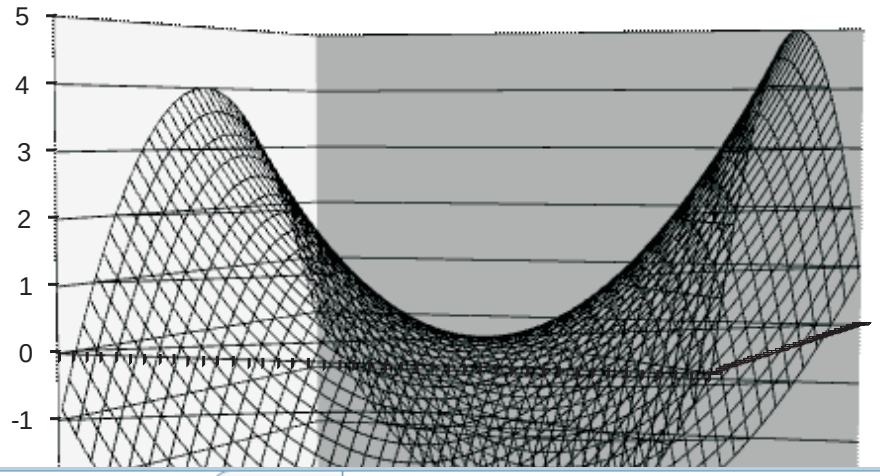
| Gemüse | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Die Angaben beziehen sich auf je 100 Gramm verzehrfertiges Nahrungsmittel | roh oder gekocht | kcal | kJ | % Kohlehydrate | Eiweiß | % Gesamtfett | % Wasser | % Ballaststoffe | mg Vitamin C | mg Calcium | mg Kalium | mg Phopsphor | mg Magnesium |
| Artischocken | roh | 57 | 239 | **11** | + | 3 | 86 | 3 | 9 | 52 | 410 | 110 | 26 |
| " | gek | 57 | 239 | **11** | + | 3 | 86 | 3 | 6 | 50 | 315 | 90 | |
| Auberginen | roh | 25 | 105 | 5 | + | 1 | 93 | 3 | 5 | 16 | 210 | 26 | 11 |
| " | gek | 16 | 67 | **3** | + | 1 | 93 | 3 | 5 | 16 | 210 | 26 | |
| Blattsellerie | roh | 20 | 84 | 4 | + | 1 | 95 | 2 | 9 | 45 | 290 | 40 | |
| Blumenkohl | roh | 28 | 117 | **8** | + | 2 | 92 | 2 | 76 | 24 | 380 | 60 | 7 |
| " | gek | 20 | 84 | 3 | + | 2 | 94 | 2 | 45 | 18 | 250 | 52 | |

hyperbolischer Paraboloid ($z = x^2 - y^2$)




MITTELWERT =SUMME(G6:G27)

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Nr. | 1994 | Beleg Nr. | | | Beträge | | 1 Einnahmen der Kirchen-gemeinde und Kirchenstiftu | | 2 Die eigene Gemeinde | | | | | | |
| 2 | | Tag | Einnahme | Ausgabe | Vorgang | nach zeitlichem Anfall | | Klingelbeutel, Opferstock, Kollekte für ortskirchliche Bedürfnisse | | 2.1 Für Kirchengemeinde / Kirchenstiftungen und weitere gemeindliche Arbeit | | | | 2.2 Einrichtungen (Kindergarten, Diakoniestation, u.ä.) | | 2.3 Fü |
| 3 | | Monat | | | | | | | | 2.1.1 Kirche, Gemeindehaus, Jugendheim, Friedhof u.ä. | | 2.1.2 Übrige Zwecke (Gottesdienst, Kirchenschmuck u.ä.) | | | | 2.3.1 Diak Aufg Unterstü |
| 4 | | | | | | Einnahme | Ausgabe | Einnahme | Ausgabe | Einnahme | Ausgabe | Einnahme | Ausgabe | Einnahme | Ausgabe | Einnahme |
| 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 6 | | | | | Überträge von voriger Seite | 35539,37 | 15001,43 | | | | | 5641 | | 3384 | 15145 | |
| 7 | 1 | 2.5. | | 65 | Geburtstagsblumen | | 15,00 | | | | | | | | | |
| 8 | 2 | " | | 66 | Geburtstagsgeschenk Vikarin | | 48,20 | | | | | | | | | |
| 9 | 3 | " | | 67 | an Landfahrer Spende | | 80,00 | | | | | | | | | |
| 10 | 4 | " | | 68 | Bewirtung | | 6,60 | | | | | | | | | |
| 11 | 5 | " | | 69 | Geschenke für Tansania | | 153,67 | | | | | | | | | |
| 12 | 6 | " | 46 | | Kollekte | 30,50 | | | | | | | | | | |
| 13 | 7 | 3.5. | | 70 | Sommerfest - Bier und Würstl | | 587,34 | | | | | | | | | |
| 14 | 8 | " | 47 | | Spende für Sommerfest | 61,70 | | | | | | | | | | |
| 15 | 9 | " | | 71 | Abscheidsgeschenk Zivi | | 35,60 | | | | | | | | | |
| 16 | 10 | " | 48 | | Spende | 100,00 | | | | | | | | | | |
| 17 | 11 | " | | 72 | Bewirtung | | 52,20 | | | | | | | | | |
| 18 | 12 | " | | 73 | Blumen | | 23,50 | | | | | | | | | |
| 19 | 13 | " | | 74 | an kath. Pfarrkirchendienst | | 3387,50 | | | | | | | | | |
| 20 | 14 | 4.5. | | 75 | Abscheidsgeschenk Putzfrauen | | 63,80 | | | | | | | | | |
| 21 | 15 | 5.5. | 49 | | Einnahmen Frühlingsfest | 1867,24 | | | | | | | | | | |
| 22 | 16 | " | | | Bank an Kasse | | | | | | | | | | | |
| 23 | 17 | " | | 76 | an Dek. Kollekte | | 87,50 | | | | | | | | | |
| 24 | 18 | " | 50 | | Habenzins Sparkasse | 4,91 | | | | | | | | | | |
| 25 | 19 | " | | 77 | Prodekanat, Gemeinde Tansania | | 877,40 | | | | | | | | | |
| 26 | 20 | 6.5. | | | Bewirtung Gäste | | 119,70 | | | | | | | | | |
| 27 | 21 | " | | | Umbuchung [illegible] | | | | | | | | | | | |
| 28 | | | | | | 37603,72 | =SUMME(G6:G27) | | | | | | | | | |
| 29 | | | | | | | | | | | | | | | | |

# Excel VBA

# Inhaltsverzeichnis

# 1 Grundlagen der VBA-Programmierung

Zu den Grundlagen der Beschäftigung mit einer Entwicklungssprache gehören Variablen, Schleifen und Verzweigungen. Ebenso gehört ein strukturiertes und überlegtes Fehlermanagement, Testen und Aufspüren eigener Fehler dazu. All dies wird in diesem Kapitel beschrieben.

Sämtliche Themen werden erläutert und einige Probleme beschrieben, die sich daraus ergeben.

Um die Theorie der Kontrollstrukturen anschaulich darzustellen, habe ich in diesem Kapitel einige Beispiele allgemeiner Natur aufgenommen.

## 1.1 Allgemeiner Aufbau der Prozeduren

Damit Sie in Excel 2007 „vernünftig“ programmieren können, sollten Sie über die Office-Schaltfläche in Excel-Optionen „die Entwicklerregisterkarte in der Multifunktionsleiste anzeigen“ lassen. Dann erscheint in der Multifunktionsleiste der neue Menüpunkt „Entwicklertools“. Dann stehen Ihnen die entsprechenden Gruppen zur Verfügung, mit deren Hilfe Sie schnell Makros aufzeichnen können, beziehungsweise Sie denn Code im Visual-Basic-Editor bearbeiten können.

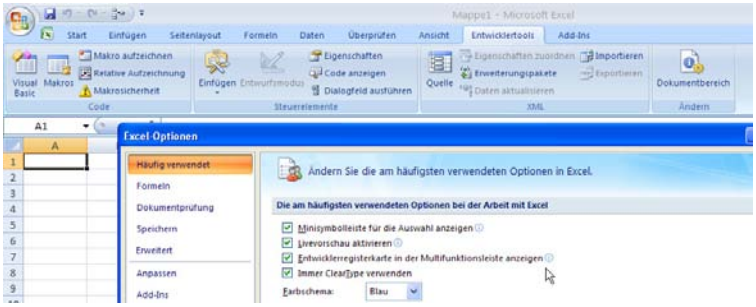



**Abbildung 1.1** Die Entwicklerregisterkarte „Entwicklertools“

Über die Schaltfläche „Visual Basic“ in der ersten Gruppe Code gelangen Sie in die Entwicklungsumgebung, wo Sie Module, UserFormen und Klassenmodule erstellen können.

Lassen Sie uns mit dem allgemeinen Aufbau von Prozeduren in Modulen beginnen.

### 1.1.1 Die Syntax einer Prozedur

```
[Private | Public] [Static] Sub Name [(ArgListe)]

[Anweisungen]

[Exit Sub]

[Anweisungen]

End Sub
```

**Tabelle 1.1** Die Elemente eines Prozedurnamens

| Teil | Beschreibung |
|---|---|
| Public | Auf die Prozedur, die Public Sub definiert ist, kann von allen anderen Prozeduren in allen Modulen zugegriffen werden. Bei Verwendung in einem Modul kann auf die Prozedur nur innerhalb des Projekts zugegriffen werden. |
| Private | Auf die Prozedur, die Private Sub definiert ist, kann nur durch andere Prozeduren aus dem Modul zugegriffen werden, in dem sie deklariert wurde. |
| Static | Die lokalen Variablen der Prozedur, die Static Sub definiert ist, bleiben zwischen Aufrufen erhalten. Das Attribut „Static“ wirkt sich nicht auf Variablen aus, die außerhalb der Prozedur deklariert wurden, auch wenn sie in der Prozedur verwendet werden. |
| Name | Erforderlich. Name der Prozedur gemäß den Standardkonventionen für Namen von Variablen: maximal 255 Zeichen, kein Schlüsselwort und eindeutig. Der Name darf nicht mit einer Ziffer beginnen und keine Satz- oder Sonderzeichen enthalten – mit Ausnahme des Unterstrichs „_“. |
| ArgListe | Variablenliste mit den Argumenten, die an die Prozedur beim Aufruf übergeben werden. Mehrere Variablen werden durch Kommas getrennt. |

| Teil | Beschreibung |
|---|---|
| Anweisungen | Eine beliebige Gruppe auszuführender Anweisungen im Rumpf der Prozedur. |

**Achtung**

Dass der Bindestrich im Prozedurnamen verboten ist, ist hinlänglich bekannt. Weniger klar ist, dass auch die deutschen Umlaute „ä“, „ö“ und „ü“ und das „ß“ verwendet werden dürfen – ein solches Programm läuft auch in den USA. Beachten Sie jedoch, dass es auf asiatischen Rechnern, die mit Zeichenersetzungsprogrammen ausgestattet sind, zu Konflikten führen kann. Um bei Prozedurnamen (nicht mit den Schlüsselwörter in Konflikt zu kommen, verwende ich gerne deutsche Bezeichnungen für Funktions- und Prozedurnamen. Dabei folge ich der Konvention SubstantivVerb oder VerbSubstantiv. Selbstredend sollte der Name einer Prozedur einen aussagekräftigen Rückschluss auf ihre Funktion erlauben. Und: CamelCasing, also die Verwendung von Groß- und Kleinschreibung, erleichtert das Lesen von Prozedurnamen, beispielsweise „NamenEintragen“, „DateinamenAuslesen“ oder „BenutzerInformationenAnzeigen“.

Wenn Sie mehrere Makros in unterschiedlichen Modulen speichern, dann sollten Sie den Modulen sprechende Namen geben.

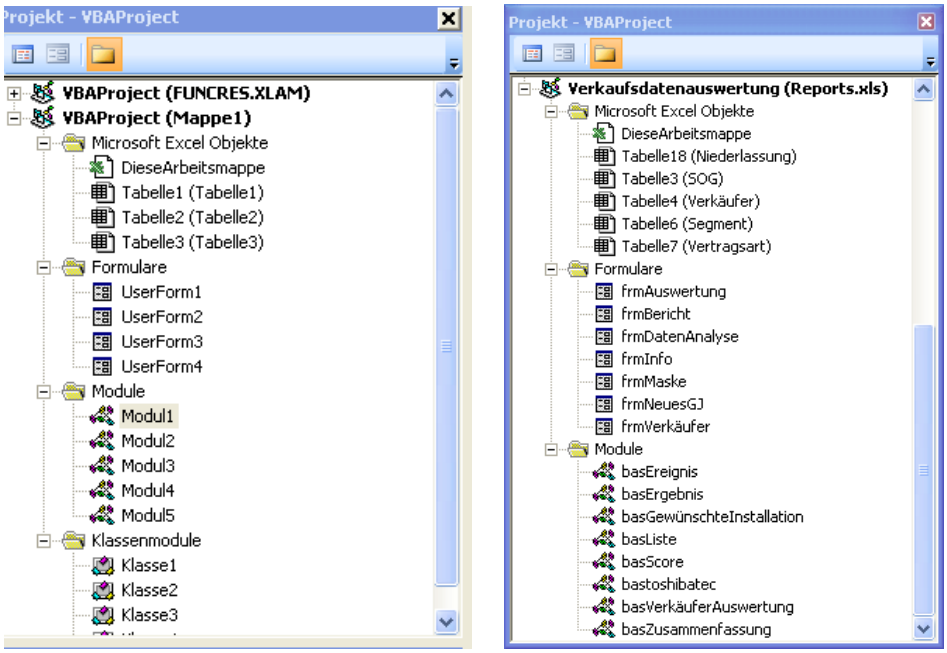



**Abbildung 1.2** Die linke Variante ist sehr unübersichtlich; vernünftige Namen erleichtern die Suche.

### 1.1.2 Kommentare

Kommentare werden mit einem Apostroph „'“ eingeleitet, das am Anfang oder innerhalb einer Zeile stehen kann. Kommentare können ebenso durch ein rem (remark) eingeleitet werden, das sich nur am Anfang der Zeile befinden darf. Kommentare, die mit rem eingeleitet werden, dürfen nicht hinter Befehlen stehen. Da der Unterstrich einen Befehl in mehrere Zeilen trennt, darf auch kein Kommentar hinter dem Unterstrich stehen. So nicht:

```
MsgBox "Beachten Sie die Firmenrichtlinien!", _  '    Meldungsfenster

    vbInformation , "compurem"
```

Entweder verwenden Sie das Anführungszeichen, oder schreiben Sie die Remark-Zeile als eigenständige Zeile.

**Tipp**

Für das Apostroph steht Ihnen in der Symbolleiste „Bearbeiten“ ein Symbol zum Ein- und Ausschalten zur Verfügung.

Kommentare erscheinen in grüner Schrift, was Sie im Menü Extras | Optionen im Blatt Editorformen unter der „Codefarbe“ Kommentartext ändern könnten.

**Tipp**
Dieses Symbol können Sie mit gedrückter <ALT>-Taste in die Standardsymbolleiste ziehen oder mit gedrückter <ALT>+<STRG>-Taste hineinkopieren.

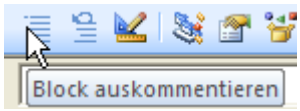



**Abbildung 1.3** Kommentare können mit Hilfe der Symbole schnell ein- und ausgeschaltet werden.

**Hinweis**
Kommentieren Sie blockweise! Kommentieren Sie alles, was Sie programmieren! Auch wenn es auf den ersten Blick nach viel Arbeit aussieht, hilft es Ihnen, doch im Nachhinein schnell einen Überblick über alte oder fremde (von anderen Programmierern geschriebene, Codeteile zu erhalten.

Schreiben Sie in ein Modul zu Beginn Ihren Namen, das Erstellungsdatum und das Datum der letzten Änderung. Dann sind Sie sicher, dass Sie die Informationen parat haben, wenn Sie herausfinden müssen, ob Sie in der aktuellen Version programmieren.

### 1.1.3 Programmzeilen

Ein automatischer Umbruch, wie von der Textverarbeitung bekannt, findet erst nach 1.024 Zeichen statt: Um einen manuellen Umbruch zu organisieren, kann und sollte der Code, wenn er länger ist, in mehrere Zeilen geteilt werden. Dies erfolgt durch eine obligatorische Leerstelle, der ein Unterstrich am Ende der Zeile folgt.

**Hinweis**
Sie dürfen maximal zehn Zeilen Code durch "_" voneinander trennen. Und der Unterstrich darf nicht innerhalb von Textteilen stehen. Vor dem Unterstrich muss eine Leerstelle stehen!

Sollen mehrere Befehle in einer Zeile geschrieben werden, dann können diese durch einen Doppelpunkt voneinander getrennt werden, was allerdings die Lesbarkeit des Codes erschwert.

## 1.2 Variablen, Konstanten und Datentypen

Die folgende Tabelle listet sämtliche Datentypen auf, die VBA zur Verfügung stellt.

**Tabelle 1.2 Eine Zusammenfassung der verschiedenen Variablentypen:**

| Datentyp | Variablentyp | Typen-kennzei-chen | Bereich | Typkür-zel | Beispiel | Speicherplatz (in Byte) |
|---|---|---|---|---|---|---|
| | Boolean | | 0 (False) und -1 (True) | f oder bln | True, False | 2 |
| Ganzzahlen | Byte | | 0 bis 255 | byt | 7, 104 | 1 |
| | Integer | % | -32.768 bis 32.767 | int | 7, 104, 1234 | 2 |
| | Long | & | -2.147.483.648 bis 2.147.483.647 | lng | 12345678 | 4 |
| Dezimalzahlen | Single | ! | $-3{,}402823 * 10^{38}$ bis $3{,}402823 * 10^{38}$ | sng | 2,7182818 | 4 |
| | Double | # | $-1{,}797693 * 10^{324}$ bis $1{,}797693 * 10^{324}$ | dbl | $12345678 * 10^{100}$ | 8 |

| Datentyp | Variablentyp | Typen-kennzei-chen | Bereich | Typkür-zel | Beispiel | Speicherplatz (in Byte) |
|---|---|---|---|---|---|---|
| | Currency | @ | -9,22 * 10 $^{14}$ bis 9,22 * 10 $^{14}$ | cur | 59,90 € | 8 |
| | Decimal | | 79 * 10 $^{28}$ bis 79 * 10 $^{28}$ | dec | | 14 |
| Datums-angaben | Date | | 1.1.100 bis 31.12.9999 | dat | 11.11. 2008 | 8 |
| Text | String | $ | Circa 2 Milliarden | str | "Konrad und Paul" | 10 + Länge der Zeichenkette |
| | Variant | | jeder numerische Wert im Bereich Double, jeder String | | | 22 + Länge der Zeichenkette |
| Objekte | Object | | alle Objektreferenzen | | | 4 |
| Benutzer-definiert | | | | | | |

**Hinweis**

Variablen sollten ein Präfix besitzen, an dem ihr Typ erkennbar ist. Diese Konvention, die von der Firma „Gregory Reddick & Associates“, einer Unternehmensberatungsfirma von Microsoft, herausgegeben wurde, ist nicht verbindlich. Allerdings arbeiten sehr viele Programmierer damit. Diese Konvention stellt eine Möglichkeit der Standardisierung für VBA-Programmierung dar.

**Tipp**

Deklarieren Sie jede Variable einzeln! Eine Deklaration der Form

```
Dim i, j, k As Integer

i = "Play it, Sam."

j = "Play As Time Goes By."

k = "Casablanca"
```

wirft erst in der letzten Zeile einen Fehler aus, da i und j vom Typ Variant deklariert sind.

**Tipp**

Das Typkennzeichen an das Ende des Variablennamens zu hängen ist möglich, aber veraltet. Verwenden Sie es nicht, sondern schreiben Sie die Variablen nach dem Schema:

```
Dim strDateiName As String

Dim intZähler As Integer

Dim lngDateienAnzahl As Long

Dim datVertragsdatum As Date

...
```

**Tipp**

Sie sollten String-Variablen nicht auf eine Textgröße begrenzen. Dieses in anderen Programmiersprachen nötige Verfahren ist hier überflüssig, da sogar verwirrend, weil VBA die überschüssigen Zeichen mit Leerzeichen auffüllt. Das folgende Beispiel gibt einen Boolean-Wert „False“ zurück:

```
Dim strZugTyp As String * 10

strZugTyp = "ICE"

MsgBox strZugTyp = "ICE"
```

Auch der Gültigkeitsbereich sollte im Präfix des Variablennamens stehen: „m“ steht dabei für modulweit, „g“ für global, also modulübergreifend. Beispielsweise:

```
Dim blnTabellenBlattVorhanden As Booleaen

Dim mstrDateiNameMitPfad As String

Public gintAbteilungsNummer As Integer
```

**Achtung**

Auch wenn sämtliche Beispiele, die aus dem Hause Microsoft kommen, nicht den Reddick-Konventionen folgen, so sollten Sie sich kein Beispiel an diesem schlechten Programmierstil nehmen, sondern die Variablen sauber deklarieren.

**Achtung**

Beachten Sie genau den Gültigkeitsbereich der Variablen. Excel hat beispielsweise 1.048.576 Zeilen, bis Office 2003 waren es mehr als 65.000 Zeilen. Um eine Excel-Tabelle zu durchlaufen, dürfen Sie nicht eine Integer-Variable verwenden! Fast alle Städte in Sachsen und Thüringen verwenden bei den Postleitzahlen eine führende „0“. Verwenden Sie deshalb eine String-Variable, wenn Sie deutsche Postleitzahlen speichern. Excel unterscheidet konsequent zwischen Text und Zahl. Dialoge geben jedoch lediglich Texte zurück. Überprüfen Sie also korrekt, ob es sich um eine Zahl oder einen Text handelt, wenn Sie Werte nach Excel schreiben oder aus Excel Daten auslesen. Seien Sie nicht knauserig mit Speicherplatz! Die Zeiten, als Programmierer jedes Byte „herumdrehen“ mussten, sind vorbei. Schießen Sie notfalls lieber über das Ziel – heute hat niemand mehr einen langsamen Rechner.

**Tipp**

Verwenden Sie nicht den Typ „Variant“. Er ist unflexibel! Entscheiden Sie sich für eine saubere Programmierung, das heißt für Text oder Zahl. Da Variant den Typ intern wechseln kann, kann es zu Fehlern kommen, da beispielsweise „+“ für die Addition als auch für die Textverkettung reserviert ist.

**Tipp**

Zwingen Sie sich zur Variablendeklaration. Dies können Sie über das Menü Extras | Optionen einschalten – dann steht zu Beginn jedes Moduls und jeder Klasse und im Codeteil der Dialoge der Befehl Option Explicit. Dies hilft, um lästige Tippfehler zu vermeiden.

### 1.2.1 Eigene Datentypen

Aus den vorgegebenen Datentypen können Sie eigene Datentypen zusammensetzen.

```
[Private | Public] Type VarName

Elementname [([Indizes])] As Typ

[Elementname [([Indizes])] As Typ]

...

End Type
```

**Beispiel**

Ein benutzerdefinierter Datentyp

```
Public Type Anschrift
   Strasse As String
   Hausnummer As String
   Postleitzahl As String
   Ort As String
End Type

Sub Verlagsliste()
   Dim Verlag1 As Anschrift
   Dim Verlag2 As Anschrift
   Dim Verlag3 As Anschrift
   Verlag1.Strasse = "Kolbergstraße"
   Verlag1.Hausnummer = "22"
   Verlag1.Postleitzahl = "80315"
   Verlag1.Ort = "München"
   MsgBox Verlag1.Ort
   With Verlag2
      .Strasse = "Kolbergstraße"
      .Hausnummer = "22"
      .Postleitzahl = "80315"
      .Ort = Verlag1.Ort
   End With

MsgBox Verlag2.Ort
Verlag3 = Verlag2
MsgBox Verlag3.Ort

End Sub
```

### 1.2.2 Konstanten

Die Syntax lautet:

```
[Public | Private] Const KonstName [As Typ] = Ausdruck
```

**Tabelle 1.3 Die Syntax der Const-Anweisung besteht aus folgenden Teilen:**

| Teil | Beschreibung |
|---|---|
| Public | Schlüsselwort, das auf Modulebene Konstanten deklariert, die allen Prozeduren in allen Modulen zur Verfügung stehen. |
| Private | Schlüsselwort, das auf Modulebene Konstanten deklariert, die nur innerhalb des Moduls verfügbar sind, in dem sie deklariert wurden. |
| KonstName | Erforderlich. Name der Konstanten. |
| Typ | Zulässige Typen sind Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String oder Variant. |
| Ausdruck | Erforderlich. Ein Literalwert, eine andere Konstante oder eine beliebige Kombination, die beliebige arithmetische oder logische Operatoren (mit Ausnahme von Is) enthält. |

**Tipp**

Greifen Sie in Excel nicht auf Spalten über ihre Nummern zu. Die Praxis zeigt, dass der Anwender möglicherweise eine weite Spalte in einer vorhandenen Tabelle einfügen möchte. Dann müssen Sie das gesamte Programm darauf hin überprüfen, welche Spaltennummer wo verwendet wird. Besser ist es, Spaltennummern in Konstanten auszulagern – sie können im Nachhinein leichter geändert werden. Das Gleiche gilt ebenso für Dateinamen und Pfade. Sie können auch von unerfahrenen VBA-Programmierern angepasst werden – der Änderungsaufwand ist also gering.

**Tipp**

Sie sollten nicht nur alle Variablen und Konstanten sauber deklarieren, sondern dies im Kopfbereich erledigen, bitte nicht im Code. Im Kopfbereich finden Sie dann die Variablen wieder, wenn Sie vergessen haben, wie eine heißt.

### 1.2.3 Datenfelder, Arrays

Datenfelder werden genauso wie andere Variablen mit Hilfe der Dim-, Static-, Private- oder Public-Anweisungen deklariert. Der Unterschied zwischen „skalaren Variablen“ (Variablen, die keine Datenfelder sind) und „Datenfeldvariablen“ besteht darin, dass Sie generell die Größe des Datenfelds angeben müssen. Ein Datenfeld, dessen Größe angegeben ist, ist ein Datenfeld fester Größe. Ein Datenfeld, dessen Größe bei Ausführung eines Programms geändert werden kann, ist ein dynamisches Datenfeld.

**Tipp**

Ob ein Datenfeld mit 0 oder 1 beginnend indiziert ist, hängt von der Einstellung der Option Base-Anweisung ab. Wenn Option Base 1 nicht angegeben ist, beginnen alle Datenfelder mit dem Index 0.

### Deklaration eines Datenfelds mit fester Größe

In der folgenden Codezeile wird ein Datenfeld fester Größe als Integer-Datenfeld mit 11 Zeilen und elf Spalten deklariert:

```
Dim intMeinDatenfeld(10, 10) As Integer
```

Das erste Argument stellt die Zeilen, das zweite Argument die Spalten dar.

**Tipp**
Deklarieren Sie Ihre Datenfelder explizit mit einem Datentyp, der nicht „Variant“ ist, um den Code so kompakt wie möglich zu machen!

Die folgenden Codezeilen vergleichen die Größe verschiedener Datenfelder:

```
' Nachstehendes Datenfeld aus Elementen des Datentyps
' Integer beansprucht 22 Bytes (11 Elemente * 2 Bytes).
ReDim intMeinIntegerDatenfeld(10) As Integer


' Nachstehendes Datenfeld aus Elementen des Datentyps
' Double beansprucht 88 Bytes (11 Elemente * 8 Bytes).
ReDim dblMeinDoubleDatenfeld(10) As Double


' Nachstehendes Datenfeld aus Elementen des Datentyps Variant
' beansprucht mindestens 176 Bytes (11 Elemente * 16 Bytes).
ReDim MeinVariantDatenfeld(10)


' Nachstehendes Datenfeld aus Elementen des Datentyps
' Integer beansprucht 100 * 100 * 2 Bytes (20.000 Bytes).
ReDim intMeinIntegerDatenfeld (99, 99) As Integer


' Nachstehendes Datenfeld aus Elementen des Datentyps
' Double beansprucht 100 * 100 * 8 Bytes (80.000 Bytes).
ReDim dblMeinDoubleDatenfeld (99, 99) As Double


' Nachstehendes Datenfeld aus Elementen des Datentyps Variant
' beansprucht mindestens 160.000 Bytes
' (100 * 100 * 16 Bytes).
ReDim MeinVariantDatenfeld(99, 99)
```

**Tipp**

Die maximale Größe eines Datenfelds hängt von Ihrem Betriebssystem sowie von dem verfügbaren Speicher ab. Durch die Verwendung eines Datenfeldes, das den für Ihr System verfügbaren RAM-Speicher überschreitet, wird Ihre Anwendung langsamer, da die Daten von der Festplatte gelesen und auf diese geschrieben werden müssen.

### Deklarieren eines dynamischen Datenfeldes

Bei der Deklaration eines dynamischen Datenfelds können Sie die Größe des Datenfelds verändern, während der Code ausgeführt wird. Verwenden Sie zur Deklaration eines Datenfelds eine der Static-, Dim-, Private- oder Public-Anweisungen, und lassen Sie die Klammern leer, beispielsweise:

```
Dim sngDatenfeld() As Single
```

**Achtung**

Sie können die ReDim-Anweisung dazu verwenden, ein Datenfeld implizit innerhalb einer Prozedur zu deklarieren. Achten Sie darauf, bei Verwendung der ReDim-Anweisung den Namen des Datenfelds richtig zu schreiben. Auch wenn sich die Option Explicit-Anweisung im Modul befindet, wird ein zweites Datenfeld erstellt, wenn Sie den Namen falsch schreiben.

Verwenden Sie in einer Prozedur innerhalb des Gültigkeitsbereichs des Datenfeldes die ReDim-Anweisung zur Änderung der Anzahl von Dimensionen, zur Definition der Anzahl der Elemente und zur Definition der oberen und unteren Grenzen jeder Dimension. Sie können die ReDim-Anweisung beliebig oft verwenden, um das dynamische Datenfeld zu ändern. Dies hat jedoch zur Folge, dass die bestehenden Werte des Datenfeldes verloren gehen. Verwenden Sie `ReDim Preserve`, um ein Datenfeld zu erweitern, ohne dass die vorhandenen Werte im Datenfeld gelöscht werden. Die folgende Anweisung vergrößert zum Beispiel das Datenfeld sngDatenfeld um zehn Elemente, ohne die aktuellen Werte der ursprünglichen Elemente zu löschen.

```
ReDim Preserve sngDatenfeld (UBound(sngDatenfeld) + 10)
```

**Hinweis**

Wenn Sie das Schlüsselwort Preserve mit einem dynamischen Datenfeld verwenden, können Sie nur die obere Grenze der letzten Dimension, aber nicht die Anzahl der Dimensionen ändern.

**Beispiel**

Das folgende Beispiel liest alle Dateien aus einem Ordner ein und speichert sie in einem Datenfeld, das anschließend ausgelesen wird:

```
Public Sub DatenFeld()
   Const SUCHPFAD As String = _
   "C:\Dokumente und Einstellungen\Eigene Bilder\"
   Dim strDateiNamen() As String
   Dim lngDateienAnzahl As Long
   Dim strGefundenerDateiName As String

   On Error GoTo ende
   lngDateienAnzahl = 0
   strGefundenerDateiName = Dir(SUCHPFAD, vbNormal)
   Do While strGefundenerDateiName <> ""
      If strGefundenerDateiName <> "." And _
         strGefundenerDateiName <> ".." Then
         If (GetAttr(SUCHPFAD & strGefundenerDateiName) And _
            vbDirectory) = vbDirectory Then
            ReDim Preserve strDateiNamen(lngDateienAnzahl)
            strDateiNamen(lngDateienAnzahl) = strGefundenerDateiName
            lngDateienAnzahl = lngDateienAnzahl + 1
         End If
      End If
      strGefundenerDateiName = Dir
   Loop       ' -- durchlaufe den Pfad und suche alle Dateien,
   ' -- die in einem Datenfeld gespeichert werden
   If strGefundenerDateiName = "" Then
      MsgBox "Es wurden keine Dateien gefunden"
   Else
      strGefundenerDateiName = ""
      For lngDateienAnzahl = 0 To UBound(strDateiNamen)
         strGefundenerDateiName = strGefundenerDateiName & _
            vbCr & strDateiNamen(lngDateienAnzahl)
      Next     ' -- das Datenfeld wird ausgelesen
      MsgBox strGefundenerDateiName
   End If
   Exit Sub
ende:
   MsgBox "Fehler:" & vbCr & Err.Description
End Sub
```

**Achtung**

Beachten Sie, dass es in VBA keine Möglichkeit gibt zu überprüfen, ob ein Array initialisiert wurde. Das obere Beispiel würde die Fehlernummer 9 (Index außerhalb des gültigen Bereiches) liefern, wenn keine Datei gefunden wird. Man muss sich hier mit einer weiteren Variablen behelfen (beispielsweise einer Boolean-Variablen), die auf True, beziehungsweise False gesetzt wird.

Übrigens könne Sie ein Array auch bei der Initialisierung füllen:

```
Dim strOrte() As Variant

strOrte = Array("Berlin", "Hamburg", "München", "Köln", _

   "Frankfurt", "Leipzig")


MsgBox strOrte(2)
```

**Achtung**
Beachten Sie, dass Sie dieses Array vom Typ Varaint deklarieren müssen! Eine Deklaration vom Typ String würde zu einem Fehler führen.

### 1.2.4 Namensschilder für Schulungen

**Beispiel (Word)**
In einer Firma werden regelmäßig für Konferenzen und Schulungen Namensschilder erstellt. Nun hat es sich als unpraktisch erwiesen, eine Dokumentvorlage mit der maximalen Anzahl (20 Teilnehmer) zu erstellen, sondern man lässt den Benutzer in eine Eingabemaske die Namen eintragen und generiert dann daraus ein Dokument, das so viele Seiten hat, wie der Benutzer Namen eingegeben hat.

