 0011 0001 | 1    |
| 050  | 32 | 0011 0010 | 2    |
| 051  | 33 | 0011 0011 | 3    |
| 052  | 34 | 0011 0100 | 4    |

|     |    |           |   |
|-----|----|-----------|---|
| 053 | 35 | 0011 0101 | 5 |
| 054 | 36 | 0011 0110 | 6 |
| 055 | 37 | 0011 0111 | 7 |
| 056 | 38 | 0011 1000 | 8 |
| 057 | 39 | 0011 1001 | 9 |
| 058 | 3A | 0011 1010 | : |
| 059 | 3B | 0011 1011 | ; |
| 060 | 3C | 0011 1100 | < |
| 061 | 3D | 0011 1101 | = |
| 062 | 3E | 0011 1100 | > |
| 063 | 3F | 0011 1111 | ? |
| 064 | 40 | 0100 0000 | @ |
| 065 | 41 | 0100 0001 | A |
| 066 | 42 | 0100 0010 | B |
| 067 | 43 | 0100 0011 | C |
| 068 | 44 | 0100 0100 | D |
| 069 | 45 | 0100 0101 | E |
| 070 | 46 | 0100 0110 | F |
| 071 | 47 | 0100 0111 | G |
| 072 | 48 | 0100 1000 | H |
| 073 | 49 | 0100 1001 | I |
| 074 | 4A | 0100 1010 | J |
| 075 | 4B | 0100 1011 | K |
| 076 | 4C | 0100 1100 | L |
| 077 | 4D | 0100 1101 | M |
| 078 | 4E | 0100 1110 | N |
| 079 | 4F | 0100 1111 | O |
| 080 | 50 | 0101 0000 | P |
| 081 | 51 | 0101 0001 | Q |
| 082 | 52 | 0101 0010 | R |
| 083 | 53 | 0101 0011 | S |
| 084 | 54 | 0101 0100 | T |
|     |    |           |   |

|     |    |           |   |
|-----|----|-----------|---|
| 085 | 55 | 0101 0101 | U |
| 086 | 56 | 0101 0110 | V |
| 087 | 57 | 0101 0111 | W |
| 088 | 58 | 0101 1000 | X |
| 089 | 59 | 0101 1001 | Y |
| 090 | 5A | 0101 1010 | Z |
| 091 | 5B | 0101 1011 | [ |
| 092 | 5C | 0101 1100 | \ |
| 093 | 5D | 0101 1101 | ] |
| 094 | 5E | 0101 1110 | ^ |
| 095 | 5F | 0101 1111 | _ |
| 096 | 60 | 0110 0000 | ' |
| 097 | 61 | 0110 0001 | a |
| 098 | 62 | 0110 0010 | b |
| 099 | 63 | 0110 0011 | c |
| 100 | 64 | 0110 0100 | d |
| 101 | 65 | 0110 0101 | e |
| 102 | 66 | 0110 0110 | f |
| 103 | 67 | 0110 0111 | g |
| 104 | 68 | 0110 1000 | h |
| 105 | 69 | 0110 1001 | i |
| 106 | 6A | 0110 1010 | j |
| 107 | 6B | 0110 1011 | k |
| 108 | 6C | 0110 1100 | l |
| 109 | 6D | 0110 1101 | m |
| 110 | 6E | 0110 1110 | n |
| 111 | 6F | 0110 1111 | o |
| 112 | 70 | 0111 0000 | p |
| 113 | 71 | 0111 0001 | q |
| 114 | 72 | 0111 0010 | r |
| 115 | 73 | 0111 0011 | s |
| 116 | 74 | 0111 0100 | t |

|     |    |           |      |
|-----|----|-----------|------|
| 117 | 75 | 0111 0101 | u    |
| 118 | 76 | 0111 0110 | v    |
| 119 | 77 | 0111 0111 | w    |
| 120 | 78 | 0111 1000 | x    |
| 121 | 79 | 0111 1001 | y    |
| 122 | 7A | 0111 1010 | z    |
| 123 | 7B | 0111 1011 | {    |
| 124 | 7C | 0111 1100 |      |
| 125 | 7D | 0111 1101 | }    |
| 126 | 7E | 0111 1110 | ~    |
| 127 | 7F | 0111 1111 | \x7f |
| 128 | 80 | 1000 0000 | Ç    |
| 129 | 81 | 1000 0001 | ü    |
| 130 | 82 | 1000 0010 | é    |
| 131 | 83 | 1000 0011 | â    |
| 132 | 84 | 1000 0100 | ä    |
| 133 | 85 | 1000 0101 | à    |
| 134 | 86 | 1000 0110 | å    |
| 135 | 87 | 1000 0111 | ç    |
| 136 | 88 | 1000 1000 | ê    |
| 137 | 89 | 1000 1001 | ë    |
| 138 | 8A | 1000 1010 | è    |
| 139 | 8B | 1000 1011 | ï    |
| 140 | 8C | 1000 1100 | î    |
| 141 | 8D | 1000 1101 | ì    |
| 142 | 8E | 1000 1110 | Ä    |
| 143 | 8F | 1000 1111 | Å    |
| 144 | 90 | 1001 0000 | É    |
| 145 | 91 | 1001 0001 | æ    |
| 146 | 92 | 1001 0010 | Æ    |
| 147 | 93 | 1001 0011 | ô    |
| 148 | 94 | 1001 0100 | ö    |

|     |    |           |      |
|-----|----|-----------|------|
| 149 | 95 | 1001 0101 | ò    |
| 150 | 96 | 1001 0110 | û    |
| 151 | 97 | 1001 0111 | ù    |
| 152 | 98 | 1001 1000 | ÿ    |
| 153 | 99 | 1001 1001 | Ö    |
| 154 | 9A | 1001 1010 | Ü    |
| 155 | 9B | 1001 1011 |      |
| 156 | 9C | 1001 1100 |      |
| 157 | 9D | 1001 1101 | ¥    |
| 158 | 9E | 1001 1110 | \xfe |
| 159 | 9F | 1001 1111 | \xd9 |
| 160 | A0 | 1010 0000 | á    |
| 161 | A1 | 1010 0001 | ì    |
| 162 | A2 | 1010 0010 | ¢    |
| 163 | A3 | 1010 0011 | £    |
| 164 | A4 | 1010 0100 | ¤    |
| 165 | A5 | 1010 0101 | ¥    |
| 166 | A6 | 1010 0110 | ¦    |
| 167 | A7 | 1010 0111 | §    |
| 168 | A8 | 1010 1000 | ¨    |
| 169 | A9 | 1010 1001 | ©    |
| 170 | AA | 1010 1010 | ª    |
| 171 | AB | 1010 1011 | «    |
| 172 | AC | 1010 1100 | ¬    |
| 173 | AD | 1010 1101 | \x7f |
| 174 | AE | 1010 1110 | ®    |
| 175 | AF | 1010 1111 | ¯    |
| 176 | B0 | 1011 0000 | °    |
| 177 | B1 | 1011 0001 | ±    |
| 178 | B2 | 1011 0010 | ²    |
| 179 | B3 | 1011 0011 | ³    |
| 180 | B4 | 1011 0100 | ´    |



|     |    |           |   |
|-----|----|-----------|---|
| 181 | B5 | 1011 0101 | μ |
| 182 | B6 | 1011 0110 | ¶ |
| 183 | B7 | 1011 0110 | · |
| 184 | B8 | 1011 1000 | „ |
| 185 | B9 | 1011 1001 | ¹ |
| 186 | BA | 1011 1010 | º |
| 187 | BB | 1011 1011 | » |
| 188 | BC | 1011 1100 | ¼ |
| 189 | BD | 1011 1101 | ½ |
| 190 | BE | 1011 1110 | ¾ |
| 191 | BF | 1011 1111 | ¿ |
| 192 | C0 | 1100 0000 | À |
| 193 | C1 | 1100 0001 | Á |
| 194 | C2 | 1100 0010 | Â |
| 195 | C3 | 1100 0011 | Ã |
| 196 | C4 | 1100 0100 | Ä |
| 197 | C5 | 1100 0101 | Å |
| 198 | C6 | 1100 0110 | Æ |
| 199 | C7 | 1100 0111 | Ç |
| 200 | C8 | 1100 1000 | + |
| 201 | C9 | 1100 1001 | É |
| 202 | CA | 1100 1010 | Ê |
| 203 | CB | 1100 1011 | Ë |
| 204 | CC | 1100 1100 | Ì |
| 205 | CD | 1100 1101 | Í |
| 206 | CE | 1100 1110 | Î |
| 207 | CF | 1100 1111 | Ï |
| 208 | D0 | 1101 0000 | Ð |
| 209 | D1 | 1101 0001 | Ñ |
| 210 | D2 | 1101 0010 | Ò |
| 211 | D3 | 1101 0011 | Ó |
| 212 | D4 | 1101 0100 | Ô |
|     |    |           |   |

|     |    |           |   |
|-----|----|-----------|---|
| 213 | D5 | 1101 0101 | Õ |
| 214 | D6 | 1101 0110 | Ö |
| 215 | D7 | 1101 0111 | × |
| 216 | D8 | 1101 1000 | Ø |
| 217 | D9 | 1101 1001 | Ù |
| 218 | DA | 1101 1010 | Ú |
| 219 | DB | 1101 1011 | Û |
| 220 | DC | 1101 1100 | Ü |
| 221 | DD | 1101 1101 | Ý |
| 222 | DE | 1101 1110 | Þ |
| 223 | DF | 1101 1111 | ß |
| 224 | E0 | 1110 0000 | à |
| 225 | E1 | 1110 0001 | á |
| 226 | E2 | 1110 0010 | â |
| 227 | E3 | 1110 0011 | ã |
| 228 | E4 | 1110 0100 | ä |
| 229 | E5 | 1110 0101 | å |
| 230 | E6 | 1110 0110 | æ |
| 231 | E7 | 1110 0111 | ç |
| 232 | E8 | 1110 1000 | è |
| 233 | E9 | 1110 1001 | é |
| 234 | EA | 1110 1010 | ê |
| 235 | EB | 1110 1011 | ë |
| 236 | EC | 1110 1100 | ì |
| 237 | ED | 1110 1101 | í |
| 238 | EE | 1110 1110 | î |
| 239 | EF | 1110 1111 | ï |
| 240 | F0 | 1110 0000 | ð |
| 241 | F1 | 1111 0001 | ñ |
| 242 | F2 | 1111 0010 | ò |
| 243 | F3 | 1111 0011 | ó |
| 244 | F4 | 1111 0100 | ô |

|     |    |           |   |
|-----|----|-----------|---|
| 245 | F5 | 1111 0101 | õ |
| 246 | F6 | 1111 0110 | ö |
| 247 | F7 | 1111 0111 | ÷ |
| 248 | F8 | 1111 1000 | ø |
| 249 | F9 | 1111 1001 | ù |
| 250 | FA | 1111 1010 | ú |
| 251 | FB | 1111 1011 | û |
| 252 | FC | 1111 1100 | ü |
| 253 | FD | 1111 1101 | ý |
| 254 | FE | 1111 1110 | þ |
| 255 | FF | 1111 1111 | ÿ |

## Tag D

# Anhang D: Die CD zum Buch

Die Service-CD-ROM, die diesem Buch beiliegt, hat drei Unterverzeichnisse:

Unter EBOOKS finden Sie dieses Ihnen vorliegende Buch komplett im HTML-Format. So können Sie z.B. eine Lektion auch mal am Laptop durcharbeiten. Daneben gibt's zwei andere Titel aus dieser Reihe als Bonus: *Active Server Pages in 21 Tagen* und *Visual InterDev in 21 Tagen*.

Die Listings im Unterverzeichnis SOURCE sind getrennt nach Tagen bzw. Anhängen in den MS-Word-Dateien (tag\_nn.doc) untergebracht, wobei nn für die Nummer des Tages (von 01 bis 21) bzw. den Anhang (AA bis AD) steht. Innerhalb der Dateien sind die Listings fortlaufend aufgeteilt. Bei einigen Tagen sind auch die Listings der Übungen enthalten.

Eine aktuelle Visual-Basic-Version zum Üben und Lernen finden Sie im Unterverzeichnis VB6. Achtung: Die CD mit der deutschsprachigen Version des Visual Basic 6 Working Model wurde unter festgelegten Lizenzbestimmungen vom Markt&Technik- Verlag produziert. Bitte beachten Sie, daß Microsoft keinerlei Garantie-, Hilfe- oder Support-Leistung für dieses Produkt bereitstellt. Als Anwender dieser Version sind Sie weder lizenzierter Nutzer von Produkten der Firma Microsoft noch haben Sie Anrecht auf Updates oder andere Vergünstigungen. Sollte die CD fehlen oder beschädigt sein, wenden Sie sich bitte an den Markt&Technik-Verlag. SENDEN SIE DIE CD AUF KEINEN FALL AN MICROSOFT.

Sie erhalten damit eine vollständige Visual-Basic-Version, mit der Sie programmieren lernen und Ihre Programme entwickeln können. Nicht erstellen lassen sich lediglich ausführbare Dateien (EXE) und ActiveX-Steuerlemente.

Das Verzeichnis DIENSTE schließlich bietet Ihnen u.a. aktuelle Browser-Software.

---

## Stichwortverzeichnis

!

- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

-

- [Visual Basic - Grundlagen der Sprache](#)

#

- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

%

- [So funktionieren komplexe Programme](#)

&

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

\*

- [Visual Basic - Grundlagen der Sprache](#)

+

- [Visual Basic - Grundlagen der Sprache](#)

, abgerundete

- [Grafik und Multimedia](#)

.AVI

- [Grafik und Multimedia](#)

.BAS

- [Erstes Arbeiten mit Visual Basic](#)

.BMP

- [Grafik und Multimedia](#)

.CHM

- [Ein Hilfe- System erstellen](#)

.CLS

- [Erstes Arbeiten mit Visual Basic](#)

## **.CUR**

- [Grafik und Multimedia](#)

## **.DOB**

- [Erstes Arbeiten mit Visual Basic](#)

## **.EMF**

- [Grafik und Multimedia](#)

## **.FRM**

- [Erstes Arbeiten mit Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)

## **.GIF**

- [Grafik und Multimedia](#)

## **.HLP**

- [Ein Hilfe- System erstellen](#)

## **.ICO**

- [Grafik und Multimedia](#)

## **.JPEG**

- [Grafik und Multimedia](#)

## **.JPG**

- [Grafik und Multimedia](#)

## **.MDB**

- [Auf Daten zugreifen](#)

## **.MID**

- [Grafik und Multimedia](#)

## **.OCX**

- [Erstes Arbeiten mit Visual Basic](#)

## **.PAG**

- [Erstes Arbeiten mit Visual Basic](#)

## **.RLE**

- [Grafik und Multimedia](#)

## **.VBP**

- [Erstes Arbeiten mit Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)

## **.WAV**

- [Grafik und Multimedia](#)

## **.WMF**

- [Grafik und Multimedia](#)

/

- [So funktionieren komplexe Programme](#)

<

- [Programme steuern](#)
- [So funktionieren komplexe Programme](#)

<=

- [Programme steuern](#)

<>

- [Programme steuern](#)

=

- [Programme steuern](#)

>

- [Programme steuern](#)
- [So funktionieren komplexe Programme](#)

>=

- [Programme steuern](#)

@

- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

^

- [Visual Basic - Grundlagen der Sprache](#)

0

- [So funktionieren komplexe Programme](#)

### **1:1-Beziehung**

- [Auf Daten zugreifen](#)

### **1:n-Beziehung**

- [Auf Daten zugreifen](#)

A

### **Abbildungsliste**

- [Formulare](#)

### **Abbrechen**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

### **Abfrage**

- [Auf Daten zugreifen](#)

### **Abs()**

- [So funktionieren komplexe Programme](#)

## **Absolutwert**

- [So funktionieren komplexe Programme](#)

## **AccessRestriction**

- [Mit Dateien arbeiten](#)

## **Activate**

- [Formulare](#)

Ereignis

- [Steuerelemente - Verwaltung](#)

## **ActiveX**

- [Erstes Arbeiten mit Visual Basic](#)
- [ActiveX- Steuerelemente](#)

## **ActiveX Data Objects**

- [Auf Daten zugreifen](#)

## **ActiveX Dokument DLL**

- [Internet-Zugriff](#)

## **ActiveX Dokument EXE**

- [Internet-Zugriff](#)

## **ActiveX-Automatisierung**

- [ActiveX- Steuerelemente](#)

## **ActiveX-Dokument-Assistent**

- [Internet-Zugriff](#)

## **ActiveX-Dokumente**

- [Internet-Zugriff](#)

## **ActiveX-Schnittstellen-Assistent**

- [ActiveX- Steuerelemente](#)

## **ActiveX-Steuerelement**

- [Erstes Arbeiten mit Visual Basic](#)
- [Dialogfelder](#)
- [ActiveX- Steuerelemente](#)
- [ActiveX- Steuerelemente](#)

## **ActiveX-Technologie**

- [ActiveX- Steuerelemente](#)

## **Add, Auflistung**

- [Objekte in Visual Basic](#)

## **Add-In-Manager**

- [ActiveX- Steuerelemente](#)

## **AddItem-Methode**



- [Mehr Power mit Maus und Steuerelementen](#)

## **Addition**

- [Visual Basic - Grundlagen der Sprache](#)

## **ADO-DataCombo-Steuerelement**

- [Auf Daten zugreifen](#)

## **ADO-Daten-Steuerelement**

- [Auf Daten zugreifen](#)
- [Auf Daten zugreifen](#)

## **ADO-Objekte**

- [Auf Daten zugreifen](#)

## **Alignment**

Eigenschaft

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

## **AllowBigSelection**

- [Mehrdimensionale Arrays](#)

## **AllowCustomize-Eigenschaft**

- [Formulare](#)

## **AllowUserResizing**

- [Mehrdimensionale Arrays](#)

## **Am Raster ausrichten**

- [Erstes Arbeiten mit Visual Basic](#)

## **Ampersand**

- [Willkommen bei Visual Basic](#)

## **Ampersand-Zeichen**

- [Steuerelemente - Verwaltung](#)

## **And**

- [Programme steuern](#)

## **Anmelde-Dialogfeld**

- [Formularvorlagen](#)
- [Formularvorlagen](#)

## **Anmerkung**

- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

## **Anweisung**

verschachtelte

- [Programme steuern](#)

## **Anwendungs-Assistent**

- [Willkommen bei Visual Basic](#)

- [Formulare](#)

Menüs

- [Menüs](#)

## **ANY**

- [Das Windows-API](#)

## **Anzeige, Steuerelement**

- [Erstes Arbeiten mit Visual Basic](#)

## **Anzeigefeld**

- [Steuerelemente - Verwaltung](#)
- [Grafik und Multimedia](#)

## **API, aufrufen**

- [Das Windows-API](#)

## **Apiload.txt**

- [Das Windows-API](#)

## **API-Typ**

- [Das Windows-API](#)

## **API-Viewer**

- [Das Windows-API](#)

## **App**

- [Objekte in Visual Basic](#)

## **Append**

- [Mit Dateien arbeiten](#)

## **Applets**

- [Internet-Zugriff](#)

## **Application Programming Interface**

- [Applikationen veröffentlichen](#)
- [Das Windows-API](#)

## **App-Objekt**

- [Formularvorlagen](#)

## **Argument**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Arkustangens**

- [So funktionieren komplexe Programme](#)

## **Array**

- [Mehr Power mit Maus und Steuerelementen](#)
- dynamische

- [Mehr Power mit Maus und Steuerelementen](#)

eindimensionales

- [Mehr Power mit Maus und Steuerelementen](#)

initialisieren

- [Mehrdimensionale Arrays](#)

mehrdimensionale

- [Mehrdimensionale Arrays](#)

mehrdimensionale Deklaration

- [Mehrdimensionale Arrays](#)

mehrdimensionales

- [Mehr Power mit Maus und Steuerelementen](#)

parallele

- [Mehr Power mit Maus und Steuerelementen](#)

statische

- [Mehr Power mit Maus und Steuerelementen](#)

## **Array()**

- [Mehrdimensionale Arrays](#)

Funktion

- [Mehr Power mit Maus und Steuerelementen](#)

## **Arraydeklaration**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Asc()**

- [So funktionieren komplexe Programme](#)

## **ASCII-Code**

- [So funktionieren komplexe Programme](#)

## **ASCII-Funktionen**

- [So funktionieren komplexe Programme](#)

## **Assistenten**

- [Willkommen bei Visual Basic](#)

## **Atn()**

- [So funktionieren komplexe Programme](#)

## **ATOM**

- [Das Windows-API](#)

## **Audio Visual Interleave**

- [Grafik und Multimedia](#)

## **Audiodatei-Player**

- [Grafik und Multimedia](#)

## **Audiogeräte**

- [Grafik und Multimedia](#)

## **Auflistung**

- [Objekte in Visual Basic](#)

Add

- [Objekte in Visual Basic](#)

Count

- [Objekte in Visual Basic](#)

Item

- [Objekte in Visual Basic](#)

Remove

- [Objekte in Visual Basic](#)

## **Aufruf**

- [So funktionieren komplexe Programme](#)

## **Aufrufliste**

- [Applikationen veröffentlichen](#)

## **Ausführen**

- [Willkommen bei Visual Basic](#)

bis Cursor

- [Applikationen veröffentlichen](#)

## **Ausgabe**

Formatieren

- [Drucken](#)

numerische

Formatieren

- [So funktionieren komplexe Programme](#)

skalieren

- [Drucken](#)

## **Ausgabesteuerung**

- [Drucken](#)

## **Ausrichtung, Steuerelement**

- [Steuerelemente - Verwaltung](#)

## **Automatisierung**

- [ActiveX- Steuerelemente](#)

## **AutoSize, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **BackColor**

- [Formulare](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **Backspace**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **BackStyle**

- [Grafik und Multimedia](#)

## **Band**

- [Formulare](#)

## **BandBorders**

- [Formulare](#)

## **Band-Objekte**

- [Formulare](#)

## **BASIC**

- [Willkommen bei Visual Basic](#)

## **Baumansicht**

- [Formulare](#)

## **Bearbeiten, Symbolleiste**

- [Erstes Arbeiten mit Visual Basic](#)

## **Beenden**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Beep**

- [Programme steuern](#)

## **Befehle, dateibezogene**

- [Mit Dateien arbeiten](#)

## **Befehlsschaltfläche**

- [Steuerelemente - Verwaltung](#)

Eingabetaste

- [Steuerelemente - Verwaltung](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Begrüßungsbildschirm**

- [Formularvorlagen](#)

- [Formularvorlagen](#)

## **Benutzer**

- [Willkommen bei Visual Basic](#)

## **Benutzerdefiniert**

- [Dialogfelder](#)

## **Benutzerdokumente**

- [Erstes Arbeiten mit Visual Basic](#)

## **Benutzeroberfläche, grafische**

- [Willkommen bei Visual Basic](#)

## **Benutzersteuerelemente**

- [Erstes Arbeiten mit Visual Basic](#)

## **Bericht anzeigen**

- [Willkommen bei Visual Basic](#)

## **Bezeichnungsfeld**

- [Erstes Arbeiten mit Visual Basic](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Bild**

Anzeige deaktivieren

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

Farbauflösung

- [Grafik und Multimedia](#)

## **Bildfeld, Steuerelement**

- [Erstes Arbeiten mit Visual Basic](#)

## **Bildfelder**

- [Grafik und Multimedia](#)

## **Bildlaufleiste**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Bonusprojekt 7: Bildlaufleisten](#)

## **Binärdateien**

- [Mit Dateien arbeiten](#)

## **Binary**

- [Mit Dateien arbeiten](#)

## **Bindungstyp**

- [Auf Daten zugreifen](#)

## **Bitmap**

- [Grafik und Multimedia](#)

laden

- [ActiveX- Steuerelemente](#)

## **BOOL**

- [Das Windows-API](#)

## **Boolean**

- [Visual Basic - Grundlagen der Sprache](#)

## **BorderColor**

- [Grafik und Multimedia](#)
- [Grafik und Multimedia](#)

## **BorderStyle**

- [Grafik und Multimedia](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

Formulareigenschaft

- [Steuerelemente - Verwaltung](#)

Linie

- [Grafik und Multimedia](#)

## **BorderWidth**

- [Grafik und Multimedia](#)
- [Grafik und Multimedia](#)

## **Browser**

- [Willkommen bei Visual Basic](#)

## **Bug**

- [Willkommen bei Visual Basic](#)

## **ByRef**

- [So funktionieren komplexe Programme](#)

## **Byte**

- [Visual Basic - Grundlagen der Sprache](#)

## **ByVal**

- [So funktionieren komplexe Programme](#)

## **C**

## **Call**

- [So funktionieren komplexe Programme](#)

## **Cancel, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Caption**

Eigenschaft

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

## Case

- [Programme steuern](#)

## cbo

- [Erstes Arbeiten mit Visual Basic](#)

## CBool()

- [So funktionieren komplexe Programme](#)

## CByte()

- [So funktionieren komplexe Programme](#)

## CCur()

- [So funktionieren komplexe Programme](#)

## CD, Spuren

- [Grafik und Multimedia](#)

## CDate()

- [So funktionieren komplexe Programme](#)

## CDAudio

- [Grafik und Multimedia](#)

## CDBl()

- [So funktionieren komplexe Programme](#)

## CDec()

- [So funktionieren komplexe Programme](#)

## cdlCCFullOpen

- [Dialogfelder](#)

## cdlCCHelpButton

- [Dialogfelder](#)

## cdlCCPreventFullOpen

- [Dialogfelder](#)

## cdlCCRGBInit

- [Dialogfelder](#)

## cdlCFANSIOnly

- [Dialogfelder](#)

## cdlCFApply

- [Dialogfelder](#)

## cdlCFBoth

- [Dialogfelder](#)



## **cdlCFEffects**

- [Dialogfelder](#)

## **cdlCFFixedPitchOnly**

- [Dialogfelder](#)

## **cdlCFForceFontExist**

- [Dialogfelder](#)

## **cdlCFHelpButton**

- [Dialogfelder](#)

## **cdlCFLimitSize**

- [Dialogfelder](#)

## **cdlCFNoFaceSel**

- [Dialogfelder](#)

## **cdlCFNoSimulations**

- [Dialogfelder](#)

## **cdlCFNoSizeSel**

- [Dialogfelder](#)

## **cdlCFNoStyleSel**

- [Dialogfelder](#)

## **cdlCFNoVectorFonts**

- [Dialogfelder](#)

## **cdlCFPrinterFonts**

- [Dialogfelder](#)

## **cdlCFScalableOnly**

- [Dialogfelder](#)

## **cdlCFScreenFonts**

- [Dialogfelder](#)

## **cdlCFTTOnly**

- [Dialogfelder](#)

## **cdlCFWYSIWYG**

- [Dialogfelder](#)

## **cdlOFNAllowMultiselect**

- [Dialogfelder](#)

