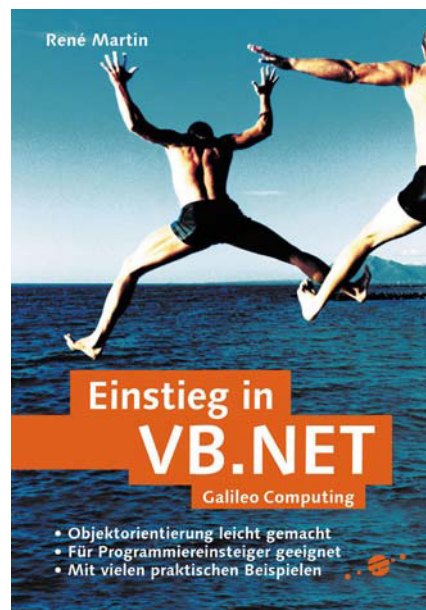


René Martin

# Leseprobe : **Einstieg in VB.NET**



Galileo Computing 

# Inhalt

Vorwort	11
Warum VB.NET?	11
Warum kein anderes Tool?	11
Warum ein weiteres Buch über VB.NET?	11
Über dieses Buch	12
Zur CD-ROM	12
Die Beispiele	12
Die Hinweise in diesem Buch	13
Die Entstehung des Buchs	14

---

<b>1</b>	<b>Die Grundlagen</b>	<b>15</b>
1.1	Nach der Installation	15
1.2	Der erste Start	16
1.3	Das erste Formular	16
1.4	Eigenschaften	17
1.4.1	Regeln für Namen	20
1.4.2	Die Eigenschaften	20
1.5	Die Toolbox	23
1.6	Das Meldungsfenster	24
1.6.1	Die Code-Eingabe für die MessageBox	25
1.6.2	Ein Wort zum »alten« Visual Basic	29
1.6.3	Die Code-Eingabe allgemein	29
1.7	Ein Textfeld	34
1.8	Texte verketteten	34
1.9	Andere Zeichen	35
1.9.1	Noch ein Wort zum Code-Tippen	36
1.9.2	Ein Wort zur Syntax	37
1.10	Variablen	38
1.11	Ein theoretischer Exkurs: Werte- und Referenztypen	44
1.12	Zuweisen von Werten an Variablen	44
1.13	Verzweigungen	45
1.14	Einige Hilfen für den Benutzer	49
1.14.1	Setze den Cursor in ein Textfeld	49
1.14.2	Ok und Abbrechen	49
1.14.3	Die Reihenfolge der Steuerelemente	49

1.15	Ein zweites Textfeld	51
1.15.1	Vergleiche	56
1.16	Rechnen in Basic	58
1.17	Optionsfelder	62
1.17.1	Schnell und ordentlich mehrere Steuerelemente platzieren	64
1.18	Ein zweites Gruppenfeld	66
1.19	Dynamische Steuerelemente	68
1.20	Das Ereignis »Klick«	69
1.21	Prozeduren	72
1.22	Ein theoretischer Exkurs: Prozeduren	74
1.23	Module	75
1.24	Ein theoretischer Exkurs: Objekte, Eigenschaften und Methoden	77
1.24.1	Objekte	77
1.24.2	Methoden	77
1.24.3	Eigenschaften	78
1.25	Zusammenfassung	82

---

## 2 Aufbauwissen VB.NET 83

2.1	Ein zweiter Dialog	83
2.2	Ein Süßigkeitenautomat	84
2.3	Eine Listbox	85
2.4	With ... End With	86
2.5	Die Liste	87
2.6	Eine ComboBox	90
2.7	Die Select-Case-Verzweigung	91
2.8	Ein theoretischer Exkurs: Ereignisse	95
2.8.1	Dynamisch ein- und ausblenden	102
2.8.2	Dynamische Größenänderung	106
2.9	Zwei völlig überflüssige Beispiele	107
2.10	Schleifen	110
2.11	For ... Next	111
2.12	Do ... Loop	112
2.13	Eine Schleife über den Süßigkeiten	114
2.14	Ein theoretischer Exkurs: Sammlungen oder Collections	115
2.15	Die For ... Each-Schleife	116
2.16	Schleifen spielerisch	122
2.16.1	Primzahlen	122
2.16.2	Fibonacci-Zahlen	127

- 2.16.3 ggT und kgV 129
- 2.16.4 Ein Scherz zum Schluss 130
- 2.17 **Rekursives Programmieren** 131
- 2.18 **Arrays (Datenfelder)** 133
  - 2.18.1 Eindimensionale Arrays 133
  - 2.18.2 Mehrdimensionale Arrays 135
- 2.19 **Konstanten** 136
- 2.20 **Funktionen** 137
- 2.21 **Zusammenfassung** 140

---

### **3 Fehler 141**

- 3.1 **Fehler abfangen** 141
- 3.2 **Programmierhilfen** 147
  - 3.2.1 IntelliSense 147
  - 3.2.2 Automatische Hilfe 148
  - 3.2.3 Automatische Prüfung 148
  - 3.2.4 Der Zwischenablagereing 148
  - 3.2.5 Kommentare 148
  - 3.2.6 Aufgabenkommentare 148
- 3.3 **Fehlersuche im Code** 149
  - 3.3.1 Ergebnisse und Zwischenergebnisse 149
  - 3.3.2 Haltepunkte 149
  - 3.3.3 Das Lokalfenster 150
  - 3.3.4 Das Überwachen-Fenster 150
  - 3.3.5 Das Direktfenster 151
  - 3.3.6 Einzelschrittmodus 151
- 3.4 **Zusammenfassung** 152

---

### **4 Zugriff auf andere Programme: Excel 153**

- 4.1 **Austausch mit Excel** 153
- 4.2 **Das Excel-Objektmodell** 153
  - 4.2.1 Das Objekt Application 153
  - 4.2.2 Zugriff auf Excel-Arbeitsmappen 155
  - 4.2.3 Zugriff auf Tabellenblätter 156
- 4.3 **Zugriff von außen auf Excel** 166

<hr/>	<b>5</b>	<b>Dateizugriff 171</b>
	5.1	Prüfen, ob eine Datei vorhanden ist 171
	5.2	Textdateien lesen und schreiben 173
	5.3	Der OpenFileDialog 176
	5.4	DriveListBox, DirListBox und FileListBox 181
	5.5	Fehler 188
	5.6	Bitte warten Sie ... 189
	5.7	Zusammenfassung 192
<hr/>	<b>6</b>	<b>Strings und Datumsangaben 195</b>
	6.1	Verketteten von Strings 195
	6.2	Strings zerlegen 195
	6.3	String und Format 206
	6.4	Datumsfunktionen 208
	6.5	Zusammenfassung 213
<hr/>	<b>7</b>	<b>Eigene Objekte 215</b>
	7.1	Objekte 215
	7.2	Eine eigene Klasse 215
	7.2.1	Members 217
	7.2.2	Eigenschaften 218
	7.2.3	Methoden 221
	7.2.4	Konstruktoren 223
	7.2.5	Überladen 223
	7.2.6	Vererbung 224
	7.2.7	Überschreiben 225
	7.2.8	Polymorphismus 227
<hr/>	<b>8</b>	<b>Eine Rechnung in Word 229</b>
	8.1	Die neue Klasse 229
	8.2	Die Klasse wird verwendet 232
	8.3	Eine Klasse für den Excel-Zugriff 235
	8.4	Eine Klasse für den Word-Zugriff 240
	8.5	Einige Objekte in Word 240
	8.5.1	Der Makrorekorder 241
	8.5.2	Application 241
	8.5.3	Methoden und Eigenschaften von Document und Documents 241
	8.5.4	Text in Word schreiben 243

- 8.5.5 Tabellen in Word 245
- 8.5.6 Eine Klasse für den Word-Zugriff 247
- 8.5.7 clsWord wird verwendet – Variante 1: Text hineinschreiben 248
- 8.5.8 Variante 2: Text in eine Dokumentvorlage an Textmarken schreiben 251
- 8.5.9 Variante 3: Eine geschützte Dokumentvorlage mit Formularfeldern 254
- 8.5.10 Word-Formulare steuern 256
- 8.6 Zusammenfassung 260

---

## **9 Datenbankzugriff 261**

- 9.1 Tabellen 261
  - 9.1.1 Datentypen 262
  - 9.1.2 Primärschlüssel 265
  - 9.1.3 Die Feldeigenschaften 265
  - 9.1.4 Standardwert 265
  - 9.1.5 Beschriftung 266
  - 9.1.6 Ändern von Datentypen 266
- 9.2 Abfragen 266
  - 9.2.1 Sortieren in einer Abfrage 267
  - 9.2.2 Filtern in einer Abfrage 269
- 9.3 Beziehungen zwischen Tabellen 271
- 9.4 Zugriff von VB.NET auf die Datenbank 272
  - 9.4.1 Gebundene Formulare 272
  - 9.4.2 Steuerung des Formulars 279
  - 9.4.3 Gebundene Formulare ohne Assistent 282
  - 9.4.4 Ungebundene Formulare 285

---

## **10 XML-Dokumente 295**

- 10.1 XML-Grundlagen 295
  - 10.1.1 Weitere Tags 299
  - 10.1.2 Der Zeichensatz 302
  - 10.1.3 Abkürzungen 303
  - 10.1.4 CDATA-Abschnitte 303
  - 10.1.5 Kommentare 305
  - 10.1.6 Attribute 306
  - 10.1.7 Mehr als zwei Möglichkeiten bei Attributen 307
  - 10.1.8 Ein Attribut oder mehrere Attribute im Dokument 310
  - 10.1.9 Attribut oder Element? 310
  - 10.1.10 Zusammenfassung von XML 311
- 10.2 XML, HTML und XSL – die Ausgabe 312
  - 10.2.1 XSL 313
  - 10.2.2 Der Aufbau einer XSL-Datei 313
  - 10.2.3 Mehrere Kindelemente anzeigen 315

10.2.4	Alle Elemente einer Ebene anzeigen	318
10.2.5	Attribute auslesen	318
10.2.6	STYLE in SPAN und DIV	319
10.2.7	Tabellen	323
10.2.8	Daten sortieren	325
10.2.9	Daten filtern	326
10.2.10	Ebenen durchlaufen	328
10.2.11	Entscheidungen	329
10.2.12	Bilder	335
10.2.13	Hyperlinks	339
10.2.14	Zusammenfassung von XSL	341
<b>10.3</b>	<b>Mit VB.NET nach XML und zurück</b>	<b>342</b>
10.3.1	Das DOM	345
10.3.2	Ein neues XML-Dokument erzeugen	345
10.3.3	Neue Elemente erzeugen	347
10.3.4	XML-Dokumente auslesen	352
10.3.5	Zugriff auf Attribute	353
10.3.6	Durchlaufen von mehreren Ebenen	354
10.3.7	Weitere XML-Elemente in VB.NET	356
10.3.8	Erzeugen von XML-Objekten und Eigenschaften	361
10.3.9	Löschen von Elementen im XML-Dokument	367
10.3.10	Transformationen von XML mit dem DOM	367
10.3.11	Zusammenfassung VB.NET und XML	370
<b>10.4</b>	<b>Ein Beispiel</b>	<b>370</b>
10.4.1	Das TreeView-Steuerelement	371
10.4.2	Daten werden eingelesen	374
10.4.3	Export in eine XML-Datei	377
<b>10.5</b>	<b>Zusammenfassung</b>	<b>382</b>

---

## **11 Grafik 383**

**Zum Schluss 389**

**Index 391**

# **Vorwort**

## **Warum VB.NET?**

Mit VB.NET ist Microsoft ein beeindruckender Schachzug gelungen. Etwa zwei Jahre lang hat ein großes Team von Entwicklern an dieser Programmiersprache gearbeitet. Herausgekommen ist ein gutes, mächtiges und vielfältiges Werkzeug, das nicht nur alles in den Schatten stellt, was aus dem Hause Microsoft kommt, sondern auch alternative, ähnliche Tools von anderen Herstellern.