Fünf Variablen werden hierfür global deklariert:

```
Dim strName1() As String

Dim strFirma1() As String

Dim strName2() As String

Dim strFirma2() As String

Dim intSeiten As Integer
```

Beim Initialisieren werden die vier Datenfelder auf die Dimension 1 festgelegt:

```
Private Sub UserForm_Initialize()

   On Error Resume Next


   ReDim strName1(1)

   ReDim strFirma1(1)

   ReDim strName2(1)

   ReDim strFirma2(1)


   Me.lblPos.Caption = "1 von 1"

   intSeiten = 1

End Sub
```

Zusätzlich wird als Beschriftungstext „1 von 1“ angezeigt. Hat nun der Benutzer die ersten Informationen ausgefüllt und klickt auf „Neue Seite“, dann werden die Daten in dem Datenfeld gespeichert, der Zähler `intSeiten` um den Wert 1 erhöht und die Dimension der Arrays um 1 vergrößert. Da die gespeicherten Daten natürlich nicht gelöscht werden dürfen, muss mit dem Schlüsselwort `ReDim Preserve` gearbeitet werden. Die Textfelder werden geleert und das Bezeichnungsfeld neu beschriftet:

```
Private Sub cmdNeu_Click()

   On Error Resume Next


   Me.cmdWeiter.Enabled = True

   Me.cmdLetzter.Enabled = True


   strName1(intSeiten) = Me.txtName1.Value

   strFirma1(intSeiten) = Me.txtFirma1.Value

   strName2(intSeiten) = Me.txtName2.Value

   strFirma2(intSeiten) = Me.txtFirma2.Value


   intSeiten = UBound(strName1) + 1


   ReDim Preserve strName1(intSeiten)

   ReDim Preserve strFirma1(intSeiten)

   ReDim Preserve strName2(intSeiten)

   ReDim Preserve strFirma2(intSeiten)


   Me.txtFirma1.Value = ""

   Me.txtName1.Value = ""

   Me.txtFirma2.Value = ""

   Me.txtName2.Value = ""

   Me.txtName1.SetFocus


   Me.lblPos.Caption = intSeiten & " von " & intSeiten


End Sub
```

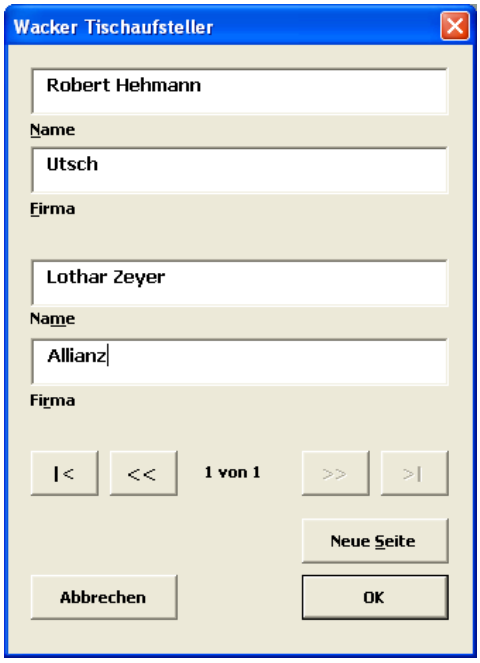



**Abbildung 1.4** Der Eingabedialog

Ähnlich funktioniert die Schaltfläche „Weiter“. Mit ihrer Hilfe werden die vier Datenfelder mit den „alten“ Werten gefüllt, da es sein könnte, dass eine Information geändert worden ist. Befindet sich der Datensatzzeiger nicht auf dem letzten Datensatz, wird der Zähler `intSeiten` um den Wert 1 erhöht, und die bereits vorhandenen Werte, in die Textfelder geschrieben:

```
Private Sub cmdWeiter_Click()

   On Error Resume Next

   strName1(intSeiten) = Me.txtName1.Value

   strFirma1(intSeiten) = Me.txtFirma1.Value

   strName2(intSeiten) = Me.txtName2.Value

   strFirma2(intSeiten) = Me.txtFirma2.Value


   If intSeiten < UBound(strName1) Then

      intSeiten = intSeiten + 1

      Me.txtName1.Value = strName1(intSeiten)

      Me.txtFirma1.Value = strFirma1(intSeiten)

      Me.txtName2.Value = strName2(intSeiten)

      Me.txtFirma2.Value = strFirma2(intSeiten)

      Me.lblPos.Caption = intSeiten & " von " & UBound(strName1)

   End If


End Sub
```

Der Sprung auf den letzten Datensatz ist klar: vorhandene Datenfelder füllen und den letzten Datensatz anzeigen:

```
Private Sub cmdLetzter_Click()

      On Error Resume Next


      strName1(intSeiten) = Me.txtName1.Value

      strFirma1(intSeiten) = Me.txtFirma1.Value

      strName2(intSeiten) = Me.txtName2.Value

      strFirma2(intSeiten) = Me.txtFirma2.Value


      intSeiten = UBound(strName1)

      Me.txtName1.Value = strName1(intSeiten)

      Me.txtFirma1.Value = strFirma1(intSeiten)

      Me.txtName2.Value = strName2(intSeiten)

      Me.txtFirma2.Value = strFirma2(intSeiten)

      Me.lblPos.Caption = intSeiten & " von " & intSeiten

End Sub
```

Ebenso der erste Datensatz:

```
Private Sub cmdErster_Click()

   On Error Resume Next


   strName1(intSeiten) = Me.txtName1.Value

   strFirma1(intSeiten) = Me.txtFirma1.Value

   Me.txtName1.Value = strName1(1)

   Me.txtFirma1.Value = strFirma1(1)


   strName2(intSeiten) = Me.txtName2.Value

   strFirma2(intSeiten) = Me.txtFirma2.Value

   Me.txtName2.Value = strName2(1)

   Me.txtFirma2.Value = strFirma2(1)


   Me.lblPos.Caption = "1 von " & intSeiten

   intSeiten = 1

End Sub
```

Auch das Bewegen um einen Datensatz nach vorne stellt keine Schwierigkeit dar. Man muss lediglich überprüfen, ob der Zeiger bereits auf dem ersten Datensatz sitzt.

Werden die Daten nun auf die Schilder geschrieben, dann muss überprüft werden, ob die Dimension der Datenfelder größer als 1 ist:

```
If UBound(strName1) > 1 Then
   Selection.WholeStory
   Selection.Copy
   For intSeiten = 2 To UBound(strName1)
      Selection.HomeKey Unit:=wdLine
      Selection.InsertBreak Type:=wdSectionBreakNextPage
      Selection.HomeKey Unit:=wdStory
      Selection.Paste
   Next
End If
```

Falls dies der Fall ist, werden so viele neue Seiten erzeugt, wie benötigt werden. Um die Textfelder zu füllen, gibt es sicherlich verschiedene Varianten, die Schleife hochzählen zu lassen. Ich habe mich für die Anzahl der Textfelder auf dem Dokument entschieden. Der Befehl „schreibe Text in ein Textfeld“ kann mit dem Makrorekorder aufgezeichnet werden. Leider kann der Text nicht direkt an die Textfelder übergeben werden, sondern man muss den Umweg über das Markieren gehen:

```
For intSeiten = 1 To ActiveDocument.Shapes.Count
   If Left(ActiveDocument.Shapes(intSeiten).Name, 8) = "Text Box" Then
      ActiveDocument.Shapes(intSeiten).Select
      If fLinks = True Then
         If fZweiMal = False Then
            ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
               Paragraphs(1).Range.Text = strName1(intArrayZähler) & _
               vbCr & strFirma1(intArrayZähler)
         Else
            ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
               Paragraphs(1).Range.Text = strName1(intArrayZähler) & _
               vbCr & strFirma1(intArrayZähler)
            fLinks = False
         End If
      Else
         If fZweiMal = False Then
            ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
               Paragraphs(1).Range.Text = strName2(intArrayZähler) & _
               vbCr & strFirma2(intArrayZähler)
         Else
            ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
```

```
                Paragraphs(1).Range.Text = strName2(intArrayZähler) & _
                vbCr & strFirma2(intArrayZähler)
             intArrayZähler = intArrayZähler + 1
             fLinks = True
          End If
       End If

       ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
          Paragraphs(1).Range.Font.Size = 22
       ActiveDocument.Shapes(intSeiten).TextFrame.TextRange. _
          Paragraphs(2).Range.Font.Size = 20

       fZweiMal = Not fZweiMal

    End If
 Next intSeiten
```

Und zum Schluss wird der Cursor an den Anfang des Dokumentes gesetzt und die Maske entladen. Hinter der Abbrechen-Schaltfläche befindet sich der Befehl, dass das Dokument geschlossen wird:

```
    ActiveDocument.Close
    Unload Me
```

Damit ich als Programmierer dennoch über Abbrechen wieder in das Dokument gelangen kann, schließe ich mit gedrückter <SHIFT>+<STRG>+<ALT>-Taste:

```
Private Sub cmdAbbrechen_MouseDown(ByVal Button As Integer, _
   ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   On Error Resume Next
   If Shift = 7 Then
      Unload Me
   Else
      ActiveDocument.Close
      Unload Me
   End If
End Sub
```

Und der Benutzer darf selbstverständlich nicht über das „x“ schließen:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   On Error Resume Next
   If CloseMode = vbFormControlMenu Then
```

```
        MsgBox "Bitte nicht über das ""x"" schließen"
        Cancel = True
    End If
End Sub
```

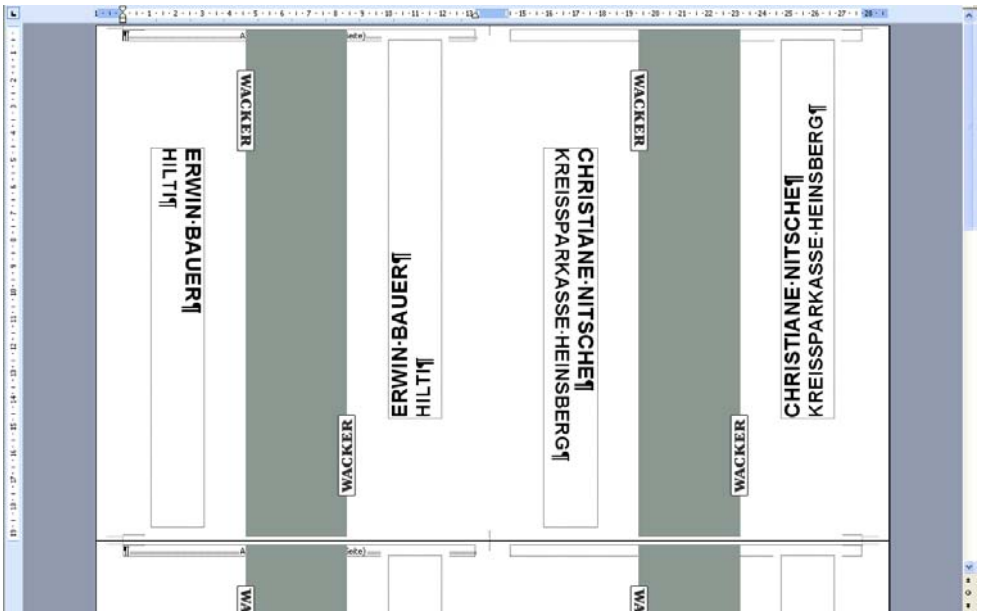



**Abbildung 1.5** Die fertigen Tischaufsteller

## 1.3 Ein- und Ausgabe

Zwei einfache Möglichkeiten zur Ein- und Ausgabe stehen Ihnen über das Meldungs- und das Eingabefenster zur Verfügung:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
InputBox(prompt[, title] [, default] [, xpos] [, ypos] _
   [, helpfile, context])
```

Wird bei der MsgBox eine Klammer verwendet, dann wird ein Wert vom Typ Integer zurückgegeben. Die InputBox gibt immer einen String-Wert zurück.

**Hinweis**
Da die InputBox unflexibel ist, sollten Sie nach Möglichkeit darauf verzichten.

**Achtung**
Wenn Sie die Ausgabe eines Textes gestalten möchten, dann sollten Sie nicht die MsgBox verwenden, sondern einen Dialog. Darauf müssen Sie auch zurückgreifen, wenn Sie einen längeren Text ausgeben möchten, d.h. einer, der länger als 1.024 Zeichen ist.

Wird das Meldungsfenster mit einer Klammer verwendet, dann muss ein Wert übergeben werden. Also beispielsweise so:

```
intAntwort = MsgBox(Prompt:="Diese Anweisung wird aufgrund eines " _
& "ungültigen Vorgangs geschlossen.", _
Buttons:=vbCritical + vbAbortRetryIgnore)
```

Wenn das Meldungsfenster jedoch keine Auswahlabfrage wie `YesNo` oder `AbortRetryIgnore` enthält, genügt zur alleinigen Anzeige folgender Befehl:

```
MsgBox Prompt:="Guten Morgen"
```

**Tipp**

Zur besseren Lesbarkeit sollten Sie die Zeilen mit der Tabulatortaste konsequent einrücken. Und: Auch wenn die beiden folgenden Zeilen gleich sind:

```
MsgBox Prompt:="Guten Morgen"
MsgBox "Guten Morgen"
```

ist die erste doch besser lesbar. Denn ein Befehl wie:

```
ActiveWorkbook.PrintOut , , 2, , , False
```

ist schwer zu verstehen.

**Tipp**

Um die Lesbarkeit zu erhöhen, sollten Sie immer die Namen der Systemkonstanten, beziehungsweise Parameter in den Code schreiben.

Ein Befehl:

```
MsgBox "Möchten Sie wirklich die Datei schließen?", _
   vbCritical + vbYesNoCancel + vbDefaultButton2 + vbSystemModal
```

ist leichter zu lesen als:

```
MsgBox "Möchten Sie wirklich die Datei schließen?", 16 + 3 + 256 + 4096
```

Oder gar:

```
MsgBox "Möchten Sie wirklich die Datei schließen?", 4371
```

Noch eleganter ist es natürlich, die Anweisung folgendermaßen zu schreiben:

```
MsgBox Prompt:="Möchten Sie wirklich die Datei schließen?", _
   Buttons:=vbCritical + vbYesNoCancel + _
   vbDefaultButton2 + vbSystemModal
```

**Hinweis**

Sollen innerhalb einer Zeichenkette Anführungszeichen geschrieben werden, so kann man sie in doppelte Anführungszeichen setzen. Oder man kann das Zeichen für Gänsefüßchen (Chr(34)) verketten. Analog steht für Zeilenwechsel vbLf oder Chr(10), für <ENTER> vbCr oder Chr(13). Der Tabulator wird durch vbTab oder Chr(9) ausgedrückt.

# 1.4 Operatoren, Verknüpfungen und Verzweigungen

Vor allem in Excel stellen Berechnungen ein zentrales Thema dar. Deshalb ist es unabdingbar die Operatoren, die VBA zur Verfügung stellt, zu kennen.

## 1.4.1 Operatoren

**Tabelle 1.4 Folgende Operatoren stehen Ihnen in VBA zur Verfügung:**

| Typ | Beschreibung | Zeichen/Operator |
|---|---|---|
| Arithmetische Operatoren | Addition | + |
| | Subtraktion | - |
| | Multiplikation | * |
| | Division | / |
| | Ganzzahlige Division | \ |
| | Potenz | ^ |
| | Modulo | Mod |
| Textverkettung | Concatenation | &, + |
| Logische Operatoren | und | AND |
| | oder | OR |
| | nicht | NOT |
| | exklusives Oder (entweder das eine oder das andere) | XOR |
| | logische Äquivalenz | EQV |
| | Implikation | IMP |
| Vergleichsoperatoren | gleich | = |
| | kleiner als | < |
| | kleiner oder gleich | <= |
| | größer als | > |
| | größer oder gleich | >= |
| | ungleich | <> |
| Vergleichsoperatoren für Text | entspricht | LIKE |
| Vergleichsoperatoren für Objekte | entspricht | IS |

Ein bekanntes Beispiel ist die Berechnung des Ostersonntags! Die Lösung dieses Problems stammt vom Mathematiker, Astronom und Physiker Carl Friedrich Gauß und sieht wie folgt aus: Die Jahreszahl sei J und J - 1900 sei a. Der Rest von a/19 wird schlicht b genannt. Jetzt wird vom Ausdruck (7*b+1)/19 der ganzzahlige Quotient genommen, der c genannt wird. Mit d wird der Rest von (11*b+4-c)/29 bezeichnet und der Quotient von a/4 mit e. Dann bleibt noch der Rest von (a+e+31-d)/7. Und dieser soll f genannt werden. Daraus folgt, dass für das Osterdatum April die Formel 25 - d - f gilt.

Soll beispielsweise von 2004 der Ostersonntag berechnet werden, so ergeben sich folgende Werte:

```
J = 2004

a = 104

b = REST(104;19) = 9

c = QUOTIENT(7*9+1;19) = 3
```

```
d = REST(11*9+4-3;29) = 13

e = QUOTIENT(104;4) = 26

f = REST(104+26+31-13;7) = 1

Ostern =DATUM(2004;4;25-11-3) = 11.04.2004
```

Analog für das Jahr 2006:

```
a = 106; b = 11; c = 4; d = 5; e = 26; f = 4; Ostern = 16.04.2006

Sub Ostersonntag()

   Dim intJahreszahl As Integer

   Dim a As Integer, b As Integer, c As Integer

   Dim d As Integer, e As Integer, f As Integer


   intJahreszahl = InputBox("Bitte eine Jahreszahl eingeben!")

   a = intJahreszahl - 1900

   b = a Mod 19

   c = (7 * b + 1) \ 19

   d = (11 * b + 4 - c) Mod 29

   e = a \ 4

   f = (a + e + 31 - d) Mod 7

   MsgBox Format(DateSerial(intJahreszahl, 4, 25 - d - f), "dd.mm.yyyy")

End Sub
```

## 1.4.2 Die Konjunktionen in VBA

**Tabelle 1.5 Die Konjunktoren in VBA**

| Wert 1 | Wert 2 | And | Or | Xor | Imp | Eqv |
|---|---|---|---|---|---|---|
| Wahr | Wahr | Wahr | Wahr | Falsch | Wahr | Wahr |
| Wahr | Falsch | Falsch | Wahr | Wahr | Falsch | Falsch |
| Falsch | Wahr | Falsch | Wahr | Wahr | Wahr | Falsch |
| Falsch | Falsch | Falsch | Falsch | Falsch | Wahr | Wahr |
| Wahr | Leer | Falsch | Wahr | Wahr | Falsch | Falsch |
| Falsch | Leer | Falsch | Falsch | Falsch | Wahr | Wahr |
| Leer | Wahr | Falsch | Wahr | Wahr | Wahr | Falsch |
| Leer | Falsch | Falsch | Falsch | Falsch | Wahr | Wahr |

**Hinweis**
Ergebnisse werden dabei immer von der rechten Seite des Gleichheitszeichens auf die linke übergeben.

Während mit dem Vergleichsoperator "=" nur exakte Gleichheit überprüft werden kann, kann mit Like mit Platzhaltern gearbeitet werden. Beispiele:

```
"Struwwelpeter" = "Struwwelpeter"
```

ergibt „True“.

```
"Struwwelpeter" = "Struwelpeter"
```

ergibt dagegen „False“. „True“ liefern folgende drei Vergleiche:

```
"Struwwelpeter" Like "Struwwel*"

"Struwwelpeter" Like "*peter"

"Struwwelpeter" Like "S?ruwwelpe?er"
```

„False“ ist das Ergebnis von folgendem Vergleich:

```
"STRUWWELPETER" Like "Struwwelpeter"
```

Wird dagegen vor den Prozeduren, im allgemeinen Deklarationsteil folgender Befehl eingefügt:

```
Option Compare Text
```

dann wird nicht zwischen Groß- und Kleinschreibung unterschieden. Das bedeutet, dass im obigen Beispiel „True“ das Ergebnis ist. Explizit unterschieden wird zwischen Groß- und Kleinbuchstaben, wenn sich vor der ersten Prozedur folgender Befehl befindet:

```
Option Compare Binary
```

## 1.5 Verzweigungen

### 1.5.1 Verzweigungen I

Die bekannteste (und vielleicht am häufigsten gebrauchte) Verzeigung kann einzeilig

```
If Bedingung Then [Anweisungen] [Else elseAnweisungen]
```

oder im Block auf mehrere Zeilen geschrieben werden:

```
If Bedingung Then
   [Anweisungen]
[ElseIf Bedingung-n Then
[elseifAnweisungen] ...
[Else
[elseAnweisungen]
End If
```

**Hinweis**
Rücken Sie die Anweisungszeilen konsequent ein! Dies erleichtert die Lesbarkeit des Codes.

### 1.5.2 Verzweigungen II

Eine (sehr selten verwendete) Verzweigung hat die folgende Syntax:

```
IIf(expr, truepart, falsepart)
```

Im Gegensatz zur If-Verzweigung stellt `IIF` eine Funktion dar, die einen Wert zurückgibt. `If` kann nach Prüfung Anweisungen ausführen (öffne die Datei, greife auf ein bestimmtes Tabellenblatt zu, lies Werte aus, …) `IIF` dagegen gibt nur einen Wert zurück:

```
strHalbjahr = IIf(intMonat <= 6, "erstes Halbjahr", "zweites Halbjahr")
```

Dies ist sicherlich kürzer als:

```
If intMonat <= 6 Then
   strHalbjahr = "erstes Halbjahr"
Else
   strHalbjahr = "zweites Halbjahr"
End If
```

### 1.5.3 Verzweigungen III

Während IIf sich nur zwischen einer von zwei Auswahlmöglichkeiten entscheiden kann, so kann die Funktion Choose aus einer Reihe von Argumenten auswählen. Die Syntax lautet:

```
Choose(Index, Auswahl-1[, Auswahl-2, ... [, Auswahl-n]])
```

Im folgenden Beispiel wählt die Funktion Auswahl zwischen vier Werten aus, die ihr übergeben wurden:

```
Function Auswahl(i As Integer) As String
   Auswahl = Choose(i, "München", "Hamburg", "Berlin", "Köln")
End Function
```

Auch diese Funktion – analog zu IIF – gibt nur einen Wer zurück und kann keine Anweisungen verarbeiten:

```
strQuartal = Choose(intMonat \ 3, "erstes Quartal", _
   "zweites Quartal", "drittes Quartal", "viertel Quartal")
```

### 1.5.4 Verzweigungen IV

Für sehr viele Fälle eignet sich die übersichtliche Select Case-Schleife:

```
Select Case Testausdruck
   [Case Ausdrucksliste-n
      [Anweisungen-n]] ...
   [Case Else
   [elseAnw]]
End Select
```

Dabei ist zu beachten, dass Vergleichsoperatoren nur mit einem IS-Statement verwendet werden können, beispielsweise:

```
Select Case Variable

Case Is > 1
```

Verschiedene Argumente können durch Kommata getrennt hintereinander geschrieben werden:

```
Case 2, 3, 4
```

Bereiche können mit To zusammengefasst werden:

```
Case 2 To 4
```

**Hinweis**

Häufig können If-Verzweigungen oder Select Case-Statements synonym verwendet werden. Ein Vorteil liegt in Case, wenn mehrere Fälle abgefragt werden:

```
Select Case strOrt

   Case "München", "Hamburg", "Berlin", "Köln"

...

If strOrt = "München" Or strOrt = "Hamburg" Or strOrt = "Berlin" _

   Or strOrt = "Köln" Then

...
```

Bei einem einzigen Fall – wenn Sie Wahrheitswerte abfragen – dann gestaltet sich jedoch eine If-Verzweigung kürzer:

```
Select Case blnDateiVorhanden

   Case False

...

If blnDateiVorhanden = False Then ...
```

Der Benutzer gibt sein Geburtsdatum ein. Daraufhin wird überprüft, ob er Geburtstag hat (oder wie viele Tage bis zu seinem Geburtstag fehlen), an welchem Wochentag er Geburtstag hatte und an welchem Wochentag er im aktuellen Jahr Geburtstag hatte oder haben wird.

**Achtung**

Um das Alter zu bestimmen, dürfen Sie nicht einfach die Jahreszahl des heutigen Datums von der Jahreszahl des Geburtsdatums abziehen. Wenn heute beispielsweise der 20. Oktober 2020 ist, der Benutzer allerdings 10. November 1980 eingibt, so ist er nicht 40 Jahre, sondern nur 39. Auch die Differenz in Tagen geteilt durch 365,25 kann zu Ungenauigkeiten führen.

```
Sub Alter_in_Jahren()

   Dim datGebdatum As Date

   Dim intAlter As Integer

   datGebdatum = InputBox("Wann bist du geboren?")

   intAlter = Year(Date) - Year(datGebdatum)

   If Month(Date) < Month(datGebdatum) Then

      intAlter = intAlter - 1
```

```
    Else
       If Month(Date) = Month(datGebdatum) Then
          If Day(Date) < Day(datGebdatum) Then
             intAlter = intAlter - 1
          End If
       End If
    End If
    MsgBox "Sie sind " & intAlter & " Jahre alt. "
End Sub
```

Oder noch eleganter:

```
Sub Alter_in_Jahren2()
   Dim datGebDatum As Date
   Dim intAlter As Integer
   datGebDatum = InputBox("Wann bist du geboren?")
   intAlter = Year(Date) - Year(datGebDatum)
   If DateSerial(Year(Date), Month(datGebDatum), _
      Day(datGebDatum)) > Date Then
         intAlter = intAlter - 1
   End If
   MsgBox "Sie sind " & intAlter & " Jahre alt. "
End Sub
```

Beide Lösungen ziehen vom aktuellen Jahr das Geburtsjahr ab. Die Differenz muss allerdings noch nicht das Lebensalter sein, denn wenn der Benutzer im laufenden Jahr noch nicht Geburtstag hatte, dann muss von der Differenz 1 abgezogen werden. Das erste Programm überprüft nun Geburtsmonat und Geburtstag, während das zweite Programm aus Geburtstag, Geburtsmonat und aktuellem Jahr ein Datum zusammensetzt und dies mit dem heutigen Tag vergleicht. In beiden Lösungen wird die Zahl 1 subtrahiert.

## 1.6 VBA-Funktionen

VBA stellt eine Reihe von Funktionen zur Verfügung, die in diesem Abschnitt beschrieben werden. Wenn Sie eine Funktion suchen, die VBA zur Verfügung stellt, dann können Sie den Objektkatalog verwenden (Menü Ansicht). Dort werden in der Bibliothek „VBA“ in der Liste der Klassen alle Funktionen aufgelistet.

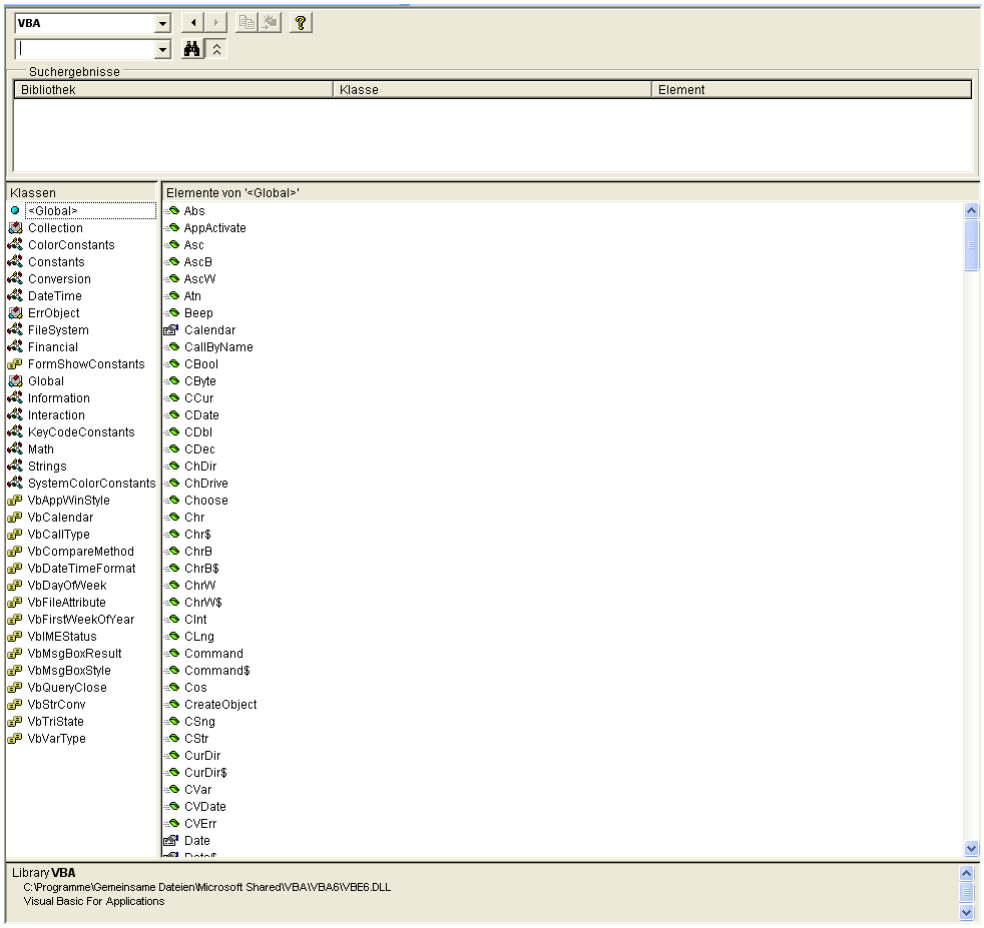



**Abbildung 1.6** Der Objektkatalog

## 1.6.1 Informationsabfragen

**Tabelle 1.6 In der folgenden Liste finden Sie sämtliche Informationsabfragen**

| Funktion | Bedeutung | Beispiel |
|---|---|---|
| IsDate | überprüft, ob es sich um ein Datum handelt. | IsDate(29.2.2007) liefert "False".<br>IsDate(28.2.2007) liefert "True". |
| IsNumeric | überprüft, ob es sich um eine Zahl handelt. | IsNumeric("drei") liefert "False".<br>IsNumeric(3) liefert "True". |
| IsNull | überprüft, ob eine Variable leer ist. | IsNull("drei") liefert "False".<br>IsNull() liefert "True". |
| IsEmpty | überprüft, ob eine Variable initialisiert wurde. | |
| IsArray | überprüft, ob es sich bei einer Variablen um ein Datenfeld handelt. | |
| IsMissing | überprüft, ob Argumente übergeben wurden. | |
| IsObject | überprüft, ob es sich um ein Objekt handelt. | |
| IsError | überprüft, ob es sich um ein Fehlerobjekt handelt. | |

### 1.6.2 Die mathematischen Funktionen

**Tabelle 1.7 Die mathematischen Funktionen:**

| Funktionsname | Bedeutung |
|---|---|
| Sqr | Quadratwurzel |
| Sin, Cos, Tan | Sinus, Cosinus, Tangens |
| Atn | der Arkustangens, die Umkehrfunktion des Tangens |
| Exp | Exponentialfunktion auf Basis e |
| Log | der natürliche Logarithmus zur Basis e |
| Abs | gibt den Absolutwert einer Zahl zurück:<br>3 = Abs(3)<br>3 = Abs(-3) |
| Int | gibt einen Wert zurück, der den gleichen Typ wie der übergebene Wert hat und den ganzzahligen Anteil einer Zahl enthält.<br>8 = Int(8.4) -9 = Int(-8.4) |
| Fix | gibt einen Wert zurück, der den gleichen Typ wie der übergebene Wert hat und den ganzzahligen Anteil einer Zahl enthält.<br>8 = Fix(8,4) -8 = Fix(-8,4) |
| Sgn | das Vorzeichen einer Zahl:<br>Wert von Zahl Rückgabewert von Sgn<br>Größer als null 1<br>Gleich null 0<br>Kleiner als null -1 |
| Round<br>(erst ab Office 2000!) | Rundet eine Zahl auf oder ab.<br>Beispiel: Round(2.4824, 2) ergibt 2,48; Round(2.4824, 1) ergibt 2,5<br>Rnd |
| Randomize | Eine Zufallszahl |

### 1.6.3 Die String-Funktionen

**Tabelle 1.8 Die Liste der wichtigsten String-Funktionen**

| Funktionsname | Bedeutung | Beispiel |
|---|---|---|
| Left$ | schneidet eine bestimmte Anzahl von Zeichen von links ab. | Left$("Hermann Melville", 4) ergibt "Herm" |
| Right$ | schneidet eine bestimmte Anzahl von Zeichen von rechts ab. | Right$(" Hermann Melville", 4) ergibt "ille" |
| Mid$ | schneidet eine bestimmte Anzahl von Zeichen aus der "Mitte" heraus, das heißt ab einer bestimmten Position. | Mid$(" Hermann Melville", 5, 4) ergibt "ann "<br>Mid$(" Hermann Melville", 5) ergibt "ann Melville" |
| InStr | überprüft, ob eine Zeichenfolge innerhalb einer Zeichenkette vorhanden ist, und gibt die Position an. | InStr("Hermann Melville ", "nn") ergibt 6<br>InStr("Hermann Melville", "l") ergibt 11<br>InStr("Hermann Melville", "y") ergibt 0 |
| Ltrim$ | löscht Leerzeichen am Anfang eines Strings. | LTrim("  Hermann Melville  ") ergibt ("Hermann Melville  ") |
| Rtrim$ | löscht Leerzeichen am Ende eines Strings. | RTrim("  Hermann Melville  ") ergibt (" Hermann Melville ") |
| Trim$ | löscht Leerzeichen am Anfang und Ende eines Strings. | Trim("  Hermann Melville  ") ergibt ("Hermann Melville ") |
| Len | ermittelt die Länge einer Zeichenkette. | Len("Hermann Melville") ergibt 16 |
| Chr | wandelt einen ASCII-Code in einen String um. | Chr(13) ergibt ¶ |
| Asc | wandelt einen String in die entsprechende Zahl des ASCII-Codes um. | Asc("A") ergibt 65 |
| Lcase | wandelt eine Zeichenkette in Kleinbuchstaben um. | Lcase("Hermann Melville") ergibt "hermann melville" |
| Ucase | wandelt eine Zeichenkette in Großbuchstaben um. | Ucase("Hermann Melville") ergibt "HERMANN MELVILLE" |
| StrComp | "vergleicht" zwei Zeichenketten, das heißt, es wird überprüft, welche zuerst im Alphabet steht. Ist die erste Zeichenkette "größer" als die zweite, wird -1 zurückgegeben, im umgekehrten Fall 1. Sind beide gleich: 0. Ist einer der beiden Strings leer, so wird null übergeben. | StrComp("Hermann Melville", "Hermann") ergibt 0<br>StrComp("Hermann", "Melville") ergibt -1<br>StrComp("Melville", "Hermann") ergibt 1 |
| Space | gibt eine Folge von Leerzeichen aus. | "Hermann" & Space(4) & "Melville" ergibt "Hermann  Melville" |
| Split | trennt eine Zeichenfolge und liefert einen Array. | Split("Hermann Melville") ergibt „Hermann“ und „Melville“ |

| Funktionsname | Bedeutung | Beispiel |
|---|---|---|
| Join | setzt eine Zeichenfolge zusammen. | |
| Filter | durchsucht eine Zeichenfolge. | |
| InStrRev | überprüft von rechts, ob eine Zeichenfolge in einer anderen vorhanden ist. | InStrRev("Hermann Melville", "y") ergibt 0<br>InStrRev("Hermann Melville", "l") ergibt 15 |

### 1.6.4 Die Uhrzeit- und Datumsfunktionen

**Tabelle 1.9 Die Liste der Datums- und Uhrzeitfunktionen:**

| Funktionsname | Bedeutung |
|---|---|
| Date | setzt das aktuelle Systemdatum ein oder stellt das Systemdatum um. |
| Now | gibt das Systemdatum und die aktuelle Systemzeit zurück. |
| Timer | gibt einen Wert vom Typ Single zurück, der die Anzahl der seit Mitternacht vergangenen Sekunden angibt. Diese Funktion wird verwendet, wenn Zeitdifferenzen berechnet werden sollen. |
| Time | setzt die aktuelle Systemzeit ein oder stellt die Systemzeit um. |
| DateSerial | gibt die fortlaufende Datumszahl eines Datums zurück. |
| DateAdd | addiert oder subtrahiert ein angegebenes Intervall zu einem oder von einem Datum. |
| DateDiff | gibt die Anzahl der Zeitintervalle zurück, die zwischen zwei Datumsangaben liegen |
| DatePart | berechnet, zu welchem Teil eines angegebenen Intervalls ein Datum gehört. |
| Day | filtert den Tag aus einem Datum. |
| Month | filtert den Monat aus einem Datum. |
| Year | filtert das Jahr aus einem Datum. |
| Weekday | gibt eine Zahl zwischen 1 und 7 zurück, die dem Wochentag entspricht. |
| Hour | filtert die Stunde aus einer Uhrzeit. |
| Minute | filtert die Minutenanzahl aus einer Uhrzeit. |
| Second | filtert die Sekundenanzahl aus einer Uhrzeit. |

**Beispiel:**
Im folgenden Beispiel wird aus einem Geburtsdatum berechnet, wie viele Tage bis heute vergangen sind oder viele Tage in diesem Jahr bis zum nächsten Geburtstag fehlen.

```
Sub Geburtstag()

   Dim intTag As Integer

   Dim intMonat As Integer

   Dim datGebDatum As Date

   Dim intAnzahlTage As Integer

   Dim strZukunft As String

   datGebDatum = InputBox("Wie lautet Ihr Geburtsdatum?")

   intTag = Day(datGebDatum)
```

```
    intMonat = Month(datGebDatum)


    If intTag = Day(Date) And intMonat = Month(Date) Then
       MsgBox "Happy Birthday"
       strZukunft = "Heute ist "
    Else
       intAnzahlTage = DateDiff("D", _
          Date, DateSerial(Year(Date) + 1, intMonat, intTag))
       If intAnzahlTage >= 365 Then
          intAnzahlTage = DateDiff("D", _
             Date, DateSerial(Year(Date), intMonat, intTag))
             strZukunft = " Geburtstag haben."
       End If
       MsgBox "Sorry, Sie haben erst in " & intAnzahlTage & _
       " Tagen Geburtstag."
    End If
    MsgBox "Sie sind an einem " & _
       Format(datGebDatum, "DDDD") & " geboren."
    If strZukunft = " Geburtstag haben." Then
       MsgBox "Sie werden in diesem Jahr an einem " & _
          Format(DateSerial(Year(Date), intMonat, intTag), _
          "DDDD") & strZukunft
    ElseIf strZukunft = "Heute ist " Then
       MsgBox "Heute ist Ihr Geburtstag." & _
          vbCr & strZukunft & Format(Date, "DDDD")
    Else
       MsgBox "Sie hatten in diesem Jahr an einem " & _
          Format(DateSerial(Year(Date), intMonat, intTag), _
          "DDDD") & " Geburtstag."
    End If


End Sub
```