## **cdlOFNCreatePrompt**

- [Dialogfelder](#)

## **cdlOFNExplorer**

- [Dialogfelder](#)

## **cdlOFNExtensionDifferent**

- [Dialogfelder](#)

## **cdIOFNFileMustExist**

- [Dialogfelder](#)

## **cdIOFNHelpButton**

- [Dialogfelder](#)

## **cdIOFNHideReadOnly**

- [Dialogfelder](#)

## **cdIOFNLongNames**

- [Dialogfelder](#)

## **cdIOFNNoChangeDir**

- [Dialogfelder](#)

## **cdIOFNNoDereferenceLinks**

- [Dialogfelder](#)

## **cdIOFNNoLongNames**

- [Dialogfelder](#)

## **cdIOFNNoReadOnlyReturn**

- [Dialogfelder](#)

## **cdIOFNNoValidate**

- [Dialogfelder](#)

## **cdIOFNOverwritePrompt**

- [Dialogfelder](#)

## **cdIOFNPathMustExist**

- [Dialogfelder](#)

## **cdIOFNReadOnly**

- [Dialogfelder](#)

## **cdIOFNShareAware**

- [Dialogfelder](#)

## **CD-Player**

- [Grafik und Multimedia](#)

## **CellPicture**

- [Mehrdimensionale Arrays](#)

## **Change, Ereignis**

- [Steuerelemente - Verwaltung](#)

## **CHAR**

- [Das Windows-API](#)

## **ChDir**

- [Mit Dateien arbeiten](#)

## **ChDrive**

- [Mit Dateien arbeiten](#)

## **CheckBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **Checked**

- [Menüs](#)
- [Menüs](#)

## **chk**

- [Erstes Arbeiten mit Visual Basic](#)

## **chkLoadTipsAtStartup()**

- [Formularvorlagen](#)

## **Choose()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **Chr()**

- [So funktionieren komplexe Programme](#)

## **CInt()**

- [So funktionieren komplexe Programme](#)

## **Circle-Methode**

- [Grafik und Multimedia](#)

## **Clear**

- [Objekte in Visual Basic](#)

## **Clear-Methode**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Click**

- [Mehr Power mit Maus und Steuerelementen](#)

Ereignis

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

## **Clipboard**

- [Objekte in Visual Basic](#)

## **CLng()**

- [So funktionieren komplexe Programme](#)

## **Close**

- [Mit Dateien arbeiten](#)

## **cmd**

- [Erstes Arbeiten mit Visual Basic](#)

## **Code**

- [Willkommen bei Visual Basic](#)

anzeigen

- [Erstes Arbeiten mit Visual Basic](#)

automatische Eingabe

- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

## **Code-Label**

- [Dialogfelder](#)

## **Collection**

- [Objekte in Visual Basic](#)

## **COLOREF**

- [Das Windows-API](#)

## **ColorMode**

- [Drucken](#)

## **Cols**

- [Mehrdimensionale Arrays](#)

## **ComboBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **Command1**

- [Erstes Arbeiten mit Visual Basic](#)

## **CommandButton**

- [Erstes Arbeiten mit Visual Basic](#)

## **Command-Eigenschaft**

- [Grafik und Multimedia](#)

## **Computeruhr**

- [Mehr Power mit Maus und Steuerelementen](#)

## **ConnectionString**

- [Bonusprojekt 9: ADO-Steuerelemente](#)

## **Container-Steuerelement**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)
- [Objekte in Visual Basic](#)

## **ControlBox, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Coolbars**

- [Formulare](#)
- [Formulare](#)

## **Copies**

- [Drucken](#)

## **Cos()**

- [So funktionieren komplexe Programme](#)

## **Cosinus**

- [So funktionieren komplexe Programme](#)

## **Count, Auflistung**

- [Objekte in Visual Basic](#)

## **Count-Eigenschaft**

- [Formulare](#)

## **CreateObject()**

- [ActiveX- Steuerelemente](#)

## **CSng()**

- [So funktionieren komplexe Programme](#)

## **CStr()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **CurDir()**

- [Mit Dateien arbeiten](#)

## **Currency**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **CurrentX**

- [Formulare](#)
- [Drucken](#)

## **CurrentY**

- [Formulare](#)
- [Drucken](#)

## **Cursor**

- [Steuerelemente - Verwaltung](#)
- [Grafik und Multimedia](#)

## **Custom**

- [Mehrdimensionale Arrays](#)

## **CVar()**

- [So funktionieren komplexe Programme](#)

# **D**

## **DAT**

- [Grafik und Multimedia](#)

## **Data**

- [Erstes Arbeiten mit Visual Basic](#)

## **DatabaseName**

- [Auf Daten zugreifen](#)

## **DataField**

- [Auf Daten zugreifen](#)

## **Date**

- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

## **DateAdd()**

- [So funktionieren komplexe Programme](#)

## **DateDiff()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **Datei**

### Befehle

- [Mit Dateien arbeiten](#)

### öffnen

- [Dialogfelder](#)
- [Dialogfelder](#)
- [Mit Dateien arbeiten](#)

### schließen

- [Mit Dateien arbeiten](#)

### schreiben

- [Mit Dateien arbeiten](#)

### sequentiell lesen

- [Mit Dateien arbeiten](#)

### sequentielle

- [Mit Dateien arbeiten](#)

### sperren

- [Mit Dateien arbeiten](#)

## **Datei-Handle**

- [Mit Dateien arbeiten](#)

## **Datei-I/O**

- [Mit Dateien arbeiten](#)

## **Dateilistenfeld**

- [Mit Dateien arbeiten](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Dateimodi**

- [Mit Dateien arbeiten](#)

## **Dateinummer**

- [Mit Dateien arbeiten](#)

freie

- [Mit Dateien arbeiten](#)

## **Datei-Steuerelemente**

- [Mit Dateien arbeiten](#)

## **Dateizugriff**

- [Mit Dateien arbeiten](#)

- [Mit Dateien arbeiten](#)

sequentieller

- [Mit Dateien arbeiten](#)

wahlfreier

- [Mit Dateien arbeiten](#)

## **Daten**

- [Mit Dateien arbeiten](#)

numerische

- [Visual Basic - Grundlagen der Sprache](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Datenbank**

zuordnen

- [Auf Daten zugreifen](#)

## **Datenbankdaten**

- [Auf Daten zugreifen](#)

## **Datenbankzugriff**

- [Formularvorlagen](#)

## **Datenformular**

- [Formularvorlagen](#)

## **Datensatz**

- [Mit Dateien arbeiten](#)

## **Datensatzlänge**

- [Mit Dateien arbeiten](#)

## **Daten-Steuererelement**

- [Auf Daten zugreifen](#)

## **Datentyp**

- [Visual Basic - Grundlagen der Sprache](#)

benutzerdefinierte

- [Mit Dateien arbeiten](#)

benutzerdefinierter

- [Mit Dateien arbeiten](#)

einbetten

- [Mit Dateien arbeiten](#)

Suffix

- [Visual Basic - Grundlagen der Sprache](#)

## **Datentypfunktionen**

- [So funktionieren komplexe Programme](#)

## **Datentypumwandlung**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **Datenübergabe**

- [So funktionieren komplexe Programme](#)

## **DatePart()**

- [So funktionieren komplexe Programme](#)

## **DateSerial()**

- [So funktionieren komplexe Programme](#)

## **DateValue()**

- [So funktionieren komplexe Programme](#)

## **Datumsfunktionen**

- [So funktionieren komplexe Programme](#)

## **Day()**

- [So funktionieren komplexe Programme](#)

## **DblClick**

- [Mehr Power mit Maus und Steuerelementen](#)

Ereignis

- [Steuerelemente - Verwaltung](#)

## **Deactivate**

- [Formulare](#)

Ereignis

- [Steuerelemente - Verwaltung](#)



## **Debug**

- [Objekte in Visual Basic](#)

## **Debug.Print**

- [Applikationen veröffentlichen](#)

## **Debug-Fenster**

- [Applikationen veröffentlichen](#)

## **Debuggen-Symbolleiste**

- [Applikationen veröffentlichen](#)

## **Debugger**

- [Willkommen bei Visual Basic](#)
- [Applikationen veröffentlichen](#)

## **Debugging**

- [Willkommen bei Visual Basic](#)

Symbolleiste

- [Erstes Arbeiten mit Visual Basic](#)

## **Debugging-Werkzeuge**

- [Applikationen veröffentlichen](#)

## **Decimal**

- [Visual Basic - Grundlagen der Sprache](#)

## **Declare**

- [Das Windows-API](#)

## **Default, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Definition anzeigen**

- [Objekte in Visual Basic](#)

## **Deklaration**

- [Visual Basic - Grundlagen der Sprache](#)  
zusammengesetzte
  - [Mit Dateien arbeiten](#)

## **Deklarationsbereich**

- [Visual Basic - Grundlagen der Sprache](#)

## **DeviceName**

- [Drucken](#)

## **DeviceType**

- [Grafik und Multimedia](#)

## **Dezimalzahlen**

- [Visual Basic - Grundlagen der Sprache](#)

## **DHTML**

- [Internet-Zugriff](#)  
Anwendungs-Assistent
  - [Internet-Zugriff](#)

## **Dialog**

- [Formularvorlagen](#)

## **Dialogfelder**

- [Dialogfelder](#)

## **DialogTitel**

- [Dialogfelder](#)

## **DialogTitle**

- [Dialogfelder](#)

## **DigitalVideo**

- [Grafik und Multimedia](#)

## **Dim**

- [Visual Basic - Grundlagen der Sprache](#)

## **dir**

- [Erstes Arbeiten mit Visual Basic](#)

## **Dir()**

- [Mit Dateien arbeiten](#)

## **Direktfenster**

- [Applikationen veröffentlichen](#)

## **Direkthilfe**

- [Ein Hilfe- System erstellen](#)
- [Ein Hilfe- System erstellen](#)

## **DirListBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **DispalyCurrentTip**

- [Formularvorlagen](#)

## **Division**

- [Visual Basic - Grundlagen der Sprache](#)  
ganzzahlige
  - [Visual Basic - Grundlagen der Sprache](#)

## **DLL**

- [Internet-Zugriff](#)
- [Das Windows-API](#)

## **Dokumentation**

- [Willkommen bei Visual Basic](#)

## **Dokumentfenster**

- [Willkommen bei Visual Basic](#)
- [Formulare](#)

## **DoNextTip()**

- [Formularvorlagen](#)

## **Doppelklick**

- [Steuerelemente - Verwaltung](#)

## **Do-Schleife**

- [Programme steuern](#)

## **Double**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **DPI**

- [Drucken](#)

## **Drag&Drop**

automatisches

- [Bonusprojekt 5: Die Übung mit der Maus](#)

manuelles

- [Mehr Power mit Maus und Steuerelementen](#)
- [Bonusprojekt 5: Die Übung mit der Maus](#)

## **Drag&Drop-Operationen**

- [Mehr Power mit Maus und Steuerelementen](#)

## **DragDrop-Ereignis**

- [Mehr Power mit Maus und Steuerelementen](#)

## **DragIcon**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Bonusprojekt 5: Die Übung mit der Maus](#)

## **Drag-Methode**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Bonusprojekt 5: Die Übung mit der Maus](#)

## **DragMode**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Bonusprojekt 5: Die Übung mit der Maus](#)

## **DragOver-Ereignis**

- [Mehr Power mit Maus und Steuerelementen](#)

## **DriveListBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **DriverName**

- [Drucken](#)

## **Dropdown-Kombinationsfeld**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Dropdown-Listen-Kombinationsfeld**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Drucken**

- [Dialogfelder](#)
- [Dialogfelder](#)

## **Drucker**

Anzahl

- [Drucken](#)

Schrifteigenschaften

- [Drucken](#)

## **Druckereigenschaften**

- [Drucken](#)

## **Druckerobjekte**

- [Drucken](#)

## **Druckerschnittstelle**

- [Drucken](#)

## **Druckertyp**

- [Dialogfelder](#)

## **Duplex**

- [Drucken](#)

## **DWORD**

- [Das Windows-API](#)

## **Dynamic Hypertext Markup Language**

- [Internet-Zugriff](#)

## **Dynamic Link Library**

- [Internet-Zugriff](#)
- [Das Windows-API](#)

## **Dynaset**

- [Auf Daten zugreifen](#)

**E**

**E**

- [So funktionieren komplexe Programme](#)

e

- [So funktionieren komplexe Programme](#)

## Eigenschaft

- [Willkommen bei Visual Basic](#)

Alignment

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

AllowCustomize

- [Formulare](#)

AutoSize

- [Steuerelemente - Verwaltung](#)

BackColor

- [Steuerelemente - Verwaltung](#)

BackStyle

- [Grafik und Multimedia](#)

BorderColor

- [Grafik und Multimedia](#)

BorderStyle

- [Steuerelemente - Verwaltung](#)

BorderStyle.Formular

- [Steuerelemente - Verwaltung](#)

BorderWidth

- [Grafik und Multimedia](#)

Cancel

- [Steuerelemente - Verwaltung](#)

Caption

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

ControlBox

- [Steuerelemente - Verwaltung](#)

Default

- [Steuerelemente - Verwaltung](#)

Enabled

- [Steuerelemente - Verwaltung](#)

FillColor

- [Grafik und Multimedia](#)

FillStyle

- [Grafik und Multimedia](#)

Font

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Drucken](#)

FontBold

- [Drucken](#)

FontCount

- [Drucken](#)

FontItalic

- [Drucken](#)

FontName

- [Drucken](#)