Während Kritiker in den vergangenen Jahren geunkt haben, dass Visual Basic keine echt objektorientierte Programmiersprache sei (womit sie übrigens Recht hatten), so ist nun der Kreis derer, die auf VB.NET-Programmierer herabsehen, sehr klein geworden. Inzwischen muss sich diese Programmiersprache nicht mehr hinter Java oder C++ verstecken. Im Gegenteil, ein Datenbankzugriff, wie er mit ADO.NET zur Verfügung gestellt wird, sucht lange seinesgleichen.

## **Warum kein anderes Tool?**

Ich gehöre zu einer Generation von Computeranwendern, die während der Schulzeit BASIC gelernt hat, im Studium mit Pascal konfrontiert wurde und sich anfangs mit den Techniken und der Syntax von Java schwer getan hat. Mir fällt der Umstieg von Visual Basic auf VB.NET nicht sehr schwer. Jeder, der schon einmal mit einer Datenbank, Programmiersprache oder Applikation aus dem Hause Microsoft zu tun hatte, der wird, wenn er sich in VB.NET einarbeiten möchte, schnell einen Zugang dazu finden.

## **Warum ein weiteres Buch über VB.NET?**

Neben mir stehen mehrere Bücher über dieses Werkzeug. Einige sind für Umsteiger konzipiert, andere für Programmierer. Mein Buch wendet sich in erster Linie an Einsteiger. Manche der bereits erschienenen Bücher haben einen Umfang von 1.000 Seiten und schrecken Anfänger somit sicherlich ab. Jedes beschreibt dabei nur einen Teil von VB.NET. Einige Bücher beschränken sich auf den Sprachkern und das Konzept von VB.NET, lassen aber Userforms völlig außer Acht. Andere behandeln lediglich Datenbanken oder Internetanwendungen. In einigen der existierenden Bücher findet sich zum Thema XML noch nicht einmal ein Eintrag im Register, und das obwohl VB.NET eine eigene Klasse dafür zur Verfügung stellt. Auch mein Buch stößt an Grenzen. Kein noch so kompetenter Programmierer wird jemals den gesamten Leistungsumfang dieser Programmiersprache vollständig kennen. Aber wenn man sich darauf einlässt, das ist meine Erfahrung, dann findet man sich relativ schnell darin zurecht und kann das Gelernte auf anderen Gebieten anwenden.



## Über dieses Buch

Drei Schwerpunkte bilden die zentralen Themen des vorliegenden Buchs.

In den ersten Kapiteln wird der **Sprachkern** erläutert. Für Umsteiger von VB auf VB.NET ist das sicherlich keine große Hürde. Interessant ist allerdings Kapitel 7, in welchem die Theorie der Objektorientierung erläutert wird.

Da sehr viele Programme Ein- und Ausgabeformulare für den Benutzer zur Verfügung stellen, nimmt dieses Thema einen großen Raum ein. Es geht um die wichtigsten **Steuerelemente**, die VB.NET zur Verfügung stellt, um das Abfangen von Benutzereingaben (Sie glauben gar nicht, welche Eingabe-Fehler manche Anwender machen) und um dynamische Dialoge zu ermöglichen, die dem User den Arbeitsalltag erleichtern.

Das dritte große Thema des Buchs ist der **Datenaustausch**. Es wird gezeigt, wie man von VB.NET auf Excel zugreift, Daten in eine Tabelle schreibt und wieder ausliest und in Word darstellt. Beschrieben wird der Datenaustausch mit einer Datenbank (am Beispiel Access). Und schließlich werden einige Möglichkeiten aufgezeigt, wie Daten in das universelle **XML-Format** geschrieben werden können, wo sie beispielsweise mit XSL(T) transformiert und in einem Browser dargestellt werden können.

Den Abschluss bildet ein kurzes Kapitel über **Grafik-Programmierung** – eigentlich mehr um weitere Möglichkeiten von VB.NET aufzuzeigen, als um dieses Thema vollständig auszuloten. Verzichtet wurde aus Platzgründen auf das Thema Internetprogrammierung, das heißt Formulare und Auswertung von Formulareingaben im Browser.

## Zur CD-ROM

Im Folgenden finden Sie viele Beispiele, die alle auf der CD-ROM vorhanden sind. Sie können von dort geöffnet und geladen werden, damit das im Text Beschriebene leichter nachvollzogen werden kann. Dort finden Sie elf Ordner mit den Namen »Kapitel01«, »Kapitel02«, ... In ihnen sind die einzelnen Dateien abgespeichert. Sie können direkt von der CD-ROM geöffnet oder aber auch auf die Festplatte kopiert werden. Beachten Sie, dass einige Dateien auf Pfade und Dateinamen zugreifen. Diese müssen möglicherweise geändert werden.

## Die Beispiele

Vielleicht wird es einige Leser erstaunen, dass dieses Buch mit der neuesten Programmiersprache auf Daten aus Office 97 (Word, Excel und Access) zugreift. Dies hat zweierlei Gründe. Zum einen erlebe ich, dass viele Firmen noch nicht von

Office 97 auf 2000 oder 2002 umgestellt haben. Die Unterschiede zwischen dem Paket 97 und der Version XP sind für den Büroalltag gering, der Aufwand und die Kosten wären zu hoch. Damit alle Beispiele dennoch von allen drei Systemen gleichermaßen nachvollzogen werden können, habe ich mich bewusst dafür entschieden, nicht das zur Zeit neueste XP-Paket zu verwenden, sondern die älteste mögliche Version. Dies hat einen programmiertechnischen Vorteil: Die Beispiele laufen auf allen Rechnern, auf denen eine dieser drei Versionen installiert ist – egal welche. Hätte ich als COM-Verknüpfungen Office 2002 eingebunden, so hätten die Anwender der älteren Versionen das Nachsehen, beziehungsweise müssten neue Verweise einbinden.

Der Nachteil dieses Vorgehens liegt auf der Hand: Wenn Sie eine neue Version von Word, Excel oder Access haben, dann heißen einige Menüpunkte anders und einige Dialoge sehen anders aus als die hier dargestellten. Aber sicherlich werden Sie sich problemlos auch darin zurecht finden. Ich habe VBA-Befehle verwendet, die in allen drei Versionen gültig sind; es würde zu weit führen, per Programmierung die installierte Version abzufragen und darauf zu reagieren. Wer in solch inhomogenen Arbeitswelten programmiert, wird dieses Problem kennen und sicherlich dafür auch schon Lösungen gefunden haben.

### Die Hinweise in diesem Buch

Folgende Tabelle gibt eine Übersicht zu den im Buch verwendeten Schreibkonventionen:

Element	Beispiel	Formatierung
Dateinamen mit oder ohne Pfad	<i>C:\Eigene Dateien\Sonstiges\Dinos.vsd</i>	<i>Kursiv</i>
Menüs	Datei • Drucken	<b>Fett</b>
spezielle Bezeichnungen, die im Text mitgelesen werden können, und Punkte auf Dialogblättern, Steuerelemente, Beschriftungen, Titel, Namen von Symbolleisten	»True« »Laufweite« »Standard«	in »Anführungszeichen«
Tasten	<b>Strg</b> + <b>A</b>	<b>Hervorgehoben</b>
Programmiercode, Schlüsselwörter und Variablennamen	<code>=Me.Close()</code>	Nichtproportional-schrift

Die Schreibkonventionen in diesem Buch

Ich habe versucht, Einrückungen des Codes so darzustellen, wie es VB.NET macht. An einigen Stellen bin ich allerdings bei sehr langen Zeilen in Konflikt mit den Zeilen des Buchs geraten. Bitte sehen Sie also darüber hinweg, wenn die Ansicht auf Ihrem Bildschirm nicht immer zu 100 % mit der Ansicht im Buch übereinstimmt.

### **Die Entstehung des Buchs**

Auf die Idee, dieses Buch zu schreiben, bin ich bei meiner ersten VB.NET-Schulung für bit und InfraServ in Gendorf/Burgkirchen (bei Burghausen) gekommen. Dort durfte ich im Frühjahr 2002 eine Woche lang die Programmiersprache für Jugendliche unterrichten. Viele der hier beschriebenen Beispiele wurden in dieser Schulung verwendet, kritische Fragen, Probleme von Ein- und Umsteigern eingearbeitet. Ich habe seit dieser Zeit viel gelernt über VB.NET, aber auch über die Themen, die sich in diesem Umfeld befinden: Applikationen, Datenbanken und XML. Dieses Wissen möchte ich weitergeben und hoffe, dass Sie beim Lesen dieses Buchs und beim Programmieren mit VB.NET ebenso viel Freude und Spaß haben werden wie ich.

**René Martin**

München, im September 2002

# 1 Die Grundlagen

*Dieses Kapitel gibt eine erste Einführung in das Arbeiten mit VB.NET. Es wird gezeigt, wie man die Programmiersprache startet, ein Formular anlegt, Steuerelemente darauf platziert und Code an die Steuerelemente bindet. Außerdem wird beschrieben, wie Variablen in VB.NET deklariert werden und die erste Verzweigung (If) wird erläutert.*

## 1.1 Nach der Installation

Ich gehe davon aus, dass Visual Studio .NET korrekt installiert ist. Bei der Installation sind Sie gefragt worden, welche Komponenten Sie haben möchten. In diesem Buch ist VB.NET das zentrale Thema. Natürlich können (und sollten) alle anderen Komponenten auch installiert werden. Einige von ihnen werden in diesem Buch erläutert, andere, wie C# oder Visual C++, sollen nicht angesprochen werden. Es sind andere Programmiersprachen, die etwa das Gleiche leisten wie VB.NET.



Abbildung 1.1 Der Startbildschirm

## 1.2 Der erste Start

Nach der Installation von Visual Studio .NET starten wir das Programm mit »Microsoft Visual Studio .NET«. Danach heißt uns ein Begrüßungsbildschirm willkommen. Natürlich wollen wir ein »Neues Projekt«. Der Begrüßungsbildschirm bietet uns eine Reihe von Möglichkeiten an.

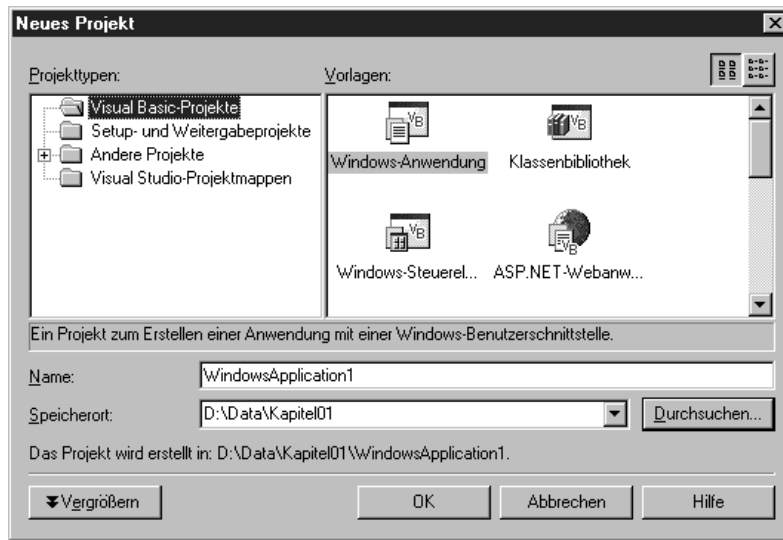


Abbildung 1.2 Ein neues Projekt wird angelegt.

Da wir mit Visual Basic programmieren und die Grundlagen erlernen möchten, beginnen wir mit einer Windows-Anwendung. Über die anderen Projekttypen werden wir noch sprechen. Unten wird ein Name verlangt. Ich vergebe den Namen »Kaffeautomat« und speichere ihn in einem bestimmten Ordner. Dies ist wichtig! Da nicht nur eine .exe-Datei gespeichert wird, sondern noch weitere Komponenten, sollten Sie vorher auf der Festplatte einen Ordner anlegen, in den die ganzen Dateien geschrieben werden. Er sollte natürlich regelmäßig gesichert werden, sodass Sie immer den letzten Stand der Anwendung zur Verfügung haben, falls einmal etwas schief läuft.