### 1.6.5 Die Funktion Format

Ihre Syntax lautet:

```
Format(Ausdruck[,Format[,firstdayofweek[,firstweekofyear]]])
```

### 1.6.6 Umwandlungsfunktionen

**Tabelle 1.10 Umwandlungsfunktionen**

| Funktion | Rückgabetyp |
|---|---|
| CBool | Boolean |
| CByte | Byte |
| CCur | Currency |
| CDate | Date |
| CDbl | Double |
| CDec | Decimal |
| CInt | Integer |
| CLng | Long |
| CSng | Single |
| CVar | Variant |
| CStr | String |

**Beispiel:**
Im nächsten Beispiel gibt der Benutzer seinen Vor- und Nachnamen ein. Diese Zeichenkette wird in Vorname und Nachname zerlegt.

```
Sub NamenZerlegen1()
   Dim strGanzName As String
   Dim strVorname As String
   Dim strNachname As String

   strGanzName = InputBox("Wie heißen Sie?", "Vor- und Zuname")
   If strGanzName = "" Then
      MsgBox "Sie haben nichts eingegeben!"
   ElseIf InStr(strGanzName, " ") = 0 Then
      MsgBox "Sie haben nur einen Namen eingegeben!"
      Exit Sub
   End If
   strVorname = Left$(strGanzName, InStr(strGanzName, " ") - 1)
   strNachname = Right$(strGanzName, Len(strGanzName) - _
      InStr(strGanzName, " "))
   MsgBox "Der Vorname lautet: " & strVorname & _
      " und der Nachname lautet: " & strNachname
End Sub
```

Der Zuname kann auch anders herausgelöst werden:

```
strNachname = Right$(strGanzName, _
   Len(strGanzName) - Len(strVorname))
```

Oder so:

```
strNachname = Mid$(strGanzName, InStr(strGanzName, " ") + 1)
```

VBA 6.0 (ab Office 2000) stellt neue String-Funktionen zur Verfügung: `Split`, `Join`, `Filter` und `InStrRev`. Mit der Funktion `Split` kann die Aufgabe ebenso gelöst werden:

```
Sub NamenZerlegen3()
   Dim strTeilnamen() As String
   Dim strGanzName As String
   strGanzName = InputBox("Wie heißen Sie?", "Vor- und Zuname")
   If strGanzName = "" Then
      MsgBox "Sie haben nichts eingegeben!"
      Exit Sub
   End If
   strTeilnamen = Split(strGanzName)
   If strTeilnamen(UBound(strTeilnamen)) = _
      strTeilnamen(0) Then
      MsgBox "Der Vorname lautet: " & strTeilnamen(0)
   Else
      MsgBox "Der Vorname lautet: " & strTeilnamen(0) & _
         " und der Nachname lautet: " & _
         strTeilnamen(UBound(strTeilnamen))
   End If
End Sub
```

Umgekehrt setzt die Funktion `Join` einen String wieder zusammen, wobei sie ebenfalls einen Array verlangt. Im folgenden Beispiel werden die einzelnen Komponenten wieder zusammengefügt:

```
Sub NamenZerlegenUndZusammensetzen()
   Dim strTeilnamen() As String
   Dim i As Integer
   Dim strGanzName As String
   strGanzName = InputBox("Wie heißen Sie?", "Alle Namen")
   strTeilnamen = Split(strGanzName)
   For i = 0 To UBound(strTeilnamen)
      strGanzName = Join(strTeilnamen, "")
   Next
```

```
   MsgBox strGanzName
End Sub


Sub DateiNamenZerlegen1()
   Dim strDateiName As String
   Dim i As Integer

   strDateiName = ActiveWorkbook.FullName
   MsgBox strDateiName

   For i = Len(strDateiName) To 1 Step -1
      If Mid(strDateiName, i, 1) = "\" Then
         strDateiName = Right(strDateiName, _
            Len(strDateiName) - i)
         Exit For
      End If
   Next i
   strDateiName = Left(strDateiName, _
      InStr(strDateiName, ".") - 1)
   MsgBox strDateiName
End Sub
```

Man kann mit `InStrRev` (ab Office 2000) direkt die Position von rechts bestimmen:

```
strDateiName = Right(strDateiName, InStrRev(strDateiName, " ") - 1)
```

Oder den Text ohne Schleife in ein Datenfeld zerlegen. Dies sieht folgendermaßen aus:

```
Sub DateiNamenZerlegen2()
   Dim strDateiName As String
   Dim strTeilNamen() As String
   strDateiName = ActiveWorkbook.FullName
   MsgBox strDateiName
   strTeilNamen = Split(strDateiName, "\")
   strDateiName = strTeilNamen(UBound(strTeilNamen))
   strDateiName = Left(strDateiName, _
      InStr(strDateiName, ".") - 1)
   MsgBox strDateiName
End Sub
```

Der Anwender gibt in ein Eingabefeld eine Funktion der Form

```
y=log(1/x)^x
```

oder

```
z=sin(x)*cos(y)
```

ein. Folgende Operatoren sind erlaubt:

```
+, -, *, / und ^
```

Neben den Variablen x, y und z und den Klammern sind die Funktionen Sin, Cos, Tan, Arcsin, Arccos, Arctan, Sinhyp, Coshyp, Tanhyp, Arcsinhyp, Arccoshyp, Arctanhyp, Exp, Ln, Log10, Abs, Fakultät und Wurzel zugelassen. Sonst nichts. Aus ihr wird in Excel ein Diagramm erzeugt. Einer Funktion, die den eingegebenen Funktionsnamen testet, wird auch die Dimension (zwei oder drei) übergeben und der Anfangswert, der berechnet werden soll:

```
Public Function Parser(Funktion As String, Dimension As Byte, _
   AnfangsWert As Double, Optional AnfangsWert_Y As Double) As String
   Dim strZeichen As String
   Dim i As Integer
   Dim j As Integer
   Parser = ""
```

Die eingegebene Funktion wird in Kleinbuchstaben verwandelt:

```
   Funktion = LCase(Funktion)
```

Zuerst wird überprüft, ob die Funktion mit "y=" oder "z=" beginnt:

```
   If Left(Funktion, 1) <> "y" And Dimension = 2 Then
      Parser = "Bitte beginnen Sie die Eingabe mit ""y"""
      Exit Function
   ElseIf Left(Funktion, 1) <> "z" And Dimension = 3 Then
      Parser = "Bitte beginnen Sie die Eingabe mit ""z"""
      Exit Function
   ElseIf Mid(Funktion, 2, 1) <> "=" Then
      Parser = "Nach dem ""y"" muss ein Gleichheitszeichen stehen"
      Exit Function
   ElseIf InStr(1, Funktion, "#") > 0 Then
      Parser = "Bitte nur gültige Zeichen eingeben!"
   Else
      Funktion = Mid(Funktion, 3)

      j = 0
```

Anschließend wird überprüft, ob die Anzahl der geöffneten Klammern mit der Anzahl der geschlossenen Klammern übereinstimmt:

```
For i = 1 To Len(Funktion)
   If Mid(Funktion, i, 1) = "(" Then
      j = j + 1
   End If
Next i
For i = 1 To Len(Funktion)
   If Mid(Funktion, i, 1) = ")" Then
      j = j - 1
   End If
Next i
If j <> 0 Then
   Parser = "Falsche Anzahl Klammern!"
   Exit Function
End If
```

Dann wird getestet, ob ungültige Zeichen eingegeben wurden:

```
For i = 1 To Len(Funktion)
   If Asc(Mid(Funktion, i, 1)) < Asc("(") Or _
      Asc(Mid(Funktion, i, 1)) < Asc("u") Then
      Parser = "Sie haben ein falsches Zeichen eingegeben!"
      Exit Function
   End If
Next i
```

Da Excel mehr Funktionen kennt als VBA, werden die eingegebenen Funktionen, die in Excel benötigt werden, in VBA-Funktionen verwandelt, die existieren, nämlich in sin, cos, tan, atn, log, sgn und sqr:

```
Funktion = Replace(Funktion, "exp", "sin")

Funktion = Replace(Funktion, "x", CStr(AnfangsWert))

Funktion = Replace(Funktion, "arcsinhyp", "sin")
Funktion = Replace(Funktion, "arccoshyp", "cos")
Funktion = Replace(Funktion, "arctanhyp", "tan")

Funktion = Replace(Funktion, "arctan", "atn")
```

```
        Funktion = Replace(Funktion, "arccos", "atn")
        Funktion = Replace(Funktion, "arcsin", "atn")

        Funktion = Replace(Funktion, "sinhyp", "sin")
        Funktion = Replace(Funktion, "coshyp", "cos")
        Funktion = Replace(Funktion, "tanhyp", "tan")

        Funktion = Replace(Funktion, "log10", "log")
        Funktion = Replace(Funktion, "ln", "log")

        Funktion = Replace(Funktion, "fakultät", "sgn")
        Funktion = Replace(Funktion, "abs", "sgn")

        Funktion = Replace(Funktion, "wurzel", "sqr")

        If Dimension = 3 Then
           Funktion = Replace(Funktion, "y", CStr(AnfangsWert_Y))
        End If
```

Da VBA als Dezimaltrennzeichen einen Punkt und kein Komma akzeptiert, wird auch geändert:

```
        Funktion = Replace(Funktion, ",", ".")
```

Mit der Excel-Methode Evaluate kann nun überprüft werden, ob die so konstruierte Funktion korrekt ist:

```
        On Error Resume Next
        i = Application.Evaluate(Funktion)
        If Err.Number <> 0 Then
           If Err.Number = 13 Then
              Parser = "Die von Ihnen eingegebene Funktion ist falsch" & _
                 vbCr & "oder der Startwert konnte nicht berechnet werden!"
           Else
              Parser = Err.Description
           End If
           Exit Function
        End If
     End If
  End Function
```

Diese Funktion wird für die automatisierte Erzeugung von Diagrammen verwendet. Dort gibt der Benutzer in ein Eingabefeld eine Funktion ein, die in Excel grafisch dargestellt wird. Um die Eingabe zu überprüfen, wird die oben beschriebene Funktion verwendet.

## 1.7 Selbst erzeugte Funktionen, Aufrufe und Parameterübergabe

Häufig werden Prozeduren an verschiedenen Stellen eines Programms verwendet. Deshalb können sie ausgelagert und von beliebigen Punkten aus aufgerufen werden.

### 1.7.1 Aufruf

Wenn eine Prozedur eine andere aufruft, dann kann dies mit dem Schlüsselwort `Call` geschehen oder einfach, indem der Name der zweiten Prozedur auftaucht. Danach wird die erste Prozedur an der Stelle weitergeführt, an der die zweite aufgerufen wurde, beispielsweise so:

```
Sub A()

   Call B

End Sub

Sub B()

   MsgBox "Nun ist Ende"

End Sub
```

Zu dem Namen der Prozedur bei der Anweisung Call B kann auch der Name des Moduls hinzugefügt werden, beispielsweise:

```
Sub A()

   Call Modul1.B

End Sub
```

Alle Lösungen funktionieren nur, wenn B Public deklariert wurde. Die Prozedur wird also nicht in einem anderen Modul gefunden, wenn B folgendermaßen aussieht:

```
Private Sub B()

End Sub
```

Im folgenden Beispiel ruft das zweite Makro das erste auf. Dieses startet nach fünf Minuten das zweite und fordert den Benutzer auf nicht einzuschlafen:

```
Public Sub NichtEinschlafen1()

   Application.OnTime Now + TimeSerial(0, 0, 2), "NichtEinschlafen2"

End Sub

Public Sub NichtEinschlafen2()

   Beep

   MsgBox "Bitte nicht einschlafen!"

   Call NichtEinschlafen1

End Sub
```

Das Makro muss natürlich noch über gestartet werden, beispielsweise beim Öffnen der Excelmappe:

```
Private Sub Workbook_Open()
   Application.OnTime Now + TimeSerial(0, 0, 2), "NichtEinschlafen1"
End Sub
```

Übrigens beenden Sie den Scherz mit der Tastenkombination <Strg>+<Pause> oder über das Symbol „Zurücksetzen“: 

**Hinweis**

Beachten Sie, dass Sie Prozeduren einen Wert übergeben können. Wenn Sie mit dem Schlüsselwort Call arbeiten und mehrere Werte übergeben, dann müssen Sie die Klammer verwenden:

```
Call NichtEinschafen(0, 10, 0)
```

Ohne `Call` dürfen Sie keine Klammer verwenden:

```
NichtEinschafen 0, 10, 0
```

Als Alternative eignet sich die Schreibweise mit den Parametern, die eine beliebige Reihenfolge erlaubt:

```
   Call NichtEinschlafen(Stunden:=0, Minuten:=10, Sekunden:=0)
   NichtEinschlafen Stunden:=0, Minuten:=10, Sekunden:=0
End Sub
Sub NichtEinschlafen(Stunden As Integer, _
   Minuten As Integer, Sekunden As Integer)
```

### 1.7.2 Globale Variablen

Wird eine Variable in mehreren Prozeduren verwendet, dann wird der Wert „global“, das heißt „modulweit“ oder „projektweit“, deklariert. Dies ist vor allem in den Ereignisprozeduren der Dialoge sehr wichtig.

Im folgenden Beispiel wird der Inhalt der Variablen `strAnzeigeText` in den beiden Prozeduren A und B, die sich im gleichen Modul befinden, verwendet.

```
Dim strAnzeigeText As String
Sub A2()
   strAnzeigeText = "Nun ist Ende"
   Call B
End Sub

Sub B2()
   MsgBox strAnzeigeText
End Sub
```

### 1.7.3 Übergabe

Soll dagegen eine Variable explizit übergeben werden, dann kann die erste Prozedur (A) diese Werte auf zweierlei Weise an Prozedur B übergeben: Werden die alten Werte in der Prozedur noch verwendet, dann geschieht die Übergabe „Call by Value“, das heißt, die alten Werte sind in der Prozedur A noch vorhanden (es werden die Werte direkt übergeben und nicht einen Verweis auf den Speicher). Wird dagegen Prozedur B mit „Call by Reference“ aufgerufen, dann wird der veränderte Werte in A verwendet. Die Prozedur B könnte so aussehen:

```
Sub B(ByRef X As Integer, ByVal Y As Integer)


End Sub
```

Im ersten Fall werden die Werte 1 und 2 von A an B übergeben. Dort wird zu ihnen ein Wert addiert – und so werden sie am Ende der Prozedur A3 angezeigt (als 3 und 4):

```
Sub A3()
   Dim X As Integer, Y As Integer
   X = 1: Y = 2
   Call B3(X, Y)
   MsgBox "x: " & X & " y: " & Y
End Sub


Sub B3(ByRef X As Integer, ByRef Y As Integer)
   X = X + 2: Y = Y + 2
End Sub
```

Anders dagegen in folgendem Fall: Die Werte werden „By Value“ übergeben, das heißt, bei der Rückgabe halten sich die alten Werte. In Prozedur A werden also 1 und 2 angezeigt.

```
Sub A4()
   Dim X As Integer, Y As Integer
   X = 1: Y = 2
   Call B4(X, Y)
   MsgBox "x: " & X & " y: " & Y
End Sub


Sub B4(ByVal X As Integer, ByVal Y As Integer)
   X = X + 2: Y = Y + 2
End Sub
```

**Hinweis**

Wird nicht nur der Inhalt einer Variablen, sondern eines Arrays übergeben, dann kann dies mit dem Befehl ParamArray geschehen. Im folgenden Beispiel werden drei Werte übergeben, wo sie in einem Datenfeld gespeichert und anschließend weiter verarbeitet werden:

```
Sub A5()
   Dim X As Integer, Y As Integer, Z As Integer
   X = 1: Y = 2: Z = 3
   Call B5(X, Y, Z)
End Sub

Sub B5(ParamArray Werte())
   Dim intErgebnis As Integer
   Dim i As Integer
   For i = LBound(Werte) To UBound(Werte)
      intErgebnis = intErgebnis + Werte(i) ^ 2
   Next
   MsgBox intErgebnis
End Sub
```

Das Ergebnis, das in B angezeigt wird, lautet 14.

Diese Technik wird verwendet, wenn die Anzahl der übergebenen Werte nicht bekannt ist, beispielsweise bei benutzerdefinierten Funktionen in Excel. Die Funktion SUMME addiert sämtliche Werte des markierten Bereichs. Eine benutzerdefinierte Funktion MITTELWERT_OHNE0 berechnet das Ergebnis aus allen Werten, die übergeben werden. Die Anzahl der Werte ist dabei unterschiedlich.

Wird eine Prozedur verwendet, um einen Wert zu übergeben und einen (nur einen!) Wert zurückzugeben, dann kann (und sollte) sie als Funktion geschrieben werden:

```
Sub A6()
   Dim X As Integer, Y As Integer, Z As Integer
   X = 1: Y = 2
   Z = B6(X, Y)
   MsgBox "Z: " & Z
End Sub
Function B6(X As Integer, Y As Integer) As Integer
   B6 = X + Y
End Function
```

Da der Wert der Funktion B6 direkt zurückgegeben wird, wird sie immer als „By Reference“ aufgerufen. Somit ist die genaue Spezifizierung überflüssig!

## 1.8 Schleifen, rekursives Programmieren

Schleifen haben die Aufgabe, eine oder mehrere Anweisungen mehrmals zu wiederholen. Dabei stehen Ihnen zwei verschiedene Arten zur Verfügung. Bei einer Zählerschleife (oder unbedingten Schleife) wird eine feste Anzahl von Schritten durchlaufen. In der Regel weiß man zu Beginn schon, wie oft diese Schleife durchlaufen

werden soll. In Bedingungsschleifen wird die Wiederholung so lange ausgeführt, bis die Bedingung erfüllt ist oder solange die Bedingung erfüllt ist.

**Achtung**
Beim Testen können leicht Endlosschleifen entstehen. Um eine solche zu unterbrechen, kann das Programm mit Hilfe der Tastenkombination <STRG>+<PAUSE> angehalten werden!

### 1.8.1 Zählerschleifen

Die Zählerschleife For ... Next besitzt folgenden Aufbau:

```
For Zähler = Anfang To Ende [Step Schritt]
   [Anweisungen]
   [Exit For]
   [Anweisungen]
Next [Zähler]
```

**Tabelle 1.11 Die Syntax für die For ... Next-Anweisung besteht aus folgenden Teilen:**

| Teil | Beschreibung |
|---|---|
| Zähler | ist eine numerische Variable, die als Schleifenzähler dient. |
| Anfang | der Startwert von Zähler |
| Ende | der Endwert von Zähler |
| Schritt | ist optional. Es ist ein Betrag, um den der Zähler bei jedem Schleifendurchlauf verändert wird. Falls kein Wert angegeben wird, ist die Voreinstellung für „Schritt“ eins. „Schritt“ lässt ganze (positive und negative) Werte und Dezimalzahlen zu. |
| Exit For | Mit diesem Befehl kann die Schleife vorzeitig beendet werden. |

VBA stellt für Objektsammlungen eine spezielle Form der Zählerschleife zur Verfügung: die For Each ... Next-Schleife. Die Syntax lautet:

```
For Each Element In Sammlung
   [Anweisungen]
   [Exit For]
   [Anweisungen]
Next [Element]
```

### 1.8.2 Bedingungsschleifen

Die Bedingung kann entweder am Anfang oder Ende einer Bedingungsschleife stehen. Man spricht auch von kopf- und fußgesteuerten Schleifen. Steht die Bedingung am Ende, dann wird die Schleife mindestens einmal durchlaufen. Steht sie am Kopf, so wird überprüft, ob die Schleife überhaupt betreten werden darf. Die Syntax lautet:

```
Do [{While | Until} Bedingung]
   [Anweisungen]
```

```
    [Exit Do]
    [Anweisungen]
Loop
```

Oder:

```
Do
    [Anweisungen]
    [Exit Do]
    [Anweisungen]
Loop [{While | Until} Bedingung]
```

**Beispiel**

Ein Anfangskapital wird zu 7,5% verzinst. Wie viele Jahre benötigt man, damit es sich verdoppelt hat? Man kann diese Aufgabe mit allen vier Möglichkeiten berechnen lassen. Der Vollständigkeit halber sollen sie angezeigt werden:

### Variante 1:

```
Sub KapitalVerdoppeln1()
    Dim curKapital As Currency
    Dim curZielKapital As Currency
    Dim intJahre As Integer

    curKapital = InputBox("Wie viel möchten Sie anlegen?")
    curZielKapital = curKapital * 2

    Do Until curKapital >= curZielKapital
        curKapital = curKapital * 1.075
        intJahre = intJahre + 1
    Loop

    MsgBox intJahre & " Jahre werden zum Verdoppeln benötigt."
End Sub
```

**Variante 2:**

```
Sub KapitalVerdoppeln2()
   Dim curKapital As Currency
   Dim curZielKapital As Currency
   Dim intJahre As Integer

   curKapital = InputBox("Wie viel möchten Sie anlegen?")
   curZielKapital = curKapital * 2

   Do While curKapital < curZielKapital
      curKapital = curKapital * 1.075
      intJahre = intJahre + 1
   Loop

   MsgBox intJahre & " Jahre werden zum Verdoppeln benötigt."
End Sub
```

**Variante 3:**

```
Sub KapitalVerdoppeln3()
   Dim curKapital As Currency
   Dim curZielKapital As Currency
   Dim intJahre As Integer

   curKapital = InputBox("Wie viel möchten Sie anlegen?")
   curZielKapital = curKapital * 2

   Do
      curKapital = curKapital * 1.075
      intJahre = intJahre + 1
   Loop While curKapital < curZielKapital

   MsgBox intJahre & " Jahre werden zum Verdoppeln benötigt."

End Sub
```

**Variante 4:**

```
Sub KapitalVerdoppeln4()
   Dim curKapital As Currency
   Dim curZielKapital As Currency
   Dim intJahre As Integer

   curKapital = InputBox("Wie viel möchten Sie anlegen?")
   curZielKapital = curKapital * 2

   Do
      curKapital = curKapital * 1.075
      intJahre = intJahre + 1
   Loop Until curKapital >= curZielKapital

   MsgBox intJahre & " Jahre werden zum Verdoppeln benötigt."
End Sub
```

Da Currency auf vier Nachkommastellen genau rechnet, sollte man korrekterweise noch überprüfe, ob die eingegebene Zahl mindesten 0,001 groß ist, da es sonst zu einem Fehlerwert 6: Überlauf oder zum Ergebnis 0 Jahre kommt.

**Hinweis**

Finanzmathematisch ist dieses Beispiel Unsinn. Wird ein Kapital x zu einem Prozentsatz von 7,5% Zinsen angelegt, dann beträgt das Kapital nach n Jahren:

$$K_{Ende} = K_{Start} \times (1 + p)^n$$

Setzt man nun $K_{Ende} = K_{Start} * 2$, dann folgt aus der Formel:

$2 = (1+p)^n$

oder:

$n = \log_{(1+p)}2 = lg(2)/lg(1+p)$

Für p = 0,075 ergibt sich aus der Formel: n ≈ 9,584. Das heißt, dass sich das Kapital bei 7,5% Zinsen nach zehn Jahren verdoppelt hat – unabhängig vom ursprünglichen Kapital.

Daneben findet sich noch (aus historischen Gründen) die While ... Wend-Schleife. Auf sie soll an dieser Stelle nicht eingegangen werden: Sie ist nicht so flexibel wie die Do ... Loop-Schleifen. Es gibt keinen Fall, in dem sie Vorteile gegenüber den anderen Schleifen hätte.

### 1.8.3 Rekursionen

Eine besondere Art der Schleifen sind Funktionen und Prozeduren, die sich selbst aufrufen. Damit sie sich nicht unendlich oft aufrufen, stehen diese Aufrufe in der Regel in einer Verzweigung, das heißt einer Bedingung. Das Standardbeispiel für Rekursionen ist das Berechnen der Fakultät. Die Fakultät einer Zahl ist das Produkt aller Zahlen (ab 1) bis zu der Zahl. Die Fakultät von 5 lautet 120, da

```
5! = 1*2*3*4*5
```

oder allgemein:

```
n! = n*(n-1)*(n-2)* ... * 2 * 1
```

Dies kann über eine Zählerschleife berechnet werden:

```
Sub Fakultät1()
   Dim intZahl As Integer
   Dim dblFakultät As Double
   Dim intZähler As Integer

   intZahl = InputBox("Bitte eine Zahl zur Fakultätsberechnung")
   dblFakultät = 1
   For intZähler = 1 To intZahl
      dblFakultät = dblFakultät * intZähler
   Next
   MsgBox "Die Fakultät von " & intZahl & _
      " lautet: " & Format(dblFakultät, "#,##0")
End Sub
```

Oder mit einer Bedingungsschleife:

```
Sub Fakultät2()
   Dim intZahl As Integer
   Dim dblFakultät As Double

   intZahl = InputBox("Bitte eine Zahl zur Fakultätsberechnung")
   dblFakultät = 1
   Do While intZahl > 1
      dblFakultät = dblFakultät * intZahl
      intZahl = intZahl - 1
   Loop

   MsgBox "Die Fakultät lautet: " & Format(dblFakultät, "#,##0")
End Sub
```

Oder schließlich rekursiv. Die Funktion Fakultät3 wird so lange aufgerufen, bis `intZahl` den Wert 1 erreicht hat (beim Herabzählen).

```
Function Fakultät3(intZahl As Integer) As Double
   If intZahl > 1 Then
      Fakultät3 = intZahl * Fakultät3(intZahl - 1)
   Else
```

```
        Fakultät3 = 1
    End If
End Function
```

Dies kann mit folgender Prozedur aufgerufen werden:

```
Sub FakultätBerechnen()
    Dim intZahl As Integer
    Dim dblFakultät As Double
    intZahl = InputBox("Bitte eine Zahl zur Fakultätsberechnung")
    dblFakultät = Fakultät3(intZahl)
    MsgBox "Die Fakultät von " & intZahl & _
        " lautet: " & Format(dblFakultät, "#,##0")
End Sub
```

Oder auch mit einer aufsteigenden Zählung:

```
    Dim intZahl As Integer
    Dim intEndZahl As Integer
Function Fakultät4(intZahl As Integer) As Double

    Fakultät4 = 1

    If intZahl <= intEndZahl Then
        Fakultät4 = intZahl * Fakultät4(intZahl + 1)
    End If
End Function



Sub FakultätBerechnen2()
    Dim intZahl As Integer
    intZahl = InputBox("Bitte eine Zahl zur Fakultätsberechnung")

    intEndZahl = intZahl
    dblFakultät = Fakultät4(1)

    MsgBox "Die Fakultät von " & intZahl & _
        " lautet: " & Format(dblFakultät, "#,##0")
End Sub
```

Zugegeben: Die erste Variante ist natürlich eleganter!

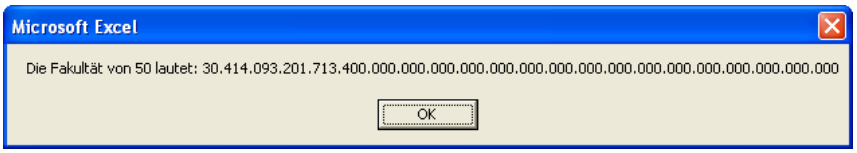



**Abbildung 1.7** 50!

**Beispiel**

Eine Schnecke sitzt vor einer 4,5 Meter hohen Mauer. Jeden Tag klettert sie 50 cm nach oben, in jeder Nacht rutscht sie 10% ihrer Gesamthöhe nach unten. Wann ist sie oben?

Also: Sie beginnt bei 0 m, klettert am ersten Tag auf 0,5 m, rutscht um 0,05 m nach unten, so dass sie den zweiten Tag bei 0,45 m beginnt. Von dort klettert sie auf 0,95 m, um in der folgenden Nacht auf 0,95 m - 0,095 m = 0,855 m zu landen. Dann klettert sie auf 1,355 m und rutscht auf 1,2195 m herunter. Und so weiter. Die Rekursion lässt sich gut in eine Schleife einbauen. Versuchen Sie verschiedene Varianten!

In dieser Aufgabe steckt ein umgangssprachliches Problem! Wenn man sagt, dass man von einer Zahl 10% abzieht, so darf nicht x - 10% gerechnet werden, sondern x - x*10%. Andererseits interessieren nicht die 10%, sondern die übrigen 90%. Und da VBA nicht mit Prozentwerten rechnen kann, so kann die Nachthöhe mit 0.9 multipliziert werden.

```
Sub Schneckenproblem1()

   Dim intTage As Integer

   Dim dblMorgenHöhe As Double, dblAbendhöhe As Double


   dblMorgenHöhe = 0

   intTage = 0


   Do While dblAbendhöhe < 4.5

      intTage = intTage + 1

      dblAbendhöhe = dblMorgenHöhe + 0.5

      dblMorgenHöhe = dblAbendhöhe * 0.9

   Loop

   MsgBox "Die Schnecke benötigt " & intTage & " Tage."

End Sub
```

Analog kann die Schleife auch begonnen werden:

```
Do Until dblAbendhöhe >= 4.5
```

Oder man formuliert die Bedingung am Schleifenrumpf:

```
Sub Schneckenproblem2()

   Dim intTage As Integer

   Dim dblMorgenHöhe As Double, dblAbendhöhe As Double
```

```
    dblMorgenHöhe = 0

    intTage = 0

    Do

       intTage = intTage + 1

       dblAbendhöhe = dblMorgenHöhe + 0.5

       dblMorgenHöhe = dblAbendhöhe * 0.9

    Loop Until dblAbendhöhe >= 4.5


    MsgBox "Die Schnecke benötigt " & intTage & " Tage."

End Sub
```

Oder analog:

```
Loop While dblAbendhöhe < 4.5
```

Horst wird gefragt, wie alt seine vier Kinder sind. Er gibt als Antwort: „Das Produkt ihrer Alter beträgt 1536, die Summe 30.“ Er überlegt und fügt hinzu: „Die Jüngste heißt Claudia.“ Wie alt sind seine Kinder?

Man kann diese Aufgabe recht einfach durch mehrere ineinander verschachtelte Schleifen lösen. Die erste Schleife läuft vom Produkt 1536 bis 1 herunter. Die zweite Schleife nur noch von der Zahl der ersten Schleife bis 1, die dritte von der Zahl der zweiten Schleife und so weiter. In jeder Schleife wird überprüft, ob der Schleifenzähler ein Teiler von 1536 ist. Falls ja, dann läuft die nächste Schleife los. Im Inneren der vierten Schleife werden die Summe und das Produkt der vier Zahlen gezogen. Beträgt die Summe 30 und das Produkt 1536, dann ist eine Lösung gefunden. Für diese Aufgabe existieren zwei Lösungen: (16/6/4/4) und (12/8/8/2). Da es eine jüngste Tochter gibt, muss die zweite Lösung die korrekte sein. Dies kann übrigens auch in die (innere) Schleife eingebaut werden.

Sehr viel schneller läuft die Schleife, wenn Sie die Zählung von unten beginnen und nach oben (maximal 30) hochzählen lassen und erst in der vierten Schleife überprüfen, ob die vier Lebensalter die Bedingungen Produkt und Summe erfüllen:

```
Sub KindAlterBerechnen()

    Const KPRODUKT As Integer = 1536

    Const KSUMME  As Integer = 30

    Dim intAlter1 As Integer

    Dim intAlter2 As Integer

    Dim intAlter3 As Integer

    Dim intAlter4 As Integer


    For intAlter1 = 1 To KSumme

       For intAlter2 = intAlter1 + 1 To KSumme

          For intAlter3 = intAlter2 To 30

             For intAlter4 = intAlter3 To 30

                If intAlter1 * intAlter2 * intAlter3 * intAlter4 = _

                   KProdukt And intAlter1 + intAlter2 + intAlter3 + _
```

```
                  intAlter4 = 30 Then
                     MsgBox "Horsts Kinder haben " & _
                     "folgendes Alter:" _
                     & vbCr & intAlter1 & " / " & _
                     intAlter2 & " / " & intAlter3 & _
                     " / " & intAlter4
                  Exit Sub
                  End If
               Next
            Next
         Next
      End Sub
```

**Beispiel**

Eine Benutzerin oder ein Benutzer wird nach ihrem oder seinem Geschlecht gefragt. Nun soll diese Frage penetrant so lange wiederholt werden, bis sie oder er ein korrektes „w“ oder „m“ eingibt.

```
Sub Begrüßung1()
   Dim strName As String
   Dim strGeschlecht As String
   Do
      strGeschlecht = InputBox("Wie lautet Ihr Geschlecht?" & _
         vbCr & "Bitte ""m"" oder ""w"" eingeben!", "Name")
      If strGeschlecht = "w" Or strGeschlecht = "W" Then
         strName = InputBox("Wie lautet Ihr Name?", "Name")
         MsgBox "Hallo, liebe " & strName
      ElseIf LCase(strGeschlecht) = "m" Then
         strName = InputBox("Wie lautet Ihr Name?", "Name")
         MsgBox "Hallo, lieber " & strName
      End If
   Loop Until LCase(strGeschlecht) = "w" Or _
      LCase(strGeschlecht) = "m"
End Sub
```

Die Loop-Schleife kann auch anders formuliert werden:

```
   Loop While LCase(strGeschlecht) <> "w" And _
      LCase(strGeschlecht) <> "m"
```

**Beispiel**

Der Benutzer gibt eine Zahl ein, die daraufhin überprüft wird, ob es sich um eine Primzahl handelt oder nicht.

Eine Primzahl ist eine Zahl, die nur durch 1 und durch sich selbst teilbar ist. Also muss man von allen Zahlen zwischen einschließlich 2 und der Zahl -1 überprüfen, ob sie ein Teiler der Zahl sind. Es genügt sogar bei √(Zahl) aufzuhören. Hätte beispielsweise die Zahl 25 einen Teiler < 5, dann hätte sie auch einen (zugehörigen) Teiler > 5.

```
Sub Primzahl()

   Dim dblZahl As Double

   Dim dblZähler As Double


   dblZahl = InputBox _

      ("Von welcher Zahl soll überprüft werden" & _

      ", ob es sich um eine Primzahl handelt?", "Prim")


   For dblZähler = 2 To Sqr(dblZahl)

      If dblZahl Mod dblZähler = 0 Then

         MsgBox dblZahl & " ist keine Primzahl"

         Exit Sub

      End If

   Next dblZähler


   MsgBox dblZahl & " ist eine Primzahl"


End Sub
```

Man könnte auch in Zweierschritten hoch zählen, da nur die ungeraden Zahlen interessant sind. Dann müssen allerdings die ersten Werte überprüft werden. Oder man arbeitet mit einer Bedingungsschleife:

```
Sub Primzahl2()

   Dim dblZahl As Double

   Dim dblZähler As Double


   dblZahl = InputBox("Von welcher Zahl soll überprüft " & _

      "werden, ob es sich um eine Primzahl handelt?", "Prim")

   dblZähler = 2

   Do Until dblZähler > Sqr(dblZahl)

      If dblZahl Mod dblZähler = 0 Then

         MsgBox dblZahl & " ist keine Primzahl." & _
```

```
            vbCr & "Ein Zähler lautet: " & dblZähler
            Exit Sub
         End If
         dblZähler = dblZähler + 1
      Loop
      MsgBox dblZahl & " ist eine Primzahl"
   End Sub
```


**Abbildung 1.8** 997.987.997 ist eine Primzahl.

**Achtung**

Übrigens: Da Mod nur Integer-Werte verarbeitet, wäre es besser, wenn man folgendermaßen überprüfen würde:

```
If cInt(dblZahl / dblZähler) = dblZahl / dblZähler Then
```


**Abbildung 1.9** 2.111.111.993 ist auch eine Primzahl.

Zwar wurde vor einiger Zeit die Fermat'sche These gelöst, aber dennoch gibt es eine Reihe von Problemen, die noch nicht gelöst sind. Das Collatz'sche Problem gehört dazu:

Man nehme eine beliebige Zahl. Ist diese Zahl gerade, dann wird sie durch 2 geteilt. Ist sie dagegen ungerade, dann wird sie mit drei multipliziert und um eins vergrößert; beispielsweise 20:

```
20 • 10 • 5 • 16 • 8 • 4 • 2 • 1 • 4 • 2 • 1 • 4 • 2 • 1 • ...
```

oder 21:

```
21 • 64 • 32 • 16 • 8 • 4 • 2 • 1 • 4 • 2 • 1 • ...
```

oder 19:

```
19 • 58 • 29 • 88 • 44 • 22 • 11 • 34 • 17 • 52 • 26 • 13 • 40 • 20 • ...
```

Jede Reihe endet schließlich in der Folge 1 → 4 → 2 → 1. Schreiben Sie ein Programm, bei dem der Benutzer eine Zahl eingibt, die mit der Collatz'schen Methode auf 1 zurückgeführt wird.

```
Sub CollatzschesProblem()
   Dim dblZahl As Double
   Dim strAText As String

   dblZahl = InputBox("Welche Zahl soll getestet werden?")
```

```
    strAText = dblZahl


    Do Until dblZahl = 1
       If dblZahl Mod 2 = 0 Then
          dblZahl = dblZahl / 2
       Else
          dblZahl = dblZahl * 3 + 1
       End If
    strAText = strAText & ", " & dblZahl
    Loop
    MsgBox strAText
End Sub
```

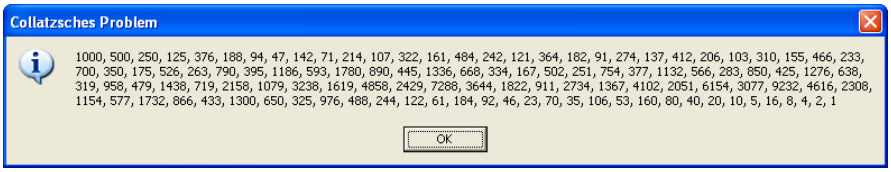



**Abbildung 1.10** Jede Zahl wird auf 4 → 2 → 1 → 4 → 2 → 1 zurückgeführt.

**Beispiel**

Gesucht ist eine Funktion, die die Quersumme einer Zahl berechnet. Daraus soll eine rekursive Funktion entwickelt werden, die die Endquersumme berechnet.