Fonts

- [Drucken](#)

FontSize

- [Drucken](#)

FontStrikeThru

- [Drucken](#)

FontTransparent

- [Drucken](#)

FontUnderline

- [Drucken](#)

ForeColor

- [Steuerelemente - Verwaltung](#)

Height

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

Icon

- [Steuerelemente - Verwaltung](#)

Left

- [Steuerelemente - Verwaltung](#)

Locked

- [Steuerelemente - Verwaltung](#)

MaxButton

- [Steuerelemente - Verwaltung](#)

MaxLength

- [Steuerelemente - Verwaltung](#)

MDIChild

- [Formulare](#)

MinButton

- [Steuerelemente - Verwaltung](#)

MousePointer

- [Steuerelemente - Verwaltung](#)

Movable

- [Steuerelemente - Verwaltung](#)

MultiLine

- [Steuerelemente - Verwaltung](#)

nach Kategorien anzeigen

- [Steuerelemente - Verwaltung](#)

Name

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

PasswordChar

- [Steuerelemente - Verwaltung](#)

Picture

- [Grafik und Multimedia](#)

ScaleMode

- [Formulare](#)

ScrollBars

- [Steuerelemente - Verwaltung](#)
- [Bonusprojekt 7: Bildlaufleisten](#)

ShowInTaskbar

- [Steuerelemente - Verwaltung](#)

StartPosition

- [Steuerelemente - Verwaltung](#)

TabIndex

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

Text

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

ToolTipText

- [Steuerelemente - Verwaltung](#)

Top

- [Steuerelemente - Verwaltung](#)

True

- [Steuerelemente - Verwaltung](#)

Visible

- [Steuerelemente - Verwaltung](#)

Width

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

WindowState

- [Steuerelemente - Verwaltung](#)

WordWrap

- [Steuerelemente - Verwaltung](#)

Wrappable

- [Formulare](#)

## **Eigenschaftfenster**

- [Erstes Arbeiten mit Visual Basic](#)

## **Eigenschaftenseiten**

- [Erstes Arbeiten mit Visual Basic](#)
- [Dialogfelder](#)

## **Einfügemarke**

- [Steuerelemente - Verwaltung](#)

## **Eingabefeld**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Eingaben**

- [Programme steuern](#)

InputBox()

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Eingabetaste**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Einzelschritt**

- [Applikationen veröffentlichen](#)

## **Ellipse**

- [Grafik und Multimedia](#)
- [Grafik und Multimedia](#)



## **Else**

- [Programme steuern](#)

## **ElseIf**

- [Programme steuern](#)

## **Elternfenster**

- [Formulare](#)

## **Elternformular**

- [Formulare](#)

## **Embedding**

- [Objekte in Visual Basic](#)

## **Empty**

- [Visual Basic - Grundlagen der Sprache](#)
- [Programme steuern](#)
- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Enabled**

- [Menüs](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **End Function**

- [Steuerelemente - Verwaltung](#)

## **End Sub**

- [Erstes Arbeiten mit Visual Basic](#)

## **End With**

- [Objekte in Visual Basic](#)

## **EndDoc-Methode**

- [Drucken](#)

## **Endlosschleife**

- [Programme steuern](#)

## **Enhanced Meta Files**

- [Grafik und Multimedia](#)

## **Enterprise Edition**

- [Willkommen bei Visual Basic](#)

## **Enum**

- [ActiveX- Steuerelemente](#)

## **Ereignis**

- [Willkommen bei Visual Basic](#)

Activate

- [Steuerelemente - Verwaltung](#)

beenden

- [Programme steuern](#)

Click

- [Steuerelemente - Verwaltung](#)

DblClick

- [Steuerelemente - Verwaltung](#)

Deactivate

- [Steuerelemente - Verwaltung](#)

Initialize

- [Steuerelemente - Verwaltung](#)

Load

- [Steuerelemente - Verwaltung](#)

Paint

- [Steuerelemente - Verwaltung](#)

Resize

- [Steuerelemente - Verwaltung](#)

Unload

- [Steuerelemente - Verwaltung](#)

## **Ereignisprozeduren**

- [Steuerelemente - Verwaltung](#)

## **Eröffnungsbildschirm**

- [Willkommen bei Visual Basic](#)

## **Err**

- [Objekte in Visual Basic](#)

## **Esc-Taste**

- [Steuerelemente - Verwaltung](#)

## **Exclusive**

- [Auf Daten zugreifen](#)

## **EXE erstellen**

- [Applikationen veröffentlichen](#)

## **EXENAME**

- [Objekte in Visual Basic](#)

## **Exit**

- [Programme steuern](#)

## **Exit Do**

- [Programme steuern](#)

## **Exit For**

- [Programme steuern](#)

## **Exp()**

- [So funktionieren komplexe Programme](#)

## **Explorer-Stil**

- [Formulare](#)

## **Exponent**

- [Visual Basic - Grundlagen der Sprache](#)

# **F**

## **False**

- [Visual Basic - Grundlagen der Sprache](#)

## **Farbauflösung**

- [Grafik und Multimedia](#)

## **Farbe**

- [Dialogfelder](#)
- [Dialogfelder](#)

definieren

- [Dialogfelder](#)

## **Farbwerte**

- [Menüs](#)

## **Feld**

- [Menüs](#)

## **Figur-Steuererelement**

- [Erstes Arbeiten mit Visual Basic](#)
- [Grafik und Multimedia](#)

## **fil**

- [Erstes Arbeiten mit Visual Basic](#)

## **FileListBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **FillColor**

- [Grafik und Multimedia](#)

## **FillStyle**

- [Grafik und Multimedia](#)
- [Mehrdimensionale Arrays](#)

## **Filter**

- [Dialogfelder](#)

## **FilterIndex**

- [Dialogfelder](#)

## **Fix**

- [So funktionieren komplexe Programme](#)

## **FixedCols**

- [Mehrdimensionale Arrays](#)

## **FixedOrder**

- [Formulare](#)

## **FixedRows**

- [Mehrdimensionale Arrays](#)

## **flexFillRepeat**

- [Mehrdimensionale Arrays](#)

## **flexFillSingle**

- [Mehrdimensionale Arrays](#)

## **flexFocusHeavy**

- [Mehrdimensionale Arrays](#)

## **flexFocusLight**

- [Mehrdimensionale Arrays](#)

## **flexFocusNone**

- [Mehrdimensionale Arrays](#)

## **FlexGrid**

- [Mehrdimensionale Arrays](#)

## **flexGridFlat**

- [Mehrdimensionale Arrays](#)

## **flexGridNone**

- [Mehrdimensionale Arrays](#)

## **flexGridRaised**

- [Mehrdimensionale Arrays](#)

## **flexHighlightAlways**

- [Mehrdimensionale Arrays](#)

## **flexHighlightNever**

- [Mehrdimensionale Arrays](#)

## **flexHighlightWithFocus**

- [Mehrdimensionale Arrays](#)

## **flexResizeBoth**

- [Mehrdimensionale Arrays](#)

## **flexResizeColumns**

- [Mehrdimensionale Arrays](#)

## **flexResizeNone**

- [Mehrdimensionale Arrays](#)

## **flexResizeRows**

- [Mehrdimensionale Arrays](#)

## **flexSelectionByColumn**

- [Mehrdimensionale Arrays](#)

## **flexSelectionByRow**

- [Mehrdimensionale Arrays](#)

## **flexSelectionFree**

- [Mehrdimensionale Arrays](#)

## **Fließkommazahlen**

- [Visual Basic - Grundlagen der Sprache](#)

## **FocusRect**

- [Mehrdimensionale Arrays](#)

## **Fokus**

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

## **Font**

- [Drucken](#)
- [Objekte in Visual Basic](#)

Eigenschaft

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **FontBold**

- [Drucken](#)

## **FontCount**

- [Drucken](#)
- [Objekte in Visual Basic](#)

## **FontItalic**

- [Drucken](#)

## **FontName**

- [Drucken](#)

## **FontSize**

- [Drucken](#)

## **FontStrikeThru**

- [Drucken](#)

## **FontTransparent**

- [Drucken](#)

## **FontUnderline**

- [Drucken](#)

## **For Each**

- [Formulare](#)

## **ForeColor**

- [Formulare](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **Form**

- [Formulare](#)

## **Form.Hide**

- [Formulare](#)

## **Form.Show**

- [Formulare](#)

## **Form\_DragDrop()**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Form\_Load()**

- [Erstes Arbeiten mit Visual Basic](#)
- [Formulare](#)

## **Form\_Unload()**

- [Formulare](#)
- [Formulare](#)

## **Form1**

- [Erstes Arbeiten mit Visual Basic](#)

## **Format()**

- [So funktionieren komplexe Programme](#)

## **FormatString**

- [Mehrdimensionale Arrays](#)
- [Mehrdimensionale Arrays](#)

## **Forms-Auflistung**

- [Formulare](#)

## **Formular**

- [Willkommen bei Visual Basic](#)
  - [Erstes Arbeiten mit Visual Basic](#)
- ausdrucken

- [Drucken](#)

benutzerdefinierte Eigenschaften

- [Formulare](#)

Eigenschaften

- [Formulare](#)

Ereignisse

- [Steuerelemente - Verwaltung](#)
- [Formulare](#)

geöffnete

- [Formulare](#)

Größenkoordinaten

- [Erstes Arbeiten mit Visual Basic](#)

in Taskleiste anzeigen

- [Steuerelemente - Verwaltung](#)

laden

- [Mehr Power mit Maus und Steuerelementen](#)

LoadPicture()

- [Grafik und Multimedia](#)

mehrere

- [Formulare](#)

Methoden

- [Formulare](#)

Positionskoordinaten

- [Erstes Arbeiten mit Visual Basic](#)

primäres

- [Formulare](#)

Status

- [Steuerelemente - Verwaltung](#)

Text

- [Formulare](#)

übergeordnetes

- [Formulare](#)

Verschieben

- [Steuerelemente - Verwaltung](#)

Visible

- [Formulare](#)

**Formular-Editor, Symbolleiste**

- [Erstes Arbeiten mit Visual Basic](#)

## **Formulareigenschaften**

- [Steuerelemente - Verwaltung](#)

## **Formularfenster**

- [Willkommen bei Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)

## **Formular-Layout-Fenster**

- [Erstes Arbeiten mit Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)

## **Formularmodul**

- [Visual Basic - Grundlagen der Sprache](#)

## **Formularschablonen**

- [Willkommen bei Visual Basic](#)

## **Formularvorlagen**

- [Willkommen bei Visual Basic](#)
- [Formularvorlagen](#)

## **For-Schleifen**

- [Programme steuern](#)

verschachtelte

- [Programme steuern](#)

## **fra**

- [Erstes Arbeiten mit Visual Basic](#)

## **Frame**

- [Erstes Arbeiten mit Visual Basic](#)

## **FreeFile()**

- [Mit Dateien arbeiten](#)

## **frm**

- [Erstes Arbeiten mit Visual Basic](#)

## **FrontPage Express**

- [Ein Hilfe- System erstellen](#)

## **Function**

- [Steuerelemente - Verwaltung](#)

## **Funktion**

Array()

- [Mehr Power mit Maus und Steuerelementen](#)

Datentypumwandlung

- [So funktionieren komplexe Programme](#)



integrierte

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

interne

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)
- [So funktionieren komplexe Programme](#)

numerische

- [So funktionieren komplexe Programme](#)

Spc()

- [Formulare](#)

Strings

- [So funktionieren komplexe Programme](#)

Tab()

- [Formulare](#)

Teilstrings

- [So funktionieren komplexe Programme](#)

trigonometrische

- [So funktionieren komplexe Programme](#)

## **Funktionsdeklaration**

- [So funktionieren komplexe Programme](#)

## **Funktionsprozedur**

- [So funktionieren komplexe Programme](#)

# **G**

## **GDI32.DLL**

- [Das Windows-API](#)

## **Get**

- [Mit Dateien arbeiten](#)

## **GetData**

- [Objekte in Visual Basic](#)

## **GetFormat**

- [Objekte in Visual Basic](#)

## **GetObject()**

- [ActiveX- Steuerelemente](#)

## **GetSystemDirectory()**

- [Das Windows-API](#)

## **GetTempPath()**

- [Das Windows-API](#)

## **GetText**

- [Objekte in Visual Basic](#)

## **GetText()**

- [Objekte in Visual Basic](#)

## **GetWindowsDirectory()**

- [Das Windows-API](#)

## **GIF-Dateien**

- [Grafik und Multimedia](#)

## **Gleich**

- [Programme steuern](#)

## **global**

- [So funktionieren komplexe Programme](#)

## **Grafik**

Skalierung

- [Formulare](#)

## **Grafikdatei, laden**

- [Grafik und Multimedia](#)

## **Grafikformate**

- [Grafik und Multimedia](#)

## **Graphics Interchange Format**

- [Grafik und Multimedia](#)

## **grd**

- [Erstes Arbeiten mit Visual Basic](#)

## **GridLines**

- [Mehrdimensionale Arrays](#)

## **GridLinesFixed**

- [Mehrdimensionale Arrays](#)

## **Größenkoordinaten, Formular**

- [Erstes Arbeiten mit Visual Basic](#)

## **Größer als**

- [Programme steuern](#)

## **Großbuchstaben**

- [So funktionieren komplexe Programme](#)

## **Gültigkeitsbereich**

- [So funktionieren komplexe Programme](#)

## **Haltepunkte**

- [Applikationen veröffentlichen](#)
- [Applikationen veröffentlichen](#)  
mehrfache
  - [Applikationen veröffentlichen](#)