## 1.3 Das erste Formular

Und schon geht es los. Normalerweise beginnt ein Projekt mit einem leeren Formular. Das muss es aber nicht – es gibt auch Programme, die im Hintergrund laufen und über gar kein Formular verfügen. Doch zum Lernen sind Formulare ein prima Einstieg. Formulare haben übrigens auch andere Namen: Man nennt sie auch UserForm, Dialog oder Eingabemaske. Egal, wie Sie sie nennen: Das Formu-

lar wird am Ende unsere Maschine sein. Oder noch besser: das Eingabefeld, auf dem der Benutzer auswählt, was er gerne haben möchte. Wäre das Formular nicht da, das heißt, hätten wir mit einem leeren Projekt begonnen, so könnten wir über den Menüpunkt **Projekt · Windows Form hinzufügen** die Maske erstellen.

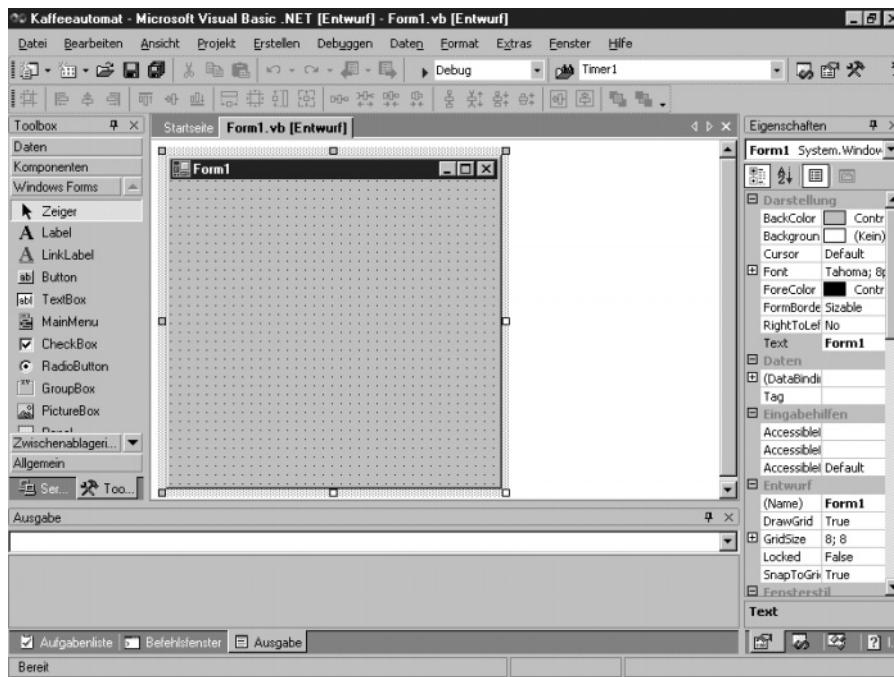


Abbildung 1.3 Das Formular

## 1.4 Eigenschaften

Zuerst sollen einige Eigenschaften des Dialogs festgelegt werden. Dies geschieht im Eigenschaftenfenster, das sich rechts unten auf dem Bildschirm befindet. Sollte es nicht dort sitzen, so kann man es über **Ansicht · Eigenschaftenfenster** (oder **F4**) anzeigen lassen. Es ist in seiner Breite variabel und kann mit der Maus vergrößert und verkleinert werden. Mit ihr kann es ebenso aus der Verankerung gelöst und wieder an seinen Platz zurückgezogen werden, wo es einrastet. Wenn das Fenster stört, weil man den Platz braucht, kann man die Pin-Nadel in der blauen Titelzeile drücken. Dann verschwindet das Fenster bis auf eine Titelzeile am Rand. Wird der Mauszeiger nun dorthin bewegt, dann zeigt sich das Fenster und es kann mit ihm gearbeitet werden. Zieht man den Mauszeiger weg, zieht es sich wieder in seine »Schlafposition« zurück.

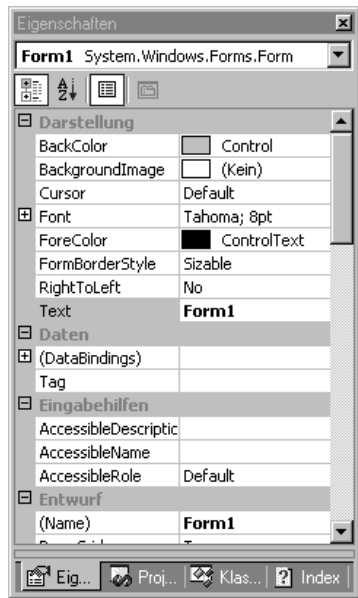


Abbildung 1.4 Das Eigenschaftenfenster

Überhaupt besitzt die Entwicklungsumgebung eine Reihe von Fenstern, die verschiebbar und andockbar sind. Einige von ihnen verbergen sich hinter Registerhenkeln, andere sind möglicherweise ausgeblendet. Finden Sie ein Fenster nicht, so sehen Sie im Menü **Ansicht** nach.

Das Eigenschaftenfenster ist in verschiedene Kategorien unterteilt. Wenn Sie auf die Kategorien verzichten wollen, dann können Sie sich mit dem Symbol nur die Eigenschaften in alphabetischer Reihenfolge anzeigen lassen.

Wenn ich allerdings in der Praxis eine Eigenschaft suche, dann gehe ich über die Kategorien, weil dort schneller klar wird, wie sie heißt und wo sie steckt. Denn normalerweise weiß ich, in welcher Kategorie sich meine Eigenschaft befindet.

Die wichtigste Eigenschaft ist der Name, ihn vergebe ich immer als Erstes. Man könnte den Dialog »Kaffee« nennen. Nun hat sich bei VB-Programmierern eine Konvention durchgesetzt. Wer so etwas sauber macht, der beginnt den Namen von Formularen mit einem »frm«. Der Grund ist einfach: Wir haben am Ende nicht nur zwei Buttons und drei Textfelder, sondern vielleicht mehrere Dutzend Variablen, Steuerelemente, Module, Funktionen und so weiter. Damit sich da noch ein Mensch auskennt, gibt man den Teilchen vernünftige Namen.

Das war fürs Erste schon alles.

Starten kann man die Maschine nun über das Symbol in der Symbolleiste mit dem blauen Pfeil, der nach rechts zeigt (wie beim Kassettenrekorder), über die Taste **F5** oder über das Menü **Debuggen • Starten**. Und schon erscheint der Dialog auf dem Bildschirm. Vielleicht sind Sie ein wenig enttäuscht, denn viel ist da nicht zu sehen. Andererseits ist das ein ganz toller Dialog für Anfänger: Sie können dort nichts kaputt machen.

Den Dialog kann man wie jeden schließen – rechts oben über das Schließen-Symbol »x«.

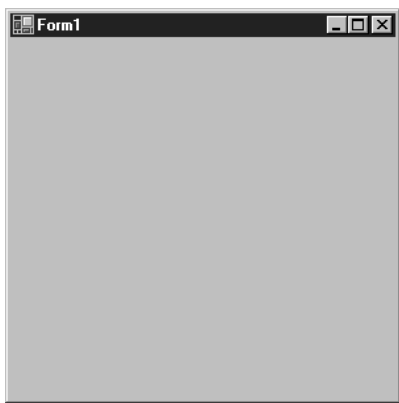


Abbildung 1.5 Das erste Formular

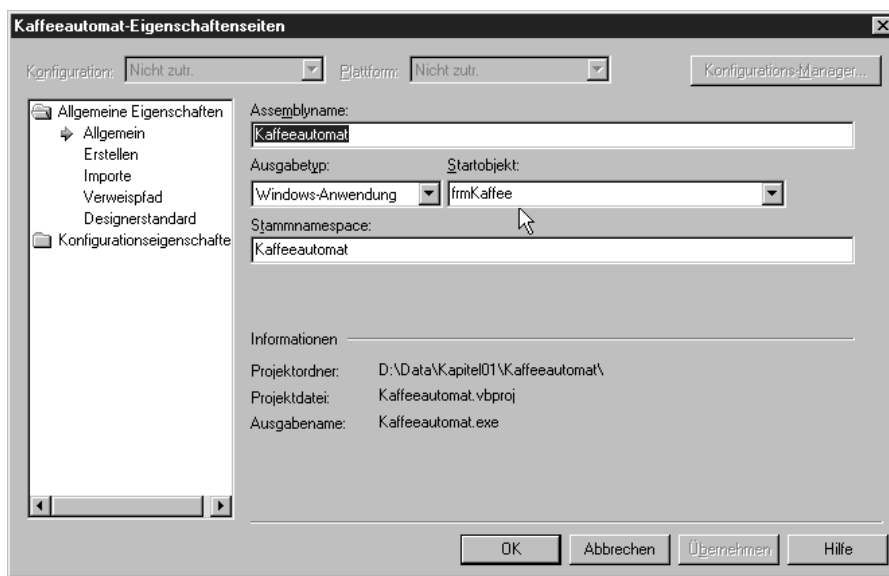


Abbildung 1.6 Hier wird geregelt, welches Formular das Startformular ist.



Achtung: Sollten Sie nach dem Umbenennen des Dialogs Fehlermeldungen erhalten oder sollte das Formular nicht mehr korrekt angezeigt werden, dann liegt es möglicherweise daran, dass das Startformular nicht korrekt eingestellt ist. Im Menü **Ansicht • Eigenschaftenseiten** in der Kategorie »Allgemein« wird festgelegt, wie der Name des Formulars lautet, wenn das Programm startet. Da wir eine Windows-Anwendung gewählt haben, wurde automatisch ein Formular erzeugt, das den Namen »Form1« erhalten hat. Nun kann es passieren, dass beim Umbenennen des Formulars noch der ursprüngliche Eintrag in der Liste zu finden ist. Das muss dann geändert werden.

#### **1.4.1 Regeln für Namen**

Der Name, der für das Formular vergeben wurde, ist nicht ganz beliebig. Er unterliegt einigen Regeln. Der Name darf bis zu 80 Zeichen lang sein und alle Zeichen enthalten, außer Leerzeichen, Satz- und Sonderzeichen: `,:;!»$$%&/(){<>}^`. Auch der Bindestrich »-« darf nicht verwendet werden. Dagegen sind Unterstrich (»\_«), Punkt (».«), Groß- und Kleinschreibung sowie die zehn Ziffern erlaubt. Folgende Namen sind gültig:

- ▶ Tee
- ▶ frmTee
- ▶ frm\_Mein\_erstes\_Formular\_das\_ich\_ganz\_alleine\_erstellt\_habe

Deutsche Umlaute (»ä«, »ö« und »ü«) sind ebenso erlaubt wie das »ß«. Wer Probleme beim internationalen Austausch fürchtet, der sollte auf Umlaute und »ß« verzichten.

#### **1.4.2 Die Eigenschaften**

Es gibt verschiedene Arten von Eigenschaften: Bei einigen Eigenschaften werden Listen von Auswahlmöglichkeiten geöffnet (beispielsweise »Cursor«). Zum Teil sind es Listen mit mehreren Registerblättern (beispielsweise »BackColor«), bei einigen Eigenschaften sind es Dialoge, die erscheinen, wenn man auf das Kästchen mit den drei Punkten klickt (beispielsweise »Font«). Einige Eigenschaften verlangen eine Eingabe (beispielsweise »Tag«), oder eine Datei, die über einen Dialog ausgewählt werden kann (beispielsweise »Icon«). Wenn Sie sich die Eigenschaften nach Kategorien anzeigen lassen, dann können einzelne Kategorien auf- oder zugeklappt werden, was der Übersichtlichkeit dient. Ebenso besitzen manche Eigenschaften mehrere Unterkategorien, die einzeln angezeigt oder in einem Block zusammengefasst werden.

Ich möchte Sie nun nicht mit einer großen, vollständigen Liste aller Eigenschaften langweilen, deshalb beschränke ich mich auf einige wichtige Eigenschaften.