Es existieren zwei Lösungsvarianten für dieses Problem. Eine unschöne Variante wandelt die Zahl in eine Zeichenkette um, die nun Zeichen für Zeichen durchlaufen wird, eine elegante Variante löst die Aufgabe „mathematisch“, das heißt: Es wird mittels geschicktem Runden die letzte Ziffer herausgelöst. Ist eine der beiden Lösungen gefunden, dann kann die Endquersumme schnell berechnet werden.

```
Sub QuerBerechnen()
    Dim dblZahl As Double
    dblZahl = InputBox("Bitte eine Zahl eingeben.", "Quersumme")
    MsgBox "Die Quersumme von " & dblZahl & " lautet: " & _
       Quersumme(dblZahl)
End Sub
```

Die unschöne Variante wandelt die Zahl in eine Zeichenkette um, die nun Zeichen für Zeichen durchlaufen wird:

```
Function Quersumme(Zahl As Double) As Long
    Dim dblÜbergabewert As Double
    Dim dblInkrement As Double
    Dim dblZähler As Long
    dblÜbergabewert = 0
```

```
    dblInkrement = 0

    For dblZähler = 1 To Len(Str$(Zahl))
       dblInkrement = Val(Mid(Str$(Zahl), dblZähler, 1))
       dblÜbergabewert = dblÜbergabewert + dblInkrement
    Next

    Quersumme = dblÜbergabewert
End Function
```


**Abbildung 1.11** Die Quersumme

Eleganter ist die „mathematische“ Lösung, die vorsieht, mittels der Funktion Fix die letzte Ziffer herauszulösen:

```
Function Quersumme2(Zahl As Double) As Long
    Dim dblÜbergabewert As Double
    Dim dblInkrement As Double
    Dim lngZähler As Long
    Dim lngLänge As Long
    Dim dblNeueZahl As Double

    dblÜbergabewert = 0
    dblInkrement = 0
    lngLänge = Fix(Log(Zahl) / Log(10)) + 1
    dblNeueZahl = Zahl

    For lngZähler = 1 To lngLänge
       dblInkrement = dblNeueZahl - Fix(dblNeueZahl / 10) * 10
       dblÜbergabewert = dblÜbergabewert + dblInkrement
       dblNeueZahl = Fix(dblNeueZahl / 10)
    Next

    Quersumme2 = dblÜbergabewert
End Function
```

Die erste Variante sieht modifiziert dann wie folgt aus:

```
Function EndQuersumme(Zahl As Double) As Long
   Dim dblÜbergabewert As Double
   Dim dblInkrement As Double
   Dim dblZähler As Long
   dblÜbergabewert = 0
   dblInkrement = 0

   For dblZähler = 1 To Len(Str$(Zahl))
      dblInkrement = Val(Mid(Str$(Zahl), dblZähler, 1))
      dblÜbergabewert = dblÜbergabewert + dblInkrement
   Next
   If dblÜbergabewert > 9 Then
      dblÜbergabewert = EndQuersumme(dblÜbergabewert)
   End If
   EndQuersumme = dblÜbergabewert
End Function
```

Die zweite könnte folgendermaßen aussehen:

```
Function EndQuersumme2(Zahl As Double) As Long
   Dim dblÜbergabewert As Double
   Dim dblInkrement As Double
   Dim lngZähler As Long
   Dim lngLänge As Long
   Dim dblNeueZahl As Double

   dblÜbergabewert = 0
   dblInkrement = 0
   lngLänge = Fix(Log(Zahl) / Log(10)) + 1
   dblNeueZahl = Zahl

   For lngZähler = 1 To lngLänge
      dblInkrement = dblNeueZahl - Fix(dblNeueZahl / 10) * 10
      dblÜbergabewert = dblÜbergabewert + dblInkrement
      dblNeueZahl = Fix(dblNeueZahl / 10)
   Next


   If dblÜbergabewert > 9 Then
```

```
      dblÜbergabewert = EndQuersumme2(dblÜbergabewert)
   End If
   EndQuersumme2 = dblÜbergabewert
End Function
```

Es werden lediglich die drei Zeilen

```
   If dblÜbergabewert > 9 Then
      dblÜbergabewert = Quersumme(dblÜbergabewert)
   End If
```

eingefügt.


**Abbildung 1.12** Die Endquersumme

**Beispiel**

Leonardo Fibonacci (1170-1250) entdeckte eine Reihe, die mit 1 und 1 beginnt. Jede folgende Zahl entsteht als Summe der beiden Vorgänger.

Also:

```
1, 1, 2, 3, 5, 8, 13, 21, 34
```

und so weiter.

Die iterative Lösung stellt sicherlich kein Problem dar. Die rekursive Lösung ruft sich selbst auf. Dabei muss man wissen, dass F(x) = F(x-1) + F(x-2).

```
Sub Fibonacci_Zahlen1()
   Dim i1 As Double, i2 As Double, i3 As Double
   Dim bytAntwort As Byte
   Dim strFibonacci As String

   i1 = 1: i2 = 1
   strFibonacci = "1, 1"

   Do Until bytAntwort = vbNo
      i3 = i1 + i2
      strFibonacci = strFibonacci & ", " & Format(i3, "#,##0")
      i1 = i2: i2 = i3
      bytAntwort = MsgBox("Die letzte Fibonaccizahl war " & _
         Format(i3, "#,##0") & _
```

```
            "." & vbCr & "Die Reihe lautet: " & _
            strFibonacci & vbCr & vbCr & _
            "Möchten Sie eine weitere sehen?", vbYesNo)
    Loop
End Sub
```

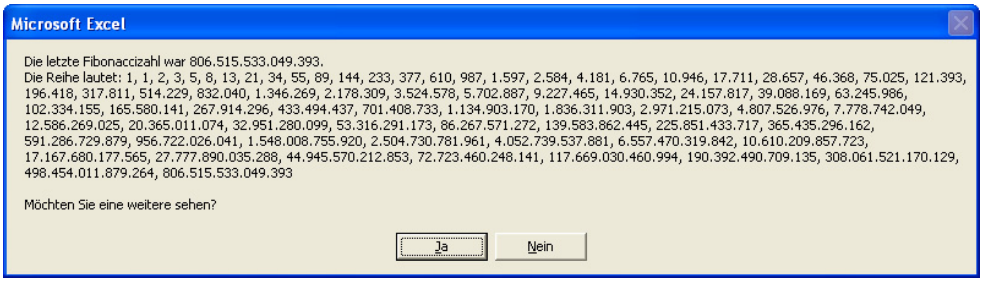



**Abbildung 1.13** Fibonacci I

Die rekursive Variante ruft sich selbst auf. Dabei muss man wissen, dass F(x) = F(x-1) + F(x-2). So ergibt sich die rekursive Funktion:

```
Function Fibonacci_Berechnen(dblAnzahl)


   If dblAnzahl < 3 Then
      Fibonacci_Berechnen = 1
   Else
      Fibonacci_Berechnen = Fibonacci_Berechnen(dblAnzahl - 1) _
      + Fibonacci_Berechnen(dblAnzahl - 2)
   End If
End Function
```

Diese Funktion könnte beispielsweise über folgende Prozedur aufgerufen werden:

```
Sub Fibonacci_Zahlen2()
   Dim dblAnzahl As Double
   dblAnzahl = InputBox("Welche Fibonaccizahl wird berechnet?")
   MsgBox "Die " & dblAnzahl & _
      ". Fibonaccizahl lautet: " & _
      Format(Fibonacci_Berechnen(dblAnzahl), "#,##0")
End Sub
```


**Abbildung 1.14** Fibonacci II

Excel hat beliebig keine beliebig tief verschachtelte Objekte – deshalb spielen rekursive Schleifen in der Praxis in VBA für Excel eine geringe Rolle. Auch für das Durchlaufen von Ordnern in beliebiger Tiefe kann diese Programmiertechnik nicht verwendet werden, da Dir innerhalb eines Programms nur einmal und nicht mehrmals benutzt werden.

Allerdings spielt die Rekursion in anderen Applikationen eine wichtige Rolle. Beispielsweise können in Visio Shapegruppe wiederum gruppiert werden. Will man auf sämtliche Shapes in beliebiger Tiefe zugreifen, dann müssen alle Gruppen der Gruppen der Gruppe … überprüft werden. Oder in Outlook. Dort können Ordner ineinander verschachtelt werden.

**Beispiel (Outlook)**

Das folgende Beispiel listet sämtliche Ordner und Unterordner und Unterunterordner … auf:

```
Sub AlleFoldersAuslisten()

   Dim olNS As NameSpace


   Set olNS = Application.GetNamespace("MAPI")

   strNamen = ""

   ListFolder olNS.Folders, 0

   MsgBox strNamen

End Sub


Sub ListFolder(parentfolder As Folders, i As Integer)

   Dim olFold As MAPIFolder


   For Each olFold In parentfolder

      strNamen = strNamen & vbCr & String(i * 2, vbTab) & _

         olFold.Name & ": " & olFold.DefaultMessageClass

      ListFolder olFold.Folders, i + 1

      DoEvents

   Next

End Sub
```

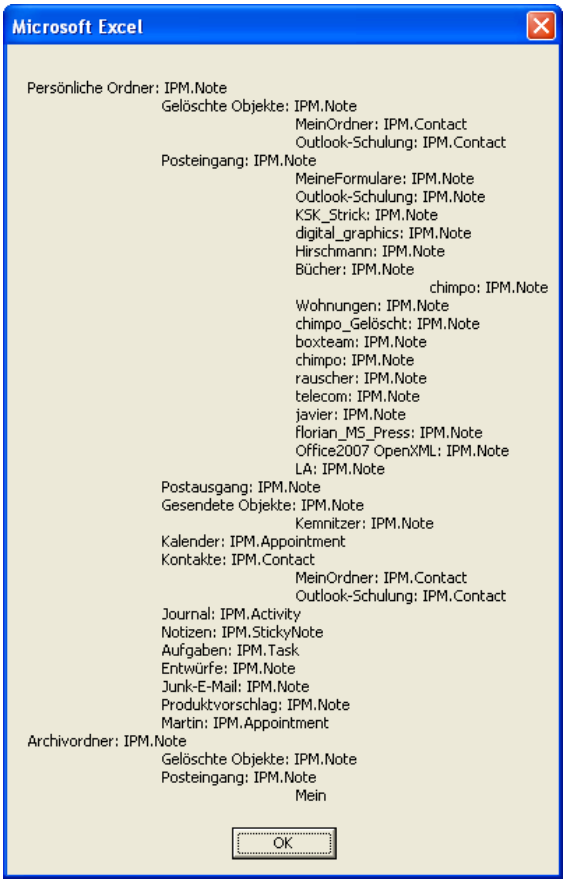



**Abbildung 1.15** Die Ordner von Outlook

## 1.9 Fehler

Das schlimme und schwierige Kapitel „Fehler“ hat zwei Komponenten: Fehler, die Sie als Programmierer beim Erstellen von Routinen machen, und Fehler, die während der Laufzeit auftreten, oder Fehler, die der Benutzer bei der Eingabe machen kann. Erstere sollten vor dem Ausliefern eines Programms gefunden werden, auf Letztere kann (und sollte) per Programmiercode reagiert werden.

### 1.9.1 Programmierfehler

An dieser Stelle folgen einige Tipps, wie man weniger Fehler machen kann.

**Tipp**

Schalten Sie im Menü Extras | Optionen im Registerblatt „Editor“ die Option „Variablendeklaration erforderlich“ ein.

Dann wird in neuen VBA-Modulen zu Beginn der Befehl

```
Option Explicit
```

auftauchen, der Sie zwingt, jede Variable zu deklarieren. So kann es nicht vorkommen, dass eine als `strEingabe` deklarierte Variable im Code als `strEingaben` durchgehen wird. Sie erhalten beim Testen sofort eine Fehlermeldung, die darauf hinweist.

**Tipp**

Rücken Sie ein! So können Anfang und Ende von With ... End With-Klammern, von Verzweigungen und Schleifen sichtbar gemacht werden. So vergessen Sie keine Zeile. Denn beim Kompilieren meldet VBA nicht immer den korrekten Fehler. In einem Programmteil:

```
Dim i As Integer

For i = 1 To 5


If i = 2 Then

Next
```

wird behauptet, dass For ohne Next sei.

**Tipp**

Schreiben Sie nach der If-Zeile gleich die End If-Zeile, nach Do While gleich Loop. Und so weiter. Auch so kann man das lästige Vergessen von Zeilen vermeiden.

**Tipp**

Vergeben Sie vernünftige Variablennamen. Strukturieren Sie Ihre Programme, indem Sie sie in verschiedene Module und Klassen unterteilen. So halten Sie Ordnung. Kommentieren Sie! Am besten jede Zeile. Schlimm genug, wenn man in einem fremden Makro auf Befehle wie Liste.Füllen stößt. Was das ist, kann nur durch den Einzelschrittmodus getestet werden.

**Tipp**

Tippfehler sind lästig. Deshalb tippe ich alle VBA-Befehle in Kleinbuchstaben ein und kontrolliere nach Betätigen der <ENTER>-Taste, ob VBA sie in Groß- bzw. Kleinschreibung umwandelt, das heißt, ob VBA sie erkennt. Falls nicht, dann liegt ein Tippfehler vor.

**Tipp**

Nicht sofort angezeigte Fehler können Sie auffinden, indem Sie mit dem Menüpunkt Debuggen | das Projekt debuggen. Sollten sich jedoch Fehler in Funktionen oder Prozeduren eingeschlichen haben, die bei einem Test gar nicht auffallen (weil beispielsweise diese Funktion oder Prozedur nicht verwendet wird), dann meldet es der Debugger trotzdem.

**Tabelle 1.12 Zum Testen stehen Ihnen folgende Optionen zur Verfügung:**

| Befehl | Menüpunkt | Tastenkombination |
|---|---|---|
| Einzelschritt | Debuggen \| Einzelschritt | <F8> |
| Einzelschritt. Unterprozeduren werden übersprungen | Debuggen \| Prozedurschritt | <SHIFT>+<F8> |
| Aktuelle Werte anzeigen | Den Mauszeiger auf die Variable setzen | |
| Haltepunkte | Debuggen \| haltepunkt ein/aus | <F9> |
| Sprung bis zur Cursorposition | Debuggen \| Ausführen bis Cursorposition | <STRG>+<F8> |
| Sprung bis zum Prozedurende | Debuggen \| Prozedur abschliessen | <SHIFT>+<STRG>+<F8> |
| Überwachungsausdrücke | Debuggen \| Überwachung hinzufügen | |
| | Ansicht \| Überwachungsfenster | |
| | Ansicht \| Lokalfenster | |

Die drei Fenster – das Überwachungsfenster, Lokalfenster und Direktfenster – dienen dazu, den Inhalt von Variablen zu überprüfen, während Sie im Einzelschritt (Taste <F8>) den Code durchlaufen

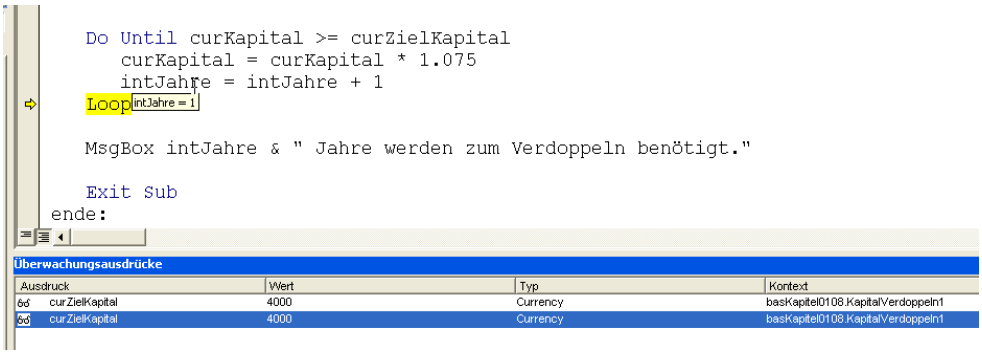



**Abbildung 1.16** Das Überwachungsfenster

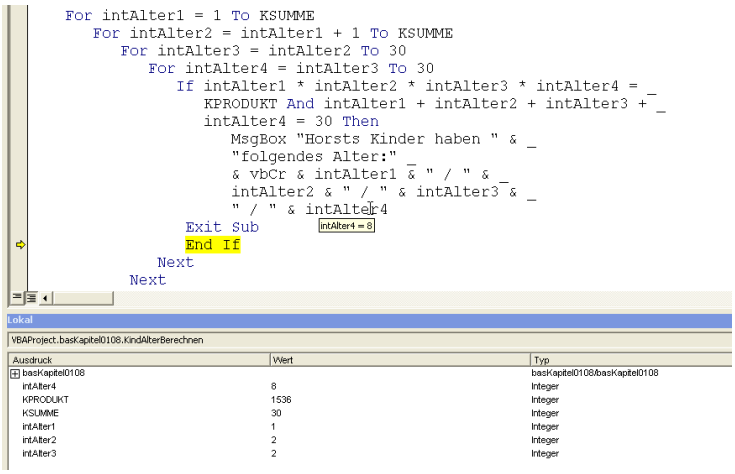



**Abbildung 1.17** Das Lokalfenster

In das Direktfenster kann mit Hilfe des „?“ der Inhalt einer oder mehrerer Variablen angezeigt werden. Ein Debug.Print im Code gibt den aktuellen Inhalt im Direktfenster an. Den Inhalt erhalten Sie auch, wenn Sie mit dem Mauszeiger über den Variablennamen zur Laufzeit fahren.

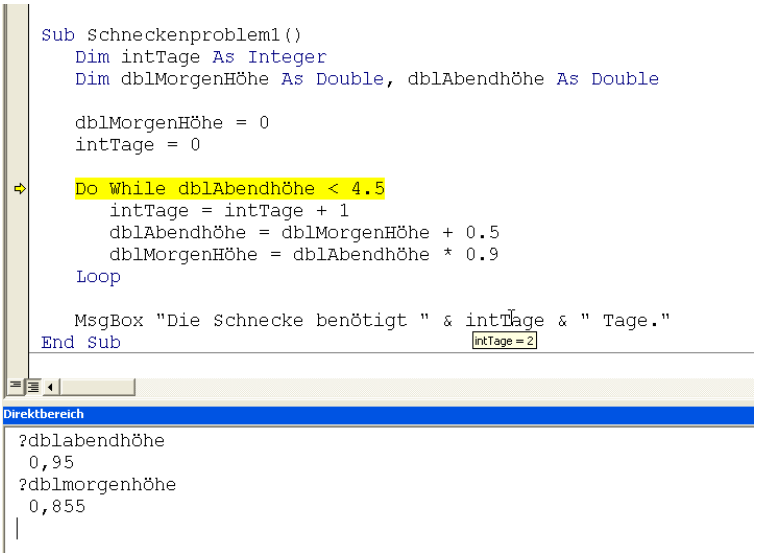



**Abbildung 1.18** Das Direktfenster

Alle verwendeten Dateien werden im Lokalfenster angezeigt. Vor allen bei tief verschachtelten Objekten ist dies eine sehr gute Hilfe.

Und schließlich können Sie Überwachungsausdrücke formulieren, deren Inhalt im Überwachungsfenster steht. Wird in Excel beispielsweise mit dem Wert der Zelle `xlZelle.Value` gearbeitet, dann könnte im Überwachungsfenster überprüft werden, welche Zelle gerade verarbeitet wird (`xlZelle.Address`) und zu welchem Blatt sie gehört (`xlZelle.Parent.Name`).

**Tipp**

Ich gehöre zu den Programmierern, die sich die Inhalte von Variablen nicht über das Direktfenster, sondern über eine MsgBox anzeigen lassen. Manchmal muss ich Codezeilen auskommentieren, um sie überspringen zu lassen – dafür steht die Symbolleiste Bearbeiten zur Verfügung.

### 1.9.2 Fehler zur Laufzeit

Gerade in Fachzeitschriften und auf Konferenzen ist schon viel über die Frage diskutiert worden, wie ein geübter Programmierer in VBA auf Fehler reagiert, die von Anwender- oder Programmseite herrühren. Sicherlich gibt es hierzu kein Patentrezept, aber ein paar Tipps. Im Folgenden beschreibe ich, welche Strategie ich in größeren Projekten anwende, um solche Fehler zu vermeiden. Schließlich soll der Anwender nicht folgende Meldung erhalten:

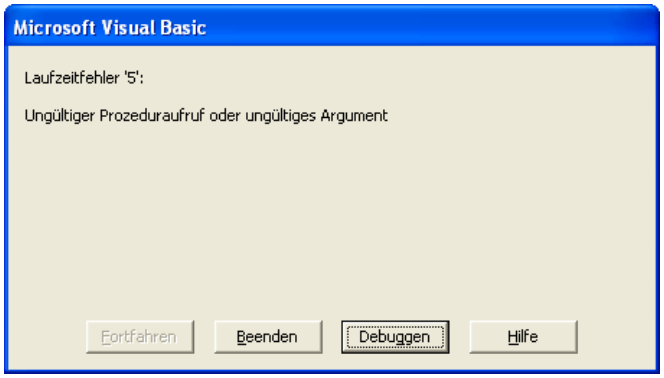



**Abbildung 1.19** oder Debuggen?

Nach der Variablendeklaration fange ich prinzipiell alle möglichen Fehler ab und reagiere adäquat darauf. Beispielsweise können Benutzereingaben durch einen einfachen If-Befehl abgefangen werden. Gibt der Benutzer beispielsweise eine Zahl ein, von der überprüft werden soll, ob es sich um eine Primzahl handelt, dann muss diese Zahl positiv sein. Also kann überprüft werden:

```
If dblEingabezahl < 1 Then

[...]
```

Ebenso können die Informationsfunktionen (IsDate, IsEmpty, IsError, IsMissing, IsNull, und IsNumeric) eingesetzt werden (sie wurden in Kapitel 1.2 aufgelistet):

```
If Not IsNumeric(dblEingabezahl) Then

[...]
```

Wird auf eine Datei zugegriffen, überprüfe ich, ob die Datei vorhanden ist. Dies leistet der Befehl Dir. Ebenso beim Speichern in einen Ordner oder beim Zugriff auf ein Laufwerk.

Werden Informationen in eine Excel-Tabelle geschrieben, dann überprüfe ich, ob auch wirklich die richtige Datei geöffnet wurde. In Excel können Informationen vor dem Anwender verborgen werden. Wie das funktioniert, wird in Kapitel 6 beschrieben. Ich überprüfe stets, ob Tabellen, die benötigt werden, wirklich vorhanden sind. Möglicherweise hat der Benutzer ein für mein Programm relevantes Tabellenblatt umbenannt oder ge-

löscht. Auch bei Datenauswertungen muss man Sorgfalt walten lassen. Wenn es dem Anwender möglich ist, Daten zu verändern, beispielsweise aus Zahlen Texte zu machen, dann sollte dies „sauber“ vor dem eigentlichen Programmstart überprüft werden.

Was aber, wenn ein ganz anderer Fehler auftritt? Ein Fehler, den Sie nicht eingeplant haben. Dann steht Ihnen der Befehl

```
On Error
```

zur Verfügung:

```
On Error Resume Next
```

Der Fehler wird übersprungen, und das Programm wird mit der Zeile fortgesetzt, die unmittelbar auf die Zeile folgt, die den Fehler verursacht. Dies ist eine gefährliche Sache, da nicht auf den Fehler adäquat reagiert wird.

```
On Error GoTo 0
```

Dieser Befehl deaktiviert alle benutzerdefinierten Fehlerroutinen. Diese Zeile kann für die Testphase und die Analyse eines Programmverhaltens interessant sein.

```
On Error GoTo Sprungmarke
```

Tritt ein Fehler auf, so wird eine Sprungmarke angesprungen, die sich in der Regel am Ende der Prozedur befindet. Dort kann auf den Fehler adäquat reagiert werden. Vor der Sprungmarke sollte allerdings das Programm beendet werden mit:

```
Exit Sub
```

Innerhalb der Sprungmarke kann der Fehler über die Fehlerobjektvariable Err spezifiziert werden.

**Tabelle 1.13** Err besitzt folgende Eigenschaften und Methoden:

| **Eigenschaften von Err** | **Beschreibung** |
|---|---|
| Number | eine spezifische Nummer des Fehlers |
| Description | Beschreibung des Fehlers, die dem Benutzer angezeigt werden kann |
| Source | Fehlerquelle |
| HelpContext, HelpFile | Hilfedatei, die an einen bestimmten Fehler gebunden ist |
| Methoden von Err | Beschreibung |
| Clear | löscht den Fehler |
| Raise | erzeugt einen Fehler |

Soll nach dem behandelten Fehler die Prozedur an der gleichen Stelle aufgerufen werden und ab dieser Stelle weitergearbeitet werden, dann geschieht dies mit dem Befehl

```
Resume
```

Dazu sollte der Fehlerwert natürlich wieder zurückgesetzt werden. Liegt kein Fehler vor, dann hat er den Wert 0. Also:

```
[...]

Err.Clear

Resume
```

Soll eine Zeile tiefer weitergearbeitet werden, dann mit

```
Resume Next
```

Über Fehler und ihre Behandlungsmöglichkeiten ließe sich noch viel sagen. In Fachzeitschriften werden diese Themen und Strategien dazu stark diskutiert. Was hier vorgestellt wurde, ist lediglich das Gerüst, das nun ausbaufähig ist.

Prozeduren beginnen in meinen Programmen mit der kompletten Liste der Variablen.

Anschließend folgt der Befehl

```
On Error GoTo ende
```

Wobei „ende“ eine Sprungmarke darstellt, die sich am unteren Teil des Programms befindet.

Im zweiten Teil werden mögliche Fehlerquellen sauber überprüft, und – falls „etwas nicht stimmt“ – es wird mit einer Meldung an den Benutzer reagiert. Das Programm wird abgebrochen, bevor es überhaupt zum Laufen kommt.

Im eigentlichen Programmteil treten nur Fehler auf, die ich nicht eingeplant habe, das heißt Dinge, mit denen ich überhaupt nicht gerechnet habe. Inzwischen weiß ich, dass ein Anwender beispielsweise zwei Mal Excel öffnen kann und die Datei bereits in einer anderen Applikation läuft. Oder dass in Excel die Seiteneinstellungen über Seitenlayout (bis Excel 2003: Datei | Seite einrichten) nicht funktionieren, wenn kein Drucker installiert ist. Auf solche Fehler reagiere ich am Ende mit:

```
    Exit Sub

ende:

    If Err.Number > 0 Then

        MsgBox "Es ist ein Fehler eingetreten :" & vbCr & _

          Err.Description, vbCritical

        End

   End If


End Sub
```

Dies könnte wie folgt aussehen:

**Beispiel**

Teile des Abfangens können über eine If-Bedingung erledigt werden, wie das folgende Beispiel zur Berechnung des kleinsten gemeinsamen Vielfachen zeigt:

```
Sub kgV()

   Dim dblZahl1 As Double

   Dim dblZahl2 As Double

   Dim dblGrößteZahl As Double

   Dim dblZähler As Double

   Const err_Überlauf = 6    'die Fehlerkonstanten

   Const err_KeineZahl = 13

   Const err_NegativeZahl = 998

   Const err_KeineGanzeZahl = 999


   On Error GoTo err_kgV


   dblZahl1 = InputBox("Von welchen Zahlen soll das " & _
```

```
        "kgV berechnet werden?", "kgV: Zahl1")
    dblZahl2 = InputBox("Von welchen Zahlen soll das " & _
        "kgV berechnet werden?", "kgV: Zahl2")

'mögliche Fehler werden erzeugt und abgefangen
    If dblZahl1 < 1 Or dblZahl2 < 1 Then
        Err.Raise err_NegativeZahl
    End If
    If Fix(dblZahl1) < dblZahl1 Or Fix(dblZahl2) < dblZahl2 Then
        Err.Raise err_KeineGanzeZahl
    End If

'der eigentliche Programmteil
    If dblZahl1 > dblZahl2 Then
        dblGrößteZahl = dblZahl1
    Else
        dblGrößteZahl = dblZahl2
    End If

    For dblZähler = dblGrößteZahl To dblZahl1 * dblZahl2
        If dblZähler Mod dblZahl1 = 0 And _
           dblZähler Mod dblZahl2 = 0 Then
            MsgBox "Das kgV von " & _
            dblZahl1 & " und " & dblZahl2 & _
            " lautet: " & dblZähler
            Exit Sub
        End If
    Next dblZähler
    Exit Sub

err_kgV:   'die Sprungmarke und Fehlerroutine
    If Err.Number = err_KeineZahl Then
        MsgBox "Es wurde keine Zahl eingegeben"
    ElseIf Err.Number = err_KeineGanzeZahl Then
        MsgBox "Es wurde keine ganze Zahl eingegeben"
```

```
   ElseIf Err.Number = err_NegativeZahl Then
      MsgBox "Es wurde eine negative Zahl eingegeben"
   ElseIf Err.Number = err_Überlauf Then
      MsgBox "Das Ergebnis kann nicht berechnet werden - es ist zu groß!"
   Else
      MsgBox "Fehlernummer: " & Err.Number & _
         vbCr & "Fehler: " & Err.Description & _
         vbCr & "Fehlerquelle: " & Err.Source
   End If
End Sub
```

**Beispiel**

Ein Benutzer tippt einen Laufwerksbuchstaben in eine Inputbox, in der sich ein Ordner befindet, in dem eine Datei liegt (\SCHWUNG\Schwung.ini). Nun kann eine Reihe Fehler überprüft werden:

- Der Benutzer hat keinen Buchstaben eingetippt.
- Das Verzeichnis existiert nicht.
- Der Benutzer hat kein Leserecht auf dieses Verzeichnis.
- Im Diskettenlaufwerk liegt keine Diskette.
- Im angegebenen Laufwerk existiert der Ordner nicht.
- Das Laufwerk existiert, der Ordner auch, allerdings ist die Datei nicht vorhanden.

Erzeugen Sie (künstliche) Fehler mit dem Befehl GettAttr, fangen Sie den Fehler ab, und melden Sie ihn dem Benutzer.

```
Sub Laufwerk_und_Ordner_und_Datei()
   Dim strLaufwerk As String

   On Error Resume Next

   strLaufwerk = InputBox("Bitte einen Laufwerksbuchstaben" & _
      " eingeben")
   strLaufwerk = Left(strLaufwerk, 1)
   strLaufwerk = UCase(strLaufwerk)

   If Asc(strLaufwerk) < Asc("A") Or _
      Asc(strLaufwerk) > Asc("Z") Then
         MsgBox "Sie haben Unsinn getippt!", _
            vbCritical, "SCHWUNG"
         End
   End If
```

```
GetAttr (strLaufwerk & ":\")
If Err.Number = 5 Then
    MsgBox "Es wurde keine Diskette/CD-ROM in Laufwerk " _
    & strLaufwerk & " eingelegt!", vbCritical, "SCHWUNG"
     End
ElseIf Err.Number = 76 Then
     MsgBox "Das Laufwerk " & strLaufwerk & _
     " existiert nicht auf Ihrem Rechner", _
     vbCritical, "SCHWUNG"
     End
ElseIf Err.Number = 70 Then
   MsgBox "Sie haben kein Leserecht auf das Laufwerk " & _
   strLaufwerk, vbCritical, "SCHWUNG"
     End
ElseIf Err.Number > 0 Then
    MsgBox "Es ist ein Fehler eingetreten auf Laufwerk " _
    & strLaufwerk & _
    vbCr & vbCr & Err.Description, vbCritical, "SCHWUNG"
    End
End If
GetAttr (strLaufwerk & ":\SCHWUNG\Schwung.ini")
If Err.Number = 76 Then
   MsgBox "Der Ordner ""SCHWUNG"" existiert nicht " & _
      "in Laufwerk " & strLaufwerk, vbCritical, "SCHWUNG"
     End
 ElseIf Err.Number = 53 Then
     MsgBox "Die Datei ""Schwung.ini"" existiert " & _
        "nicht in " & strLaufwerk & ":\SCHWUNG", _
        vbCritical, "SCHWUNG"
     End
 ElseIf Err.Number > 0 Then
      MsgBox "Es ist ein Fehler eingetreten beim " & _
        "Zugriff auf " & strLaufwerk & _
        ":\SCHWUNG\Schwung.ini" & vbCr & vbCr & _
```

```
                Err.Description, vbCritical, "SCHWUNG"
            End
        End If
End Sub
```

Die Fehler können ebenso protokolliert und in eine Textdatei oder Datenbank geschrieben werden, von wo aus sie ausgelesen und auf sie reagiert wird. Ich verwende das Meldungsfenster, weil ich weiß, dass mich der Anwender bald darauf anrufen oder mir eine E-Mail schicken wird, in der er den Fehler beschreibt. Anhand der Fehlernummer und der Beschreibung des Fehlers (`Err.Description`) kann relativ schnell ermittelt werden, welche Art von Fehler aufgetreten ist.

Ich versuche zu vermeiden, eigene Fehler zu erzeugen. Man kann beispielsweise von Word mit dem Befehl

```
Set xlApp = GetObject(, "Excel.Application")
```

auf das geöffnete Excel zugreifen. Falls Excel nicht offen ist, kann dies wie folgt abgefangen werden:

```
   Dim xlApp  As Object
   Const err_Excel_LäuftNicht = 429
   On Error Resume Next
   Set xlApp = GetObject(, "Excel.Application")
   If Err.Number = err_Excel_LäuftNicht Then
      Set xlApp = CreateObject("Excel.Application")
   End If
   On Error GoTo err_handler
[...]
   Set xlApp = Nothing
[...]
   Exit Sub
err_handler:
```

[...]

Das ist – meiner Meinung nach – kein sauberer Code. Wie programmiertechnisch sauber auf eine andere Applikation zugegriffen wird, soll hier nicht beschrieben werden.