## **HBildlaufleiste**

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Height**

- [Drucken](#)
- [Grafik und Multimedia](#)
- [Objekte in Visual Basic](#)  
Eigenschaft
  - [Erstes Arbeiten mit Visual Basic](#)
  - [Steuerelemente - Verwaltung](#)

## **HelpContextID**

- [Ein Hilfe- System erstellen](#)

## **HelpFile**

- [Ein Hilfe- System erstellen](#)
- [Ein Hilfe- System erstellen](#)

## **Hexadezimalsystem**

- [Menüs](#)

## **Hide**

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

## **HighLight**

- [Mehrdimensionale Arrays](#)

## **Hilfe**

- [Erstes Arbeiten mit Visual Basic](#)
- [Dialogfelder](#)
- [Dialogfelder](#)

## **Hilfesysteme, HTML**

- [Ein Hilfe- System erstellen](#)

## **Hintergrundfarbe, Steuerelement**

- [Steuerelemente - Verwaltung](#)

## **hsb**

- [Erstes Arbeiten mit Visual Basic](#)

## **HScrollBar**

- [Erstes Arbeiten mit Visual Basic](#)

## **HTML**

- [Internet-Zugriff](#)
- [Internet-Zugriff](#)
- [Ein Hilfe- System erstellen](#)

## **HTML-Hilfedatei**

- [Ein Hilfe- System erstellen](#)

## **http**

- [Internet-Zugriff](#)

## **Hüllzeilen**

- [Menüs](#)

## **HyperText Markup Language**

- [Internet-Zugriff](#)
- [Ein Hilfe- System erstellen](#)

## **Hypertext Transfer Protocol**

- [Internet-Zugriff](#)

## **Hypertext-Sprung**

- [Ein Hilfe- System erstellen](#)

I

## **I/O**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)
- [Mit Dateien arbeiten](#)

## **Icon**

- [Grafik und Multimedia](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **IE Animated Button**

- [Internet-Zugriff](#)

## **IE Popup Menu**

- [Internet-Zugriff](#)

## **IE Preloader**

- [Internet-Zugriff](#)

## **IE Super Label**

- [Internet-Zugriff](#)

## **IE Timer**

- [Internet-Zugriff](#)

## **If**

- [Programme steuern](#)

## **If TypeOf**

- [So funktionieren komplexe Programme](#)
- [Objekte in Visual Basic](#)

## **If...Else-Anweisungen, verschachtelte**

- [Programme steuern](#)

## **Ignorieren**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **IIf()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **Image**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Image-Eigenschaft**

- [Formulare](#)

## **ImageList**

- [Formulare](#)

## **img**

- [Erstes Arbeiten mit Visual Basic](#)

## **Index**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Auf Daten zugreifen](#)

## **Info-Dialogfeld**

- [Formularvorlagen](#)

## **INI-Dateien**

- [Das Windows-API](#)

## **Initialize, Ereignis**

- [Steuerelemente - Verwaltung](#)

## **In-Place-Aktivierung**

- [Objekte in Visual Basic](#)

## **Input**

- [Mit Dateien arbeiten](#)

## **Input #**

- [Mit Dateien arbeiten](#)

## **Input/Output**

- [Mit Dateien arbeiten](#)

## **InputBox()**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Instanz**

- [Objekte in Visual Basic](#)

## **Int**

- [So funktionieren komplexe Programme](#)

## **intButton**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Integer**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **Integerumwandlung**

- [So funktionieren komplexe Programme](#)

## **Internet**

- [Willkommen bei Visual Basic](#)

## **Internet Service Provider**

- [Internet-Zugriff](#)

## **Internet-Browser**

- [Willkommen bei Visual Basic](#)

## **Internet-Explorer-Steuerelemente**

- [Internet-Zugriff](#)

## **Internet-Transfer**

- [Internet-Zugriff](#)

## **Internet-Zugriff**

- [Internet-Zugriff](#)

## **interpretiert**

- [Willkommen bei Visual Basic](#)

## **Interval**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Intranet**

- [Internet-Zugriff](#)

## **intShift**

- [Mehr Power mit Maus und Steuerelementen](#)

## **InvisibleAtRunTime**

- [ActiveX- Steuerelemente](#)

## **Is**

- [Programme steuern](#)

## **Is TypeOf**

- [Drucken](#)

## **IsDate()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **IsEmpty()**

- [So funktionieren komplexe Programme](#)

## **IsNull()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **IsNumeric()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **ISP**

- [Internet-Zugriff](#)

## **Item, Auflistung**

- [Objekte in Visual Basic](#)

## **Iteration**

- [Programme steuern](#)

# **J**

## **Java**

- [Internet-Zugriff](#)

## **JPEG-Dateien**

- [Grafik und Multimedia](#)

# **K**

## **Kalenderoperationen**

- [Dialogfelder](#)

## **Kanal**

- [Mit Dateien arbeiten](#)

## **Kapselung**

- [Objekte in Visual Basic](#)
- [Internet-Zugriff](#)

## **Kernel**

- [Das Windows-API](#)

## **KERNEL32.DLL**

- [Das Windows-API](#)

## **KeyDown**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)
- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **KeyPress**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **KeyPreview**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **KeyUp**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **KeyUp-Ereignis**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Key-Wert**

- [Formulare](#)

## **Kill**

- [Mit Dateien arbeiten](#)

## **KillDoc**

- [Drucken](#)

## **Kindfenster**

- [Willkommen bei Visual Basic](#)  
minimieren
  - [Formulare](#)

## **Kindformulare**

- [Formulare](#)

## **Klasse**

- [So funktionieren komplexe Programme](#)
- [Objekte in Visual Basic](#)
- [ActiveX- Steuerelemente](#)

## **Klassenmodule**

- [Erstes Arbeiten mit Visual Basic](#)

## **Klassenprozeduren**

- [So funktionieren komplexe Programme](#)

## **Kleinbuchstaben**

- [So funktionieren komplexe Programme](#)

## **Kleiner als**

- [Programme steuern](#)



## Klicks

- [Mehr Power mit Maus und Steuerelementen](#)

## Kombinationsfeld

- [Mehr Power mit Maus und Steuerelementen](#)

einfaches

- [Mehr Power mit Maus und Steuerelementen](#)
- [Mehr Power mit Maus und Steuerelementen](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## Kommentar

- [Willkommen bei Visual Basic](#)
- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

## Kompilieren

- [Applikationen veröffentlichen](#)

## kompiliert

- [Willkommen bei Visual Basic](#)

## Konkatenation

- [Visual Basic - Grundlagen der Sprache](#)

## Konstante

- [Visual Basic - Grundlagen der Sprache](#)

benannte

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## Konstanten-Suffixe

- [Visual Basic - Grundlagen der Sprache](#)

## Kontextstring

- [Ein Hilfe- System erstellen](#)

## Kontrollkästchen

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

Status

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## Kreis

- [Grafik und Multimedia](#)

## Kreise

- [Grafik und Multimedia](#)

## **Label**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Dialogfelder](#)

## **Ländereinstellungen**

- [Visual Basic - Grundlagen der Sprache](#)

## **Laufwerkslistenfeld**

- [Mit Dateien arbeiten](#)
  - Steuerelement
    - [Erstes Arbeiten mit Visual Basic](#)

## **lbl**

- [Erstes Arbeiten mit Visual Basic](#)

## **LCase()**

- [So funktionieren komplexe Programme](#)

## **Left**

Eigenschaft

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Left()**

- [So funktionieren komplexe Programme](#)

## **Len()**

- [So funktionieren komplexe Programme](#)

## **Let**

- [Visual Basic - Grundlagen der Sprache](#)

## **Like**

- [Programme steuern](#)

## **lin**

- [Erstes Arbeiten mit Visual Basic](#)

## **Line**

- [Erstes Arbeiten mit Visual Basic](#)

## **Line Input #**

- [Bonusprojekt 6: Dateien lesen und schreiben](#)

## **Line-Methode**

- [Grafik und Multimedia](#)

## **Linien, Steuerelement**

- [Erstes Arbeiten mit Visual Basic](#)

## **Linienbreite**

- [Grafik und Multimedia](#)

## **Linienfarbe**

- [Grafik und Multimedia](#)

## **Linienformat**

- [Grafik und Multimedia](#)

## **Linien-Steuerelement**

- [Grafik und Multimedia](#)

## **Linking**

- [Objekte in Visual Basic](#)

## **ListBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **ListCount-Eigenschaft**

- [Mehr Power mit Maus und Steuerelementen](#)

## **List-Eigenschaft**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Listenansicht**

- [Formulare](#)

## **Listenelemente**

- [Mehr Power mit Maus und Steuerelementen](#)

Anzahl

- [Mehr Power mit Maus und Steuerelementen](#)

## **Listenfeld**

- [Mehr Power mit Maus und Steuerelementen](#)

einfaches

- [Mehr Power mit Maus und Steuerelementen](#)
- [Mehr Power mit Maus und Steuerelementen](#)

Initialisierung

- [Mehr Power mit Maus und Steuerelementen](#)

löschen

- [Mehr Power mit Maus und Steuerelementen](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Listen-Steuerelemente**

- [Mehr Power mit Maus und Steuerelementen](#)

## **ListIndex-Eigenschaft**

- [Mehr Power mit Maus und Steuerelementen](#)

## **ListView**

- [Formulare](#)

## **Load**

- [Mehr Power mit Maus und Steuerelementen](#)

Ereignis

- [Steuerelemente - Verwaltung](#)

## **LoadPicture**

- [Erstes Arbeiten mit Visual Basic](#)

## **LoadPicture()**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)
- [Grafik und Multimedia](#)

## **Lock Read**

- [Mit Dateien arbeiten](#)

## **Lock Read Write**

- [Mit Dateien arbeiten](#)

## **Lock Write**

- [Mit Dateien arbeiten](#)

## **Locked, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **LockType-Argument**

- [Mit Dateien arbeiten](#)

## **Log()**

- [So funktionieren komplexe Programme](#)

## **Logarithmus, natürlicher**

- [So funktionieren komplexe Programme](#)

## **lokal**

- [So funktionieren komplexe Programme](#)

## **Long**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **Loop**

- [Programme steuern](#)

## **LPCSTR**

- [Das Windows-API](#)

## **LPSTR**

- [Das Windows-API](#)

## **lst**

- [Erstes Arbeiten mit Visual Basic](#)

## **LTrim()**

- [So funktionieren komplexe Programme](#)

## **M**

## **Main()**

- [Formulare](#)

## **Mapi32.txt**

- [Das Windows-API](#)

## **Matrix**

- [Mehrdimensionale Arrays](#)

## **Mausbewegungen**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Mausereignisse**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Mehr Power mit Maus und Steuerelementen](#)

## **Mauszeiger**

- [Mehr Power mit Maus und Steuerelementen](#)

Umriss

- [Steuerelemente - Verwaltung](#)

## **MaxButton, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Maximier-Schaltfläche**

- [Steuerelemente - Verwaltung](#)

## **MaxLength, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **mciModeNotOpen**

- [Grafik und Multimedia](#)

## **mciModePause**

- [Grafik und Multimedia](#)

## **mciModePlay**

- [Grafik und Multimedia](#)

## **mciModeReady**

- [Grafik und Multimedia](#)

## **mciModeRecord**

- [Grafik und Multimedia](#)

## **mciModeStop**

- [Grafik und Multimedia](#)

## **mcuModeSeek**

- [Grafik und Multimedia](#)

## **MDI**

- [Willkommen bei Visual Basic](#)
- [Formulare](#)
- [Formulare](#)

## **MDIChild-Eigenschaft**

- [Formulare](#)

## **MDI-Formular hinzufügen**

- [Formulare](#)

## **Mehrfachauswahlen**

- [Bonusprojekt 4: Mehrfachauswahlen in Listefeldern](#)

## **Meldungsfeld**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

Icon

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

Schaltflächen

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Menü**

Anwendungs-Assistent

- [Menüs](#)

Dialogfeld

- [Menüs](#)

Trennlinie

- [Menüs](#)

## **Menü-Editor**

- [Menüs](#)

## **Menüleiste**

- [Willkommen bei Visual Basic](#)
- [Menüs](#)

## **Menüoption**

anlegen

- [Menüs](#)

## **Menüoptionen**

markierte

- [Menüs](#)

## **MessageBeep**

- [Das Windows-API](#)

## **Metadateien**

- [Grafik und Multimedia](#)

## **Methode**

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

Circle

- [Grafik und Multimedia](#)

EndDoc

- [Drucken](#)

Line

- [Grafik und Multimedia](#)

NewPage

- [Drucken](#)

PaintPicture

- [Drucken](#)

Print

- [Drucken](#)

PrintForm

- [Drucken](#)

PSet

- [Grafik und Multimedia](#)

## **Microsoft ADO Data Control 6.0**

- [Bonusprojekt 9: ADO-Steuerelemente](#)

## **Microsoft Calendar Control 8.0**

- [Dialogfelder](#)

## **Microsoft Common Dialog Control 6.0**

- [Dialogfelder](#)

## **Microsoft Developer's Network**

- [Erstes Arbeiten mit Visual Basic](#)

## **Microsoft Internet Controls**

- [Internet-Zugriff](#)

## **Microsoft Internet Transfer Control 6.0**

- [Internet-Zugriff](#)

## **Microsoft Multimedia Control 6.0**

- [Grafik und Multimedia](#)

## **Microsoft Office**

- [Willkommen bei Visual Basic](#)

## **Microsoft Winsock Control 6.0**

- [Internet-Zugriff](#)

## **Mid()**

- [So funktionieren komplexe Programme](#)