Bis jetzt ist unser Dialog noch grau und langweilig. Ich hätte gerne etwas mehr Farbe. So etwas wird im Eigenschaftensfenster eingestellt. Beginnen wir in der Kategorie »Darstellung«. Mit »BackColor« ist die Hintergrundfarbe gemeint. Ein Mausklick auf die Eigenschaft »BackColor« zeigt einen Pfeil rechts daneben an. Ein Klick darauf öffnet ein weiteres Fenster mit drei Registerblättern. In den Blättern »System«, »Web« und »Benutzerdefiniert« sind schon Farben vorgegeben. Nur im Blatt »Benutzerdefiniert« sind am unteren Rand noch einige weiße Felder. Ein Klick mit der rechten Maustaste öffnet den Farben-Dialog, wo eine der 16 Millionen Farben eingestellt werden kann. Das reicht fürs Erste.

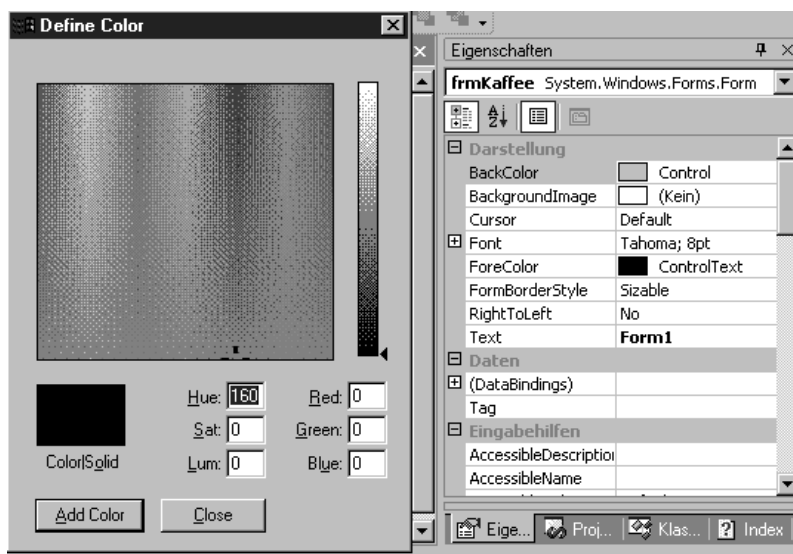
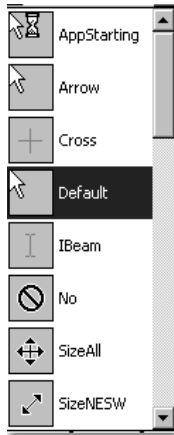


Abbildung 1.7 Die Farbe kann geändert werden.

Ein Tipp: Als ich begonnen habe zu programmieren, waren meine ersten Masken bunt. Manchmal habe ich sogar die Kunden gefragt, welche Farbe sie gerne möchten. Das fand ich sehr Spaßig. Inzwischen bin ich davon abgekommen. Der Grund ist einfach: Jeder Anwender kennt graue Bildschirmmasken. Auf einer bekannten Maske findet er sich leichter zurecht. Deshalb sind heute alle Masken, die ich für andere Menschen programmiere, grau. Wenn Sie möchten, dann können Sie natürlich einen bunten Kaffeeautomaten gestalten. Aber seien Sie gewarnt: Seriös sieht es nicht aus!

Mit »Cursor« könnte man den Mauszeiger verändern. Auch das sieht lustig aus, macht aber wenig Sinn. Warum sollte der Benutzer ein No-Symbol erhalten? Dann denkt er vielleicht, dass er etwas nicht tun darf. Im Ernst: Ich verwende den Mauszeiger, wenn Programme sehr lange laufen. Dann setze ich zu Beginn den

Mauszeiger auf `AppStarting` oder `WaitCursor` (die Sanduhr) und wenn das Programm fertig gearbeitet hat, dann wird der Cursor natürlich wieder auf `Default` gesetzt, allerdings per Programmierung. Wie man das macht, verrate ich noch.



**Abbildung 1.8** Die verschiedenen Cursor

»Font« ist interessant, nicht? Wo bitte ist denn eine Schrift auf dem Dialog? Die Antwort lautet: Alles, was später auf die Maske gesetzt wird, erhält diese Schrift als Standardschrift. Sie kann natürlich wieder geändert werden. Das Gleiche gilt natürlich auch für »ForeColor«.

Über »FormBorderStyle« kann eingestellt werden, ob der Benutzer den Rand verändern kann oder nicht, ob er seine Maske größer oder kleiner ziehen darf.

»RightToLeft« ist für Sprachen, die von rechts nach links geschrieben werden.

»Text« meint die Beschriftung in der Titelzeile. Diese ändere ich immer ab – beispielsweise in »Kaffeeautomat«.

Etwas weiter unten finden Sie den Namen des Formulars. Er ist wichtig, weil der Dialog über seinen Namen gesteuert wird. Noch einmal und ganz deutlich: Zuerst wird der Name festgelegt und dann wird er nicht mehr geändert! Denn möglicherweise taucht im Code der Name irgendwo auf. Eine Namensänderung wird weder erkannt noch umgesetzt und deshalb würden Fehler folgen.

Eine Hilfe zum Basteln stellen die `DrawGrid`, die Gitternetzpunkte dar. Ihre Entfernung kann eingestellt werden, ebenso wie die Option, ob neue Steuerelemente daran einrasten. Das ist eine prima Hilfe, wenn viele Steuerelemente auf dem Dialog sitzen.

Man könnte eine Hilfedatei an das Formular binden, das Icon links oben in der Titelzeile ändern, die beiden Symbole zum Vergrößern und Verkleinern wegblenden und nicht in der Taskbar anzeigen.

Weiter unten im Eigenschaftfenster wird die Größe festgelegt. Man könnte sie auch durch Ziehen an den drei weißen Eckanfassern rechts und unten vergrößern. »StartPosition« gibt an, wo die Maske gestartet wird, und »WindowState«, ob sie in der normalen Fenstergröße erscheint.

Dies sind die wichtigsten Eigenschaften – wenn auch nicht alle. Zunächst genügt dies.

So, unsere Maske ist nun schön, bunt und lustig, sie hat nur einen fundamentalen Fehler: Sie kann nichts! Das sollten wir ändern.

## 1.5 Die Toolbox

Am linken Rand findet sich ein weiteres Fenster: Die Toolbox. Sollte sie nicht sichtbar sein, so könnte man sie über das Hammer- und Schraubenschlüssel-Symbol herholen oder natürlich auch wieder über das Menü **Ansicht • Toolbox** **[Strg]** + **[Alt]** + **[X]**. Wer nicht genug Platz auf dem Bildschirm hat, der kann über das Pin-Nadel-Symbol steuern, dass die Toolbox nur in einer schmalen Leiste sichtbar ist. Fährt der Benutzer mit der Maus über diese Zeile, dann erscheint die Toolbox; befindet sich der Cursor an einer anderen Stelle, dann verschwindet sie wieder.

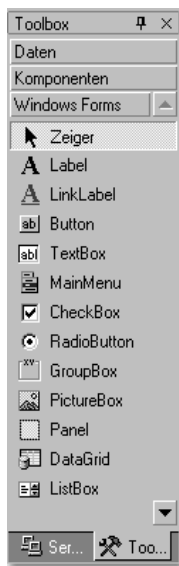


Abbildung 1.9 Die Toolbox

Da wir zuerst einfache Steuerelemente auf der Maske platzieren, benötigen wir das Register »Windows Forms«. Das vierte Steuerelement in dieser Liste ist der Button. Er wird angeklickt. Danach wird auf der Seite ein Rechteck aufgezogen und schon ist der erste Button da. Man könnte aber auch mit gedrückter Maustaste aus der Toolbox heraus einen Button auf das Formular ziehen.

Ich hätte gerne zwei Buttons, einen für »OK« und einen für »Abbrechen«. Normalerweise sitzen sie rechts unten.

Was haben wir gelernt? Zuerst wird der Name geändert. Den einen Button nenne ich »butOk«, den anderen »butAbbrechen«. Das »but« steht dabei für »Button« oder auf Deutsch für Befehlsschaltfläche. Manche Programmierer verwenden übrigens auch das alte Kürzel »cmd« (Commandbutton) oder »btn«. Aber das ist jedem selbst überlassen, der programmiert.

Damit der Benutzer weiß, was die Buttons tun, müssen sie beschriftet werden. Dafür ist die Eigenschaft »Text« zuständig. Der Text erscheint auf der Befehlsschaltfläche und erklärt die Funktion des Buttons.

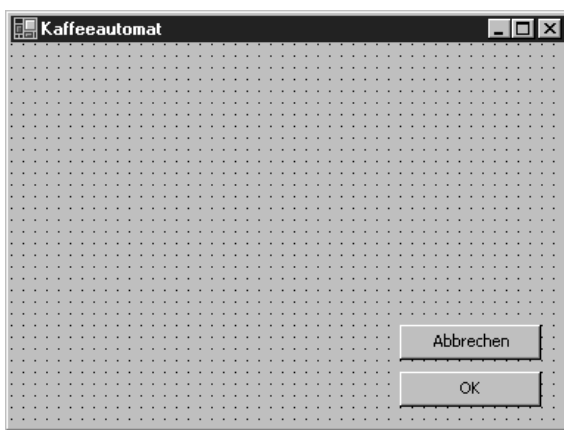


Abbildung 1.10 Das Formular mit zwei Buttons

## 1.6 Das Meldungsfenster

Der Benutzer soll das Formular über die Abbrechen-Schaltfläche schließen können. Mit einem Doppelklick auf den Button gelangen Sie in den Code oder genauer: Sie gelangen zwischen die beiden Zeilen

```
Private Sub butAbbrechen_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles butAbbrechen.Click
```

```
End Sub
```

Die lange erste Zeile steht nur für das auslösende Ereignis, wenn der Benutzer auf die Schaltfläche klickt. Das Objekt, welches den Befehl auslöst, heißt »butAbbrechen«. Das Ereignis »Click« steht für das Mausclicken des Benutzers, aber auch dafür, dass der Benutzer mit der Tabulator-Taste auf den Button »springen« und dann mit `Enter` den Code auslösen kann, der sich dahinter verbirgt.

In diesen Code muss der Befehl für »Schließe die Form«. Die Form trägt den Namen »frmKaffee«. Gibt man den Namens dieses Objekts ein und fügt danach einen Punkt an, dann erscheint eine kleine Auswahlliste von Dingen, die ich bisher noch nicht erläutert habe. Nach dem Punkt muss »ActiveForm« ausgewählt werden. Und diesem Objekt kann befohlen werden, dass es sich schließen soll (natürlich wieder mit einem Punkt). »Close« heißt dies in der Sprache von VB.NET. Die ganze Zeile lautet dann:

```
frmKaffee.ActiveForm.Close
```

Es geht ein klein wenig einfacher. Der erste Teil kann mit »Me« abgekürzt werden. Also:

```
Me.Close
```

Diesen Befehl kann man überall verwenden, ohne zu wissen, wie die Userform heißt. Ein Test beweist, dass man mit »Abbrechen« das Formular schließen kann.

Zurück zum Formular gelangt man über das Register. Das eine ist für die Entwurfsansicht zuständig, das heißt dort wird gebastelt. Im anderen steht der Code.