Laut „Microsoft Office 2000 Visual Basic-Programmierhandbuch“ sind für das Error-Objekt die ersten 512 Zahlen als Fehlernummern reserviert. Die übrigen Zahlen (bis 65.536) stehen dem Benutzer für eigene Fehler zur Verfügung. Dies ist nicht korrekt, da in einigen Applikationen Fehlernummern vergeben sind, die größer als 512 oder negativ sind. Dennoch kann man sich die ersten 512 Fehlernummern mit ihrer Bedeutung auflisten lassen:

```
Sub Fehlernummern()
   Dim i As Integer
   Dim strFehlerListe As String
   On Error Resume Next

   For i = 1 To 512
```

```
      Err.Raise i

         strFehlerListe = strFehlerListe & vbCr & i & ":" _

            & vbTab & Err.Description

         Err.Clear

      Next

      MsgBox strFehlerListe

   End Sub
```

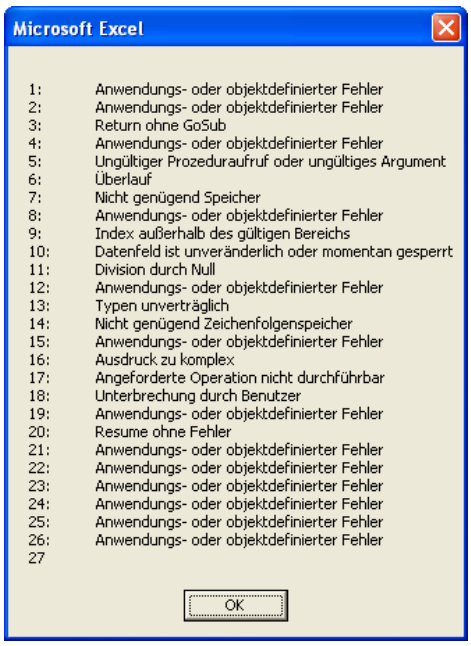



**Abbildung 1.20** Einige vorbelegte Fehlernummern

Dieses Ergebnis kann man sich natürlich in eine Excel-Datei schreiben lassen.

Zugegeben: An einigen – wenigen – Stellen ist es nötig, Fehler zu generieren.

**Beispiel (Word)**

In einer Firma liegen Dokumentvorlagen auf dem Server, die der Anwender über eine spezifische Word-Maske holen kann. Einige dieser Vorlagen sind ungeschützt, andere sind geschützt. Liegt ein Schutz auf den Vorlagen, dann entweder ohne Kennwort, damit der Anwender sie öffnen kann, um mit dem Dokument Serienbriefe zu erstellen, oder mit einem Kennwort (hier „Kennwort“), damit der Anwender sie nicht öffnen kann. Nun existiert noch ein dritter Fall: Der Anwender kann eigene Vorlagen erstellen – mit und ohne Schutz und natürlich mit eigenem Kennwort, wenn diese Vorlage beispielsweise abteilungsweise verwendet wird.

Der Befehl

```
strPasswort = " Kennwort "

ActiveDocument.Unprotect Password:=strPasswort
```

öffnet das geschützte Dokument, ganz gleich, ob das Passwort vergeben oder kein Passwortschutz verwendet wurde. Liegt ein anderes Passwort auf dem Dokument wird ein Fehler erzeugt. Problematisch ist der erste Fall, weil nach Beendigung der Routine wird per Programmierung wieder ein Schutz eingeschaltet – er soll genauso sein, wie er vorher war:

1. Fall: War das Dokument offen, dann bleibt es offen.

2. Fall: War das Dokument ohne Passwort geschützt, dann ist es hinterher wieder ohne Passwort geschützt.

3. Fall: War das Dokument mit dem Passwort „Kennwort“ geschützt, dann ist es hinterher wieder mit demselben Passwort geschützt.

4. Fall: War das Dokument mit einem anderen Passwort geschützt, dann erhält der Anwender einen Hinweis, weil per Programmierung das Dokument nicht geöffnet werden kann.

Der erste Fall ist schnell überprüft. Schwieriger sind die übrigen drei Fälle:

Liegt ein Schutz auf dem Dokument, dann wird überprüft, ob kein Kennwort auf dem Dokument liegt. Es wird mit irgendeinem beliebigen Kennwort (hier: „@@@Rene Martin@@@”) geöffnet, von dem unwahrscheinlich ist, dass dieses Kenwort von einem anderen Anwender vergeben wurde. Kann man es mit diesem Kennwort öffnen, dann lag kein Kennwort auf dem Dokument, das heißt – es wird ebenfalls ohne Kennwort geschützt. Ist der Befehl misslungen, wird ein Fehler erzeugt. Diese kann abgefangen werden. Nun wird versucht, das Dokument mit dem Passwort „Kennwort“ zu öffnen. Gelingt dies, dann ist das Passwort bekannt. Falls nicht, dann liegt ein anderes, benutzerseitig vergebenes Kennwort auf dem Dokument. Dies wird nun dem Anwender mitgeteilt, da nicht alle Informationen in das Dokument geschrieben werden können. Leider ist das Überprüfen nur über das Abfangen der Fehler möglich, wie der unten stehende Code zeigt:

```
Sub SchutzTest()
   Dim fGeschützt As Boolean
   Dim fSchutzOhnePW As Boolean
   Dim fSchutzMitITKennwort As Boolean
   Dim strPasswort As String
   On Error Resume Next
   strPasswort = "Kennwort"
   fSchutzMitITKennwort = True
   Err.Clear
   If ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
   ' -- wenn ein Schutz auf dem Dokument liegt
      fGeschützt = True

      ActiveDocument.Unprotect Password:="@@@Rene Martin@@@"
      If Err.Number = 0 Then
         fSchutzOhnePW = True
      Else
         fSchutzOhnePW = False
      End If

      Err.Clear
      If fSchutzOhnePW = False Then
         ActiveDocument.Unprotect Password:=strPasswort
         If Err.Number = 0 Then
            fSchutzMitITKennwort = True
```

```
      Else
         fSchutzMitITKennwort = False
       MsgBox "Der Befehl kann nicht vollständig ausgeführt werden," _
       & vbCr & _
         "da auf dem Dokument ein anderes Passwort liegt.", _
         vbInformation, "Kreissparkasse"
         On Error Resume Next
         ' -- falls ein anderes Passwort auf dem Dokument liegt.
      End If
   End If


End If
Err.Clear


' -- ********************
' --       tue was
' -- ********************
If fSchutzMitITKennwort = True Then
   MsgBox "Hier tue ich was!"
End If
' -- schütze so wie vorher:
If fGeschützt = False Then
   ' -- das Dokument war nicht geschützt und bleibt offen
Else
   ' -- das Dokument war geschützt
   If fSchutzOhnePW = True Then
       ' -- ohne IT-Kennwort
       ActiveDocument.Protect Type:=wdAllowOnlyFormFields
    Else
       If fSchutzMitITKennwort = True Then
       ' -- mit IT-Kennwort
          ActiveDocument.Protect _
             Type:=wdAllowOnlyFormFields,
             Password:=strPasswort, _
             NoReset:=True
```

```
            End If

        End If

    End If


End Sub
```

## 1.10 Fazit

Wenn Sie Variablen sauber deklarieren, wenn Sie sich an hausinterne, eigene oder die Reddick-Konvention halten, strukturiert programmieren, Fehlermöglichkeiten sauber abfangen und sehr gut testen, dann kann (fast) nichts mehr schief gehen.

# 2 Dialoge

Ähnlich wie in Kapitel 1 („Sprachkern“) muss man in einem VBA-Profi-Buch sicherlich niemandem mehr erzählen, wie ein Dialog erstellt wird, welche Eigenschaften eingestellt werden und wie man auf Steuerelemente zugreift beziehungsweise sie dynamisch verändert.

An dieser Stelle sollen vielmehr einige Beispiele aus meiner Praxis beschrieben werden, wie Dynamik in Dialogen realisiert wird, wie auf Dialoge Eingabefehler vom Anwender abgefangen werden und worauf dabei zu achten ist.

# 2.1 Dialoge

## 2.1.1 Der Verkäuferstamm einer Firma

Um eine Datenauswertung vorzunehmen, wird ein Dialog erstellt. Die wichtigste Eigenschaft des Dialogs ist der Name – ich vergebe ihn immer zuerst, hier: `frmVerkäufer`.

**Hinweis**

In den letzten Jahren bin ich davon abgekommen, Dialoge farblich zu gestalten oder mit speziellen Effekten zu versehen – ich versuche, die Gestaltung ähnlich den Microsoft-Dialogen zu realisieren. Der Grund ist einfach: Der Anwender kommt leichter damit zurecht, wenn auf einer grauen Maske die Buttons so platziert sind, wie er es von seinen Anwendungsprogrammen gewohnt ist. Dies ist jedoch lediglich ein grobes Muster; bedauerlicherweise halten sich auch die Dialoge in MS Office nicht an Standards, sondern variieren von Applikation zu Applikation. Man muss sich beispielsweise nur den Dialog Schriftart in Word, PowerPoint und in Excel (bis Office 2003: Format | Zeichen) ansehen.

Den Elementen werden konsequent Namen nach der Reddick-Konvention zugewiesen – in diesem Beispiel: `lblListe`, `cboKategorie`, `txtName`, `txtZiel`, `txtZielGrossMargin`, `txtZielService`, `txtZielNewCopierers`, `txtZielReplacement`, `txtZielNeukunde`, `cmdNL`, `cmdVertriebsbeauftragter`, `cmdStandort`, `cmdVerkäufer`, `cmdLöschen`, `cmdUmsatz`, `cmdInfo` und `cmdAbbrechen`. Auch die Bezeichnungsfelder erhalten vernünftige, das heißt sprechende Namen: `lblZiel`, `lblName` …

Die Abbrechen-Schaltfläche sitzt jeweils unten rechts, die übrigen Schaltflächen werden darüber oder daneben angeordnet. Um einen gleichen Abstand zu erhalten, kann im Menü Format die Größe angeglichen, die Buttons können ausgerichtet und der Abstand angeglichen werden. Zum Teil arbeite ich mit den numerischen Werten im Eigenschaftsfenster, damit die Abstände, Größen und Positionswerte exakt sind. Das ist sehr viel Arbeit, aber der Kunde wird zuerst die Maske sehen, – und – falls sie ihm nicht gefällt, muss ich nachträglich Änderungen vornehmen.

Im Menü Ansicht | Aktivierreihenfolge werden die Steuerelemente so angeordnet, dass zuerst die Bezeichnungsfelder, dann die Textfelder in der Liste stehen. Der Grund ist einfach: Jedes Bezeichnungsfeld verfügt über einen Accelerator, einen Buchstaben, mit dessen Hilfe der Anwender mit der Tastatur in das nächste Textfeld springen kann.

Der Accelerator des Bezeichnungsfeldes „lblName“ lautet „N“. Damit erhält das „N“ in „Name des neuen Verkäufers“ einen Unterstrich. Drückt nun der Anwender <ALT>+<N>, dann springt der Cursor nicht auf das Label, sondern auf das Steuerelement, das in der Liste der Aktivierreihenfolge als nächstes unter „lblName“ steht.

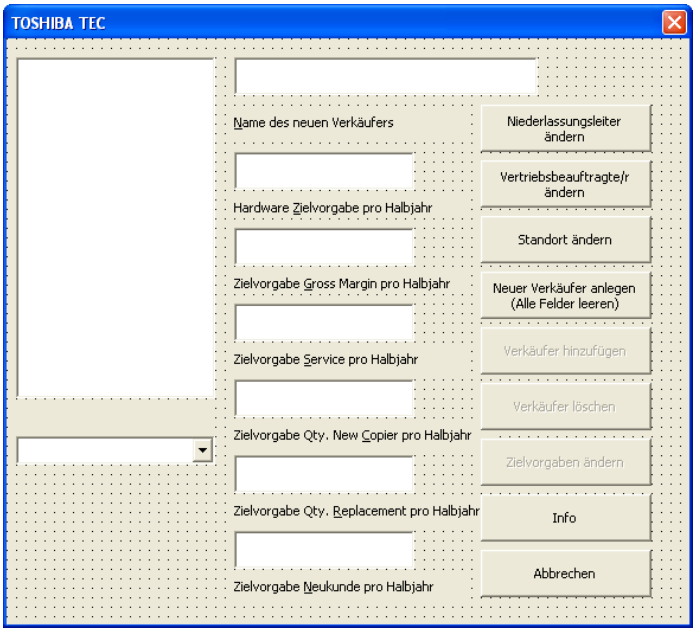



**Abbildung 2.1** Die Eingabemaske für die Namen der Verkäufer

### 2.1.2 Die Daten werden ausgelesen

Drei Variablen werden modulweit, das heißt, für das gesamte Formular deklariert:

```
Dim xlBlatt As Worksheet

Dim xlZelle As Range

Dim intZeilen As Integer
```

Dabei gehe ich davon aus, dass wohl kaum mehr als 32.000 Verkäufer oder 32.000 Kategorien benötigt werden (deshalb deklariere ich die Variable vom Typ „Integer“). Beim Starten werden Combobox und Listenfeld gefüllt:

```
Private Sub UserForm_Initialize()

   Dim i As Integer

   Set xlBlatt = Application.ThisWorkbook.Worksheets("Verkäufer")


   Set xlZelle = xlBlatt.Range("N1")

   intZeilen = xlZelle.CurrentRegion.Columns.Count


   For i = 1 To intZeilen - 1   ' -- die Kategorien

      Me.cboKategorie.AddItem xlZelle.Offset(0, i).Value

   Next


   Set xlZelle = xlBlatt.Range("A1")

   intZeilen = xlZelle.CurrentRegion.Rows.Count
```

```
    For i = 1 To intZeilen - 1

       Me.lstListe.AddItem xlZelle.Offset(i, 0).Value

    Next


End Sub
```

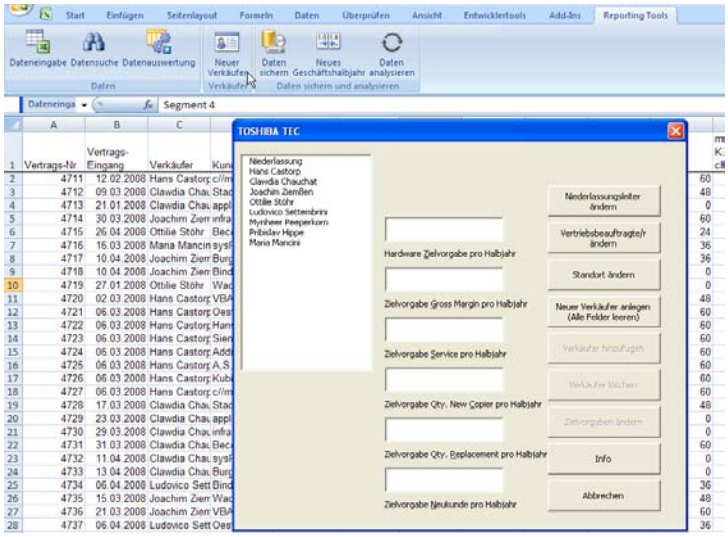



**Abbildung 2.2** Die Daten werden aus dem Blatt ausgelesen.

Zwar könnte man den Verkäuferkategorien auf dem Excel-Blatt einen Namen zuweisen und diesen fest verknüpft in die Liste aufnehmen – das Einlesen per Code hat den Vorteil, dass neue Kategorien lediglich hinzugefügt werden müssen; sie werden dann automatisch eingelesen.

Die Verkäuferkategorien stehen nebeneinander, da Verkäufer aus der Liste auch gelöscht werden können.

### 2.1.3 Neue Daten hinzufügen

Die Befehlsschaltflächen „Verkäufer hinzufügen“, „Verkäufer löschen“ und „Zielvorgaben ändern“ wurden in den Eigenschaften auf `Enabled = False` gesetzt – man hätte dies auch im Code im Ereignis `UserForm_Initialize` erledigen können. Ebenso sind die Textfelder inaktiv.

Wird nun einer der Verkäufer ausgewählt, werden seine Daten eingelesen und formatiert angezeigt. Die Variable i ermittelt die Indexnummer des gewählten Eintrags.

```
Private Sub lstListe_Change()


    Dim i As Integer

    On Error Resume Next


    Me.cmdUmsatz.Enabled = True

    i = Me.lstListe.ListIndex + 1

    Me.txtZiel.Value = Format(xlZelle.Offset(i, 1).Value, "#,##0.00")

    Me.txtZielGrossMargin.Value = _

       Format(xlZelle.Offset(i, 2).Value, "#,##0.00")
```

```
    Me.txtZielService.Value = _
       Format(xlZelle.Offset(i, 3).Value, "#,##0.00")
    Me.txtZielNewCopierers.Value = _
       Format(xlZelle.Offset(i, 4).Value, "#,##0.00")
    Me.txtZielReplacement.Value = _
       Format(xlZelle.Offset(i, 5).Value, "#,##0.00")
    Me.txtZielNeukunde.Value = _
       Format(xlZelle.Offset(i, 6).Value, "#,##0.00")

    Call DeaktAlle

    If Me.lstListe.ListIndex >= 0 Then
       If Me.lstListe.Value = "Niederlassung" Then
          Me.cboKategorie.Visible = False
       ElseIf xlZelle.Offset(Me.lstListe.ListIndex + 1, _
          7).Value = "" Then
          Me.cboKategorie.Visible = True
          Me.cboKategorie.Value = ""
          Exit Sub
       Else
          Me.cboKategorie.Visible = True
          Me.cboKategorie.ListIndex = _
             xlZelle.Offset(Me.lstListe.ListIndex + 1, 7).Value - 2
       End If
    End If
End Sub
```

Wurde „Niederlassung“ gewählt, dann ist die ComboBox unsichtbar; bei einer Auswahl eines Verkäufers wird seine zugehörige Kategorie angezeigt. Sie wird über den Wert, der in der Spalte „H“ steht, ermittelt und als Listindex der Dropdown-Liste verwendet.

Zugegeben: dieses Beispiel hat in puncto Skalierbarkeit einen kleinen Nachteil: werden neue Kategorien hinzugefügt, dann müssen sie entweder ans Ende der Liste gesetzt werden oder die Nummerierung der Spalten muss erneut angepasst werden. Das Dilemma könnte über Konstanten oder über eine Schleife gelöst werden (siehe Kapitel 1).

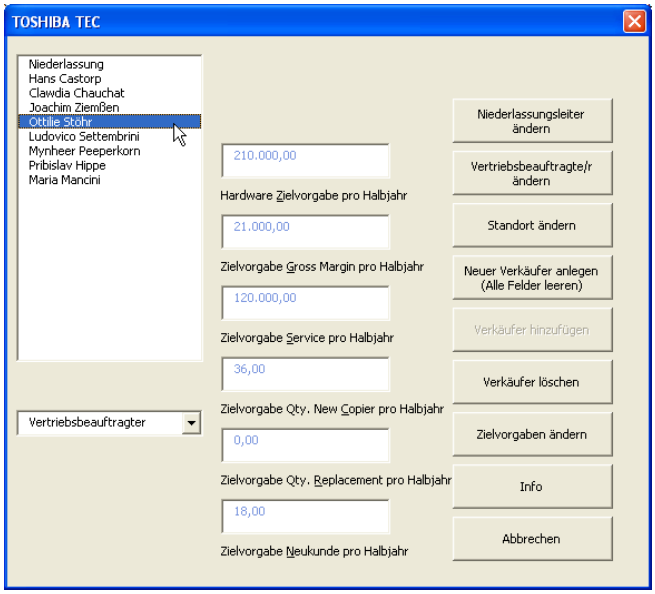



**Abbildung 2.3** Nur bei den Namen wird das Kombinationsfeld sichtbar.

Das Makro „DeaktAlle“ deaktiviert alle Textfelder und die drei Befehlsschaltflächen. Es wird an mehreren Stellen aufgerufen:

```
Private Sub DeaktAlle()
   Dim ctl As Control

   For Each ctl In Me.Controls
      If Left(ctl.Name, 3) = "txt" Then
         ctl.Locked = True
         ctl.ForeColor = &H80000003
      End If
   Next
   Me.cmdUmsatz.Caption = "Zielvorgaben ändern"
   Me.cmdOk.Enabled = False
   Me.cmdLöschen.Enabled = True
End Sub
```

Auch hier bietet sich eine Alternative an: satt der Codezeile:

```
      If Left(ctl.Name, 3) = "txt" Then
```

kann geschrieben werden:

```
      If TypeName(ctl) = "TextBox" Then
```

Mit Hilfe der Befehlsschaltfläche `cmdNL` kann der Name des Niederlassungsleiters geändert werden. Die Neueingabe erfolgt über eine einfache, unflexible InputBox, die in diesem Fall genügt, weil davon auszugehen ist, dass der Niederlassungsleiter nicht sehr häufig geändert wird:

```
Private Sub cmdNL_Click()
   On Error Resume Next
   If MsgBox("Der aktuelle Niederlassungsleiter lautet: """ & _
      Application.ActiveWorkbook.Worksheets("Verkäufer"). _
      Range("J2").Value & """." & vbCr & "Möchten Sie ihn ändern?", _
      vbInformation + vbYesNo, "TOSHIBA TEC") = vbYes Then

         Application.ActiveWorkbook.Worksheets("Verkäufer"). _
         Range("J2").Value = InputBox("Bitte geben Sie den Namen " &_
         vbCr & "des neuen Niederlassungsleiters ein!", "TOSHIBA TEC", _
         Application.ActiveWorkbook.Worksheets("Verkäufer"). _
         Range("J2").Value)
      Unload Me
   End If
End Sub
```

Analog werden der Vertriebsbeauftragte und der Standort geändert.

### 2.1.4 Daten löschen

Scheidet ein Verkäufer aus, dann kann sein Name aus der Liste gelöscht werden – allerdings auch hier nach vorheriger Nachfrage. Die „Niederlassung“ darf allerdings nicht gelöscht werden:

```
Private Sub cmdLöschen_Click()
   On Error Resume Next
   If Me.lstListe.Value = "Niederlassung" Then
      MsgBox "Sie dürfen die Niederlassung nicht löschen!", _
      vbInformation, "TOSHIBA TEC"
      Exit Sub
   End If

   If MsgBox("Möchten Sie """ & Me.lstListe.Value & _
      """ wirklich löschen?", vbInformation + vbYesNo, _
      "TOSHIBA TEC") = vbYes Then
      xlZelle.Offset(Me.lstListe.ListIndex + 1, _
         1).EntireRow.Delete shift:=xlUp
      Set xlZelle = Nothing
      Set xlBlatt = Nothing
      Unload Me
```

```
    End If

End Sub
```

Beim Löschen wird die Zeile gelöscht – die übrigen Zeilen werden nach oben geschoben. Wird ein neuer Verkäufer angelegt, dann werden alle Felder geleert, das Namensfeld und sein Bezeichnungsfeld werden sichtbar, und der Cursor wird mit der Methode `SetFocus` in das Namensfeld gesetzt.

```
Private Sub cmdAlleLeeren_Click()

    Dim ctl As Control

    On Error Resume Next

    For Each ctl In Me.Controls

        If Left(ctl.Name, 3) = "txt" Then

            ctl.Visible = True

            ctl.Locked = False

            ctl.Value = ""

            ctl.ForeColor = &H80000007

        End If

    Next

    Me.lblName.Visible = True

    Me.cmdOk.Enabled = True

    Me.cboKategorie.Visible = True

    Me.cboKategorie.ListIndex = 1

    Me.txtName.SetFocus

End Sub
```

Hinter dem Umsatz-Button liegen zwei Funktionen: Die Textfelder werden eingeschaltet, damit die Zielvorgaben für einen Verkäufer geändert werden können, oder die Angaben werden für einen neuen Verkäufer geändert. Im ersten Fall werden alle Felder durchlaufen.

### 2.1.5 Plausibilitätsprüfung

Alle Textfelder außer des Feldes `txtZielNeukunde` werden aktiviert (`Locked = False`) und die Schriftfarbe auf Schwarz gesetzt. Welcher Code sich hinter welcher Farbe verbirgt, kann leicht aus den Eigenschaften herauskopiert werden. Anschließend wird auch die Beschriftung der Befehlsschaltfläche geändert:

```
Private Sub cmdUmsatz_Click()

    Dim i As Integer

    Dim ctl As Control

    On Error Resume Next

    If Me.cmdUmsatz.Caption = "Zielvorgaben ändern" Then

        For Each ctl In Me.Controls

            If Left(ctl.Name, 3) = "txt" And _

                ctl.Name <> "txtZielNeukunde" Then
```

```
            ctl.Locked = False
            ctl.ForeColor = &H80000007
         End If
      Next

      Me.txtZiel.SetFocus
      Me.txtZiel.SelStart = 0
      Me.txtZiel.SelLength = Len(Me.txtZiel.Value)
      Me.cmdUmsatz.Caption = "Zielvorgaben für" & Chr(10) & _
      Me.lstListe.Value & " ändern"
```

**HInweis**

Diese drei Zeilen verwende ich als Standardprüfung für sämtliche Textfelder, in denen Zahlen eingegeben werden sollen. Ich überprüfe in der Reihenfolge:

1. Wurde etwas eingeben
2. Wurde eine Zahl eingegeben (`IsNumeric`), siehe oben?
3. Ist die Zahl plausibel (beispielsweise größer als 0)?

Im zweiten Fall werden zuerst alle Eingaben überprüft. Die Methode `SetFocus` setzt den Cursor in das Eingabefeld. Mit `SelStart = 0` und `SelLength = Len("Inhalt des Textfeldes")` wird ein vorhandener, aber falscher Inhalt markiert.

```
   Else
      If Me.lstListe.ListIndex = -1 Then
          MsgBox "Sie haben keinen Verkäufer ausgewählt!", _
            , "TOSHIBA TEC"
          Exit Sub
      End If

      If Me.txtZiel.Value = "" Then
          MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
          Me.txtZiel.SetFocus
          Exit Sub
      ElseIf IsNumeric(Me.txtZiel.Value) = False Then
          MsgBox "Sie haben keine gültige Zahl eingegeben", , _
            "TOSHIBA TEC"
          Me.txtZiel.SetFocus
          Me.txtZiel.SelStart = 0
          Me.txtZiel.SelLength = Len(Me.txtZiel.Value)
```

```
            Exit Sub
        ElseIf Me.txtZielNewCopierers.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielNewCopierers.SetFocus
            Exit Sub
        ElseIf Me.txtZielReplacement.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielReplacement.SetFocus
            Exit Sub
        ElseIf Me.txtZielService.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielService.SetFocus
            Exit Sub
        ElseIf Me.txtZielService.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielService.SetFocus
            Exit Sub
        ElseIf Me.txtZielGrossMargin.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielGrossMargin.SetFocus
            Exit Sub
        ElseIf Me.txtZielNeukunde.Value = "" Then
            MsgBox "Sie haben kein neues Ziel eingegeben", , "TOSHIBA TEC"
            Me.txtZielNeukunde.SetFocus
            Exit Sub
        ElseIf IsNumeric(Me.txtZielNewCopierers.Value) = False Then
           MsgBox "Bitte geben Sie eine gültige Zahl ein!", _
              vbInformation, "TOSHIBA TEC"
           Me.txtZielNewCopierers.SetFocus
           Me.txtZielNewCopierers.SelStart = 0
           Me.txtZielNewCopierers.SelLength = _
              Len(Me.txtZielNewCopierers.Value)
           Exit Sub
[...]
        End If
```

Auch dies könnte selbstredend eleganter mit einer Schleife gelöst werden:

```
For Each ctl In Me.Controls
   If TypeName(ctl) = "TextBox" Then
      If ctl.Value = "" Then
         MsgBox "Sie haben kein neues Ziel eingegeben"
         ctl.SetFocus
         Exit Sub
      ElseIf IsNumeric(ctl.Value) = False Then
         MsgBox "Bitte geben Sie eine gültige Zahl ein!"
         ctl.SetFocus
         ctl.SelStart = 0
         ctl.SelLen = Len(ctl.Value)
         Exit Sub
      End If
   End If
Next ctl
```

Beachten Sie, dass Sie bei dieser Lösung nicht abfragen dürfen:

```
   If TypeName(ctl) = "TextBox" And ctl.Value = "" Then
```

da beide Teile der If-Verzweigung gleichzeitig abgeprüft werden. Da Bezeichnungsfelder (Labels) keinen Value besitzen, würde der zweite Teil einen Fehler auslösen. Zwei ineinander geschachtelte If-Verzweigungen haben den Vorteil, dass der zweite Teil nur dann aufgerufen wird, wenn der erste Teil mit True abgearbeitet wurde.

Erst nachdem alle möglichen Fehlerquellen überprüft wurden, werden die Werte der Textfelder ausgelesen, in Zahlen vom Wert „Double“ konvertiert und in die entsprechenden Zellen geschrieben:

```
For i = 1 To intZeilen - 1
   If Me.lstListe.Value = xlZelle.Offset(i, 0).Value Then

      xlZelle.Offset(i, 1).Value = CDbl(Me.txtZiel.Value)
      xlZelle.Offset(i, 2).Value = _
         CDbl(Me.txtZielGrossMargin.Value)
      xlZelle.Offset(i, 3).Value = CDbl(Me.txtZielService.Value)
      xlZelle.Offset(i, 4).Value = _
         CDbl(Me.txtZielNewCopierers.Value)
      xlZelle.Offset(i, 5).Value = _
         CDbl(Me.txtZielReplacement.Value)
      If Me.lstListe.Value <> "Niederlassung" Then
         xlZelle.Offset(i, 6).Value = _
```

```
                    CDbl(Me.txtZielNeukunde.Value)

                End If

                Exit For

            End If

        Next

        Me.txtZiel.Value = ""

        Call lstListe_Change

        Me.lstListe.SetFocus

    End If

End Sub
```

Auch hier gilt: Eine Schleife und das Auslagern der Werte in Konstanten erleichtern die Skalierbarkeit des Programms.

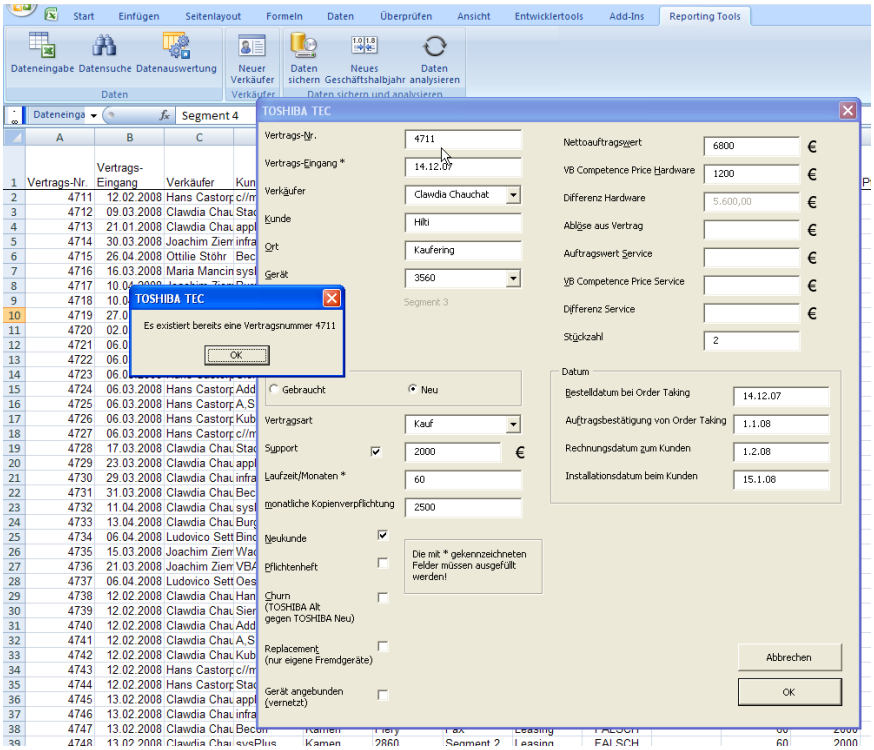



**Abbildung 2.4** Die Daten werden vor dem Eintragen überprüft.

Analog arbeitet der Button „Verkäufer hinzufügen" – zuerst werden die eingegebenen Informationen abgefragt, anschließend die Werte eingetragen:

```
Private Sub cmdOk_Click()

    On Error Resume Next

    If IsNumeric(Me.txtZiel.Value) = False Then

        MsgBox "Bitte geben Sie eine gültige Zahl ein!", _

            vbInformation, "TOSHIBA TEC"

        Me.txtZiel.SetFocus
```

```
      Me.txtZiel.SelStart = 0
      Me.txtZiel.SelLength = Len(Me.txtZiel.Value)
      Exit Sub
   ElseIf IsNumeric(Me.txtZielNewCopierers.Value) = False Then
      MsgBox "Bitte geben Sie eine gültige Zahl ein!", _
         vbInformation, "TOSHIBA TEC"
      Me.txtZielNewCopierers.SetFocus
      Me.txtZielNewCopierers.SelStart = 0
      Me.txtZielNewCopierers.SelLength = _
         Len(Me.txtZielNewCopierers.Value)
      Exit Sub
   ElseIf IsNumeric(Me.txtZielReplacement.Value) = False Then
[...]
   End If

   xlZelle.Offset(intZeilen, 0).Value = Me.txtName.Value
   xlZelle.Offset(intZeilen, 1).Value = CDbl(Me.txtZiel.Value)
   xlZelle.Offset(intZeilen, 1).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 2).Value = CDbl(Me.txtZielGrossMargin.Value)
   xlZelle.Offset(intZeilen, 2).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 3).Value = CDbl(Me.txtZielService.Value)
   xlZelle.Offset(intZeilen, 3).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 4).Value = _
      CDbl(Me.txtZielNewCopierers.Value)
   xlZelle.Offset(intZeilen, 4).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 5).Value = CDbl(Me.txtZielReplacement.Value)
   xlZelle.Offset(intZeilen, 5).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 6).Value = CDbl(Me.txtZielNeukunde.Value)
   xlZelle.Offset(intZeilen, 6).NumberFormatLocal = "#.##0"
   xlZelle.Offset(intZeilen, 7).Value = Me.cboKategorie.ListIndex + 2

   Set xlZelle = Nothing
   Set xlBlatt = Nothing
   Unload Me
End Sub
```

Wenn Sie sehr viele Felder haben, dann können Sie über das Namenspräfix abfragen, ob es sich um ein Textfeld handelt, oder mit Hilfe der Eigenschaft `TypeName`:

```
Dim c As Control
For Each c In Me.Controls
    If TypeName(c) = "TextBox" Then
        MsgBox c.Name
    End If
Next
```

## 2.2 Ein weiteres Bsp zu vielen Steuerelementen: ein Bewertungsformular

Auf einem Bewertungsformular befinden sich mehrere Reihen (Kategorien) von Kontrollkästchen. Sie sind jeweils mit den Notenwerten 1 bis 5 bezeichnet. Der Benutzer kann nun aus jeder Reihe ein Kästchen auswählen oder zwei nebeneinander liegende auswählen (zum Beispiel für die Noten 2 oder 2–3). Es ist aber verboten, aus einer Reihe drei Kästchen anzuklicken, ebenso ist es nicht erlaubt, zwei Kästchen auszuwählen, die nicht nebeneinander liegen.