## **MIDI-Sequencer**

- [Grafik und Multimedia](#)

## **MinButton, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Minimier-Schaltfläche**

- [Steuerelemente - Verwaltung](#)

## **MkDir**

- [Mit Dateien arbeiten](#)

## **mnu**

- [Erstes Arbeiten mit Visual Basic](#)

## **Mod**

- [Visual Basic - Grundlagen der Sprache](#)

## **mod**

- [Erstes Arbeiten mit Visual Basic](#)

## **Mode**

- [Grafik und Multimedia](#)

## **Modul**

- [Visual Basic - Grundlagen der Sprache](#)

## **Module**

- [Erstes Arbeiten mit Visual Basic](#)

## **Modulo**

- [Visual Basic - Grundlagen der Sprache](#)

## **Month()**

- [So funktionieren komplexe Programme](#)

## **MouseDown**

- [Mehr Power mit Maus und Steuerelementen](#)

## **MouseMove**

- [Mehr Power mit Maus und Steuerelementen](#)

## **MousePointer**

- [Mehr Power mit Maus und Steuerelementen](#)

- [Objekte in Visual Basic](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)



## **MouseUp**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Movable, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Move-Methode**

- [Mehr Power mit Maus und Steuerelementen](#)

## **MSDN**

- [Erstes Arbeiten mit Visual Basic](#)

## **MsgBox()**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **MsgBox-Anweisung**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **MSINFO32.EXE**

- [Formularvorlagen](#)

## **MSN**

- [Internet-Zugriff](#)

## **MultiLine, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Multimedia-Steuerelement**

- [Grafik und Multimedia](#)

Befehle

- [Grafik und Multimedia](#)

## **Multimedia-Werkzeuge**

- [Grafik und Multimedia](#)

## **Multiple Document Interface**

- [Willkommen bei Visual Basic](#)
- [Formulare](#)

## **Multiplikation**

- [Visual Basic - Grundlagen der Sprache](#)

## **MultiSelect-Eigenschaft**

- [Bonusprojekt 4: Mehrfachauswahlen in Listenfeldern](#)

## **Multitasking**

- [Willkommen bei Visual Basic](#)

## **Musical Instrument Digital Interface**

- [Grafik und Multimedia](#)

## **n:n-Beziehungen**

- [Auf Daten zugreifen](#)

## **Nach Kategorien**

- [Steuerelemente - Verwaltung](#)

## **Name, Eigenschaft**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Name-Eigenschaft**

- [Erstes Arbeiten mit Visual Basic](#)

## **Neubooten**

- [Das Windows-API](#)

## **Neues Projekt**

- [Willkommen bei Visual Basic](#)

Neu

- [Willkommen bei Visual Basic](#)

Vorhanden

- [Willkommen bei Visual Basic](#)

## **NewPage**

- [Drucken](#)

## **Not**

- [Programme steuern](#)

## **Notation, wissenschaftliche**

- [Visual Basic - Grundlagen der Sprache](#)

## **Notify-Eigenschaft**

- [Grafik und Multimedia](#)

## **Now**

- [So funktionieren komplexe Programme](#)

## **NULL**

- [Das Windows-API](#)

## **Null**

- [Visual Basic - Grundlagen der Sprache](#)
- [Programme steuern](#)
- [So funktionieren komplexe Programme](#)

## **Nullstring**

- [Visual Basic - Grundlagen der Sprache](#)

## **Object**

- [Visual Basic - Grundlagen der Sprache](#)
- [So funktionieren komplexe Programme](#)

## **Object Linking and Embedding**

- [Formulare](#)
- [Objekte in Visual Basic](#)

## **Objekt**

- [Willkommen bei Visual Basic](#)
- [Objekte in Visual Basic](#)

anzeigen

- [Erstes Arbeiten mit Visual Basic](#)

Bildschirm

- [Formulare](#)

einfügen

- [Objekte in Visual Basic](#)

Listenfeld

- [Menüs](#)

persistentes

- [Objekte in Visual Basic](#)

Screen

- [Formulare](#)

## **Objektkatalog**

- [Objekte in Visual Basic](#)

## **Objektklasse**

- [Objekte in Visual Basic](#)

## **Objektnamen, Präfix**

- [Erstes Arbeiten mit Visual Basic](#)

## **objektorientiert**

- [Objekte in Visual Basic](#)

## **Objekttypen**

- [Erstes Arbeiten mit Visual Basic](#)

## **Objektvariable**

- [So funktionieren komplexe Programme](#)

## **ODBC**

- [Formularvorlagen](#)

## **ODBC-Anmeldung**

- [Formularvorlagen](#)

- [Formularvorlagen](#)

## **OLE**

- [Erstes Arbeiten mit Visual Basic](#)
- [Objekte in Visual Basic](#)

Speichern

- [Objekte in Visual Basic](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **OLE-Objekt**

- [Formulare](#)

## **OLE-Steuerelement**

- [Objekte in Visual Basic](#)

## **On Error Goto**

- [Dialogfelder](#)

## **OOP**

- [Objekte in Visual Basic](#)

## **Open**

- [Mit Dateien arbeiten](#)

## **Open Database Connectivity**

- [Formularvorlagen](#)

## **Operator, überladener**

- [Visual Basic - Grundlagen der Sprache](#)

## **Operatoren**

- [Visual Basic - Grundlagen der Sprache](#)

bedingter

- [Programme steuern](#)

logischer

- [Programme steuern](#)

Reihenfolge

- [Visual Basic - Grundlagen der Sprache](#)

## **opt**

- [Erstes Arbeiten mit Visual Basic](#)

## **Option**

aktivieren

- [Menüs](#)

markierte

- [Menüs](#)

## **Option Compare Text**

- [Programme steuern](#)

## **Option Explicit**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **OptionButton**

- [Erstes Arbeiten mit Visual Basic](#)

## **Optionen-Dialogfeld**

- [Formularvorlagen](#)
- [Formularvorlagen](#)

## **Optionsfeld**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)  
Steuerelement
  - [Erstes Arbeiten mit Visual Basic](#)

## **Or**

- [Programme steuern](#)

## **Ordner**

- [Mit Dateien arbeiten](#)

## **Ordner wechseln**

- [Erstes Arbeiten mit Visual Basic](#)

## **Orientation**

- [Formulare](#)
- [Drucken](#)

## **Output**

- [Mit Dateien arbeiten](#)

## **Overlay**

- [Grafik und Multimedia](#)

**P**

## **Page**

- [Drucken](#)

## **Paint, Ereignis**

- [Steuerelemente - Verwaltung](#)

## **PaintPicture-Methode**

- [Drucken](#)

## **Paket- und Weitergabe-Assistent**

- [Applikationen veröffentlichen](#)

## **PaperBin**

- [Drucken](#)

## **PaperSize**

- [Drucken](#)

## **PasswordChar**

- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **Path**

- [Objekte in Visual Basic](#)

## **pic**

- [Erstes Arbeiten mit Visual Basic](#)

## **Picture**

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

- [Formulare](#)

- [Grafik und Multimedia](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **PictureBox**

- [Erstes Arbeiten mit Visual Basic](#)

## **Pixel**

- [Erstes Arbeiten mit Visual Basic](#)

## **Port**

- [Drucken](#)

## **Positionskoordinaten, Formular**

- [Erstes Arbeiten mit Visual Basic](#)

## **PrevInstance**

- [Objekte in Visual Basic](#)

## **Print**

- [Drucken](#)

- [Objekte in Visual Basic](#)

## **Print #**

- [Mit Dateien arbeiten](#)

## **Printer**

- [Drucken](#)

- [Objekte in Visual Basic](#)

## **Printer-Objekt**

- [Drucken](#)

## **Printers.Count**

- [Drucken](#)

## **Printers-Auflistung**

- [Drucken](#)

## **PrintForm**

- [Drucken](#)

Nachteile

- [Drucken](#)

## **Print-Methode**

- [Formulare](#)

Positionierung

- [Formulare](#)

## **PrintQuality**

- [Drucken](#)

## **Private**

- [Steuerelemente - Verwaltung](#)
- [So funktionieren komplexe Programme](#)

## **Private Sub**

- [Steuerelemente - Verwaltung](#)

## **Professional Edition**

- [Willkommen bei Visual Basic](#)

## **Programm, ausführen**

- [Willkommen bei Visual Basic](#)

## **Programmcode**

- [Erstes Arbeiten mit Visual Basic](#)

## **Programmiersprache**

- [Willkommen bei Visual Basic](#)

## **Programmierung**

- [Willkommen bei Visual Basic](#)

ereignisorientierte

- [Willkommen bei Visual Basic](#)

objektorientierte

- [Objekte in Visual Basic](#)

## **Programmwartung**

- [Willkommen bei Visual Basic](#)

## **Projekt**

- [Willkommen bei Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)

erstellen

- [Applikationen veröffentlichen](#)

speichern

- [Erstes Arbeiten mit Visual Basic](#)

## **Projektbeschreibungsdatei**

- [Erstes Arbeiten mit Visual Basic](#)

## **Projekteigenschaften**

- [Visual Basic - Grundlagen der Sprache](#)

## **Projektfenster**

- [Erstes Arbeiten mit Visual Basic](#)

Objekttypen

- [Erstes Arbeiten mit Visual Basic](#)

## **Property Get**

- [Formulare](#)

## **Property Let**

- [Formulare](#)

## **Protokoll**

- [Internet-Zugriff](#)

## **Prototyp**

- [Willkommen bei Visual Basic](#)

## **Prozedur**

- [Erstes Arbeiten mit Visual Basic](#)

allgemeine

- [So funktionieren komplexe Programme](#)

Aufruf

- [So funktionieren komplexe Programme](#)

beenden

- [Programme steuern](#)

Listenfeld

- [Menüs](#)

öffentliche

- [So funktionieren komplexe Programme](#)

private

- [So funktionieren komplexe Programme](#)

## **Prozeduraufruf**



- [So funktionieren komplexe Programme](#)

## **Prozedurnamen**

- [So funktionieren komplexe Programme](#)

## **PSet**

- [Grafik und Multimedia](#)

## **Public**

- [So funktionieren komplexe Programme](#)

## **Pulldown-Menü**

- [Menüs](#)

## **Put**

- [Mit Dateien arbeiten](#)

# **Q**

## **QBasic**

- [Willkommen bei Visual Basic](#)

## **Quadrat**

- [Grafik und Multimedia](#)

## **Quadratwurzel**

- [So funktionieren komplexe Programme](#)

## **Quellprogramm**

- [Erstes Arbeiten mit Visual Basic](#)

# **R**

## **Radius**

- [Grafik und Multimedia](#)

## **Rahmen**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Random**

- [Mit Dateien arbeiten](#)

## **Raster**

- [Erstes Arbeiten mit Visual Basic](#)

anzeigen

- [Erstes Arbeiten mit Visual Basic](#)

## **RDO**

- [Auf Daten zugreifen](#)

## **Read**

- [Mit Dateien arbeiten](#)

## **Read Write**

- [Mit Dateien arbeiten](#)

## **ReadFromFile**

- [Objekte in Visual Basic](#)

## **Rechteck**

- [Grafik und Multimedia](#)

## **Recordset**

- [Auf Daten zugreifen](#)

## **RecordSource**

- [Auf Daten zugreifen](#)

## **Referenz**

- [So funktionieren komplexe Programme](#)

## **Registerkarten**

- [Formularvorlagen](#)

## **Registrierung, ändern**

- [Formularvorlagen](#)

## **relational**

- [Auf Daten zugreifen](#)

## **Remote Data Objects**

- [Auf Daten zugreifen](#)

## **Remove, Auflistung**

- [Objekte in Visual Basic](#)

## **RemoveItem**

- [Mehr Power mit Maus und Steuerelementen](#)

## **res**

- [Erstes Arbeiten mit Visual Basic](#)

## **Resize**

- [Formulare](#)

Ereignis

- [Steuerelemente - Verwaltung](#)

## **Resize-Eigenschaft**

- [Formulare](#)

## **Ressourcen**

- [Willkommen bei Visual Basic](#)

Sprache

- [Willkommen bei Visual Basic](#)

**Right()**

- [So funktionieren komplexe Programme](#)

**RLE-Dateien**

- [Grafik und Multimedia](#)

**RmDir**

- [Mit Dateien arbeiten](#)

**RowHeigthMin**

- [Mehrdimensionale Arrays](#)

**Rows**

- [Mehrdimensionale Arrays](#)

**RTF-Hilfe**

- [Ein Hilfe- System erstellen](#)

**RTrim()**

- [So funktionieren komplexe Programme](#)

**S**

**SaveSetting**

- [Formularvorlagen](#)

**SaveToFile**

- [Objekte in Visual Basic](#)

**ScaleHeight**

- [Drucken](#)

**ScaleLeft**

- [Drucken](#)

**ScaleMode**

- [Drucken](#)

**ScaleMode-Eigenschaft**

- [Formulare](#)

**ScaleTop**

- [Drucken](#)

**ScaleWidth**

- [Drucken](#)

**ScaleX**

- [Grafik und Multimedia](#)

**ScaleY**

- [Grafik und Multimedia](#)

## **Scanner**

- [Grafik und Multimedia](#)

## **Schaltfläche**

- [Erstes Arbeiten mit Visual Basic](#)

Eigenschaften

- [Erstes Arbeiten mit Visual Basic](#)

## **Schleifen**

- [Programme steuern](#)

## **Schleifenanweisungen**

- [Programme steuern](#)

## **Schleifenvariable**

- [Programme steuern](#)