Auch die Ok-Schaltfläche soll eine Funktion haben. Klickt der Benutzer auf »Ok«, so soll er eine Meldung erhalten, dass es nun Kaffee gibt. Ein Doppelklick führt zu dem Code. Zwischen die beiden Zeilen

```
Private Sub butOk_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles butOk.Click
```

```
End Sub
```

wird der Befehl für das Meldungsfenster geschrieben. Er lautet:

```
MessageBox
```

### **1.6.1 Die Code-Eingabe für die MessageBox**

Wie geht es dann weiter? Die MessageBox ist ein Objekt, und nach Objekten steht immer ein Punkt, also:

```
MessageBox.
```

Schon erscheint eine der tollen Hilfen der Programmiersprache: Es wird angezeigt, wie es weitergeht, das heißt, es wird gezeigt, was die `MessageBox` kann – sie kann angezeigt werden:

```
MessageBox.Show
```

Ohne zu weit vorgehen zu wollen, sei Folgendes gesagt: `Show` ist eine Methode, das heißt, sie bewirkt etwas. Nach einer Methode steht immer eine Klammer. Wird also nach `MessageBox.Show` die öffnende Klammer eingegeben, dann erscheint eine weitere Hilfe: ein Quickinfo. Es verrät, dass zuerst ein »text« stehen muss. Mehr noch: »text« wird erklärt als »Der im Meldungsfenster anzuzeigende Text«. Und der wird – wie in vielen anderen Programmiersprachen auch – in Anführungszeichen gesetzt. Also beispielsweise:

```
MessageBox.Show("Achtung: Jetzt gibt es Kaffee!")
```

Das reicht aus, die Klammer wird geschlossen, nun speichern und starten. Tatsächlich: Jetzt gibt es Kaffee. Vielleicht hat es Sie gewundert, dass das Meldungsfenster zwölf Varianten zur Verfügung stellt und dass die erste nicht nur einen Eintrag, sondern sieben Parameter benötigt. Der Hintergrund ist folgender: Man kann dieser Funktion eine oder mehrere Varianten geben. Unsere Möglichkeit – »melde nur Text« – finden Sie unter der Nummer 6. Sind Sie also unsicher, ob Ihre Eingabe korrekt ist, so wechseln Sie mit den Pfeiltasten auf der Tastatur oder mit der Maus nach oben und unten. So schön es ist, dass uns verschiedene Varianten zur Verfügung stehen, so knauserig ist VB.NET: Es verlangt unbedingt eine der vorgeschlagenen Möglichkeiten.

Man kann die Meldung allerdings noch verfeinern. Wird die letzte Klammer gelöscht, oder noch besser: Wenn Sie vor die schließende Klammer klicken, dann können Sie mit dem Symbol »Zeigt die Parameterinformation ... an« das Quick-Info wieder herholen. Es geht noch weiter. Nach dem »text« folgt als Trennzeichen ein Komma und danach wird die »caption« eingegeben. Auch sie steht in Anführungszeichen, weil auch sie ein String, das heißt eine Zeichenkette, ist. Wie wäre es mit »Meine Kaffeemaschine«?

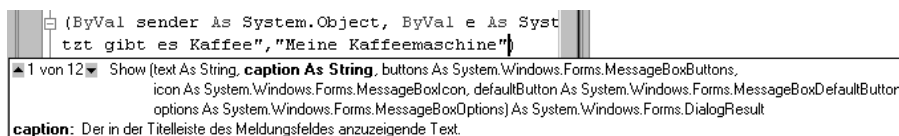
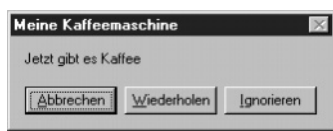


Abbildung 1.11 Das Meldungsfenster – die Caption

Nach dem zweiten Komma erscheint eine Liste der Möglichkeiten für die »buttons«. Alle beginnen mit »MessageBoxButtons«. Die sechs Möglichkeiten stehen für die sechs Varianten der Schaltflächen. Es bedeuten:

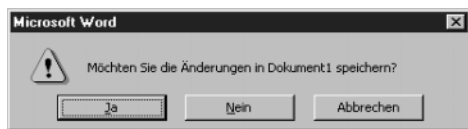
Bezeichnung	Bedeutung
AbortRetryIgnore	Abbrechen, Wiederholen, Ignorieren
OK	OK
OKCancel	OK, Abbrechen
RetryCancel	Wiederholen, Abbrechen
YesNo	Ja, Nein
YesNoCancel	Ja, Nein, Abbrechen

**Tabelle 1.1** Die verschiedenen Schaltflächen im Meldungsfenster



**Abbildung 1.12** Eine Variante des Meldungsfensters

Die anderen Schaltflächen kennen Sie von anderen Anwendungen:



**Abbildung 1.13** Ein Beispiel aus Word

Der nächste Parameter verlangt ein Symbol. Die Symbole beginnen mit »MessageBoxIcon.« und heißen:

Bezeichnung	Bedeutung	Symbol
Asterisk	Stern	
Error	Fehler	
Exclamation	Ausrufezeichen	
Hand	Hand	
Information	Information	

**Tabelle 1.2** Die verschiedenen Symbole im Meldungsfenster



Bezeichnung	Bedeutung	Symbol
None	Kein Symbol	
Question	Fragezeichen	
Stop	Stop	
Warning	Warnung	

**Tabelle 1.2** Die verschiedenen Symbole im Meldungsfenster (Forts.)

Achtung: Unter Windows funktionieren nur vier der Symbole: »Exclamation«, »Information«, »Question« und »Warning« – die übrigen Varianten zeigen eine dieser vier Möglichkeiten an.



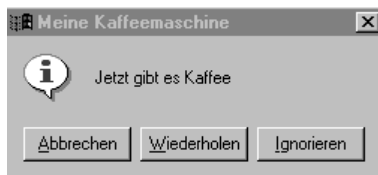
**Abbildung 1.14** Die vier möglichen Symbole

Soll der Fokus nicht auf der ersten Schaltfläche sitzen, sondern auf einer anderen, dann wird »MessageBoxDefaultButton« auf »Button1«, »Button2« oder »Button3« gesetzt. Damit kann der Benutzer leichter mit der `[Enter]`-Taste die wichtigste Taste aktivieren. Und schließlich könnte nach dem letzten Komma die Laufrichtung des Textes festgelegt werden, was für arabische oder hebräische PCs interessant ist.

Die ganze Zeile sieht dann folgendermaßen aus:

```
Private Sub cmdOK(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdOK.Click
    MessageBox.Show("Achtung: Jetzt gibt es Kaffee!",
    "Meine Kaffeemaschine", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Information, MessageBoxDefaultButton.Buttons2,
    MessageBoxOptions.DefaultDesktopOnly)
End Sub
```

Das Ergebnis sieht dann folgendermaßen aus:









**Abbildung 1.15** Das Meldungsfenster


### 1.6.2 Ein Wort zum »alten« Visual Basic

Noch immer werden die Visual Basic-Befehle unterstützt. Das heißt, man könnte statt »MessageBox« den »alten« Befehl »MsgBox« verwenden, dessen Syntax zwar ähnlich, aber dennoch etwas anders ist. In diesem Buch wird darauf verzichtet, die alten Befehle zu erklären. Wer sie kennt und unbedingt verwenden möchte – bitte schön. Die VB.NET-Befehle haben den großen Vorteil, dass sie durchgängig gleich strukturiert sind, das hilft enorm beim Erlernen.


### 1.6.3 Die Code-Eingabe allgemein

Ein paar theoretische Bemerkungen zur Code-Eingabe:

Texteingabe und -korrektur im Codefenster erfolgen nach den gleichen Regeln wie in der Textverarbeitung: Zu korrigierende Zeichen werden mit der Korrekturtaste  oder der Taste  gelöscht, Text kann an beliebigen Stellen eingefügt werden. Mit der  kann in den Überschreibmodus gewechselt werden: Der Cursor erscheint dann als Rechteck und löscht beim Schreiben die früheren Zeichen. Um Text zu markieren, können Sie die Maus verwenden und mit gedrückter linker Maustaste über den Text ziehen oder mit gedrückter -Taste + Pfeiltasten den Cursor über den Text bewegen. Der gesamte Code, das heißt das ganze Modul, kann auch mit  +  oder dem Menü **Bearbeiten · Alles Auswählen** markiert werden. Zum Kopieren und Ausschneiden kann das Menü **Bearbeiten · Kopieren** und **Bearbeiten · Einfügen** verwendet werden, respektive: **Bearbeiten · Ausschneiden** und **Bearbeiten · Einfügen**. Kopierte Textteile landen in der Toolbox im »Zwischenablagering«. Von dort können mit Hilfe der rechten Maustaste also auch ältere Texte, die sich noch im Zwischenspeicher befinden, herausgeholt werden.

Textbausteine gibt es nicht, dafür aber die zweite Möglichkeit, Code-Schnipsel im Register »Allgemein« in der Toolbox abzulegen. Texte im Code können markiert und in die Toolbox gezogen werden, oder man drückt beim Herausziehen die -Taste – dann werden sie hineinkopiert. In der Toolbox wird nun der Beginn angezeigt. Die Texte können jederzeit wieder herausgezogen werden.

Werden Textteile kopiert, dann erscheinen sie im »Zwischenablagering«. Auch dort können sie jederzeit mit gedrückter Maustaste herausgezogen werden.

Man kann markierte Textteile innerhalb des Codes mit der Maus verschieben – mit der Drag&Drop-Funktion. Ebenso können Textteile mit der -Taste durch Ziehen kopiert werden.

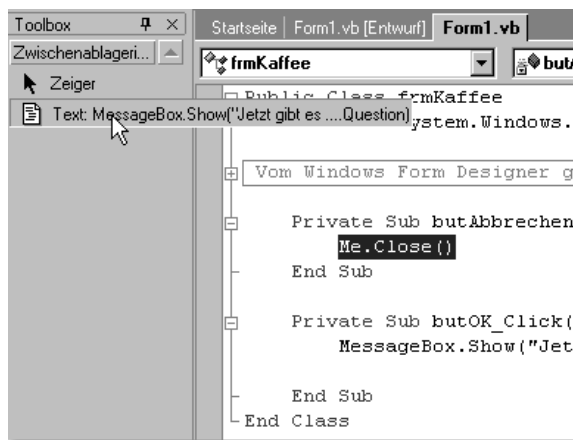


Abbildung 1.16 Eine der vielen angenehmen Hilfen in VB.NET

Wenn sich der Cursor am Anfang einer Zeile befindet und dort `[Enter]` gedrückt wird, dann wird eine neue Codezeile oberhalb eingefügt. Die vier Pfeiltasten bewegen den Cursor Zeile für Zeile oder Zeichen für Zeichen nach oben, unten, links oder rechts. Die Tasten `[Bild ↑]` beziehungsweise `[Bild ↓]` blättern eine Bildschirmseite nach oben oder unten. Mit `[Strg] + [↑]` beziehungsweise `[Strg] + [↓]` blättern Sie in die nächste oder vorhergehende Prozedur. Der Einzug wird automatisch vorgenommen, was sehr hilfreich ist.

Das alles dürfte aus der Textverarbeitung bekannt sein.

### Lange Textzeilen

Normalerweise entspricht eine Textzeile einem Befehl. Nach dem Drücken von `[Enter]` wird die Eingabe beendet und der eingegebene Text wird auf seine Richtigkeit überprüft. Problemlos können die Anweisungszeilen durch Leerzeilen voneinander getrennt werden – leere Zeilen werden übergangen und haben keinerlei Funktion, außer dass sie beim Lesen helfen.

Es kann nun passieren, dass ich eine sehr lange Befehlszeile eingeben muss, zum Beispiel:

```
MessageBox.Show("Möchten Sie, dass Ihr Computer zerstört
wird?", " Odyssee 2000", MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly)
```

Ein automatischer Umbruch, wie von der Textverarbeitung bekannt, findet erst nach 1.024 Zeichen statt: Der Text fließt bis dahin immer weiter nach rechts. Um

einen manuellen Umbruch zu organisieren, kann der Text in mehrere Zeilen geteilt werden. Diese werden durch eine Leerstelle (sie ist unbedingt nötig!), der ein Unterstrich am Ende der Zeile folgt, getrennt.

```
MessageBox.Show _  
("Ihr Computer wird nun zerstört", _  
"Odyssee 2000", MessageBoxButtons.YesNoCancel, _  
MessageBoxIcon.Warning, _  
MessageBoxDefaultButton.Button1, _  
MessageBoxOptions.DefaultDesktopOnly)
```

Die Option könnte über **Extras · Optionen** ausgeschaltet werden. Dort finden Sie im Ordner **Text-Editor · Basic** die Einstellung **Zeilenumbruch**. Mit ihrer Hilfe könnte man mehrzeilig schreiben, was aber sicherlich nur wenige Basic-Programmierer tun.

Umgekehrt können mehrere logische Programmierzeilen in eine Textzeile geschrieben werden. Dann werden sie mit einem Doppelpunkt (»:«) getrennt, wie beispielsweise folgende drei Befehlszeilen:

```
MessageBox.Show("Achtung!") : Beep() : Beep()
```

Hiervon ist in der Regel abzuraten, weil der Code sehr schnell unübersichtlich wird.