**Variante 1:**

```
Sub WerteTest()
   Dim intZeile As Integer
   Dim intSpalte As Integer
   intZeile = Mid(Me.ActiveControl.Name, 4, 2)
   intSpalte = Mid(Me.ActiveControl.Name, 6, 2)
   Select Case intSchalter(intZeile)
   Case 2

      If Me.ActiveControl.Value = True Then
         Me.ActiveControl.Value = False
         intWert(intZeile) = intWert(intZeile) + intSpalte
         MsgBox "Es wurden bereits zwei Kästchen angekreuzt!"
         intSchalter(intZeile) = 2
      Else

         intSchalter(intZeile) = 1
         intWert(intZeile) = intWert(intZeile) - intSpalte
      End If
   Case 1
```

```
      If Me.ActiveControl.Value = True Then
            intSchalter(intZeile) = 2
         If Abs(intWert(intZeile) - intSpalte) > 1 Then
            Me.ActiveControl.Value = False
            MsgBox "Dieses Kästchen kann nicht angekreuzt werden!"
            intSchalter(intZeile) = 1
            intWert(intZeile) = intWert(intZeile) + intSpalte
         Else
            intWert(intZeile) = intWert(intZeile) + intSpalte
         End If
      Else
         intSchalter(intZeile) = intSchalter(intZeile) - 1
         intWert(intZeile) = intWert(intZeile) - intSpalte
      End If
   Case 0
      If Me.ActiveControl.Value = True Then
         intWert(intZeile) = intSpalte
         intSchalter(intZeile) = intSchalter(intZeile) + 1
      Else
         intSchalter(intZeile) = intSchalter(intZeile) - 1
         intWert(intZeile) = 0
      End If
   End Select
End Sub
```

Dabei werden global deklariert:

```
Option Explicit
Dim intWert(1 To 6) As Integer
Dim intSchalter(1 To 6) As Integer
```

Diese Prozedur wird von allen Kontrollkästchen aufgerufen:

```
Private Sub chk0101_Click()
   WerteTest
End Sub
Private Sub chk0102_Click()
   WerteTest
End Sub
```

```
Private Sub chk0103_Click()
   WerteTest
End Sub
[...]
```

### Variante 2:

```
Sub WerteTest02()
   Dim intZeile As Integer
   Dim intSpalte As Integer
   Dim bytSchalter As Byte
   Dim ctl As Control

   fMeldungErfolgt = False
   intZeile = Mid(Me.fraGesamt.ActiveControl.Name, 4, 2)
   intSpalte = Mid(Me.fraGesamt.ActiveControl.Name, 6, 2)

   For Each ctl In Me.fraGesamt.Controls
      If TypeName(ctl) = "CheckBox" Then
         If Mid(ctl.Name, 4, 2) = intZeile Then
            If ctl.Value = True Then
               If bytSchalter = 2 Then
                  Me.fraGesamt.ActiveControl.Value = False
                  If fMeldungErfolgt = False Then
                     MsgBox "Es wurden bereits zwei Kästchen angekreuzt!"
                     fMeldungErfolgt = True
                  End If
                  Exit Sub
               ElseIf bytSchalter = 1 Then
                  If Abs(Mid(ctl.Name, 6, 2) - intSpalte) > 1 Then
                     Me.fraGesamt.ActiveControl.Value = False
                     If fMeldungErfolgt = False Then
                        MsgBox "Dieses Kästchen kann " & _
                           nicht angekreuzt werden!"
                        fMeldungErfolgt = True
```

```
                End If

                Exit Sub

            Else

                bytSchalter = 2

            End If

        Else

            bytSchalter = 1

        End If

    End If

End If

Next


End Sub
```

Diese Prozedur benötigt ebenfalls die Deklaration einer globalen Variablen `fMeldungErfolgt`. Sie dient dazu, dass nicht mehrmals ein Meldungsfenster angezeigt wird:

```
Dim fMeldungErfolgt As Boolean
```

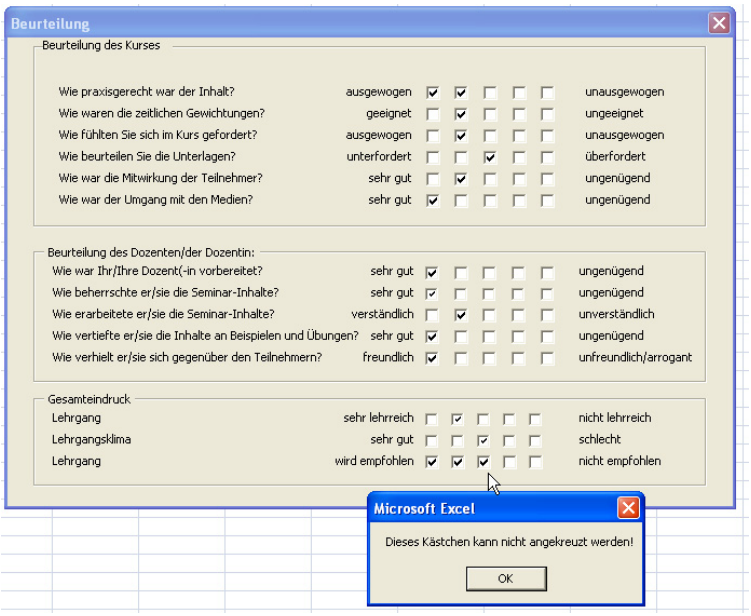



**Abbildung 2.5** Die freie Auswahl wird eingeschränkt.

Und schließlich kann auch diese Prozedur von mehreren Kontrollkästchen aufgerufen werden:

```
Private Sub chk1201_Click()

   WerteTest02

End Sub
```

```
Private Sub chk1202_Click()

   WerteTest02

End Sub


Private Sub chk1203_Click()

   WerteTest02

End Sub

[...]
```

### 2.2.1 Eingaben überprüfen

Sie können auch schon bei der Eingabe überprüfen, ob das, was der Anwender eingegeben hat, plausibel ist. Darf in einem Textfeld beispielsweise nur eine Zahl eingegeben werden, dann wird überprüft, ob der ASCII-Code zwischen dem ASCII-Code für 0 und dem für 9 liegt. Falls nicht, wird überprüft, ob ein Komma eingegeben wurde, was schließlich erlaubt ist. Da jedoch maximal ein Komma pro Zahl erlaubt ist, wird auch dieser Fall geprüft. Im folgenden Beispiel wird die Eingabe eines Punktes durch ein Komma ersetzt:

```
Private Sub txtAblöse_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)

   On Error Resume Next

   If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then

      If KeyAscii = Asc(",") Or KeyAscii = Asc(".") Then

         If InStr(1, Me.txtAblöse.Value, ",") Then

            KeyAscii = 0

         Else

            KeyAscii = Asc(",")

         End If

      Else

         KeyAscii = 0

      End If

   End If

   ' -- nur Zahleneingabe

End Sub
```

Dennoch gelingt es dem Anwender, Texte in das Textfeld einzugeben – die Aktionen Kopieren und Einfügen können nicht deaktiviert werden und werden umgekehrt nicht als KeyPress-Ereignis aufgerufen. Deshalb wird beim Verlassen des Feldes überprüft:

```
Private Sub txtAblöse_Exit(ByVal Cancel As MSForms.ReturnBoolean)

   If txtAblöse.Value <> "" Then

      If IsNumeric(Me.txtAblöse.Value) = False Then

         MsgBox "Bitte nur Zahlen eingeben!", vbInformation, _
```

```
            "TOSHIBA TEC"
         Me.txtAblöse.SelStart = 0
         Me.txtAblöse.SelLength = Len(Me.txtSupport.Value)
         Me.txtAblöse.SetFocus
         Cancel = True
      End If
   End If
   ' -- nur Zahleneingabe
End Sub
```

Und zur Sicherheit wird natürlich hinter der OK-Schaltfläche noch einmal geprüft. Dann kann wirklich (fast) nichts mehr schief gehen.

Ebenso sollten Sie bei Combo-Boxen die Eigenschaft „Style“ von „fmStyleDropDownCombo“ auf „fmStyle-DropDownList“ setzen, wenn Sie nicht möchten, dass der Anwender andere Werte eingibt als diejenigen, die Sie ihm in der Liste zur Verfügung stellen. Oder Sie fangen dies per Programmierung ab – falls der Anwender auch andere Werte eingeben darf. Im folgenden Beispiel wurden zuerst Leerzeichen entfernt (es gibt Benutzer, die tippen konsequent ein Leerzeichen nach jedem Wort!) und anschließend die Frage gestellt, ob das eingegebene Gerät wirklich verwendet werden soll. Falls er sich dafür entscheidet, wird diesem Gerät der Typ „Zubehör“ zugewiesen:

```
   Me.cboGeräte.Value = Trim(Me.cboGeräte.Value)
   If Me.cboGeräte.ListIndex = -1 Then
      If MsgBox("Das von Ihnen eingegebene Gerät" & vbCr & vbCr & _
      vbTab & """" & Me.cboGeräte.Value & """" & vbCr & vbCr & _
      "existiert nicht in der Liste." & vbCr & vbCr & _
      "Möchten Sie es dennoch verwenden?", _
      vbInformation + vbYesNo, "TOSHIBA TEC") = vbYes Then
            Me.cboSegment.Caption = "Zubehör"
      Else
            Me.cboGeräte.SetFocus
            Cancel = True
      End If
   Else
      Me.cboSegment.Caption = _
      ActiveWorkbook.Worksheets("Segment"). _
      Range("F1").Offset(Me.cboGeräte.ListIndex + 1, 0).Value
   End If
```

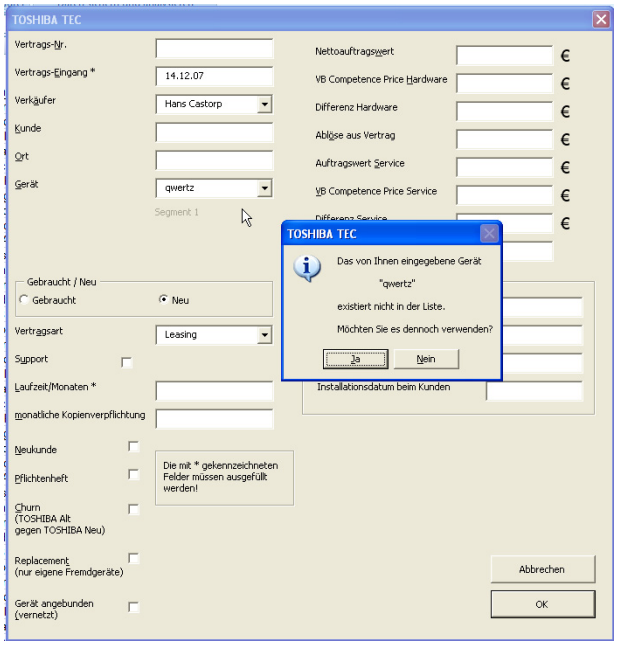



**Abbildung 2.6** Eingaben in einem Kombinationsfeld werden überprüft.

### 2.2.2 Dialog schließen

Manchmal soll beim Schließen eines Dialogs eine bestimmte Aktion ausgeführt werden. Um zu verhindern, dass der Benutzer über das Systemmenü „x“ die Maske schließt, kann dies im Ereignis `QueryClose` abgefangen und unterbunden werden:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   If CloseMode = vbFormControlMenu Then
      MsgBox "Bitte nur über die Schaltfläche ""Abbrechen""" & vbCr & _
      "oder mit der Taste <esc> schließen.", vbInformation, "TOSHIBA TEC"
      Cancel = True
   End If
End Sub
```

## 2.3 Viele „Kleinigkeiten“

### 2.3.1 Mehrspaltige Listenfelder

Wenn Sie in einem Listenfeld oder einem Kombinationsfeld mehrere Spalten anzeigen lassen möchten, dann können Sie dies über einen Array realisieren:

```
Dim strLänder(1 To 10, 1 To 10) As String
   strLänder(1, 1) = "Tschechische Republik": strLänder(1, 2) = "Prag"
   strLänder(2, 1) = "Estland": strLänder(2, 2) = "Tallinn"
   strLänder(3, 1) = "Zypern": strLänder(3, 2) = "Nikosia"
```

```
strLänder(4, 1) = "Lettland": strLänder(4, 2) = "Riga"
strLänder(5, 1) = "Litauen": strLänder(5, 2) = "Vilnius"
strLänder(6, 1) = "Ungarn": strLänder(6, 2) = "Budapest"
strLänder(7, 1) = "Malta": strLänder(7, 2) = "Valetta"
strLänder(8, 1) = "Polen": strLänder(8, 2) = "Warschau"
strLänder(9, 1) = "Slowenien": strLänder(9, 2) = "Ljubljana"
strLänder(10, 1) = "Slowakische Republik"
strLänder(10, 2) = "Bratislava"

Me.cmbListe.ColumnCount = 2
Me.cmbListe.List = strLänder
Me.cmbListe.ListIndex = 0
```

Abgefangen werden sie beispielsweise so:

```
MsgBox "Es wurde Nr. " & Me.cmbListe.ListIndex + 1 & " gewählt:" _
& vbCr & Me.cmbListe.Column(0) & " " & Me.cmbListe.Column(1)
```

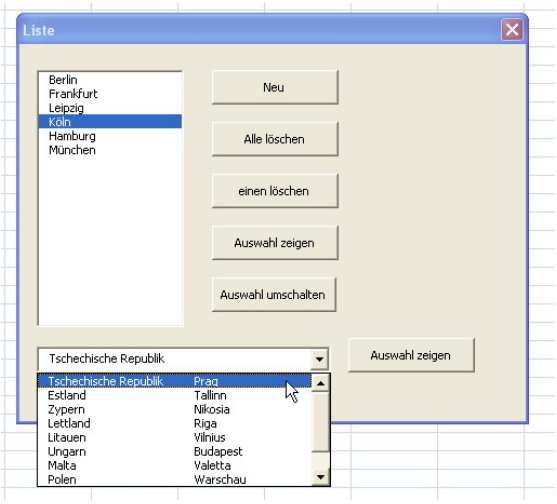



**Abbildung 2.7** Die Auswahl wird getroffen und abgefangen.

### 2.3.2 Mauszeiger auf UserForm

Es gibt sicherlich keine Notwendigkeit, dem Anwender einen andern Mauszeiger zur Verfügung zu stellen als den Standardmauszeiger. Deshalb ändere ich auch nicht den „Mouse-Icon“ und den „MousePointer“ in den Eigenschaften der UserForm. Allerdings ändere ich nicht nur in den Applikationen den Standardmauszeiger und beschrifte die Statuszeile in Excel so:

```
Application.Cursor = xlWait
Application.ScreenUpdating = False
Application.StatusBar = "Makro läuft. Bitte warten Sie ..."
```

in Word folgendermaßen:

```
Application.System.Cursor = wdCursorWait
Application.StatusBar = ""
Application.ScreenUpdating = False
```

sondern zeige auch die „Sanduhr“ auf der UserForm an:

```
        Me.MousePointer = fmMousePointerHourGlass
```

Dies wird natürlich am Ende – wenn das Programm korrekt abgelaufen ist oder wenn ein Fehler aufgetreten ist – wieder zurückgesetzt:

```
    Application.Cursor = xlDefault
    Application.ScreenUpdating = True
    Application.StatusBar = False
    Me.MousePointer = fmMousePointerDefault
```

Übrigens – damit der Benutzer in sehr lang andauernden Programmen sieht, dass etwas passiert, kann mit einer Befehlsschaltfläche ein Fortschrittszeiger simuliert werden:

```
Public Sub VerlaufAnzeigen()
    Const MAXVERLAUF As Double = 114

    With frmAuswertung.cmdVerlauf
       .Visible = True
          sngVerlauf = sngVerlauf + 2.5
          If sngVerlauf >= 100 Then
             .Caption = "99 %"
          Else
             .Width = .Width + MAXVERLAUF / 40
             .Caption = CInt(sngVerlauf) & " %"
          End If
          frmAuswertung.Repaint
    End With
End Sub
```

Gestartet wird das Programm mit:

```
sngVerlauf = 0
Call VerlaufAnzeigen
```

Und an mehreren Stellen im Code wird das Programm angestoßen über:

```
Call VerlaufAnzeigen
```

### 2.3.3 Das Rechenergebnis der Eingabe dynamisch anzeigen

Häufig soll der Benutzer bei der Eingabe von Zahlenwerten schon das Ergebnis in einem geschützten Textfeld oder auf einem Bezeichnungsfeld sehen. Verwenden Sie hierzu das Ereignis `Change`. Dieses Ereignis reagiert auch auf das Löschen von Zeichen. Um mögliche fehlerhafte Eingaben abzufangen, muss darauf reagiert werden. Es ist sicherlich nicht elegant – aber für diesen Zweck völlig ausreichend, mit `On Error Resume Next` auf Fehler in der Eingabe zu reagieren.

```
Private Sub txtAuftragswert_Change()

   On Error Resume Next

   Me.txtDifferenzHardware.Value = _

      Format(CDbl(Me.txtAuftragswert.Value) - _

         CDbl(Me.txtTransferpreis.Value), "#,##0.00")

End Sub
```

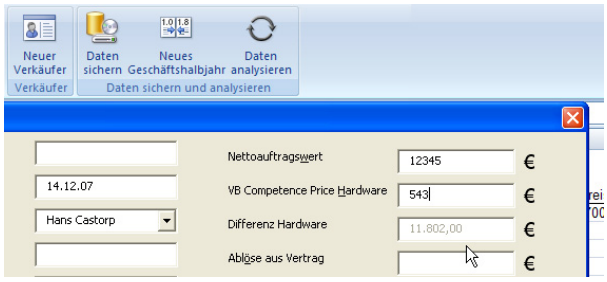



**Abbildung 2.8** Bei der Eingabe der Daten wird das Ergebnis berechnet und dem Benutzer angezeigt.

### 2.3.4 Andere Informationen dynamisch anzeigen

In einer Maske trägt der Benutzer seinen Vor- und Zunamen ein. Daraus sollen in einem Feld seine Initialen und seine E-Mail-Adresse generiert werden. Dies muss bei jeder Eingabe beziehungsweise Änderung der entsprechenden Felder vorgenommen werden:

```
Private Sub txtVorname_Change()

   Call InitialEintragen

End Sub


Private Sub txtZuname_Change()

   Call InitialEintragen

End Sub
```

Die entsprechende Routine erledigt beides. Dabei wird im Namen überprüft, ob sich dort Sonderzeichen befinden, die für die Mail-Adresse nicht zugelassen sind. Sie werden automatisch transformiert – also aus „René“ wird „Rene“, aus „Müller“ „wird „Mueller“ und so weiter.

```
Sub InitialEintragen()

   ' -- diese Prozedur bildet das Initial und erzeugt die Email-Adresse

   Dim strNameGanz As String
```

```
Dim strName() As String
Dim i As Integer
On Error Resume Next

Me.txtInitial.Value = Left(Me.txtVorname.Value, 1) & _
   Left(Me.txtZuname.Value, 1)
' -- die Initialen werden gebildet

strNameGanz = Me.txtVorname.Value & "." & Me.txtZuname.Value
ReDim strName(Len(strNameGanz))

For i = 1 To Len(strNameGanz)
   strName(i) = Mid(strNameGanz, i, 1)
Next

For i = 1 To Len(strNameGanz)
   Select Case Mid(strNameGanz, i, 1)

   ' -- KLeinbuchstaben:
   Case "ä"
      strName(i) = "ae"
   Case "ö", "æ", "ø"
      strName(i) = "oe"
   Case "ü"
      strName(i) = "ue"
   Case "ß"
      strName(i) = "ss"
   Case "ç"
      strName(i) = "c"
   Case "à", "á", "â", "ã", "å"
      strName(i) = "a"
   Case "ë", "é", "è"
      strName(i) = "e"
   Case "ì", "í", "î", "ï"
      strName(i) = "i"
```

```
        Case "ñ"
           strName(i) = "n"
        Case "ò", "ó", "ô", "õ"
           strName(i) = "o"
        Case "ù", "ú", "û"
           strName(i) = "u"
        Case "ÿ", "ý"
           strName(i) = "y"
        Case " "
           strName(i) = "-"

        ' -- Großbuchstaben:
        Case "À", "Á", "Â", "Ã", "Ä", "Å"
           strName(i) = "A"
        Case "Ä", "Æ"
           strName(i) = "Ae"
...
     End Select
     Next

     For i = 1 To Len(strNameGanz)
        strName(0) = strName(0) & strName(i)
     Next

     Me.txtEmail.Value = strName(0)
     If Len(Me.txtEmail.Value) > 40 Then
        Call EmailMehrzeilig
     Else
        Call EmailEinzeilig
     End If
  End Sub
```

Und schließlich gibt es einige Mitarbeiterinnen, die ganz lange Namen haben. Extra für sie wird das Mail-Feld dynamisch vergrößert. Als Obergrenze wurden hier 40 Zeichen angesetzt:

```
  Private Sub EmailMehrzeilig()
     Me.lblEmail.Top = Me.lblInfo.Top
```

```
    Me.lblInfo.Visible = False

    Me.txtEmail.MultiLine = True

    Me.txtEmail.EnterKeyBehavior = True

    Me.txtEmail.Height = 36

End Sub


Private Sub EmailEinzeilig()

    Me.lblEmail.Top = 287

    Me.lblInfo.Visible = True

    Me.txtEmail.MultiLine = False

    Me.txtEmail.EnterKeyBehavior = False

    Me.txtEmail.Height = 18

End Sub
```

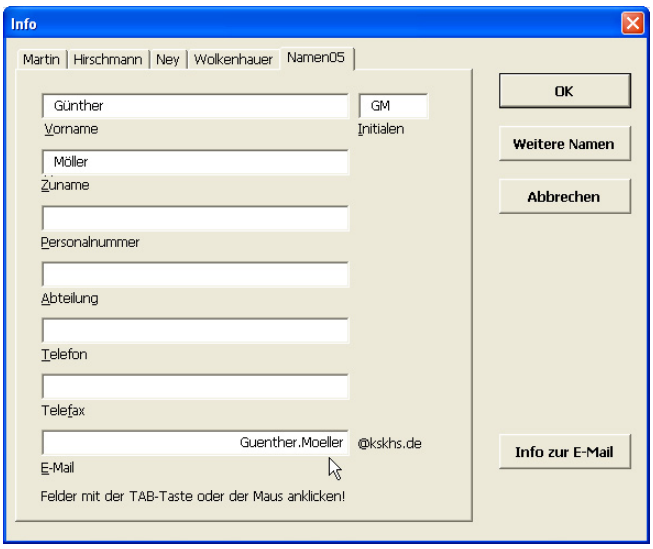



**Abbildung 2.9** Die Initialen und die E-Mail-Adresse werden automatisch generiert.

### 2.3.5 Datenmerken (Suchen/Weitersuchen)

Ein Benutzer gibt auf einer Suchmaske Informationen ein, nach denen gesucht werden soll. Da er eine Schaltfläche „Weitersuchen“ hat, muss der gesuchte Werte zwischengespeichert werden. Sicherlich kann man dieses Problem auch über eine globale Variable oder eine als `Static` deklarierte Variable lösen. Dennoch habe ich mich für einen anderen Weg entschieden – der auch an anderen Stellen verwendet werden kann. Auch wenn ein Steuerelement unsichtbar ist, so kann man ihm dennoch Eigenschaften zuweisen und auslesen. Und genau das wird an dieser Stelle benutzt.

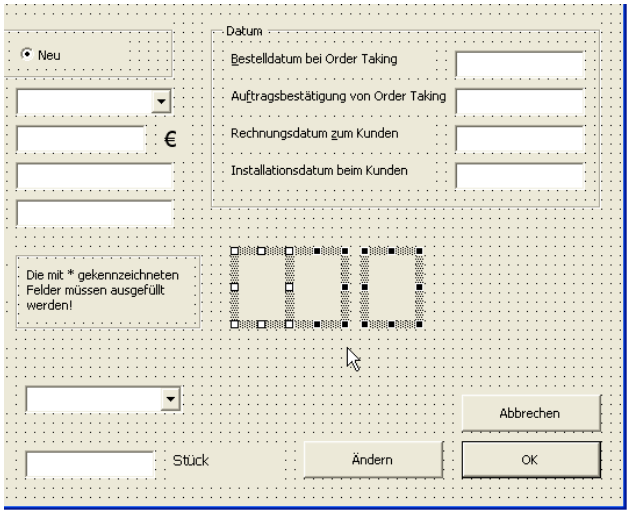



**Abbildung 2.10** Drei unsichtbare Bezeichnungsfelder können mit Informationen gefüllt werden.

Wurde in ein Suchfeld etwas eingegeben, dann wird dieser Suchwert auf das Bezeichnungsfeld geschrieben. Nach ihm kann anschließend gesucht werden:

```
ElseIf Me.cmdOk.Caption = "Suchen" Then
   If Me.txtKunde.Value <> "" Then
      Me.lblKunde.Caption = Me.txtKunde.Value

      For lngZähler = 1 To lngZeilen - 1
         If InStr(xlZelle.Offset(lngZähler, 3).Value, _
            Me.txtKunde.Value) > 0 Then
            Me.lblPos.Caption = lngZähler
            Call ZeigeDaten(lngZähler, xlBlatt)
            ' - die gefundenen Daten werden angezeigt.
            Me.cmdOk.Caption = "Weitersuchen"
            Me.cmdÄndern.Visible = True
            Me.txtVertragsNummer.SetFocus
            Call AlleControlsHer(Me)
            ' -- zeigt alle Felder
            Exit Sub
         End If
      Next
   ElseIf Me.txtVertragsNummer.Value <> "" Or _
      Me.txtVertragseingang.Value <> "" Then
      Me.lblVertragseingang.Caption = Me.txtVertragseingang.Value
```

```
For lngZähler = 1 To lngZeilen - 1
If IsDate(Me.txtVertragseingang.Value) = False Then
   datDatum = Date + 1
Else
   datDatum = CDate(Me.txtVertragseingang.Value)
End If
   If CStr(xlZelle.Offset(lngZähler, 0).Value) = _
      Me.txtVertragsNummer.Value Or _
      xlZelle.Offset(lngZähler, 1).Value = datDatum Then
      Me.lblPos.Caption = lngZähler
      Call AlleControlsHer(Me)
      If Me.txtVertragseingang.Value <> "" Then
         Me.cmdOk.Caption = "Weitersuchen"
      Else
         Me.cmdOk.Visible = False
      End If
      Call ZeigeDaten(lngZähler, xlBlatt)
      Me.cmdÄndern.Visible = True
...
```

**Achtung**

Beachten Sie in diesem Beispiel, dass, wenn Sie Zahl in ein Textfeld oder in ein Bezeichnungsfeld schreiben und es danach wieder auslesen, dass auf den Wert als Zeichenkette (String) zugegriffen wird. Sie sollte mit `CInt`, beziehungsweise `CDbl` diesen in eine Zahl umwandeln.

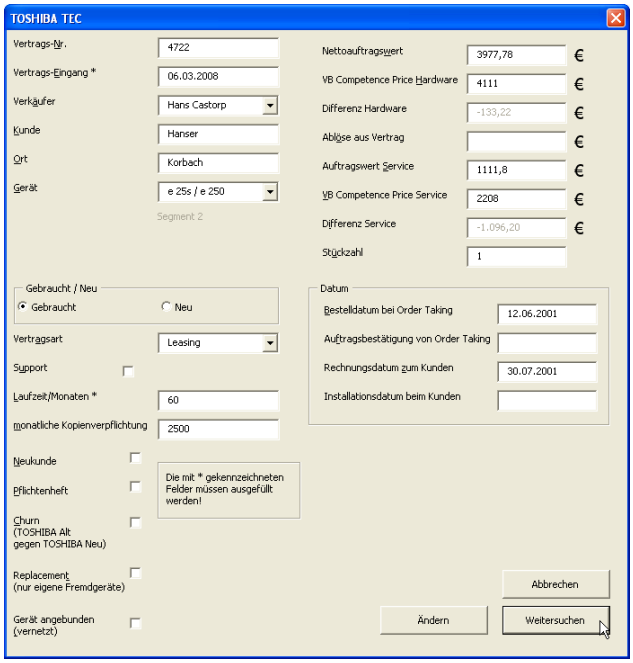



**Abbildung 2.11** Suchbegriffe werden eingegeben, die Werte werden „gemerkt“, und der Benutzer kann weitersuchen.

## 2.3.6 Alberne Spielereien?

### Bilder laden

Angenommen, Sie möchten Bilder (beispielsweise Logos) nicht fest auf einer Maske einfügen, dann können Sie diese auch zur Laufzeit laden lassen. Obwohl in der Liste der Eigenschaften des Bildfeldes `Picture` steht, kann nicht die Eigenschaft

```
imgKunst.Picture = strBildPfad & "Botti.bmp"
```

gesetzt werden. Man muss mit der Methode LoadPicture arbeiten:

```
imgKunst.Picture = LoadPicture(strBildPfad & "Botti.bmp")
```

Auch wenn das nachfolgende Beispiel etwas albern anmutet, so lasse ich häufig Logos von Dateien dynamisch einfügen, die auf der Festplatte liegen, damit die Exceldatei nicht zu groß wird. Dies muss natürlich sauber überprüft werden – am besten mit dem Befehl Dir.

In unserem Beispiel soll ein Klick auf eine Bildbeschriftung (Label) ein anderes Bild laden und den vorhandenen Text ändern.

```
Option Explicit

Dim strInfo(1 To 7) As String

Dim strBildPfad As String


Private Sub lblBildInfo_Click()

   Select Case Left(lblBildInfo.Caption, 5)

   Case "Leona"

      lblBildInfo.Caption = strInfo(2)

      imgKunst.Picture = LoadPicture(strBildPfad & "Botti.bmp")
```

```
      Case "Botti"
         lblBildInfo.Caption = strInfo(3)
         imgKunst.Picture = LoadPicture(strBildPfad & "Michel.bmp")
      Case "Miche"
         lblBildInfo.Caption = strInfo(4)
         imgKunst.Picture = LoadPicture(strBildPfad & "Rembr.bmp")
      Case "Rembr"
         lblBildInfo.Caption = strInfo(5)
         imgKunst.Picture = LoadPicture(strBildPfad & "VanGogh.bmp")
      Case "Van G"
         lblBildInfo.Caption = strInfo(6)
         imgKunst.Picture = LoadPicture(strBildPfad & "Seurat.bmp")
      Case "Seura"
         lblBildInfo.Caption = strInfo(7)
         imgKunst.Picture = LoadPicture(strBildPfad & "Monet.bmp")
      Case Else
         lblBildInfo.Caption = strInfo(1)
         imgKunst.Picture = LoadPicture(strBildPfad & "Vinci.bmp")
      End Select
   End Sub

   Private Sub UserForm_Initialize()
      strBildPfad = "C:\Eigene Dateien\Eigene Bilder\"

      strInfo(1) = "Leonardo da Vinci" & vbCr & vbCr & _
      "ital. Maler, Bildhauer, Architekt, Kunsttheoretiker " & _
      "Naturforscher u. Mechaniker" & vbCr & _
      "*1452-1519; zuerst Schüler Verocchios in Florenz, " & _
      " dann (1482-99) in Mailand am Hofe, nach Etappen in " & _
      "Florenz, Mailand und Rom; zuletzt in Frankreich; " & _
      "universales Genie der Renaissance. In seinen Gemälden " & _
      "verband L. Körper und Raum durch Umrisse verschleiernde" & _
      " Lichtwirkung (sfumato). Hptw.: ""Abendmahl"", " & _
      """Mona Lisa"" und ""Anna Selbdritt"""
```

```
   strInfo(2) = "Botticelli, Sandro" & vbCr & vbCr & _
   "ital. Maler *1444 - 1510;" & vbCr & _
   "neben wichtigen Porträts " & _
   "religiöse und allegorische Bilder. Hauptw.: " & _
   """Frühling"", ""Geburt d. Venus"", Madonnen, Fresken in" & _
   " der Sixtinischen Kapelle."

   strInfo(3) = "Michelangelo Buonarotti" & vbCr & vbCr & _
[...]
   strInfo(7) = "Manet, Edouard" & vbCr & vbCr & _
[...]
   imgKunst.Picture = LoadPicture(strBildPfad & "Vinci.bmp")
   lblBildInfo.Caption = strInfo(1)
End Sub
```

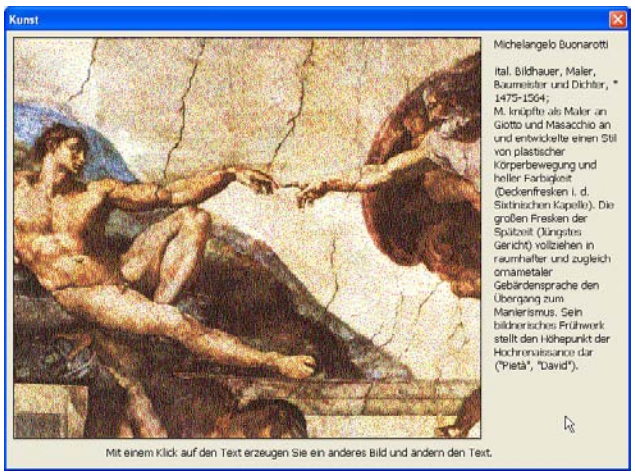



**Abbildung 2.12** Verschiedene Bilder werden geladen.

Mit dieser Technik kann eine kleine Bildvorschau eines Ordners gebaut werden:

```
Option Explicit
Option Compare Text
Dim i As Integer
Dim strPfad As String
Dim strBilder() As String
Private Sub cmdWeiter_Click()
   On Error Resume Next
   If i = UBound(strBilder) Then
      i = 0
```

```
   Else
      i = i + 1
   End If

   Me.imgBild.Picture = LoadPicture(strPfad & strBilder(i))

End Sub

Private Sub cmdZurück_Click()
   If i = 0 Then
      i = UBound(strBilder)
   Else
      i = i - 1
   End If

   Me.imgBild.Picture = LoadPicture(strPfad & strBilder(i))

End Sub

Private Sub UserForm_Initialize()
   On Error Resume Next
   Dim strBildname As String

   strPfad = "D:\Eigene Dateien\Eigene Bilder\Renes Lieblingsbilder\"
   strBildname = Dir(strPfad)
   i = 0
   Do While strBildname <> ""
      If strBildname <> "." And strBildname <> ".." Then
         Select Case Right(strBildname, 3)
            Case "bmp", "jpg", "gif", "tif", "ico", "pcx", "bmp", "wmf"
               If (GetAttr(strPfad & strBildname) And vbNormal) = _
                  vbNormal Then
                  ReDim Preserve strBilder(i)
                  strBilder(i) = strBildname
                  i = i + 1
```

```
                End If
            End Select
        End If
        strBildname = Dir
    Loop

    If i = 0 Then
        Me.imgBild.Visible = False
        Me.cmdWeiter.Visible = False
        Me.cmdZurück.Visible = False
    Else
        i = 0
        Me.imgBild.Picture = LoadPicture(strPfad & strBilder(i))
    End If
End Sub
```

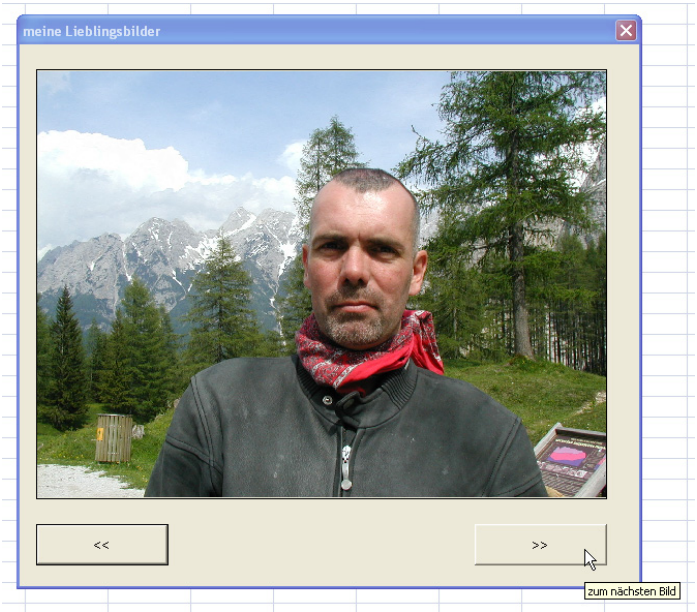



**Abbildung 2.13** Der Bilderdialog

Klickt der Benutzer im ersten Beispiel mit gedrückter <STRG>-Taste auf ein Bild, dann erscheinen Kommentare zum Bild, bei gedrückter <SHIFT>-Taste der Name des Malers und bei <ALT> das Erstellungsjahr. Werden mehrere Tasten gedrückt, dann erfolgt ein Hinweis, dass nur eine Taste zu drücken ist.

```
Private Sub imgKunst_MouseDown _
    (ByVal Button As Integer, _
    ByVal Shift As Integer, _
    ByVal X As Single, ByVal Y As Single)
```

```
    Select Case Shift
    Case 0
    Case 1
       MsgBox "Leonardo da Vinci"
    Case 2
       MsgBox "Die einen halten sie für einen Ausdruck " & _
       "kosmischer Sanftmut und Güte, die anderen halten " & _
       "das Gemälde für puren Kitsch."
    Case 4
       MsgBox "Die Mona Lisa wurde 1503 gemalt."
    Case Else
       MsgBox "Bitte nur eine Taste drücken!"
    End Select
End Sub
```

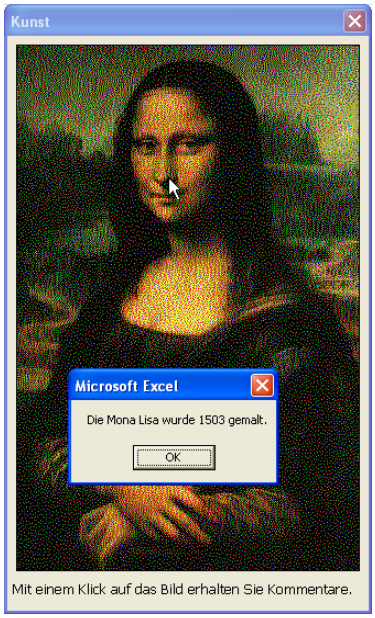



**Abbildung 2.14** Auch der Mausklick mit <SHIFT>, <STRG> oder <ALT> kann abgefangen werden.

Übrigens scheint VBA an dieser Stelle einen Bug zu haben: Klickt der Benutzer auf das Bild, dann wird zwar der Text geändert, allerdings nicht mehr das Bild!

### 2.3.7 UserFormen vergrößern und verkleinern

Auf einer Userform befindet sich eine Befehlsschaltfläche mit der Beschriftung „Erweitern“. Klickt der Benutzer auf diese Schaltfläche, dann wird die Form größer, und der Text ändert sich in „Verkleinern“. Ein weiterer Klick verkleinert die Form und ändert den Text erneut in „Vergrößern“.