## **Schrift**

- [Erstes Arbeiten mit Visual Basic](#)

## **Schriftart**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)
- [Dialogfelder](#)
- [Dialogfelder](#)

Flags

- [Dialogfelder](#)

## **Schrifteigenschaften**

- [Drucken](#)

## **Screen**

- [Formulare](#)
- [Objekte in Visual Basic](#)

## **ScrollBars**

- [Bonusprojekt 7: Bildlaufleisten](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **SDI**

- [Willkommen bei Visual Basic](#)
- [Formulare](#)
- [Formulare](#)

## **Select Case**

- [Programme steuern](#)

## **SelectionMode**

- [Mehrdimensionale Arrays](#)

## **SelLength**

- [Objekte in Visual Basic](#)

## **SelStart**

- [Objekte in Visual Basic](#)

## **SelText**

- [Objekte in Visual Basic](#)

## **SendKeys**

- [Formularvorlagen](#)

## **SendKeys-Anweisung**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Set**

- [ActiveX- Steuerelemente](#)

## **Set Printer**

- [Drucken](#)
- [Drucken](#)

## **SetData**

- [Objekte in Visual Basic](#)

## **SetText**

- [Objekte in Visual Basic](#)

## **Shape**

- [Erstes Arbeiten mit Visual Basic](#)

## **Shape-Eigenschaft**

- [Grafik und Multimedia](#)

## **Shared**

- [Mit Dateien arbeiten](#)

## **Shell-Links**

- [Dialogfelder](#)

## **Shift**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Shift-Status**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Shortcut**

- [Willkommen bei Visual Basic](#)
- [Menüs](#)

## **Show**

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

## **ShowColor**

- [Dialogfelder](#)

## **ShowFont**

- [Dialogfelder](#)

## **ShowHelp**

- [Dialogfelder](#)
- [Ein Hilfe- System erstellen](#)

## **ShowInTaskbar, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **ShowOpen**

- [Dialogfelder](#)
- [Dialogfelder](#)

## **ShowPrinter**

- [Dialogfelder](#)
- [Dialogfelder](#)

## **ShowSave**

- [Dialogfelder](#)

## **shp**

- [Erstes Arbeiten mit Visual Basic](#)

## **sichtbar**

- [So funktionieren komplexe Programme](#)

## **Sin()**

- [So funktionieren komplexe Programme](#)

## **Single**

- [Visual Basic - Grundlagen der Sprache](#)
- [Visual Basic - Grundlagen der Sprache](#)

## **Single Document Interface**

- [Willkommen bei Visual Basic](#)
- [Formulare](#)

## **Sinus**

- [So funktionieren komplexe Programme](#)

## **Skalierung**

- [Formulare](#)

## **Skriptsprache**

- [Internet-Zugriff](#)

## **Snapshot**

- [Auf Daten zugreifen](#)

## **sngX**

- [Mehr Power mit Maus und Steuerelementen](#)

## **sngY**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Software-Service**

- [Erstes Arbeiten mit Visual Basic](#)

## **Sorted-Eigenschaft**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Mehr Power mit Maus und Steuerelementen](#)

## **SourceSafe**

- [Erstes Arbeiten mit Visual Basic](#)

## **Spc()**

- [Formulare](#)

## **Speichern**

- [Erstes Arbeiten mit Visual Basic](#)

## **Speichern unter**

- [Dialogfelder](#)
- [Dialogfelder](#)

## **SQL**

- [Formularvorlagen](#)

## **Sqr()**

- [So funktionieren komplexe Programme](#)

## **Standarddialog-Steuerelement**

- [Dialogfelder](#)

## **Standarddrucker**

- [Drucken](#)

## **Standard-EXE-Anwendung**

- [Erstes Arbeiten mit Visual Basic](#)

## **Standardformulare**

- [Willkommen bei Visual Basic](#)

## **Standardobjekt**

- [Formulare](#)

## **Standard-Startobjekt**

- [Visual Basic - Grundlagen der Sprache](#)

## **Standardsymbolleiste**

- [Erstes Arbeiten mit Visual Basic](#)

## **Startmenüelemente**

- [Applikationen veröffentlichen](#)

## **Startobjekt**

- [Visual Basic - Grundlagen der Sprache](#)

## **StartSysInfo**

- [Formularvorlagen](#)

## **StartUpPosition, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Statusleiste**

- [Willkommen bei Visual Basic](#)

## **StatusUpdate-Ereignis**

- [Grafik und Multimedia](#)

## **Steuerelement**

- [Steuerelemente - Verwaltung](#)
- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

aktivieren

- [Steuerelemente - Verwaltung](#)

Anzahl

- [Formulare](#)

Anzeige

- [Erstes Arbeiten mit Visual Basic](#)

Ausrichtung

- [Steuerelemente - Verwaltung](#)

Befehlsschaltfläche

- [Erstes Arbeiten mit Visual Basic](#)

benutzerdefinierte

- [Erstes Arbeiten mit Visual Basic](#)

Bezeichnungsfeld

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

Bildfeld

- [Erstes Arbeiten mit Visual Basic](#)

Breite

- [Steuerelemente - Verwaltung](#)

Dateilistenfeld

- [Erstes Arbeiten mit Visual Basic](#)

Daten



- [Erstes Arbeiten mit Visual Basic](#)

Eigenschaft

- [Willkommen bei Visual Basic](#)

Eigenschaften

- [Erstes Arbeiten mit Visual Basic](#)

Eigenschaftenfenster

- [Erstes Arbeiten mit Visual Basic](#)

einfügen

- [Erstes Arbeiten mit Visual Basic](#)

Figur

- [Erstes Arbeiten mit Visual Basic](#)

HBildlaufleiste

- [Erstes Arbeiten mit Visual Basic](#)

Hintergrundfarbe

- [Steuerelemente - Verwaltung](#)

Höhe

- [Steuerelemente - Verwaltung](#)

Kombinationsfeld

- [Erstes Arbeiten mit Visual Basic](#)

Kontrollkästchen

- [Erstes Arbeiten mit Visual Basic](#)

Laufwerkslistenfeld

- [Erstes Arbeiten mit Visual Basic](#)

Linien

- [Erstes Arbeiten mit Visual Basic](#)

Listenfeld

- [Erstes Arbeiten mit Visual Basic](#)

Optionsfeld

- [Erstes Arbeiten mit Visual Basic](#)

Position

- [Steuerelemente - Verwaltung](#)

Präfix

- [Erstes Arbeiten mit Visual Basic](#)

Rahmen

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

Schriftart

- [Steuerelemente - Verwaltung](#)

TabStrip

- [Formularvorlagen](#)

Text

- [Steuerelemente - Verwaltung](#)

Textfeld

- [Erstes Arbeiten mit Visual Basic](#)

übergeben

- [So funktionieren komplexe Programme](#)

VBildlaufleiste

- [Erstes Arbeiten mit Visual Basic](#)

Verzeichnislistenfeld

- [Erstes Arbeiten mit Visual Basic](#)

Vordergrundfarbe

- [Steuerelemente - Verwaltung](#)

Werkzeugsammlung

- [Steuerelemente - Verwaltung](#)

Zeiger

- [Erstes Arbeiten mit Visual Basic](#)

Zeitgeber

- [Erstes Arbeiten mit Visual Basic](#)

## **Steuerelement-Ereignisse**

- [Steuerelemente - Verwaltung](#)

## **Steuerelementfeld**

- [Objekte in Visual Basic](#)

## **Steuerelementfokus**

- [Steuerelemente - Verwaltung](#)

## **Steuerelement-Objekte**

- [Objekte in Visual Basic](#)

## **Steuerformular**

- [Formulare](#)

## **Steuermenü**

- [Steuerelemente - Verwaltung](#)

## **Steuervariable**

- [So funktionieren komplexe Programme](#)

## **Str()**

- [So funktionieren komplexe Programme](#)

## Stretch-Eigenschaft

- [Grafik und Multimedia](#)

## String

- [Visual Basic - Grundlagen der Sprache](#)

Deklaration

- [Visual Basic - Grundlagen der Sprache](#)

kürzen

- [So funktionieren komplexe Programme](#)

Länge

- [So funktionieren komplexe Programme](#)

leerer

- [Visual Basic - Grundlagen der Sprache](#)

umwandeln

- [So funktionieren komplexe Programme](#)

## Stringausgaben, formatieren

- [So funktionieren komplexe Programme](#)

## String-Funktionen

- [So funktionieren komplexe Programme](#)

## Stringverknüpfung

- [Visual Basic - Grundlagen der Sprache](#)

## StrReverse()

- [So funktionieren komplexe Programme](#)

## STRUCTURE

- [Das Windows-API](#)

## Structured Query Language

- [Formularvorlagen](#)

## Sub

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## Subtraktion

- [Visual Basic - Grundlagen der Sprache](#)

## Symbolleiste

- [Willkommen bei Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)
- [Formulare](#)

mehrzeilig

- [Formulare](#)

## Syntax

- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

## Syntaxfarbgebung

- [Bonus-Projekt 1: Steuerelemente, Eigenschaften und Ereignisse](#)

## Syntaxfehler

- [Applikationen veröffentlichen](#)

## System, booten

- [Das Windows-API](#)

## System-Info

- [Willkommen bei Visual Basic](#)
- [Formularvorlagen](#)

## Systemobjekte

- [Objekte in Visual Basic](#)

## T

## Tab()

- [Formulare](#)

## Tabelle

- [Auf Daten zugreifen](#)  
erstellen
  - [Auf Daten zugreifen](#)Schleifen
  - [Mehrdimensionale Arrays](#)

## Tabellen-Steuerelement

- [Mehrdimensionale Arrays](#)
- [Mehrdimensionale Arrays](#)  
Bilder
  - [Mehrdimensionale Arrays](#)

## TabIndex

- Eigenschaft
  - [Steuerelemente - Verwaltung](#)
  - [Steuerelemente - Verwaltung](#)

## Tabulatortaste

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## TabStrip-Steuerelement

- [Formularvorlagen](#)

## Tan()

- [So funktionieren komplexe Programme](#)

## **Tangens**

- [So funktionieren komplexe Programme](#)

## **Tastatur**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Tastencodes**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

senden

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **tbsOptions\_Click()**

- [Formularvorlagen](#)

## **TCP**

- [Internet-Zugriff](#)

## **TCP/IP**

- [Internet-Zugriff](#)

## **Teilstring-Funktionen**

- [So funktionieren komplexe Programme](#)

## **Testprozeß**

- [Willkommen bei Visual Basic](#)

## **Text**

- [Steuerelemente - Verwaltung](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)
- [Steuerelemente - Verwaltung](#)

Steuerelement

- [Steuerelemente - Verwaltung](#)

## **TextBox**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Textcursor**

- [Steuerelemente - Verwaltung](#)
- [Formulare](#)

plazieren

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Textdateien**

mehrsprachige

- [Willkommen bei Visual Basic](#)

## **Text-Eigenschaft**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Textfeld**

- [Steuerelemente - Verwaltung](#)

Standardwert

- [Steuerelemente - Verwaltung](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

Textausrichtung

- [Steuerelemente - Verwaltung](#)

Zeichenmenge

- [Steuerelemente - Verwaltung](#)

## **Textstrings**

- [Visual Basic - Grundlagen der Sprache](#)

## **Time**

- [So funktionieren komplexe Programme](#)

## **Timer**

- [Erstes Arbeiten mit Visual Basic](#)
- [Mehr Power mit Maus und Steuerelementen](#)

Eigenschaften

- [Mehr Power mit Maus und Steuerelementen](#)

## **Timer-Ereignis**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Timer-Funktion**

- [So funktionieren komplexe Programme](#)

## **TimeSerial()**

- [So funktionieren komplexe Programme](#)

## **TimeValue()**

- [So funktionieren komplexe Programme](#)

## **Tipofday.txt**

- [Formularvorlagen](#)

## **Tips und Tricks**

- [Formularvorlagen](#)
- [Formularvorlagen](#)
- [Formularvorlagen](#)

## **Tips-Auflistung**

- [Formularvorlagen](#)

## **Titelleiste, Inhalt**

- [Erstes Arbeiten mit Visual Basic](#)

## **Title**

- [Objekte in Visual Basic](#)

## **tmr**

- [Erstes Arbeiten mit Visual Basic](#)

## **To**

- [Programme steuern](#)

## **ToolBar**

- [Formulare](#)

## **Toolbars**

- [Formulare](#)

## **ToolboxBitmap**

- [ActiveX- Steuerelemente](#)

## **ToolTip**

- [Ein Hilfe- System erstellen](#)

## **ToolTipText, Eigenschaft**

- [Steuerelemente - Verwaltung](#)

## **Top, Eigenschaft**

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **TrackDefault**

- [Drucken](#)

## **Transfer Control Protocol / Internet Protocol**

- [Internet-Zugriff](#)

## **Transmission Control Protocol**

- [Internet-Zugriff](#)

## **TreeView**

- [Formulare](#)

## **Trennlinie**

- [Menüs](#)

## **Trigonometrie**

- [So funktionieren komplexe Programme](#)

## **True**

- [Visual Basic - Grundlagen der Sprache](#)

## **Twip**

- [Erstes Arbeiten mit Visual Basic](#)

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **TwipsPerPixelX**

- [Objekte in Visual Basic](#)

## **TwipsPerPixelY**

- [Objekte in Visual Basic](#)

## **txt**

- [Erstes Arbeiten mit Visual Basic](#)

## **typ**

- [Erstes Arbeiten mit Visual Basic](#)

## **Type-Anweisung**

- [Mit Dateien arbeiten](#)