### **Kommentare und optische Gliederungen**

Häufig wünscht der Programmierer seine Programme zu kommentieren; sei es, um sie besser zu strukturieren, sei es, um anderen Programmierern, die damit weiterarbeiten, schnell einen Überblick zu verschaffen. Manchmal will ein Kunde nach Monaten, dass ich an dem alten Programm etwas ändere. Woher soll ich aber wissen, was ich vor Monaten gemacht habe? In so einem Fall sehe ich in den Kommentaren nach. Sollte jemand mal mit meinen Programmen weiterarbeiten, dann könnte er an den Kommentaren erkennen, was ich gemacht und mir dabei gedacht habe. Zwar gebe ich normalerweise meine Programme nicht her, aber man könnte es theoretisch tun. Einmal hatte ich sogar den Fall, dass eine Firma behauptete, ich hätte etwas falsch programmiert. Da konnte ich ihnen zeigen, dass sie nicht Recht hatten, denn im Kommentar stand, was wir damals ausgemacht hatten.

Kommentare können überall eingefügt werden: Sie werden mit einem Apostroph »'« eingeleitet, der innerhalb einer Zeile stehen kann:

```
MessageBox.Show ("Habe nun ach..." 'Dies ist von Goethe!
```

oder am Anfang einer Zeile:

```
' Nun folgt ein Meldungsfenster:  
MessageBox.Show _  
    ("Philosophie, Juristerei und Medizin")  
' Dies war das Meldungsfenster
```

Mit dem Anführungszeichen (Apostroph) meine ich den Haken über dem Zahlenzeichen, neben dem Ä, nicht den Akzent über dem Ü. Der ist für meinen Vornamen reserviert.

Kommentare können ebenso am Beginn einer Zeile durch ein »rem« (remark) eingeleitet werden:

```
Rem Nun folgt ein Meldungsfenster:  
MessageBox.Show("und leider auch Theologie")  
Rem Dies war das Meldungsfenster
```

Dieses »rem« wirkt etwas altmodisch und erfordert mehr Tipparbeit als Apostrophe. Sonst gibt es keinen Grund, auf »rem« zu verzichten. Ein kleiner Unterschied besteht zwischen »rem« und dem Apostroph: Das Apostroph kann am Ende einer Zeile stehen und diese kommentieren, »rem« nicht!

Während des Programmierens kann mit »rem« oder Apostroph eine Zeile ausgeschaltet werden – besonders, wenn sie später noch verwendet wird.

Eine praktische Hilfe finden Sie in der Symbol-Leiste »Text-Editor«. Dort gibt es zwei Symbole »Die ausgewählten Textzeilen auskommentieren« und »Hebt den Kommentar der ausgewählten Textzeilen auf.« Dies ist praktisch, wenn man ein Programm testen möchte, aber will, dass einige bestimmte Zeilen nicht ausgeführt werden. Diese kann man dann markieren und auskommentieren, also ausschalten.



Abbildung 117 Die beiden Symbole zum Kommentieren

Kommentare erscheinen in grüner Schrift, was Sie im Menü **Extras • Optionen** im Ordner **Umgebung • Schriftarten und Farben** unter **Elementvordergrund Kommentartext** ändern können. Wenn Sie es unbedingt verändern müssen, dann sollten Sie nicht Schwarz nehmen, denn die Kommentare sollen auffallen. Kommentare werden nicht abgearbeitet, sondern dienen zur Erläuterung, sodass der Programmierer bestimmte Codestellen entweder schnell findet oder Erläuterungen zum Code hat.

Auch Rot ist eine schlechte Farbe für Kommentarzeilen, denn in Rot erscheinen Fehler. Und schließlich gibt es noch Farben, die das Auge nicht so gerne mag ... Vielleicht bleiben wir doch bei Grün.

Ein beliebtes Gestaltungsmittel, das Ordnung schafft, sind Kommentare in Kästen mit »\*«, »-« oder »=«. Etwa so:

```
! * * * * *
! *
! *      Und nun für unsere Goethe-Freunde      *
! *
! * * * * *
```

```
MessageBox.Show _
    ("durchaus studiert mit heißem Bemühn.")
```

```
! * * * * *
! *
! *      Heute: Faust I                          *
! *
! * * * * *
```

Zugegeben: Solch ein gestalterisches Werk erfordert (Tipp-)Arbeit, strukturiert allerdings das Programm sehr gut.

### Einrücken von Zeilen

Eine weitere Gliederungsmöglichkeit ist das Einrücken von Zeilen, das Basic für uns vornimmt. Dadurch werden Ebenen gekennzeichnet, was bei Verschachtelungen hilfreich ist, und die Übersichtlichkeit erhöht. Ich halte diese Einstellung für wichtig, da sie die zusammengehörigen Blöcke deutlich macht. Man könnte sie über **Extras • Optionen** (Kategorie **Text-Editor • Basic • Tabstopps**) deaktivieren.

### Gliederungen

Bei allen Gliederungen, die VB.NET uns zur Verfügung stellt, kann der Code selbst in Teile untergliedert werden. Dazu kann er mit der Maus markiert werden. Nun befindet sich im Kontextmenü die Option »Gliederung«. Wird sie ausgewählt, so kann dieser Teil über die drei Pünktchen am linken Rand ein- und ausgeschaltet werden. Gerade bei längeren Projekten ist dies eine sinnvolle Hilfe.

Nochmals meine Bitte: Auch wenn ich hier im Buch nur relativ wenig kommentiere (sonst wird der Code zu lang), bitte kommentieren Sie immer. Und alles!

## 1.7 Ein Textfeld

Nun soll der Benutzer seinen Namen in ein Textfeld eintragen dürfen und müssen. Aus der Werkzeugsammlung wird eine TextBox ausgewählt und auf dem Formular aufgezogen. Damit der Benutzer weiß, was er eingeben muss, wird ein Label ausgewählt und darunter platziert. Das Textfeld erhält den Namen »txtName«, der Text ist leer, das heißt, der Inhalt der Eigenschaft »Text« wird in der Eigenschaftsliste gelöscht. Das Bezeichnungsfeld heißt »lblName«, sein Text lautet »Name«. Die Eigenschaft »Text« bedeutet beim Textfeld etwas anderes als beim Label oder beim Button. Bei den letzten beiden hat der Text nur dekorative Funktion, während beim Textfeld mit »Text« jener Text gemeint ist, den der Benutzer eingibt.

Der Benutzer tippt also seinen Namen ein und soll nach einem Klick auf die Schaltfläche namentlich einen Kaffee erhalten. Also wird verknüpft:

```
MessageBox.Show("Achtung: " & txtName.Text & _  
"Jetzt gibt es Kaffee!")
```

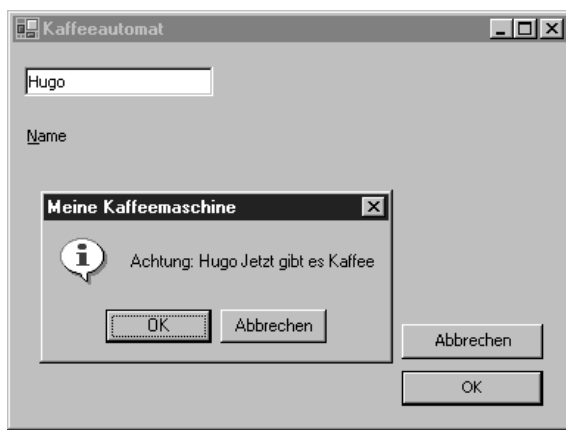


Abbildung 1.18 Der Dialog

Warum darf hier nicht einfach nur »txtName« stehen? Die Antwort ist einfach: »txtName« ist nur das Objekt, also das Textfeld selbst. Welche Eigenschaft ausgelesen wird, muss noch zusätzlich hinzugefügt werden. Denn man könnte ja auch die Eigenschaft Schriftgröße oder Hintergrundfarbe abfragen.

## 1.8 Texte verketteten

Das »&« ist von Excel bekannt? Nein? Es wird Verknüpfungsoperator genannt. Damit werden Zeichenketten verkettet. Das bedeutet: Mit einem Pluszeichen werden Zahlen verkettet (3 + 4 = 7), mit dem kaufmännischen Und-Zeichen (»&«)

werden Texte verkettet (»drei« & »vier« = »dreivier«). Das sieht nicht nur lustig aus – es wird auch häufig verwendet, etwa wenn ein Speicherort und ein Dateiname miteinander verkettet werden müssen, damit man auf eine bestimmte Datei zugreifen kann.

Manche Programmierer verwenden auch das Pluszeichen. Das klappt auch in VB.NET. Ich persönlich halte die beiden Dinge lieber auseinander, also »+« für Addition von Zahlen, »&« für Verkettung von Texten. Dafür gibt es ein tolles Fremdwort: Konkatenation. Es heißt auf Deutsch Verkettung.

## 1.9 Andere Zeichen

Was ist aber nun, wenn ich den Namen in der ersten Zeile, das Heißgetränk in der zweiten Zeile haben möchte? Wenn ich also einen Zeilenumbruch zwischen die beiden Zeilen bekommen möchte? Einfach `Enter` drücken geht nicht, da sonst eine neue Zeile eingefügt würde. Also muss das Zeichen für `Enter` eingegeben werden. In der letzten Visual Basic-Version hieß es »vbCr«. »Cr« steht dabei für »carriage Return«, also Wagenrücklauf. Das funktioniert immer noch. Man kann also schreiben:

```
MessageBox.Show("Achtung: " & txtName.Text & _  
    vbCR & "Jetzt gibt es Kaffee!")
```

Es macht in einem Meldungsfenster keinen Unterschied – man könnte auch »CrLf« eingeben. »Lf« steht für »Linefeed«. Oder beide zusammen: »vbCrLf«. Ins neue VB.NET übersetzt heißt dieser Befehl

```
Environment.NewLine
```

Oder ganz exakt:

```
System.Environment.NewLine
```

Er ist natürlich etwas umständlich zu tippen, hat aber den großen Vorteil, dass er in jedem System funktioniert, und das sollte doch das Ziel eines VB-Projekts sein. Übrigens: Wenn Sie die ASCII-Tabelle im Kopf haben, dann können Sie auch die Funktion »Chr« verwenden. `Enter` hat den Wert 34, »Lf« hat 10, man kann also auch schreiben:

```
Chr(13)
```

oder

```
Chr(10)
```



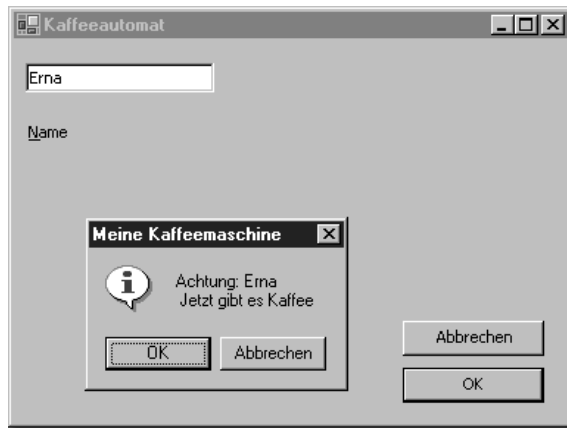


Abbildung 1.19 Der Umbruch funktioniert.

### 1.9.1 Noch ein Wort zum Code-Tippen

Wenn Sie die Beispiele abtippen (oder andere Beispiele eingeben), dann müssen Sie ein paar Kleinigkeiten beachten.

Groß- und Kleinschreibung sind in VB.NET austauschbar. Es ist also gleichgültig, ob Sie `messagebox`, `MESSAGEBOX` oder `MessageBox` schreiben. Ich erwähne das deshalb, weil es nicht nur meine eigene Faulheit beschreibt. Es hat noch eine andere wichtige Funktion. Wenn ich Code eingabe, dann tippe ich die bekannten Befehle alle in Kleinbuchstaben. Denn sobald ich die Zeile verlasse, werden die Syntax und die einzelnen Befehle überprüft. Ein Befehl wie »`messagebox`« wird als korrekt erkannt und in die richtige Schreibweise gedreht. »`messageboxo`« dagegen führt beim Verlassen der Zeile nicht nur dazu, dass der Befehl mit einer Schlangenlinie versehen wird, es werden auch nicht – anders als erwartet – die Kleinbuchstaben in Groß- und Kleinbuchstaben verwandelt. Ich persönlich finde das eine prima Hilfe, mit der ich schon viele Tippfehler schnell gefunden habe.

Leerzeichen sind ein kleines Problem. Merkregel: Lieber eins zu viel als eins zu wenig. Sind zu viele Leerzeichen in einer Zeile, dann löscht das Programm die überflüssigen beim Verlassen der Zeile. Also besser:

```
messagebox.show ( "Achtung: " & txtName.Text &
"Jetzt gibt es Kaffee!" )
```

als folgende Zeile:

```
messagebox.show("Achtung: "&txtName.Text&"Jetzt gibt es
Kaffee!")
```

Denn diese führt zu einem Fehler, weil VB.NET »denkt«, dass

```
txtName.Text&"Jetzt gibt es Kaffee!"
```

ein Ausdruck ist. Das Programm »glaubt«, dass das kaufmännische »und« Teil des Inhalts des Textfelds ist. Auf den ersten Blick sieht dies ziemlich doof aus, es steckt aber Methode dahinter.

### 1.9.2 Ein Wort zur Syntax

Wörter mit einem »x« und einem »y« sind klasse. Sie sind meistens griechisch, kein Mensch kennt sie und man kann sie überall verwenden, das macht viel her! Nein, im Ernst: Der Begriff »Syntax« ist bekannt. Er bedeutet nichts anderes als die Reihenfolge der Elemente. In einem Satz dürfen Sie sagen:

»Es war eine lange, stürmische Nacht.«

oder

»Eine lange, stürmische Nacht war es.«

Allerdings ist folgende Reihenfolge nicht möglich:

»lange Nacht, stürmische es war.«

Zwar kann man den Satz mit ein wenig Mühe noch verstehen, aber es ist anstrengend. Deshalb reden und schreiben wir lieber »korrektes« Deutsch und programmieren »korrektes« VB.NET.

In einer Programmiersprache bedeutet »Syntax« die Abfolge von bestimmten Elementen. Das heißt:

Nach der Funktion `MessageBox.Show` muss eine Klammer stehen. Zuerst »(« und am Ende »)«. Dazwischen steht der »text«, der gemeldet wird. Er ist notwendig, ohne ihn geht es nicht. Optional, das heißt möglich, aber ohne Zwang, können die übrigen Parameter stehen. Und wenn, dann bitte schön in dieser Reihenfolge. Soll keine »caption« angegeben werden, dann könnte man dies folgendermaßen schreiben:

```
MessageBox.Show _  
("Achtung: Jetzt gibt es Kaffee!", ,  
MessageBoxButtons.YesNoCancel, _  
MessageBoxIcon.Warning,)
```

Der Parameter fehlt einfach. Die Reihenfolge ist festgelegt. Nun gibt es allerdings Funktionen mit sehr vielen Parametern. Damit man nicht umständlich viele Kommata hintereinander schreiben muss, gibt es noch eine zweite Schreibweise dafür:

```

MessageBox.Show(text:="Jetzt gibt es Kaffee!", _
caption:="Meine Kaffeemaschine", _
buttons:=MessageBoxButtons.YesNoCancel, _
Icon:=MessageBoxIcon.Information, _
defaultbutton:=MessageBoxDefaultButton.Buttons2, _
options:=MessageBoxOptions.DefaultDesktopOnly)

```

Das ist zwar viel mehr Tipparbeit, aber deutlich angenehmer zu lesen.

Apropos angenehmer zu lesen: In VB.NET ist eine Zeile ein Befehl oder ein Befehl eine Zeile. Da die Codezeilen hier im Buch nicht immer in eine Zeile passen, habe ich sie häufig umgebrochen. Dies können Sie auch tun, wenn Sie die Beispiele aus dem Buch nachtippen, Sie können die Befehle aber auch in einer Zeile schreiben.

## 1.10 Variablen

Manchmal kann es passieren, dass das Programm sich den eingegebenen Namen »merken« muss. Sei es, weil er an mehreren Stellen verarbeitet wird, sei es, weil er erst später verarbeitet wird. Deshalb gibt es in jeder Programmiersprache Variablen. Der Wert

```
txtName.Text
```

soll vorübergehend mit Hilfe einer Variablen gespeichert werden. Damit bin ich wieder bei meinem Begriff von »Sauberkeit«. Eine Variable sollte sauber deklariert werden. Das passiert am Anfang der Prozedur, also in unserem Fall nach:


```
Private Sub cmdOk(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdOk.Click
```

Die Syntax lautet:

```
Dim Variablenname as Variablentyp
```

beispielsweise:

```
Dim Benutzer As String
```

Es genügt übrigens auch hier, nach dem Schlüsselwort `As` die ersten drei Buchstaben »str« einzutippen. Dann springt die Markierung auf den korrekten Variablentyp. Mit  kann das Ergebnis bestätigt werden.

Für den Namen der Variablen gibt es bestimmte Regeln. Der Name darf bis zu 255 Zeichen lang sein und alle Zeichen enthalten außer Leer-, Satz- und Sonderzeichen: `.,;:-!>$$%&/(){<>}\^`. Auch der Bindestrich »-« darf nicht verwendet werden. Dagegen sind der Unterstrich (»\_«), Groß- und Kleinschreibung (VB.NET

macht keine Unterschiede) sowie die zehn Ziffern erlaubt, das erste Zeichen darf allerdings keine Ziffer sein. Der Name der Variablen muss selbstverständlich eindeutig sein (es dürfen nicht zwei Variablen gleich heißen) und darf nicht mit einem Schlüsselwort übereinstimmen. Schlüsselwörter sind:

- ▶ alle Befehle und Anweisungen: GoTo, Dim, As, If, While ...
- ▶ alle Funktionen: Sin, Cos, Date, Year ...
- ▶ alle logischen Operatoren: And, Or, Not ...
- ▶ andere Statements: AddressOf, As, Declare, Is, Let ...

Folgender Name ist gültig:

`x`, `Hugo`, `Mein_erster_Platzhalter`, `R2D2` oder `H2SO4`

Auch die Namen der Variablen sollten ein Präfix besitzen, an dem ihr Typ erkennbar ist. Diese Konvention, die von der Firma Gregory Reddick & Associates, einer Unternehmensberatung von Microsoft, herausgegeben wurde, ist nicht verbindlich. Allerdings arbeiten sehr viele Programmierer damit (ich zum Beispiel). Die Konvention stellt eine Möglichkeit der Standardisierung für VB.NET-Programmierung dar. Sie müssen sich natürlich nicht daran halten, sollten sich aber, wenn Sie zu mehreren an einem Projekt programmieren, über die Benennung der Variablen einig sein.

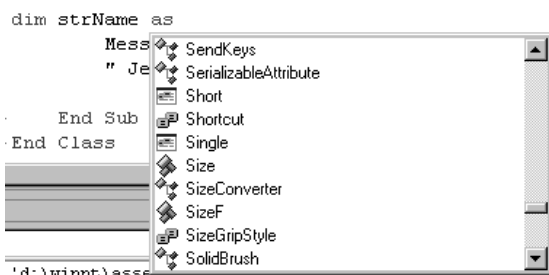
Hier die Liste der Variablentypen:

Datentyp	Variablentyp	Erklärung
Text	String	Zeichenkette, also beliebig viele Buchstaben und Ziffern
	Char	nur ein Zeichen (Unicode 16-Bit-Zeichen)
Zahlen ohne Kommastellen	Byte	ganze Zahl zwischen 0 und 255
	Short	ganze Zahl zwischen -32.768 und 32.767
	Integer	ganze Zahl zwischen -2.147.483.648 und 2.147.483.647
	Long	ganze Zahl zwischen -9.223.372.036.854.775.808 und 9.223.372.036.854.775.807
Zahlen mit Kommastellen	Single	einfach genaue Fließkommazahl
	Double	doppelt genaue Fließkommazahl
	Decimal	Fließkommazahl (96-Bit-Zahl)

**Tabelle 1.3** Liste der Variablentypen in VB.NET

Datentyp	Variablentyp	Erklärung
Wahrheitswert	Boolean	»Wahr« oder »Falsch«
Datumsangaben	Date	Datum
	DateTime	Datum mit Uhrzeitangabe
Objekte	Object	Objekte, wie andere Programme, Tabellen, Dateien ...

**Tabelle 1.3** Liste der Variablentypen in VB.NET (Forts.)



**Abbildung 1.20** Eine Variable wird deklariert.

Wer sich die Liste genau ansieht und mit der Auswahlliste in VB.NET vergleicht, stellt fest, dass ich nicht alle Typen mit diesem blauen Kästchen-Symbol erklärt habe. Das ist korrekt. Einige Typen (Int16, Int32, Int64, UInt16 ...) sind interne Datentypen und stehen zum Austausch mit C# zur Verfügung, sie sind also für uns uninteressant. Eine ganze Reihe von Elementen dieser Liste hat andere Symbole. Über einige davon werden wir noch sprechen. Doch zuerst möchte ich, dass die Datentypen jedem klar sind. Ich nenne ein Element und Sie überlegen sich, wie man es deklarieren würde.

Nehmen wir ein Beispiel. Schuhgröße?

Byte.

Richtig. Lebensalter?

Byte.

Richtig. Postleitzahlen?

Short.

Falsch. Short läuft bis 32.767. Damit könnten wir keine mittel- und süddeutschen Postleitzahlen (südlich von Detmold) eingeben. Schade um die Bayern.

Okay, Integer.

Auch das ist ungünstig: die Sachsen machen ein Problem. Leipzig Innenstadt hat beispielsweise die Postleitzahl 04105. Ich würde String verwenden. In Österreich oder der Schweiz sieht es in puncto Postleitzahlen anders aus: Dort sind Postleitzahlen vierstellig und haben keine führende Null. Bankleitzahl?

Bankleitzahlen haben maximal acht Stellen. Meine beginnt mit einer »7«. Short ist zu kurz, Integer müsste passen.

Korrekt. Kontonummer?

Uh, die können lang sein: Double?

Richtig. Kontonummern können bis zu zehn Ziffern lang sein. Also wäre Integer zu kurz. Mit Long kommt man auch aus. Aber sicher ist sicher. Double ist besser.

Telefonnummern?

Auch Double?

Nein. Denken Sie mal an die führende Null bei der Vorwahl, an den Schrägstrich oder das Leerzeichen. Das muss auch als String eingegeben werden; und Hausnummern ebenfalls. Ich wohne in der Magnus-Hirschfeld-Straße 7b.

Wenn Sie mit Präfixen arbeiten, dann stehen Ihnen folgende Präfixe zur Verfügung:

Variablentyp	Kürzel	Beispiel	Speicherplatz
String	str	strName	
Char	char	charZeichen	nur ein Zeichen (Unicode 16-Bit-Zeichen)
Byte	byt	bytAlter	ganze Zahl zwischen 0 und 255
Short			ganze Zahl zwischen -32.768 und 32.767
Integer	int	intArtikelnummer	ganze Zahl zwischen -2.147.483.648 und 2.147.483.647
Long	lng	lngKontonummer	ganze Zahl zwischen -9.223.372.036.854.775.808 und 9.223.372.036.854.775.807
Single	sng	sngGewicht	einfach genaue Fließkommazahl
Double	dbl	dblDistanz	doppelt genaue Fließkommazahl
Decimal	dec	decWarp	Fließkommazahl (96-Bit-Zahl)

**Tabelle 1.4** Die Liste der Variablentypen und ihrer Präfixe

Variablentyp	Kürzel	Beispiel	Speicherplatz
Boolean	f	FSchalter	»Wahr« oder »Falsch«
Date	dat	DatDatum	Datum
DateTime	tim	TimUhrzeit	Datum mit Uhrzeitangabe

**Tabelle 1.4** Die Liste der Variablentypen und ihrer Präfixe (Forts.)

Mehrere Variablen können gleichzeitig deklariert werden. Entweder vom gleichen Typ:

```
Dim intZahl1, intZahl2, intZahl3 As Integer
```

oder als unterschiedliche Typen:

```
Dim x As Integer, decGewicht As Decimal, _
    strName As String
```

Ein Wort zu Char: Eine Variable vom Typ String wird folgendermaßen deklariert:

```
Dim strText As String = "Harry Potter"
```

Wenn Sie diesen Typ verwenden, dann dürfen Sie nicht wie bei String analog schreiben:

```
Dim charZeichen as Char = "Y"
```

sondern müssen es mit dem Litteral »c« kennzeichnen:

```
Dim charZeichen as Char = "Y"c
```

Aber vielleicht können wir auch ganz auf diesen Datentyp verzichten und verwenden lediglich String.