## **TypeOf**

- [So funktionieren komplexe Programme](#)

# **U**

## **UCase()**

- [So funktionieren komplexe Programme](#)

## **UDP**

- [Internet-Zugriff](#)

## **Überwachungsfenster**

- [Applikationen veröffentlichen](#)

## **Ungleich**

- [Programme steuern](#)

## **Uniform Resource Locator**

- [Willkommen bei Visual Basic](#)
- [Internet-Zugriff](#)

## **Unload**

- [Mehr Power mit Maus und Steuerelementen](#)
- [Formulare](#)

Ereignis

- [Steuerelemente - Verwaltung](#)

## **Unterklasse**

- [ActiveX- Steuerelemente](#)

## **Untermenü, Kennzeichner**

- [Menüs](#)

## **Untermenüeinträge**

- [Willkommen bei Visual Basic](#)



## **Unterstrich**

- [Steuerelemente - Verwaltung](#)

## **URL**

- [Willkommen bei Visual Basic](#)
- [Internet-Zugriff](#)

## **User Datagram Protocol**

- [Internet-Zugriff](#)

## **USER32.DLL**

- [Das Windows-API](#)

## **V**

## **Val()**

- [So funktionieren komplexe Programme](#)

## **Value**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Variable**

- [Visual Basic - Grundlagen der Sprache](#)
  - globale
    - [So funktionieren komplexe Programme](#)
  - lokale
    - [Visual Basic - Grundlagen der Sprache](#)
    - [So funktionieren komplexe Programme](#)

## **Variablenamen**

- [Visual Basic - Grundlagen der Sprache](#)

## **Variablen-Arrays**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Variablendeklarationen**

- [Visual Basic - Grundlagen der Sprache](#)

## **Variablen-Gültigkeitsbereiche**

- [So funktionieren komplexe Programme](#)

## **Variablenübergabe**

Referenz

- [So funktionieren komplexe Programme](#)

Wert

- [So funktionieren komplexe Programme](#)

## **Variant**

- [Visual Basic - Grundlagen der Sprache](#)

## **VariantHeight**

- [Formulare](#)

## **VarType()**

- [So funktionieren komplexe Programme](#)
- [So funktionieren komplexe Programme](#)

## **VBA**

- [Willkommen bei Visual Basic](#)

## **vbAbort**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbAbortRetryIgnore**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbErrorHourGlass**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbArrow**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbArrowQuestion**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbCancel**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbCentimeters**

- [Formulare](#)

## **vbCharacters**

- [Formulare](#)

## **vbContainerPosition**

- [Formulare](#)

## **vbContainerSize**

- [Formulare](#)

## **vbCritical**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbCrosshair**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbCustom**

- [Mehr Power mit Maus und Steuerelementen](#)

## **VB-Datenformular**

- [Formularvorlagen](#)

## **vbDefaultButton1**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbExclamation**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbHimetric**

- [Formulare](#)

## **vbHourglass**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbIbeam**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbIconPointer**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbIgnore**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **VBildlaufleiste, Steuerelement**

- [Erstes Arbeiten mit Visual Basic](#)

## **vbInches**

- [Formulare](#)

## **vbInformation**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbKeyBack**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbKeyReturn**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbKeyTab**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbIIColor**

- [Grafik und Multimedia](#)

## **vbLPCustom**

- [Grafik und Multimedia](#)

## **vbLPDefault**

- [Grafik und Multimedia](#)

## **vbLPLarge**

- [Grafik und Multimedia](#)

## **vbLPLargeShell**

- [Grafik und Multimedia](#)

## **vbLPMonochrome**

- [Grafik und Multimedia](#)

## **vbLPSmall**

- [Grafik und Multimedia](#)

## **vbLPSmallShell**

- [Grafik und Multimedia](#)

## **vbLPVGAColor**

- [Grafik und Multimedia](#)

## **vbMillimeters**

- [Formulare](#)

## **vbNo**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbNoDrop**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbOK**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbOKCancel**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbOKOnly**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbPixels**

- [Formulare](#)

## **vbPoints**

- [Formulare](#)

## **vbQuestion**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbRetry**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbRetryCancel**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **VBScript**

- [Internet-Zugriff](#)

## **vbSizeAll**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbSizeNESW**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbSizeNS**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbSizeNWSE**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbSizePointer**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbSizeWE**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbStartUpManual**

- [Formulare](#)

## **vbStartUpOwner**

- [Formulare](#)

## **vbStartUpScreen**

- [Formulare](#)

## **vbStartUpWindowsDefault**

- [Formulare](#)

## **vbTwips**

- [Formulare](#)

## **vbUpArrow**

- [Mehr Power mit Maus und Steuerelementen](#)

## **vbUser**

- [Formulare](#)

## **VBX-Steuerelemente**

- [ActiveX- Steuerelemente](#)

## **vbYes**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbYesNo**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **vbYesNoCancel**

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## **Vcr**

- [Grafik und Multimedia](#)

## **Vergleichsoperatoren**

- [Programme steuern](#)

## **Verknüpfung**

- [Objekte in Visual Basic](#)

## **Verschiebeoperationen**

- [Mehr Power mit Maus und Steuerelementen](#)

## **Versionsverwaltung**

- [Erstes Arbeiten mit Visual Basic](#)

## **Verzeichnis**

- [Mit Dateien arbeiten](#)

## **Verzeichnislistenfeld**

- [Mit Dateien arbeiten](#)

Steuerelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Video-Clips**

- [Grafik und Multimedia](#)

## **Videodateien, digitale**

- [Grafik und Multimedia](#)

## **Videodatei-Player**

- [Grafik und Multimedia](#)

## **Videodisc**

- [Grafik und Multimedia](#)

## **Videodisc-Player**

- [Grafik und Multimedia](#)

## **Video-Player**

- [Grafik und Multimedia](#)

## **Videorecorder**

- [Grafik und Multimedia](#)

## **VisData-Fenster**

- [Auf Daten zugreifen](#)

## **Visible**

- [Bonus-Projekt 3: Benutzereingaben und bedingte Logik](#)

Eigenschaft

- [Steuerelemente - Verwaltung](#)

## **Visual Basic Enterprise Edition**

- [Willkommen bei Visual Basic](#)

## **Visual Basic für Applikationen**

- [Willkommen bei Visual Basic](#)

## **Visual Basic Professional Edition**

- [Willkommen bei Visual Basic](#)

## **Visual Basic, Daten**

- [Visual Basic - Grundlagen der Sprache](#)

## **Visual Data Manager**

- [Auf Daten zugreifen](#)

## **Visual Studio**

- [Willkommen bei Visual Basic](#)

## **Visual-Basic-Menü**

- [Menüs](#)

## **Visual-Basic-Programm**

- [Willkommen bei Visual Basic](#)

## **Visual-Basic-Projekt**

- [Willkommen bei Visual Basic](#)

## **Visual-Basic-Standarddrucker**

- [Drucken](#)

## **Vordergrundfarbe, Steuerelement**

- [Steuerelemente - Verwaltung](#)

## **Voreinstellung, Symbolleiste**

- [Erstes Arbeiten mit Visual Basic](#)

## **Vorhanden**

- [Willkommen bei Visual Basic](#)

## **vsb**

- [Erstes Arbeiten mit Visual Basic](#)

## **VScrollBar**

- [Erstes Arbeiten mit Visual Basic](#)

# **W**

## **Wait-Eigenschaft**

- [Grafik und Multimedia](#)

## **Wave-Datei**

- [Grafik und Multimedia](#)

## **Webbrowser**

- [Willkommen bei Visual Basic](#)
- [Formularvorlagen](#)
- [Internet-Zugriff](#)

## **Weekday()**

- [So funktionieren komplexe Programme](#)

## **Werkzeugsammlung**

- [Erstes Arbeiten mit Visual Basic](#)
- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## **Wert**

- [So funktionieren komplexe Programme](#)  
ganzzahliger

- [Visual Basic - Grundlagen der Sprache](#)

serieller

- [So funktionieren komplexe Programme](#)

## Wertebereich

- [Visual Basic - Grundlagen der Sprache](#)

## WhatsThisButton

- [Ein Hilfe- System erstellen](#)

## WhatsThisHelp

- [Ein Hilfe- System erstellen](#)

## WhatsThisHelpID

- [Ein Hilfe- System erstellen](#)

## Width

- [Drucken](#)
- [Grafik und Multimedia](#)
- [Objekte in Visual Basic](#)

Eigenschaft

- [Erstes Arbeiten mit Visual Basic](#)
- [Steuerelemente - Verwaltung](#)

## Wiederholen

- [Eingabe/Ausgabe: Tastatur und Bildschirm](#)

## Wildcard

- [Programme steuern](#)

## Win32.api

- [Das Windows-API](#)

## Windows Help Viewer

- [Ein Hilfe- System erstellen](#)

## Windows-API

- [Applikationen veröffentlichen](#)
- [Das Windows-API](#)
- [Das Windows-API](#)

## Windows-API-Routinen

- [Das Windows-API](#)

## Windows-Standarddrucker

- [Drucken](#)

## WindowState, Eigenschaft

- [Steuerelemente - Verwaltung](#)

## Windows-Steuerelemente, Ereignisse



- [Willkommen bei Visual Basic](#)

## **WinHelp**

- [Ein Hilfe- System erstellen](#)
- [Ein Hilfe- System erstellen](#)

## **WinHelp-Hilfedatei**

- [Ein Hilfe- System erstellen](#)

## **WinSock**

- [Internet-Zugriff](#)

## **With..End With**

- [Objekte in Visual Basic](#)

## **WordWrap**

- [Mehrdimensionale Arrays](#)  
Eigenschaft
  - [Steuerelemente - Verwaltung](#)

## **World Wide Web**

- [Internet-Zugriff](#)

## **Wrappable-Eigenschaft**

- [Formulare](#)

## **Write**

- [Mit Dateien arbeiten](#)

## **Write #**

- [Mit Dateien arbeiten](#)

## **WWW-Seiten**

- [Internet-Zugriff](#)

## **X**

## **Xor**

- [Programme steuern](#)

## **Y**

## **Y2K-Fehler**

- [Visual Basic - Grundlagen der Sprache](#)

## **Year()**

- [So funktionieren komplexe Programme](#)

## **Z**

## **Zahlen**

- [Visual Basic - Grundlagen der Sprache](#)

ganze

- [Visual Basic - Grundlagen der Sprache](#)

## **Zahlensystem, hexadezimal**

- [Menüs](#)

## **Zeichenmethoden**

- [Grafik und Multimedia](#)

## **Zeichen-Steuererelemente**

- [Grafik und Multimedia](#)

## **Zeiger**

- [Steuererelemente - Verwaltung](#)

Steuererelement

- [Erstes Arbeiten mit Visual Basic](#)

## **Zeile**

fortsetzen

- [Steuererelemente - Verwaltung](#)

Fortsetzung

- [Bonus-Projekt 1: Steuererelemente, Eigenschaften und Ereignisse](#)

## **Zeitgeber, Steuererelement**

- [Erstes Arbeiten mit Visual Basic](#)

## **Zeitstempel**

- [So funktionieren komplexe Programme](#)

## **Zeitwerte**

- [So funktionieren komplexe Programme](#)

## **Zelle**

- [Mehrdimensionale Arrays](#)

## **Zoom**

- [Drucken](#)

## **Zugriff**

wahlfreier

- [Mit Dateien arbeiten](#)

- [Mit Dateien arbeiten](#)

## **Zugriffsbeschränkungen**

- [Mit Dateien arbeiten](#)

## **Zuweisung**

- [Visual Basic - Grundlagen der Sprache](#)
-

# Inhaltsverzeichnis

## Woche 1 im Überblick

Tag 1 Willkommen bei Visual Basic

Tag 2 Erstes Arbeiten mit Visual Basic

Tag 3 Steuerelemente - Verwaltung

Bonus-Projekt 1:  
Steuerelemente, Eigenschaften und Ereignisse

Tag 4 Menüs

Tag 5 Visual Basic - Grundlagen der Sprache

Bonusprojekt 2: Variablen und Ausdrücke

Tag 6 Programme steuern

Tag 7 Eingabe/Ausgabe: Tastatur und Bildschirm

Bonus-Projekt 3: Benutzereingaben und bedingte Logik

## Woche 2 im Überblick

Tag 8 So funktionieren komplexe Programme

Tag 9 Dialogfelder

Tag 10 Mehr Power mit Maus und Steuerelementen

Bonusprojekt 4: Mehrfachauswahl in Listefeldern

Bonusprojekt 5: Die Übung mit der Maus

Tag 11 Formulare

Tag 12 Mit Dateien arbeiten

## Bonusprojekt 6: Dateien lesen und schreiben

### Tag 13 Drucken

### Tag 14 Grafik und Multimedia

## Bonusprojekt 7: Bildlaufleisten

## Woche 3 im Überblick

### Tag 15 Formularvorlagen

### Tag 16 Objekte in Visual Basic

### Tag 17 ActiveX-Steuerelemente

## Bonusprojekt 8: Elemente, die Ihre Applikationen verschönern

### Tag 18 Auf Daten zugreifen

## Bonusprojekt 9: ADO-Steuerelemente

### Tag 19 Internet-Zugriff

### Tag 20 Ein Hilfe-System erstellen

### Tag 21 Applikationen veröffentlichen

## Bonustag 22 Mehrdimensionale Arrays

## Bonustag 23 Das Windows-API

## Anhang A: Lösungen

## Anhang B: Operator-Prioritäten

## Anhang C: ASCII-Tabelle

## Anhang D: Die CD zum Buch

## Stichwortverzeichnis

---