Übrigens könnte man auf die Variablendeklaration sogar verzichten (dies wird implizite Variablendeklaration genannt). Man könnte unter **Projekt · Eigenschaften** einstellen, dass keine Variablendeklaration erforderlich ist (Kategorie »Erstellen«). Dort könnte man die Option Explicit auf »Off« setzen. Dann wird die Variable einfach im Code verwendet. Dies hat allerdings einige Nachteile:

Wird eine Variable deklariert und wird sie im Quellcode falsch geschrieben, dann wird dies gekennzeichnet. Wird eine Variable nicht deklariert, so bekommt sie den Datentyp Object zugewiesen, der sehr viel mehr Speicherplatz benötigt. Insgesamt wird das Programm langsamer und fehleranfälliger. Das heißt: Das Risiko, dass das Programm abstürzt, ist größer. Also, Variablen immer deklarieren! Es hat nur Vorteile.

```

Private Sub butOK_Click(ByVal sender As Sys
    Dim strName As String
    strName = txtName.Text
    Message[Der Name 'strName' wurde nicht deklariert.] txtName
    " Jetzt gibt es Kaffee", "Meine Kaffeeen

End Sub
. Class

```

Abbildung 1.21 Eine Variable wurde falsch geschrieben.

Vielleicht stellen Sie sich die ganze Zeit die Frage, warum wir dieses seltsame Spiel treiben. Die Antwort wurde oben schon gegeben. Gerechnet werden kann nur mit einer Zahl. Deshalb muss ich vorher wissen, ob die Eingabe als Berechnung weiterverarbeitet wird oder nicht. Nun könnte man argumentieren, dass dann zwei Typen genügen. Es gibt aber noch einen zweiten Grund für eine saubere Variablendeklaration: Speicherplatz. Je kleiner die Variablen deklariert sind, umso weniger Speicher benötigen sie. Wie viel sie brauchen, steht oben in der Liste. Zwar ist es bei zwei oder drei Variablen egal, ob sie vom Typ »Byte« oder »Double« sind, dafür sind unsere Rechner schon schnell genug. Aber bei vielen Hundert Variablen macht sich der Speicherplatz (oder nicht mehr vorhandene Arbeitsspeicher) in der Geschwindigkeit bemerkbar.

Zwar könnte man nun argumentieren, dass die Sache mit den Typen in VB.NET gar nicht so wichtig ist, weil das Programm die Typen automatisch konvertiert. Das sollten Sie jedoch vermeiden, da der Code instabiler wird. Zwar funktioniert:

```

Dim i As Integer = 7
MessageBox.Show("Die Zahl lautet: " & i)

```

besser ist aber folgende Lösung:

```

Dim i As Integer = 7
MessageBox.Show("Die Zahl lautet: " & _
    i.ToString)

```

Man kann sich den »Fehler« anzeigen lassen, indem in **Projekt • Eigenschaften** im Ordner **Erstellen • Option Strict** eingeschaltet wird.

Also: Variablen sollten korrekt, so klein wie möglich und so groß wie nötig deklariert werden.



## 1.11 Ein theoretischer Exkurs: Werte- und Referenztypen

Die Typen in VB.NET werden in zwei Kategorien unterteilt: Wertetypen und Referenztypen. Wertetypen benötigen weniger Speicherplatz und sorgen für einen schnelleren Zugriff – dafür fehlen ihnen einige Eigenschaften, wie beispielsweise die Vererbung. Diese ist in den Referenztypen vorhanden. Allerdings benötigen sie mehr Speicherplatz. Wenn eine Variable einen Referenztyp beinhaltet, dann wird Bezug auf Daten genommen. Das heißt, die Variable fungiert als Zeiger (Pointer) auf diese Daten und wird dem Stack zugewiesen, während die Daten an anderer Stelle (im Heap) verwaltet werden. Alle Referenztypen sind vom Typ Objekt abgeleitet. Werden Wertetypen in Referenztypen umgewandelt, dann sind sie auch vom Objekttyp abgeleitet. Wird ein Wertetyp wie ein Referenztyp behandelt, so spricht man von Boxing (das Gegenteil heißt Unboxing). Es gibt einen Unterschied in der erstmaligen Verwendung von Werte- und Referenztypen, auf den wir weiter unten noch eingehen werden. Sonst ist der Unterschied nicht bedeutend.

## 1.12 Zuweisen von Werten an Variablen

Wir deklarieren:

```
Dim strName as String
```

und füllen dann die Variable

```
strName = "Bilbo"
```

Der Befehl »Dim« stammt übrigens aus der Urzeit von BASIC und steht für »Dimensions«.

Wichtig ist, dass die Übergabe des Wertes immer von rechts (das heißt rechts vom Gleichheitszeichen) nach links geschieht. Das Gleichheitszeichen wird bei Vergleichen verwendet:

```
If strName= "Bilbo" Then
```

und auch bei Zuweisungen:

```
strName= "Bilbo"
```

Man kann dies auch schon bei der Deklaration erledigen:

```
Dim strName as String = "Bilbo"
```

Ist eine Variable einmal von einem Typ festgelegt, dann kann der Typ nicht mehr zur Laufzeit geändert werden, wohl aber der Inhalt. Das heißt, die Variable kann nacheinander verschiedene Werte annehmen:

```
strName = "Bilbo"  
strName = "Frodo"  
strName = "Gollum"
```

Nun enthält die Variable `strName` den Wert »Gollum«. Natürlich macht dieses Beispiel keinen Sinn, denn normalerweise würde erst die Variable weiterverarbeitet, bevor der Inhalt ganz geändert wird.

Man kann den Inhalt aber auch nur zum Teil ändern. Will man an den Vornamen einen Zunamen hängen, so geschieht das Verketteten rechts vom Gleichheitszeichen, die Zuweisung wieder nach links:

```
strName = "Bilbo"  
strName = strName & " Beutlin"
```

Jetzt ist die Variable `strName` mit dem Wert »Bilbo Beutlin« gefüllt.

Man kann es auch anders schreiben:

```
strName &= " Beutlin"
```

Wem diese Schreibweise nicht gefällt, der kann natürlich auch die etwas längere verwenden.

Zahlen werden natürlich ebenso deklariert:

```
Dim intZahl As Integer
```

Gefüllt wird die Variable `intZahl` mit dem Wert 8:

```
intZahl = 8
```

Oder in einem Schritt:

```
Dim intZahl As Integer = 8
```

### 1.13 Verzweigungen

Nun hat der Benutzer seinen Namen eingegeben und dieser ist an eine Variable übergeben worden. Was geschieht jedoch, wenn der Benutzer seinen Namen nicht eingibt? Dann soll er einen Hinweis erhalten, dass er seinen Namen eingeben muss, sonst gibt es keinen Kaffee – wie im richtigen Leben eben.

Diese Abfrage, oder genauer: diese Entscheidung, wird mit einer Verzweigung vorgenommen. Wahrscheinlich kennen Sie diese Anweisung schon, sie lautet:

```
If txtName.Text = "" Then MessageBox.Show _  
("Hey, Benutzer, dein Name fehlt! ")
```

Dann würde der Benutzer nach der Schelte seinen Kaffee erhalten. Das soll er aber nicht. Ich verwende die oben beschriebene einzeilige Syntax sehr selten, lieber schreibe ich die If-Verzweigung auf mehrere Zeilen verteilt. Drückt man nach der Bedingung

```
If txtName.Text = "" Then
```

die , dann erscheint zwei Zeilen tiefer das Ende der Verzweigung:

```
End If
```

Zwischen diese beiden Zeilen kann man die Anweisung schreiben:

```
If txtName.Text = "" Then  
    MessageBox.Show _  
    ("Hey, Benutzer, dein Name fehlt! ")  
End If
```

Damit erhält der Benutzer immer noch Kaffee. Aber man hat einen besseren Überblick und kann mehr als eine Befehlszeile verwenden (zum Beispiel, wenn man den Benutzer dreimal hintereinander beschimpfen möchte).

Wenn kein Name eingegeben wurde, dann reagiere so, ansonsten gibt es Kaffee. Dieses umgangssprachliche »sonst« kann auch in den Code eingebaut werden – mit dem Befehl Else:

```
If txtName.Text = "" Then  
    MessageBox.Show _  
    ("Hey, Benutzer, dein Name fehlt! ")  
Else  
    MessageBox.Show(Achtung: " & txtName.Text & _  
    "Jetzt gibt es Kaffee! ")  
End If
```

Sollen mehr als zwei Fälle geprüft werden, dann kann dies mit der Anweisung ElseIf erledigt werden. Soll beispielsweise Hugo Tee und Maria Schokolade bekommen, dann ist so vorzugehen:

```
If txtName.Text = "" Then  
    MessageBox.Show _  
    ("Hey, Benutzer, dein Name fehlt! ")
```

```

ElseIf txtName.Text = "Hugo" Then
    MessageBox.Show(Achtung: " & txtName.Text & _
        "Jetzt gibt es Tee! ")
ElseIf txtName.Text = "Maria" Then
    MessageBox.Show(Achtung: " & txtName.Text & _
        "Jetzt gibt es Schokolade! ")
Else
    MessageBox.Show(Achtung: " & txtName.Text & _
        "Jetzt gibt es Kaffee! ")
End If

```

Überprüft wird der Code von oben nach unten. Zuerst wird also geprüft, ob nichts eingegeben wurde. Dann wird Hugo überprüft, dann Maria und schließlich der Rest.

Im Übrigen können Verzweigungen auch ineinander geschachtelt werden. Dazu bräuchte man allerdings noch eine zweite Information, die man überprüfen könnte. Das Ganze sieht dann beispielsweise so aus:

```

If txtName.Text = "" Then
    MessageBox.Show _
        ("Hey, Benutzer, dein Name fehlt! ")
ElseIf txtName.Text = "Hugo" Then
    If CInt(txtAlter.Text) < 10 Then
        MessageBox.Show(Achtung: " & _
            txtName.Text & "Jetzt gibt es Tee!")
    Else
        MessageBox.Show(Achtung: " & _
            txtName.Text & _
            "Jetzt gibt es Tee mit Rum!")
    End If
ElseIf txtName.Text = "Maria" Then
    If CInt(txtAlter.Text) < 10 Then
        MessageBox.Show(Achtung: " & _
            txtName.Text & _
            "Jetzt gibt es Schokolade! ")
    Else
        MessageBox.Show(Achtung: " & _
            txtName.Text & _
            "Jetzt gibt es Schokolade mit Cognac! ")
    End If
Else

```

```
        MessageBox.Show(Achtung: " & txtName.Text & _  
        "Jetzt gibt es Kaffee! ")  
    End If
```



Abbildung 1.22 Ein dezenter Hinweis

Die allgemeine Syntax für die If-Verzweigung lautet:

```
If Bedingung Then  
    [Anweisung]  
ElseIf Bedingung Then  
    [Anweisung]  
ElseIf Bedingung Then  
    [Anweisung]  
ElseIf Bedingung Then  
    [Anweisung]  
Else  
    [Anweisung]  
End If
```

Achtung: ElseIf wird in einem Wort zusammengeschrieben, während End If zwei Schlüsselwörter sind. Aber das ist fast kein Problem, denn nach der Zeile

```
If Bedingung Then  
    erscheint automatisch  
End If
```