```
Dim dblFormgröße As Double
```

```
Private Sub UserForm_Initialize()

   dblFormgröße = Me.Height

   Me.cmdErweitern.Caption = "Erweitern"

End Sub


Private Sub cmdErweitern_Click()

   If Me.Height = dblFormgröße Then

      Me.Height = dblFormgröße * 1.5

      Me.cmdErweitern.Caption = "Reduzieren"

   Else

      Me.Height = dblFormgröße

      Me.cmdErweitern.Caption = "Erweitern"

   End If

End Sub
```

Sicherlich kennen Sie dieses Beispiel von Word – dort kann der Anwender im Suchendialog Teile ein- und ausblenden:

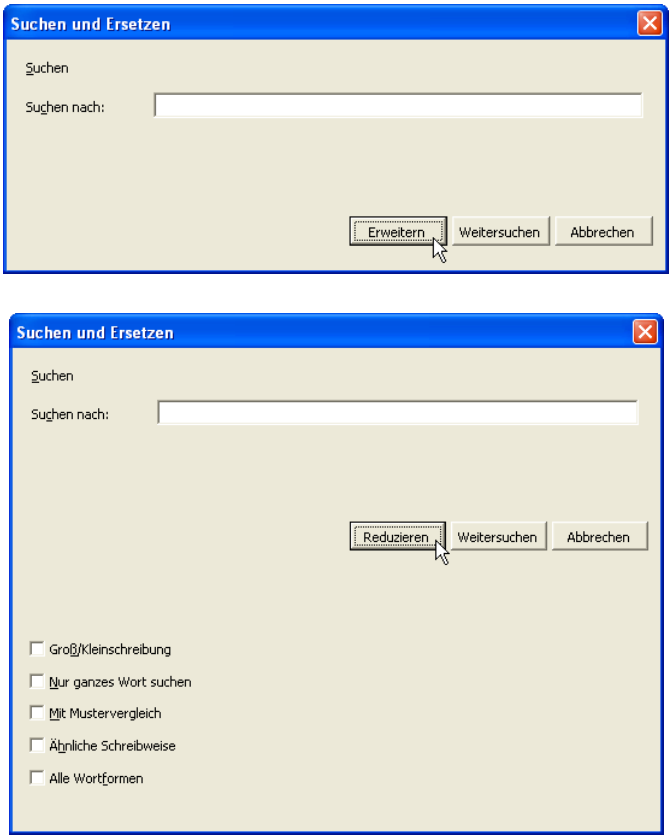



**Abbildung 2.15** Mit „Erweitern“ bzw. „Reduzieren“ kann der Dialog vergrößert und verkleinert werden.

**Beispiel**

Diese Technik habe ich mir zunutze gemacht, um auf einem Dialog die maximale Anzahl Elemente in einem Listenfeld anzeigen zu können. Es benötigt einige Versuche und ein wenig Algebra, aber damit kann das Listenfeld auf dem Dialog vergrößert oder verkleinert werden – je nach Anzahl der angezeigten Ordner:

```
Public Sub ListeVerändernMitUnterverzeichnis()

   Dim intZeilenOben As Integer

   ' -- Wenn Unterverzeichnisse vorhanden, wird diese Prozedur aufgerufen

   ' -- und vergrößert oder verkleinert die Liste und zeigt die

   ' -- Unterverzeichnisse an

   On Error Resume Next


   intZeilenOben = St.Zeilenoben(strStandort)


   With frmAbteilungen

      If .fraUnterverzeichnisse.Visible = True Then

         .fraHauptgruppen.Height = 81 - (5 - intZeilenOben) * 15

         .fraUnterverzeichnisse.Top = 85 - (5 - intZeilenOben) * 15

         .lstListe.Top = 116 - (5 - intZeilenOben) * 15

         .lstListe.Height = 359 - 115 + (5 - intZeilenOben) * 15

      Else

         .lstListe.Top = 86 - (5 - intZeilenOben) * 15

         .lstListe.Height = 359 - 85 + (5 - intZeilenOben) * 15

      End If

   End With

End Sub
```

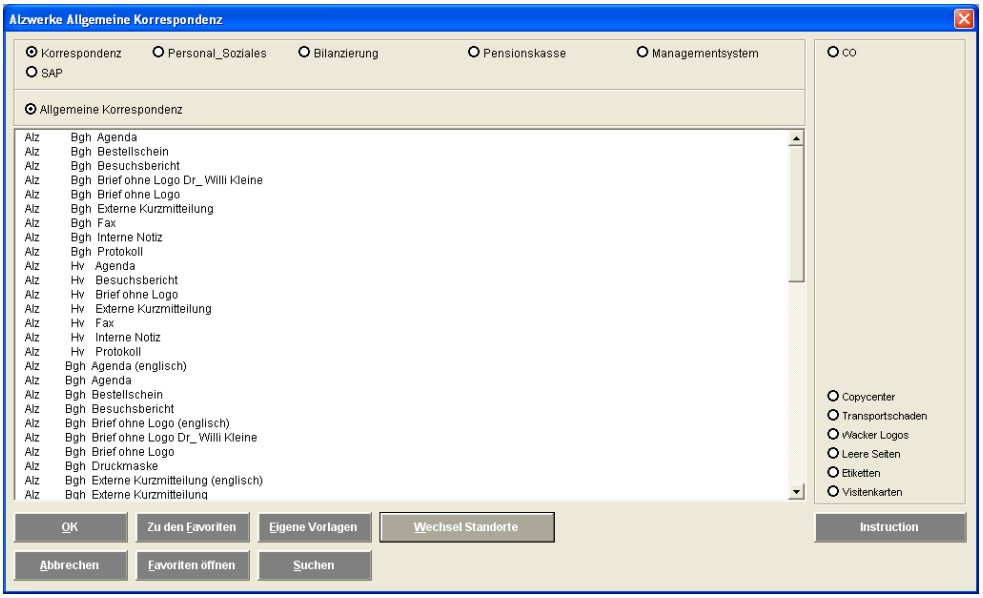



**Abbildung 2.16** Es werden nur zwei Zeilen im ersten Feld angezeigt …

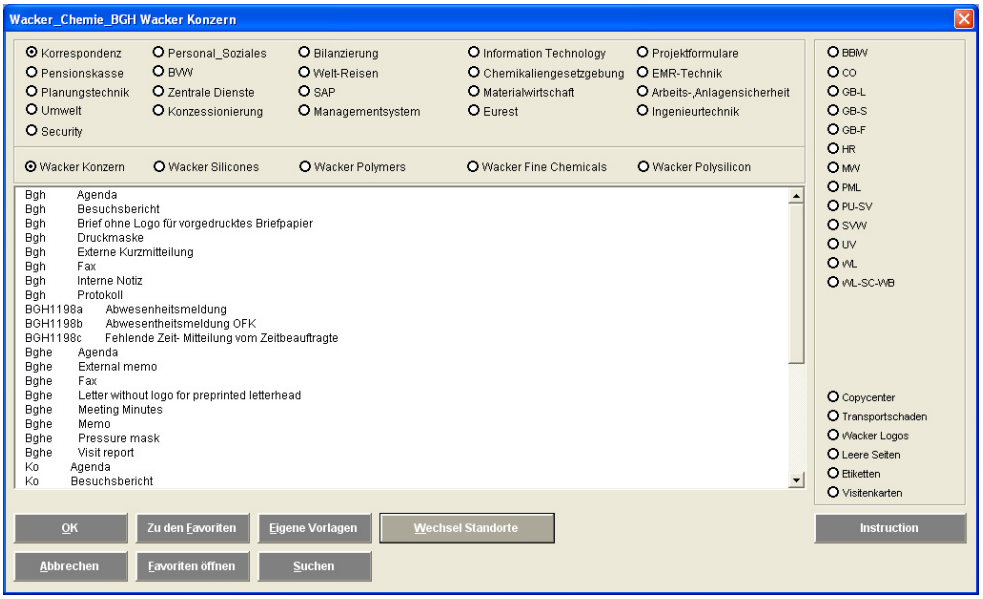



**Abbildung 2.17** … oder fünf.

## Die Farbe der Beschriftung ändern

Die Farbe der Beschriftung eines Bezeichnungsfeldes ändert sich, wenn der Mauszeiger darüber fährt.

```
Private Sub lblBezeichnung_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
      Me.lblBezeichnung.ForeColor = &HFF0000
End Sub
```

```
Private Sub UserForm_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
      Me.lblBezeichnung.ForeColor = &H80000012
End Sub
```

Dies hat Microsoft selbst in Access 97 verwendet – danach ist es allerdings verschwunden und wird bei Microsoft in keiner Anwendung mehr benutzt.

### Hüpfende Buttons (ein alberner Scherz)

Auf einem Formular befindet sich eine Befehlsschaltfläche. Versucht der Benutzer darauf zu klicken, dann springt sie zur Seite. Versucht er, auf die neue Schaltfläche zu klicken, dann sitzt sie wieder an der alten Position.

#### Variante 1:

```
Private Sub UserForm_Initialize()
   cmdGehaltserhöhung1.Visible = True
   cmdGehaltserhöhung2.Visible = False
End Sub

Private Sub cmdGehaltserhöhung1_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
   cmdGehaltserhöhung1.Visible = False
   cmdGehaltserhöhung2.Visible = True
End Sub

Private Sub cmdGehaltserhöhung2_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
   cmdGehaltserhöhung1.Visible = True
   cmdGehaltserhöhung2.Visible = False
End Sub
```

#### Variante 2:

```
Private Sub cmdGehaltserhöhung3_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
```

```
    If cmdGehaltserhöhung3.Left = 12 Then
       cmdGehaltserhöhung3.Left = 120
    Else
       cmdGehaltserhöhung3.Left = 12
    End If
End Sub
```

**Variante 3:**

```
Private Sub cmdGehaltserhöhung4_MouseMove _
   (ByVal Button As Integer, ByVal Shift As Integer, _
   ByVal X As Single, ByVal Y As Single)
   cmdGehaltserhöhung4.Move Left:=132 - cmdGehaltserhöhung4.Left
End Sub
```

Auch wenn dies ein alberner Scherz ist, den man häufig im Internet sehen kann – man kann mit Hilfe dieses Beispiels hervorragend die Begriffe „Ereignis“, „Eigenschaft“ und „Methode“ erläutern und auch zeigen, dass Eigenschaften und Methoden manchmal gar nicht weit auseinander liegen.

### 2.3.8 Steuerelemente dynamisch erzeugen

Bislang wurde die Dynamik in Dialogboxen dadurch erzeugt, dass Eigenschaften von vorhandenen Steuerelementen geändert wurden. Zum Beispiel ändert ein Klick auf eine Befehlsschaltfläche die Eigenschaft Größe (`Height`) der Dialogbox. Das führt dazu, dass ein größerer Teil angezeigt wird. Im Folgenden wird erläutert, wie ein Ereignis (zum Beispiel ein Klick auf eine Befehlsschaltfläche) nicht die Eigenschaft eines vorhandenen Objekts ändert, sondern ein neues Steuerelement erzeugt. Das folgende Dialogblatt besteht neben zwei Beschriftungen und einem Textfeld aus drei Befehlsschaltflächen. Eine der Befehlsschaltfläche wird zur Laufzeit beschriftet:

```
Private Sub UserForm_Initialize()
   Me.cmdAnzeige.Caption = "Erweitern"
End Sub
```

Beim Vergrößern der Userform werden neue Steuerelemente erzeugt. Das neue Steuerelement muss deklariert werden. Da mehrere Ereignisse darauf zugreifen, wird es vor der ersten Prozedur deklariert:

```
Dim NeuesSteuerelement As Control
```

Tabelle 2.1 Die neuen Kontrollkästchen (Chk1–Chk5) besitzen folgende Eigenschaften:

| Name | High | Left | Top | Width |
|---|---|---|---|---|
| Chk1 | 15 | 10 | 180 | 140 |
| Chk2 | 15 | 10 | 200 | 140 |
| Chk3 | 15 | 10 | 220 | 140 |
| Chk4 | 15 | 10 | 240 | 140 |
| Chk5 | 15 | 10 | 260 | 140 |

```
Private Sub cmdAnzeige_Click()
   If Me.Height = 165 Then
      Me.Height = 265
      Me.cmdAnzeige.Caption = "Reduzieren"

      Set NeuesSteuerelement = Controls.Add("forms.checkbox.1")
         With NeuesSteuerelement
            .Left = 10
            .Top = 180
            .Width = 140
            .Height = 15
         End With
   Else
      Me.Height = 165
      Me.cmdAnzeige.Caption = "Erweitern"
   End If
End Sub
```

Statt des Namens des Dialogblatts muss `Forms` stehen; das zugehörige Objekt (die Eigenschaft) ist eine Textbox, die Anzahl ist 1.

Natürlich sollte dem Kontrollkästchen eine Beschriftung zugewiesen werden:

```
[...]
        .Caption = "Groß-/Kleinschreibung"
        .Accelerator = "ß"
[...]
```

Besser wäre es, dem neuen Objekt einen selbst definierten Namen zu geben. Also:

```
Set NeuesSteuerelement = Controls.Add("forms.Checkbox.1", "Chk1")
```

Über einen global deklarierten Zähler `intZähler` können weitere Steuerelemente hinzugefügt werden:

```
Dim NeuesSteuerelement As Control
Dim intZähler As Integer

Private Sub cmdAnzeige_Click()
   If Me.Height = 165 Then
      Me.Height = 325
      Me.cmdAnzeige.Caption = "Reduzieren"
```

```
        For intZähler = 1 To 5
        Set NeuesSteuerelement = _
        Controls.Add("forms.checkbox.1", "Chk" & intZähler)
        With NeuesSteuerelement
           .Left = 10
           .Top = 160 + Zähler * 20
           .Width = 140
           .Height = 15
           Select Case intZähler
              Case 1
                 .Caption = "Groß-/Kleinschreibung"
                 .Accelerator = "ß"
              Case 2
                 .Caption = "Nur ganzes Wort suchen"
                 .Accelerator = "N"
              Case 3
                 .Caption = "Mit Mustervergleich"
                 .Accelerator = "M"
              Case 4
                 .Caption = "Ähnliche Schreibweise"
                 .Accelerator = "h"
              Case 5
                 .Caption = "Alle Wortformen"
                 .Accelerator = "f"
              End Select
           End With

        Next intZähler
     Else
  [...]
```

Die Anzahl der Steuerelemente auf dem Dialogblatt beträgt:

```
  Controls.Count
```

Das Alternativereignis der Befehlsschaltfläche soll nun diese wieder löschen:

```
  Else
     For intZähler = 1 To 5
```

```
      Me.Controls.Remove "Chk" & intZähler
   Next intZähler

   MsgBox "Alle weg", vbCritical

   Me.Height = 165
   Me.cmdAnzeige.Caption = "Erweitern"
End If
```

Das Meldungsfenster dient zum Unterbrechen des Codes, um zu zeigen, dass die Steuerelemente wirklich gelöscht wurden. Und schließlich kann über eine Schaltfläche abgefragt werden, welche Kontrollkästchen angekreuzt wurden:

```
Private Sub cmdWeitersuchen_Click()
   Dim i As Integer

   ausgabetext = ""

   For i = 0 To Controls.Count - 1
      If Left(Controls(i).Name, 3) = "Chk" Then
         If Controls(i).Value = True And Controls(i).Visible = True Then
            ausgabetext = ausgabetext & vbCr & Controls(i).Caption
         End If
      End If
   Next
   If ausgabetext = "" Then
      MsgBox "Es wurde nichts angeklickt"
   Else
      MsgBox "Folgende Kontrollkästchen sind angeklickt:" _
        & vbCr & ausgabetext
   End If
End Sub
```

Beim Initialisieren einer Userform werden ihr zwei Textfelder, zwei Bezeichnungsfelder und zwei Befehlsschaltflächen hinzugefügt.

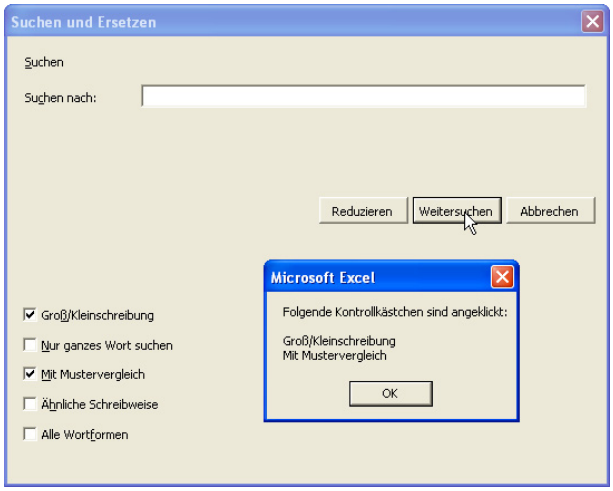



**Abbildung 2.18** Eine dynamisch erzeugte Userform

### 2.3.9 Multiseiten mit neu programmierten Steuerelementen

Ein Multiseiten-Blatt besteht aus zwei Seiten. Auf der ersten Seite befinden sich einige Steuerelemente. Klickt der Benutzer auf das zweite Blatt, dann werden einige Optionsfelder hinzugefügt. Eine Schaltfläche fragt nun die einzelnen Optionen ab.

Damit man einem Steuerelement ein Ereignis zuweisen kann, muss dieses global mit `WithEvents` deklariert werden. Beispielsweise so:

```
Dim WithEvents ctlOk As CommandButton
```

Die Namen der Steuerelemente werden so definiert. Und so können ihnen nun die bekannten Ereignisse hinzugefügt werden. Dabei ist zu beachten, dass auf andere Steuerelemente, die zu Beginn noch nicht existieren, nicht mit

```
Me.txtName.Value
```

zugegriffen werden kann, sondern nur mit:

```
Me.Controls("txtName").Value
```

Im Ereignis `Initialize` der Userform werden Textfelder und Bezeichnungsfelder hinzugefügt. Dabei sollten die wichtigsten Eigenschaften gesetzt werden:

```
Private Sub UserForm_Initialize()
   Dim ctl As Control

Set ctl = Controls.Add("forms.textbox.1", "txtName", True)
   With ctl
      .Left = 10
      .Top = 5
      .Width = 100
      .Height = 20
   End With
```

```
Set ctl = Controls.Add("forms.label.1", "lblName", True)
   With ctl
      .Left = 10
      .Top = 30
      .Caption = "Name"
      .Accelerator = "N"
   End With
[...]
```

Damit man einem Steuerelement ein Ereignis zuweisen kann, muss dieses global mit `WithEvents` deklariert werden:

```
Option Explicit
Dim WithEvents ctlAbbrechen As CommandButton
Dim WithEvents ctlOk As CommandButton
Dim WithEvents ctlAlter As TextBox
Private Sub UserForm_Initialize()
[...]
   Set ctlAlter = Controls.Add("forms.textbox.1", "txtAlter", True)
   With ctlAlter
      .Left = 10
      .Top = 60
      .Width = 100
      .Height = 20
   End With

   Set ctl = Controls.Add("forms.label.1", "lblAlter", True)
   With ctl
      .Left = 10
      .Top = 85
      .Caption = "Alter"
      .Accelerator = "A"
   End With

   Set ctlAbbrechen = Controls.Add("forms.commandbutton.1", _
      "cmdAbbrechen", True)
   With ctlAbbrechen
```

```
        .Left = 100
        .Top = 120
        .Caption = "Abbrechen"
        .Accelerator = "A"
        .Width = 60
        .Height = 20
        .Cancel = True
    End With

    Set ctlOk = Controls.Add("forms.commandbutton.1", _
        "cmdOk", True)
    With ctlOk
        .Left = 170
        .Top = 120
        .Caption = "Ok"
        .Accelerator = "O"
        .Width = 60
        .Height = 20
        .Default = True
    End With
End Sub
```

Die Namen der Steuerelemente sind definiert. Und so können ihnen nun die bekannten Ereignisse hinzugefügt werden. Dabei ist zu beachten, dass auf andere Steuerelemente, die zu Beginn noch nicht existieren, nicht mit

```
Me.txtName.Value
```

zugegriffen werden kann, sondern nur mit:

```
Me.Controls("txtName").Value
```

Im Folgenden drei Ereignisse von drei der Steuerelemente:

```
Private Sub ctlAbbrechen_Click()
    End
End Sub

Private Sub ctlOk_Click()
    MsgBox Me.Controls("txtName").Value & " ist " & _
    Me.Controls("txtAlter").Value & " Jahre alt."
End Sub
```

```
Private Sub ctlAlter_KeyPress _
   (ByVal KeyAscii As MSForms.ReturnInteger)
   If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
      KeyAscii = 0
   End If
End Sub
```

### 2.3.10 Das Multiseiten-Objekt (allgemein)

Das Multiseiten-Objekt trägt den Namen `tabNamen`. Es besitzt einige Steuerelemente. Klickt der Benutzer auf die Schaltfläche, wird ein neues Blatt hinzugefügt.

**Hinweis**
Die Nummerierung der Blätter beginnt bei 0!

```
Private Sub cmdMehrNamen_Click()
   Dim intTabs As Integer
   On Error Resume Next

   intTabs = Me.tabNamen.Tabs.Count
   ' -- zähle die vorhandenen Blätter
   Me.tabNamen.Tabs.Add ' -- füge ein neues hinzu
   Me.tabNamen.Tabs(intTabs).Caption = _
      "Namen" & Format(intTabs + 1, "00")
   ' -- die Beschriftung des neuen Blatts (vorläufig)
   Me.txtVorname.SetFocus
   ' -- setze den Focus auf das Vornamen-Feld
   Me.tabNamen.Value = intTabs
   ' -- springe das neue Blatt an
End Sub
```

Wechselt der Benutzer zwischen den Blättern, dann wird auf das Ereignis „Change“ reagiert, die neuen Benutzerinformationen werden gespeichert und die alten geladen. Hier wird mit einer Klasse gearbeitet:

```
Private Sub tabNamen_Change()
   Dim strTab As String
   Dim strSpeicherort As String
   On Error Resume Next

   strTab = Me.lblNamen.Caption
   ' -- greife den letzten Namen
```

```
    Set Info = New clsInfo
    strSpeicherort = Info.BenutzervorlagenSpeicherort & "KSKInfo.ini"

    ' -- trage den letzten Namen ein:
    Info.WertEintragen strSpeicherort, strTab, "Vorname", _
       frmInfo.txtVorname.Value
    Info.WertEintragen strSpeicherort, strTab, "Zuname",
       frmInfo.txtZuname.Value
...
    ' -- lies Namen aus
    frmInfo.txtVorname.Value = Info.Vorname
    frmInfo.txtZuname.Value = Info.Zuname
...
End Sub
```

Das Gleiche wird natürlich über die OK-Schaltfläche erledigt – mit Hilfe werden ebenfalls die Daten permanent gespeichert. Auf einige Besonderheiten wurde bereits eingegangen – die Initialen werden automatisch generiert, ebenso wie die E-Mail-Adresse. Und beim Start wird der erste Name als Standard angezeigt.

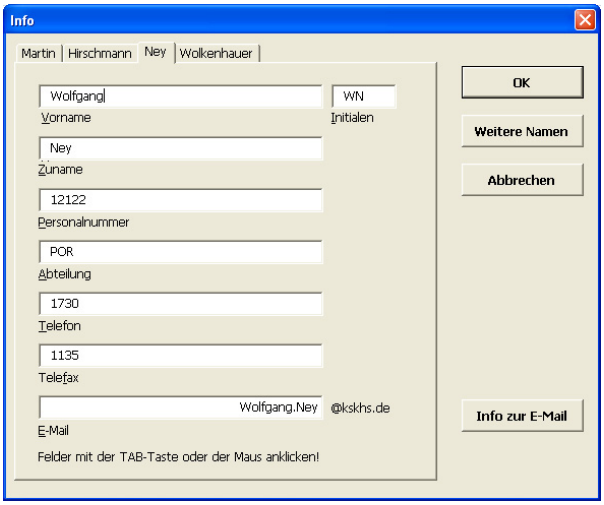



**Abbildung 2.19** Beliebig viele Namen können hinzugefügt werden.

## 2.4 Excel-Dialoge

Excel stellt die Möglichkeit zur Verfügung, auf deren Standarddialoge zuzugreifen. Die Kollektion „Dialogs“ ist eine Eigenschaft von „Application“. Nach Eingabe der Klammer erhält man die komplette Liste aller vordefinierten Dialoge. Zum Anzeigen wird die Methode `Show` verwendet:

```
Application.Dialogs(xlDialogAlignment).Show
```

**Achtung**

Sollen Werte voreingestellt werden, so muss man herausfinden, wie diese Optionen heißen. Die AutoComplete-Liste verrät darüber nichts. In der Hilfe findet sich leider keine Referenz über alle Optionen. Man muss ein wenig probieren

In Excel kann ebenfalls abgefangen werden, ob OK oder Abbrechen gedrückt wurde. Dies entspricht den Werten `True` und `False`.

Dialoge können sicherlich an bestimmte Funktionalitäten gebunden werden, wie beispielsweise an Schaltflächen von selbst erstellten Dialogboxen oder an bestimmte Ereignisse. Ebenso können die Standardeinstellungen verändert werden. Sie werden über eine Argumenten liste hinter der Methode Show eingegeben.

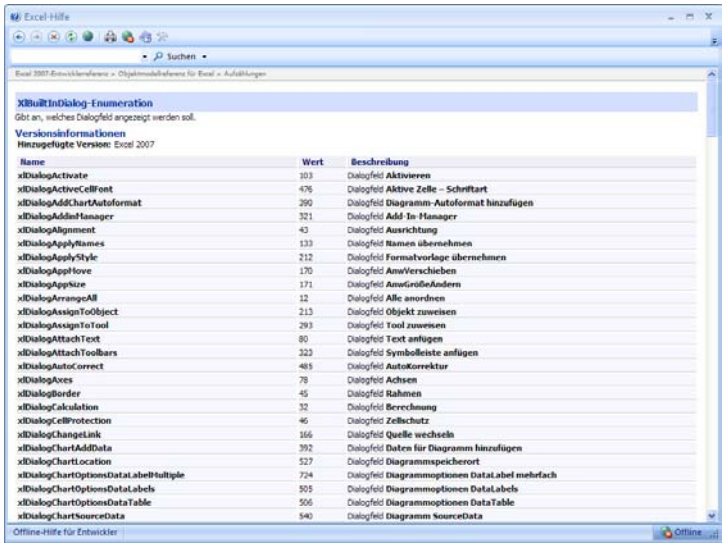



**Abbildung 2.20** Die Liste der Dialoge

```
Dim dlg As Dialog

Set dlg = Application.Dialogs(xlDialogFormatFont)

dlg.Show arg1:="Fraktur BT", arg2:=27, arg3:=True
```

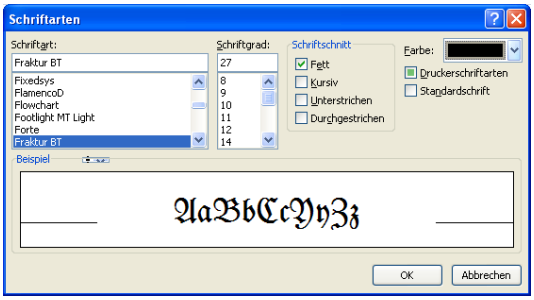



**Abbildung 2.21** Der Dialog mit den Voreinstellungen wird gestartet.

So kann jeder Dialog, der über ein Icon eines Symbols der Ribbons aufgerufen wird, beziehungsweise bis Office 2003 hinter einem Menüpunkt steht, umbelegt werden.

**Hinweis**

Weitere Informationen zu den Dialogen finden Sie im Kapitel Workbook. Dort wird der Speichern-, Öffnen-, Löschen- und Drucken-Dialog beschrieben.

## 2.5 Überwachte Ordner

**Beispiel**

In einer Firma soll in Excel überwacht werden, ob eine Datei in einen bestimmten Ordner gespeichert wird. Falls dies der Fall ist, dann wird eine Maske angezeigt, mit deren Hilfe der Anwender Informationen über seinen Bericht in einer Datenbank speichern kann. Da der Benutzer sowohl „speichern“ als auch den Menüpunkt „Speichern unter“ wählen kann, müsste beides abgefangen werden – beides entspricht jedoch dem Befehl `WorkbookBeforeSave`.

```
Public WithEvents xlApp As Application



Private Sub xlApp_WorkbookBeforeSave(ByVal Wb As Workbook,
   ByVal SaveAsUI As Boolean, Cancel As Boolean)
   Dim dlg As Dialog
   If InStr(1, Wb.FullName, ":\") > 0 Then
      Wb.Save
   Else
      Call DateiSpeichernunter
   End If
End Sub
```

Da der Benutzer auch im Nachhinein die Maske aufrufen kann, wird natürlich noch eine weitere Prozedur eingefügt, mit deren Hilfe er die Maske starten kann. Und es wird überprüft, ob die Datei bereits gespeichert wurde, was man an den Benutzereigenschaften feststellen kann.

```
   Private Function GibtEsVariable() As Boolean
       ' -- diese Funktion überprüft, ob das aktuelle Dokument eine
       ' -- Dokumenteneigenschaft hat
       ' -- und folglich bereits in die Datenbank eingetragen wurde
       Dim i As Integer
       Dim fSchalter As Boolean

       fSchalter = False
       For i = 1 To Application.ActiveWorkbook. _
           CustomDocumentProperties.Count
           If Application.ActiveWorkbook.CustomDocumentProperties(i). _
               Name = "Datensatznummer" Then
               fSchalter = True
```

```
            End If

        Next

        GibtEsVariable = fSchalter

    End Function
```

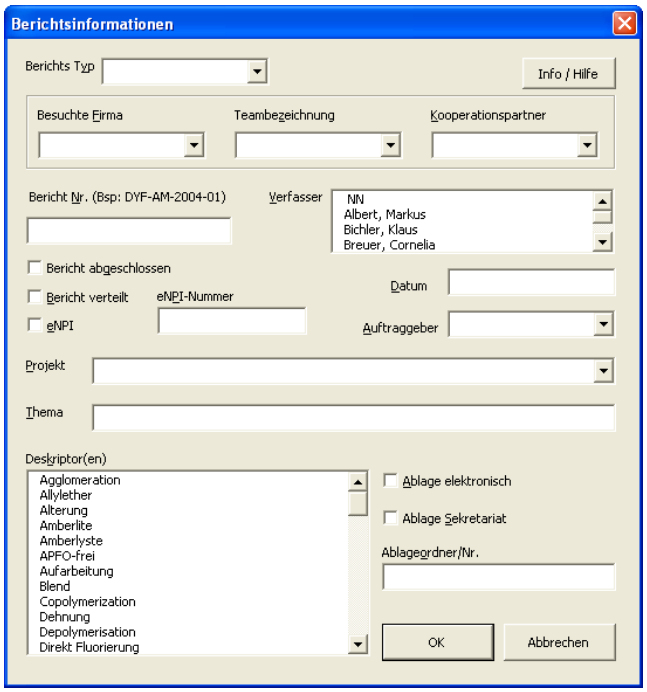



**Abbildung 2.22** Speichert der Anwender eine Datei in den richtigen Ordner, dann wird die Maske gezeigt, mit deren Hilfe Daten in die Datenbank geschrieben werden können.

Eine zweite Maske greift auf die Datenbank zu, erstellt aus den Suchkriterien einen SQL-String, mit dessen Hilfe die entsprechenden Informationen gefunden und nach Excel geschrieben werden können:

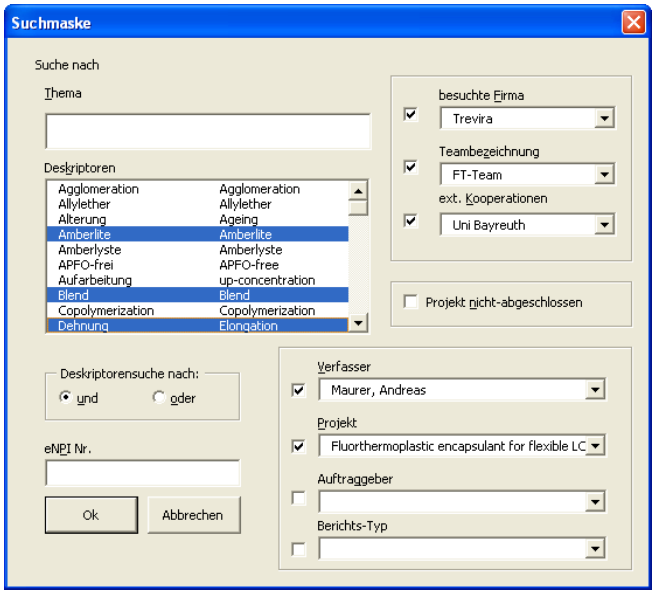



**Abbildung 2.23** Die Suchmaske

### 2.5.1 Fazit

Wenn Sie dem Anwender bequeme Möglichkeiten zur Eingabe schaffen wollen und gleichzeitig berechnete Ergebnisse bei der Eingabe anzeigen lassen möchten, dann sind Dialoge eine hervorragende Möglichkeit. Auch für formatierte Anzeigen der Versionsnummer, des Auftraggebers oder anderer Informationen eignen sich diese Masken.

Beachten Sie jedoch, dass nicht jeder Anwender die Dialoge so verwendet wie Sie. Das bedeutet zum einen, dass Sie sowohl die Benutzer im Blickwinkel haben sollten, die gerne mit der Maus arbeiten (also mit der Tabulatortaste von Eingabefeld zu Eingabefeld springen oder mit gedrückter <ALT>-Taste ein Steuerelement aufrufen), als auch solche, die mit der Maus arbeiten. Es gibt Anwender, die versuchen, in Textfelder, die nur Zahlen- oder Datumseingaben zulassen, Texte hineinzuschreiben (oftmals wissen sie es einfach nicht besser). Es gibt Anwender, die schreiben in Kombinationsfelder Informationen, die nicht in der Liste stehen (und wenn es auch nur ein Leerzeichen am Ende des Textes ist). Fangen Sie solche Fehler ab! Fangen Sie solche Fehler so gut wie möglich ab – am besten mehrmals: bei der Eingabe, beim Verlassen von Feldern, beim Bestätigen der OK-Schaltfläche.

Versuchen Sie also, Dialoge so ergonomisch wie möglich zu halten! Gehen Sie auf die Wünsche der Kunden ein, für die Sie programmieren! Denn der Anwender sieht nur die Masken, auf denen er die Daten erfasst, nicht den Code, der darunter liegt. Und wird sie eben an dieser Oberfläche messen.

# 3 Der Makrorekorder

Es ist sicherlich unmöglich, das Objektmodell von Excel in seiner gesamten Breite und mit allen Objekten zu beschreiben und erläutern. Dies erscheint mir auch nicht nötig, da per Makrorekorder (fast) alle Excel-Befehle aufgezeichnet werden können. Dennoch: Wer größere Projekte in Excel erstellt, der sollte den aufgezeichneten Code lediglich als Gerüst verwenden. Denn:

- Der Makrorekorder zeichnet nicht alles auf
- Der Makrorekorder zeichnet an einigen (wenigen) Stellen fehlerhaften Code auf
- Der Makrorekorder zeichnet meistes zu viel Code auf
  (sämtliche Befehle eines Dialogs)
- Der Makrorekorder zeichnet die Bewegungen am Bildschirm auf.

## 3.1 Der Makrorekorder

Ein wichtiges Hilfsmittel, das einen ersten Zugang zu den Objekten von Excel liefert, ist der Makrorekorder. Seine Bedienung ist denkbar simpel und erinnert an einen Kassettenrekorder oder DVD-Player:

4. Stellen Sie sicher, dass die Entwicklerregisterkarte in der Multifunktionsleiste angezeigt wird (in den Excel-optionen)

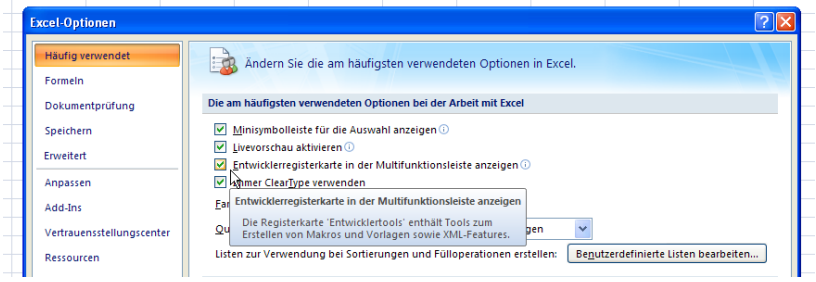



**Abbildung 3.1** Die Entwicklerregisterkarte wird aktiviert.

5. Wechseln Sie in Entwicklerregisterkarte und klicken Sie auf die Schaltfläche „Makro aufzeichnen“.
6. Vergeben Sie einen Namen. Er wird später zum Namen der Prozedur, darf also keine Sonderzeichen und kein Leerzeichen enthalten.
7. Wählen Sie aus, ob Sie das Makro in „Dieser Arbeitsmappe“, einer „Neuen Arbeitsmappe“ oder in der „Persönlichen Arbeitsmappe“ speichern möchten. Zu Testzwecken empfiehlt sich „Diese Arbeitsmappe“.

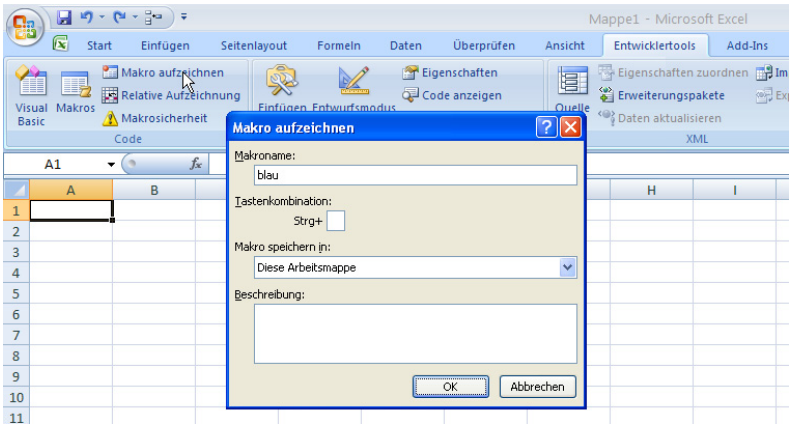



**Abbildung 3.2** Der Makrorekorder wird gestartet.

8. Starten Sie Ihre „Aktionen“ (in unserem Falle: formatiere eine Zelle „blau“)

**Achtung**

Beachten Sie, dass der Makrorekorder „gnadenlos“ alle Befehle aufzeichnet. Das bedeutet: Wenn Sie nicht wissen, wo sich ein bestimmter Befehl befindet, dann sollten Sie ihn vorher suchen und dann erst die Aufzeichnung beenden, da sonst sämtliche Befehle mit aufgezeichnet werden, was zu einem langen Code führen kann.

9. Beenden Sie den Makrorekorder über die Registerregisterkarte „Entwicklertools“ und dort den Befehl „Aufzeichnung beenden“.

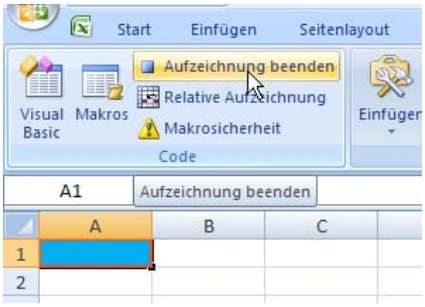



**Abbildung 3.3** Der Makrorekorder wird beendet.

**Achtung**
Wenn Sie vergessen die Aufzeichnung zu beenden, dann läuft er trivialerweise weiter und zeichnet auf und zeichnet auf und zeichnet auf, …

10. Sie finden Ihren Code am schnellsten über die Schaltfläche „Makros“, die sich in der Registerkarte „Ansicht“ und „Entwicklertools“ verbirgt.

**Hinweis**
Da der Makrorekorder manchmal (aber nicht immer) neue Module anlegt, ist es häufig schwierig das aufgezeichnete Makro im Visual Basic Editor zu finden. Deshalb sollten Sie Sie das Makro über Excel aufrufen – denn so gelangen Sie gleich zu richtigen Modul und dort an die gewünschte Stelle.

Das Makro sieht dann wie folgt aus:

```
Sub blau()
'
' blau Makro
'
'
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 15773696
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
End Sub
```

**Tipp**
Die Kommentare sind überflüssig und können gelöscht werden.

Der Code kann nun weiter verarbeitet werden.

Eine Besonderheit hat der Makrorekorder aufzuweisen. Angenommen die Zelle F17 ist ausgewählt. Wenn Sie aufzeichnen „klicke auf A1“, dann zeichnet der Makrorekorder auf:

```
Range("A1").Select
```

Das ist eine „absolute“ Sprunganweisung. Das heißt – unabhängig von der Cursorposition wird sich die Markung auf die Zelle A1 des aktuellen Blattes bewegen. Wenn Sie dies nicht möchten, können Sie die Schaltfläche „Relative Aufzeichnung“ aktivieren – dann zeichnet er auf:

```
ActiveCell.Offset(-16, -5).Range("A1").Select
```

Das heißt: ausgehend von der aktuellen Position 16 Zeilen nach oben und fünf Spalten nach links. Dieser Befehl würde zu einem Fehler führen, wenn der Cursor beispielsweise auf der Zelle B7 sitzt.

Das bedeutet: Je nachdem, was Sie benötigen, sollten Sie die relativen oder absoluten Anweisungen aufzeichnen. Auch innerhalb einer Makroaufzeichnen ist ein Wechseln möglich.

Und: Bis Excel 2003 befanden sich die Befehle für den Makrorekorder im Menü Extras | Makro.

## 3.2 Der zweifelhafte Code des Makrorekorders

Mithilfe des Makrorekorders kommt man an (fast) alle internen Befehle. Das Ergebnis des Makrorekorders hat jedoch einige entscheidende Nachteile.

### 3.2.1 Zu viel Code

Wenn Sie einen Excel-Befehl aufzeichnen, dann zeichnet das Programm zu viel Code auf. Angenommen, Sie möchten eine Papierseite ins Querformat drehen. Dann zeichnet Excel (in der Version 2007) auf:

```
Sub Makro1()

    With ActiveSheet.PageSetup

        .PrintTitleRows = ""

        .PrintTitleColumns = ""

    End With

    ActiveSheet.PageSetup.PrintArea = ""

    With ActiveSheet.PageSetup

        .LeftHeader = ""

        .CenterHeader = ""

        .RightHeader = ""

        .LeftFooter = ""

        .CenterFooter = ""

        .RightFooter = ""

        .LeftMargin = Application.InchesToPoints(0.708661417322835)

        .RightMargin = Application.InchesToPoints(0.708661417322835)

        .TopMargin = Application.InchesToPoints(0.78740157480315)

        .BottomMargin = Application.InchesToPoints(0.78740157480315)

        .HeaderMargin = Application.InchesToPoints(0.31496062992126)

        .FooterMargin = Application.InchesToPoints(0.31496062992126)

        .PrintHeadings = False
```

```
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = 360
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
[...]       .Draft = False
        .PaperSize = xlPaperA4
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
[...]
```

Der Rest braucht hier nicht wiederholt zu werden

Excel zeichnet sämtliche Informationen der vier Registerblätter Seitenlayout | Seite einrichten (bis Excel 2003: Datei | Seite einrichten) auf. Nun liegt es an Ihnen, den überflüssigen Code zu entfernen und auf den nötigen Befehl zu reduzieren. In diesem Fall lautet er:

```
Sub Makro1()
    With ActiveSheet.PageSetup
        .Orientation = xlLandscape
    End With
End Sub
```

oder noch knapper:

```
    ActiveSheet.PageSetup.Orientation = xlLandscape
```

### 3.2.2 Nicht immer der beste Code

Häufig zeichnet Excel die US-amerikanische Schreibweise auf. Wird in eine Zelle eine Summe geschrieben, dann lautet der aufgezeichnete Befehl:

```
ActiveCell.FormulaR1C1 = "=SUM(R[-4]C:R[-1]C)"
```

Es würde auch mit der deutschen Schreibweise

```
ActiveCell.FormulaLocal = "=SUMME(A1:A3)"
```

funktionieren. Wenn Sie eine Zelle benutzerdefiniert mit der Kategorie „Währung“ formatieren, dann lautet der aufgezeichnete Befehl:

```
Selection.NumberFormat = "#,##0.00 $"
```

Besser lesbar für uns Mitteleuropäer ist sicherlich:

```
Selection.NumberFormatLocal = "#.##0,00 €"
```

Auch das im oberen Abschnitt beschrieben Beispiel „gehe zu Zelle A1“ in Abhängigkeit von der aktuellen Position – oder genauer: gehe 15 Zeilen nach oben und fünf Spalten nach links:

```
ActiveCell.Offset(-16, -5).Range("A1").Select
```

ist eines der vielen Befehle, die überarbeitet werden sollten. Der interne Verweis `Range("A1")` ist überflüssig. Es genügt die Zeile:

```
ActiveCell.Offset(-16, -5).Select
```

### 3.2.3 Nicht immer Code

Einige Dinge zeichnet Excel gar nicht auf. Beispielsweise wenn Sie versuchen, die Eigenschaften der Datei (Office-Menü Vorbereiten | Eigenschaften, bis Excel 2003: Datei | Eigenschaften) aufzuzeichnen, dann liefert der Makrorekorder ein leeres Makro. Man muss nun wissen, wie der Befehl „Eigenschaften“ heißen könnte und vor allem – zu welchem Objekt er gehört. Ähnlich ist es mit dem Löschen eines Tabellenblattes. Wenn Sie mit dem Makrorekorder aufzeichnen „lösche Tabelle1“, dann lauten die Codezeilen:

```
Sheets("Tabelle1").Select

ActiveWindow.SelectedSheets.Delete
```

Würden Sie dies dem Anwender weitergeben oder in ein Programm einbauen, dann würde er stets gefragt werden, ob er das Blatt wirklich löschen möchte. Das ist kein sauberes Vorgehen!

### 3.2.4 Fehlerhafter Code

Wenn Sie die Datenüberprüfung (bis Excel 2003: Gültigkeit) mit einer benutzerdefinierten Formel aufzeichnen, dann erhalten Sie folgenden Code:

```
With Selection.Validation
    .Delete
    .Add Type:=xlValidateCustom, AlertStyle:=xlValidAlertStop, _
     Operator:=xlBetween, Formula1:="<heute()"
    .IgnoreBlank = True
    .InCellDropdown = True
    .InputTitle = ""
    .ErrorTitle = ""
    .InputMessage = ""
    .ErrorMessage = ""
    .ShowInput = True
    .ShowError = True
End With
```

Wenden Sie nun diesen Code auf einen anderen Bereich an, dann sieht das die Fehlermeldung folgendermaßen aus:

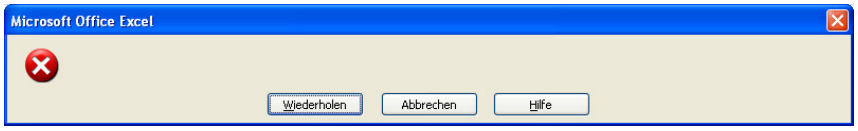



**Abbildung 3.4** Der Ergebnis des Makrorekorders

Was bei dieser Funktion noch klappt, das scheitert, beispielsweise bei folgendem Problem: Der Benutzer darf am Ende eines Textes kein Leerzeichen eingeben. Der Makrorekorder zeichnet auf:

```
With Selection.Validation
    .Delete
    .Add Type:=xlValidateCustom, AlertStyle:=xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="=rechts(A1;1)<>"" """
```

Beachten Sie dabei, dass einfache Anführungszeichen als doppelte Anführungszeichen geschrieben werden müssen. Auch wenn der Makrorekorder die deutschen Funktionsnamen aufzeichnet, muss in den VBA-Code der englische Funktionsname eingegeben werden. Und: in der deutschen Schreibweise steht das Semikolon als Trennzeichen – in der US-amerikanischen das Komma. Es ist also nur so korrekt:

```
Selection.Validation.Add _
   Type:=xlValidateCustom, AlertStyle:=xlValidAlertStop, _
   Formula1:="=Right(A1,1)<>"" """
```

### 3.2.5 Excel-VBA „hilft“ in VBA nicht immer

Wenn Sie den aufgezeichneten Code weiter verarbeiten möchten und schreiben selbst das Objekt „Selection“, tippen einen Punkt, dann kann man Excel nicht dazu bewegen, die komplette Liste der Eigenschaften und Methoden anzuzeigen. Wenn Sie dagegen „sauber“ programmieren, wie im nächsten Kapitel beschrieben, dann werden automatisch die Elemente aufgelistet.

### 3.2.6 Excel zeichnet „unscharf“ auf

Und damit sind wir beim zweiten Problem. Angenommen, Sie suchen den Befehl, mit dem ein Zellhintergrund gelb eingefärbt wird. Excel wird Ihnen – wenn Sie das Symbol verwenden – folgende Zeile aufzeichnen:

```
With Selection.Interior
    .ColorIndex = 6
    .Pattern = xlSolid
End With
```

Zugegeben: Die Eigenschaft „Pattern“, die auf „Solid“ gesetzt wird, stört an dieser Stelle wenig. Störender wirkt sich hingegen das Objekt „Selection“ aus. Es bedeutet, dass der Cursor auf dieser Zelle sitzen muss und dass das zugehörige Tabellenblatt selektiert ist. Wenn Sie dies möchten – beispielsweise dem Benutzer ein Symbol zur Verfügung stellen, mit dessen Hilfe er komplexe, immer wiederkehrende Formatierungen mit einem Mausklick durchführen kann, so ist der Befehl gerechtfertigt. In den meisten anderen Fällen jedoch nicht. Sie kopieren beispielsweise Daten von einer Datei in eine andere und markieren das Ergebnis gelb. Dies sollte ohne „Selection“ geschehen, da Sie nicht am Bildschirm programmieren, sondern Werte oder Formate in eine Zelle schreiben.

An vielen Stellen zeichnet der Makrorekorder Dinge wie „ActiveCell“, „Selection“ oder „ActiveChart“ auf. Auch andere per Makrorekorder aufgezeichnete Befehle (Objekte) wie „Sheets("Tabelle1")“, „Range("A2")“ und so weiter sind im Sinne einer Objektorientierung nicht elegant und bergen eine Menge Fehler .

**Hinweis**

Vermeiden Sie das Objekt „Selection“ und Ähnliche und die Methode „Activate“ (oder „Select“). Der Code wird langsam, unübersichtlich, ist schlecht zu pflegen und fehleranfällig. In meinen gesamten Programmen tauchen diese Befehle überhaupt nicht auf. Zugegeben – vielleicht in der letzten Codezeile, damit der Cursor nach der Auswertung der Tabellenblätter auf einem bestimmten Blatt in Zelle „A1“ sitzt. Aber sonst nicht!

Damit man völlig ohne diese Methoden auskommen kann beziehungsweise die Objekte korrekt setzt, muss man das Objektmodell kennen.

## 3.3 Fazit

Der Makrorekorder ist ein praktisches Werkzeug mit man schnell an Code – oder genauer – an die eigentlichen Objekte, Eigenschaften und Methoden gelangt. Für einen Anfänger sicherlich völlig ausreichend; ein Profi sollte jedoch immer den Code überarbeiten, das heißt:

- Voraussetzungen überprüfen
- Fehler abfangen
- unnötigen Ballast löschen
- unnötige Kommentare entfernen
- Objekte „sauber“ adressieren

# 4 Das Excel-Objektmodell: Application

Es ist sicherlich unmöglich, das Objektmodell von Excel in seiner gesamten Breite und mit allen Objekten zu beschreiben und erläutern. Dies erscheint mir auch nicht nötig, da per Makrorekorder (fast) alle Excel-Befehle aufgezeichnet werden können. Dennoch: Wer größere Projekte in Excel erstellt, der sollte den aufgezeichneten Code lediglich als Gerüst verwenden. Und sollte den Aufbau der Objekte und ihre Hierarchie gut kennen.

Denn auch wenn Excel mehr als 1.500 Objekte mit einer Unmenge an Eigenschaften und Methoden zur Verfügung stellt, so genügen doch vier zentrale Objekte, mit denen (fast) alles in Excel programmiert werden kann: Application, Workbook, Worksheet und Range.

In diesem Kapitel geht es um das „oberste“ Objekt der Objekthierarchie – um Excel selbst – um die Anwendung, oder: `Application`.

## 4.1 Oberstes Objekt von Excel: Application

Das oberste Objekt von Excel heißt „Application“. Dieses Objekt hat eine Reihe von Eigenschaften:

```
Application.Name
```

liefert den Namen des Programms, also „Microsoft Excel“.

```
Application.Path
```

liefert den Pfad der Programmdatei „Excel.exe“.

```
Application.Caption
```

gibt den Text der Titelzeile zurück. Dieser kann, da es sich um eine Eigenschaft handelt, geändert werden, beispielsweise in

```
Application.Caption = "openOffice.org"
```

Eine ganze Reihe von Grundeinstellungen findet sich im Objekt „Application“:

```
AddIns, AlertBeforeOverwriting, AltStartupPath, AnswerWizard, AskToUpdateLinks, Assistant,
AutoPercentEntry, CalculationVersion, CellDragAndDrop, ControlCharacters, Cursor […]
```

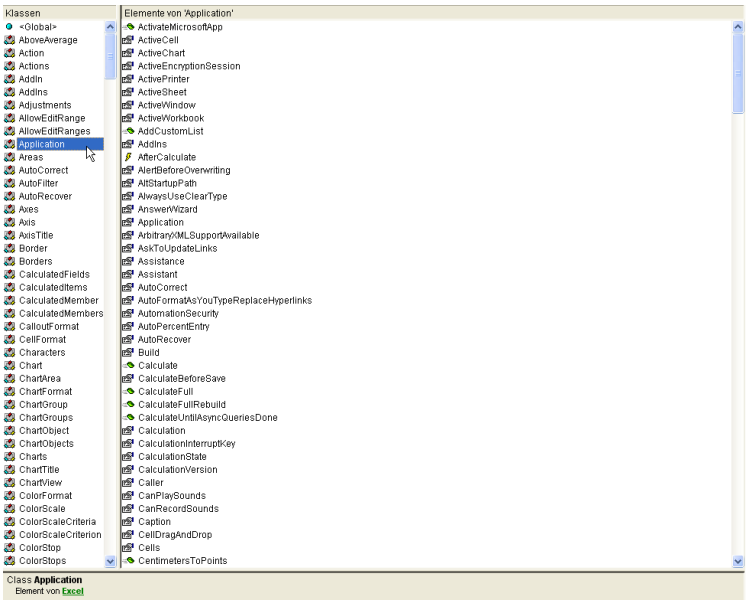



**Abbildung 4.1** Das Objekt Application

**Tabelle 4.1** Ein paar interessante Meldungs- und Darstellungseigenschaften des Objekts Application

| Eigenschaft | Werte | Beschreibung |
|---|---|---|
| AlertBeforeOverwriting | True/False | Meldungen werden eingeblendet, bevor nicht leere Zellen während einer Drag & Drop-Operation überschrieben werden. |
| AskToUpdateLinks | True/False | fragt den Benutzer, ob beim Öffnen von Dateien mit Verknüpfungen diese Verknüpfungen aktualisiert werden sollen. |

| Eigenschaft | Werte | Beschreibung |
|---|---|---|
| DisplayAlerts | True/False | Warnungen und Meldungen werden angezeigt (beispielsweise beim Löschen von Tabellenblättern). |
| Cursor | xlDefault, xlIBeam, xlNorthwestArrow, xlWait | der Mauszeiger |
| ScreenUpdating | True/False | die Bildschirmaktualisierung |
| StatusBar | | die Statuszeile |

**Tabelle 4.2 Einige wichtige Grundeinstellungen von Excel, die geändert und abgefragt werden können**

| Eigenschaft | Beschreibung |
|---|---|
| EditDirectlyInCell | direkte Zellbearbeitung |
| EnableAutoComplete | Autovervollständigen ist aktiviert. |
| ErrorCheckingOptions | Fehlerprüfung ist aktiviert. |
| ExtendList | erweitert Formatierungen für Formeln und Daten |
| MoveAfterReturn | Die aktive Zelle wird nach Drücken der Taste <ENTER> verschoben. |
| MoveAfterReturnDirection | die Richtung des Verschiebens |
| AutoPercentEntry | Einträge in Zellen, die als Prozentzahlen formatiert sind, werden bei der Eingabe automatisch mit 100 multipliziert. |
| Calculation | die Berechnungsart (xlCalculationAutomatic oder xlCalculationManual) |
| ReferenceStyle | die Zellbezüge (xlA1 oder xlR1C1) |
| DecimalSeparator | Dezimaltrennzeichen |
| ThousandsSeparator | Tausendertrennzeichen |
| FixedDecimal | Formatiert eine feste Anzahl an Dezimalstellen. |
| StandardFont | die Standardschriftart |
| StandardFontSize | die Standardschriftgröße |
| DefaultSaveFormat | das Standardspeicherformat (xlExcel7, xlTextWindows, ...) |
| | |
| PromptForSummaryInfo | Beim Speichern einer Datei werden die Dateiinformationen abgefragt. |
| | |
| ShowStartupDialog | Der Aufgabenbereich wird angezeigt. |
| ShowToolTips | QuickInfo ist aktiviert. |
| ShowWindowsInTaskbar | Jede geöffnete Arbeitsmappe wird in der Taskleiste angezeigt. |
| | |
| NetworkTemplatesPath | Netzwerkpfad der Vorlagen |
| TemplatesPath | lokaler Vorlagenordner |
| StartupPath | Startordner |
| UserLibraryPath | Pfad zum Speicherort, an dem die COM-Add-Ins installiert sind |
| LibraryPath | Pfad des Bibliotheksordners |
| die dargestellte Größe von Excel:<br>Height, Width, Left, Top<br>die verwendbare Größe von Excel:<br>UsableHeight und UsableWidth | |
| WindowState | die Darstellung (xlMaximized, xlMinimized und xlNormal) |

**Tabelle 4.3 Einige Sammlungen von Excel:**

| Sammlung | Beschreibung |
|---|---|
| CommandBars | Symbolleisten und Menüleiste(n) (bis Excel 2003) |
| Charts | Diagramme |
| AddIns | Add-Ins |
| CustomListCount, GetCustomListContents | benutzerdefinierte Listen<br>sie werden gelöscht mit:<br>DeleteCustomList |
| PreviousSelections | die Liste der letzten Markierungen |
| RecentFiles | zuletzt geöffnete Dateien<br>sie können angezeigt werden mit:<br>DisplayRecentFiles |
| Windows | offene Fenster |
| Names | vom Anwender definierte Namen |
| Workbooks | (alle geöffneten) Arbeitsmappen |

**Tabelle 4.4 Der Zugriff auf interne Dialoge ist möglich über:**

| Name | Beschreibung |
|---|---|
| Dialogs | die Sammlung aller Dialoge |
| FileDialog (msoFileDialogFilePicker) | Datei \| Öffnen (Dateien) |
| FileDialog(msoFileDialogFolderPicker) | Datei \| Öffnen (Ordner) |
| FileDialog (msoFileDialogOpen) | Datei \| Öffnen (allgemein) |
| FileDialog (msoFileDialogSaveAs) | Datei \| Speichern unter |
| GetOpenFilename | Datei \| Öffnen |
| GetSaveAsFilename | Datei \| Speichern unter |

**Tabelle 4.5 Einige wichtige Methoden von Excel:**

| Methode | Beschreibung |
|---|---|
| Calculate | Neuberechnung |
| Volatile | Eine benutzerdefinierte Funktion wird neu berechnet (. |
| Repeat | Wiederholen |
| Undo | Rückgängig |
| Run | Ein Makro kann gestartet werden. |
| | |
| OnKey | Tastenbelegung |
| OnRepeat | beim Wiederholen |
| OnTime | bei einer bestimmten Uhrzeit |
| Wait | Bei einer bestimmten Uhrzeit wird ein Makro unterbrochen. |
| OnUndo | bei Rückgängig |
| EnableCancelKey | Es kann abgefangen werden, ob der Benutzer das Makro mit <STRG> + <UNTERBR> unterbricht. |
| Quit | Beenden |

**Tabelle 4.6 Einige interessante interne Befehle**

| Befehl | Beschreibung |
|---|---|
| International | länderspezifische Einstellungen, beispielsweise:<br>xlCountryCode, xlCountrySetting, xlCurrencyCode, xl24HourClock, xlMetric, xlDecimalSeparator … (siehe Kapitel 20) |
| LanguageSettings | einige Spracheinstellungen |
| PathSeparator | verwendetes Trennzeichen |
| Hinstance | die Instanzzugriffsnummer, mit der Excel aufgerufen wurde |
| Hwnd | die Fensterzugriffsnummer der obersten Ebene des Microsoft Excel-Fensters |
| OperatingSystem | Betriebssystem |
| UserName | Benutzername |
| OrganizationName | Name der Organisation |
| ProductCode | Produktcode von Excel |
| | |
| MathCoprocessorAvailable | mathematischer Koprozessor verfügbar |
| MouseAvailable | Maus verfügbar |
| | |
| Interactive | interaktiver Modus (Maus und Tastatur können blockiert werden) |
| UserControl | Wurde Excel vom Benutzer oder per Programmierung gestartet? |
| Visible | Ist Excel sichtbar? |
| Version | die Versionsnummer von Excel |
| | |
| WorksheetFunction | Excel-Funktionen können in VBA verwendet werden (wird in Kapitel 8 erläutert). |

**Hinweis**

Wenn Sie in Excel programmieren, dann müssen Sie für die Sammlungen nicht das Objekt „Applikation“ schreiben. VBA erkennt „Applikation“ als oberste Ebene und macht die Schreibung somit überflüssig. Die letzten beiden Zeilen sind äquivalent:

```
Dim xlDateien As Workbooks


Set xlDateien = Application.Workbooks


Set xlDateien = Workbooks
```

**Tipp**

Wenn Sie gerne Application schreiben möchten, dann können Sie „appl“ schreiben und danach die <STRG>+<LEERTASTE> drücken – VBA vervollständigt den Ausdruck zu „Application“.

### 4.1.1 Die Version

Manchmal ist es wichtig die Version zu erfahren, in der die Programme laufen. Gerade in Firmen, die von einer Version auf eine andere umstellen, muss überprüft werden, in welcher Version das Programm läuft. Die kann mit der Eigenschaft Version geschehen:

```
MsgBox Application.Version
```

**Achtung**

Beachten Sie, dass nach der Installation eines Servicepacks die Versionsnummer nicht mehr 12.0 für Excel 2007 oder 11.0 für Excel 2003 lautet, sondern beispielsweise 12.0a. Deshalb sollten Sie im Code anfragen:

```
If Left(Application.Version, 2) = "12" Then
```

### 4.1.2 Die Bildschirmanzeige

Die Bildschirmanzeige kann ausgeschaltet werden:

```
Application.ScreenUpdating = False
```

Danach sieht der Benutzer nicht mehr, wenn der Cursor bewegt wird. Auch das lästige Flackern, das aufgrund von Bildschirmaktualisierungen entsteht (beispielsweise beim Öffnen und Schließen von Dateien) wird dadurch vermieden. Am Ende des Programms sollte es selbstredend wieder eingeschaltet werden:

```
Application.ScreenUpdating = True
```

Übrigens schalte ich bei lange laufenden Makros den Mauszeiger auf „Sanduhr“:

```
Application.Cursor = xlWait
```

und am Ende wieder „zurück“:

```
Application.Cursor = xlDefault
```

Ebenso beschrifte ich die Statuszeile

```
Application.StatusBar = "Makro läuft. Bitte warten Sie ..."
```

das am Ende des Programms wieder ausgeschaltet wird:

```
Application.StatusBar = False
```

### 4.1.3 Warnmeldungen

Eigene Excelaktionen lösen Warnmeldungen aus, beispielsweise das Überschreiben einer Datei oder das Löschen von Tabellenblättern. Sie können diese Meldung ausschalten mit:

```
Application.DisplayAlerts = False

ActiveSheet.Delete

Application.DisplayAlerts = True
```

Sie sollten es nach dem durchgeführten Befehl sofort wieder einschalten.

Erstaunlicherweise warnt Excel beim Verschieben von Zellen in einen anderen Bereich, in dem sich bereits Daten befinden. Jedoch löst der Befehl

```
xlBereich2.Cut Destination:=xlBereich1
```

keine Fehlermeldung aus. Wollte man die Fehlermeldung unterdrücken, die für den Benutzer beim Verschieben angezeigt wird, dann könnte man dies beim Starten von Excel oder beim Öffnen einer Datei mit folgendem Befehl erledigen:

```
Application.AlertBeforeOverwriting = False
```

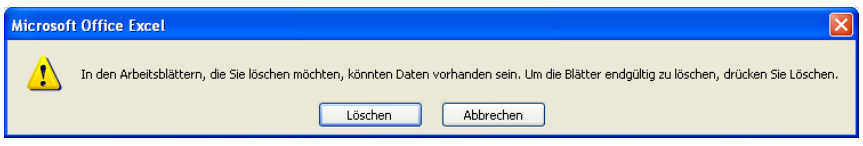



**Abbildung 4.2** Dieser Warnhinweis kann unterdrückt werden.

### 4.1.4 Evaluate

Werte von Funktionen können mit `Evaluate` ausgewertet werden. Beachten Sie, dass der Funktionsname in der englischen Schreibweise vorliegen muss. So liefern beispielsweise

```
MsgBox Application.Evaluate("=SUM(A1:A5)")

MsgBox Application.Evaluate("=AVERAGE(A1:A5)")

MsgBox Application.Evaluate("=COUNTA(A1:A5)")
```

die Summe, der Durchschnitt und die Anzahl der Zahlen der Zellen A1:A5. Und wer benötigt so etwas? An anderer Stelle wird der Benutzer über einen Dialog aufgefordert eine Formel einzugeben. Diese wird in die US-amerikanische Schreibweise konvertiert. Um nun zu überprüfen, ob der Benutzer eine falsche Formel eingegeben hat, kommt `Evaluate` zum Einsatz:

```
On Error Resume Next

MsgBox Application.Evaluate("=SUM(A1 bis A5)")

If Err.Number <> 0 Then MsgBox "Ihre Formel ist falsch"

Err.Clear

MsgBox Application.Evaluate("=AVERAGE(A1:A5)")

If Err.Number <> 0 Then MsgBox "Ihre Formel ist falsch"

Err.Clear
```

### 4.1.5 InputBox

Nicht nur die Klasse VBA sondern auch Application verfügt über ein Eingabefenster – die Methode `InputBox`. Der Unterschied zwischen `VBA.InputBox` und `Application.InputBox` ist jedoch, dass `Application.InputBox` noch als letzten Parameter den optionalen Wert `Type` aufweist. Dabei steht.

**Tabelle 4.7 Der Parameter Type der Inputbox**

| Wert | Bedeutung |
|---|---|
| 0 | Formel |
| 1 | Zahl |
| 2 | Text (Zeichenfolge) |
| 4 | Wahrheitswert (True oder False) |
| 8 | Zellbezug, z. B. ein Range-Objekt |
| 16 | Fehlerwert, z. B. #NV |
| 64 | Wertearray |

Es können auch Kombinationen mehrere Werte vorgenommen werden:

```
strAntwort = Application.InputBox(Prompt:="Was nun?", Type:=1 + 2)

MsgBox strAntwort
```

Interessant wird die InputBox, wenn Zellbereiche eingegeben werden sollen:

```
On Error Resume Next

Application.DisplayAlerts = False

Set xlBereich = Application.InputBox(Prompt:="Welcher Bereich?", Type:=8)
```

```
If xlBereich Is Nothing Then

   MsgBox "Es wurde kein Bereich oder ein falscher Bereich eingegeben!"

Else

   MsgBox xlBereich.Address

End If

Application.DisplayAlerts = True
```

## 4.2 Fazit

Auf den ersten Blick sieht das oberste Objekt Application bedeutungslos aus. Jedoch verbergen sich einige wichtige Eigenschaften und Methoden dahinter, die für die Programmierung nützliche sein können.

# 5 Das Excel-Objektmodell: Workbook

Um auf eine Datei zuzugreifen, benötigt man das Objekt Workbook. Dies ist sowohl beim Zugriff auf offene Dateien wichtig als auch um Dateien zu öffnen. Im Wesentlichen werden die Methoden über das Objekt Workbook aufgerufen, die sich in Office 2007 in der Office-Schaltfläche befinden (bis Office 2003 im Menü Datei): neue Datei, Datei öffnen, speichern, drucken und schließen.

## 5.1 ActiveWorkbook und ThisWorkbook

Die aktuelle Excel-Datei heißt

```
Application.ActiveWorkbook
```

oder:

```
Application.ThisWorkbook
```

Der Unterschied ist wichtig: ThisWorkbook bezieht sich auf die Datei, in der sich die Makros befinden, ActiveWorkbook dagegen auf die „oberste“ Datei, das heißt die Datei in Excel, die Sie gerade sehen, wenn Sie von Excel aus das Makro starten. Sie muss nicht notwendigerweise die Datei sein, in der sich die Makros befinden..

### 5.1.1 Alle offenen Dateien

Alle offenen Dateien können auf folgende Art angesprochen werden:

```
Application.Workbooks
```

Man kann mit einem Zähler alle Dateien durchlaufen lassen und sie somit ansprechen:

```
For i = 1 To Application.Workbooks.Count
   MsgBox Application.Workbooks(i).Name
Next
```

oder direkt alle Objekte ansprechen, wie im folgenden Beispiel:

```
Sub Alle_Dateien()
   Dim xlsDatei As Workbook
   Dim strDatName As String

   For Each xlsDatei In Workbooks
      strDatName = strDatName & vbCr & xlsDatei.Name
   Next

   MsgBox strDatName
End Sub
```

Excel besitzt für die Sammlung der Workbooks, beziehungsweise für ein Workbook, folgende Methoden:

### 5.1.2 Schließen

Eine einzelne Datei wird mit der Methode `Close` geschlossen:

```
Workbooks(1).Close(SaveChanges, FileName, RouteWorkbook)
```

Der Parameter `SaveChanges` kann den Wert `True` annehmen. Dann wird die Datei unter ihrem Dateinamen gespeichert. Besitzt die Datei noch keinen Dateinamen, dann wird der Benutzer nach dem Namen gefragt. `False` dagegen schließt die Datei ohne nachzufragen.

**Hinweis**

Der Parameter SaveChanges ist wichtig, wenn Sie eine Datei per Programmierung öffnen. Einige Formeln und Funktionen werden beim Öffnen neu berechnet, beispielsweise =HEUTE() oder =JETZT(). Dann würde beim Schließen – auch wenn in der Datei nichts geändert wurde – nachgefragt werden, ob die Änderungen gespeichert werden sollen. Um diesen Dialog zu umgehen, müssen Sie den Parameterwert False übergeben.

Wird die Datei gespeichert, könnte man sie bei Schließen unter einem anderen Namen speichern. dafür dient der Parameter `FileName`.

**Hinweis**

Beachten Sie, dass auch die Sammlung Workbooks über eine Methode Close verfügt. Mit ihr werden sämtliche geöffnete Dateien geschlossen. Diese enthält allerdings – anders als Workbooks(1).Close keine optionalen Parameter.

### 5.1.3 Speichern

Eine Datei wird mit der Methode `Save`, `SaveAs` oder `SaveCopyAs` gespeichert:

```
Workbooks(1).Save

Workbooks(1).SaveAs(Filename, FileFormat, Password, WriteResPassword, ReadOnlyRecommended,
CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodePage, TextVisualLayout)

Workbooks(1).SaveCopyAs(Filename)
```

Während Save keinen Parameter besitzt – die Datei also ohne Nachfragen unter ihrem Dateinamen speichert, verfügen `SaveAs` und `SaveCopyAs` unter einem bestimmten (oder anderen) Dateinamen.

**Achtung**

Besitzt die Datei noch keinen Dateinamen, dann wird die Datei unter dem Namen, der in der Titelzeile steht an den aktuellen Speicherort gespeichert. Also beispielsweise in Eigene Dateien unter den Dateinamen Mappe3.xlsx.

Wenn Sie eine Datei mit dem Namen „Test“ unter dem Dateinamen „Versuch“ mit der Methode `SaveAs` speichern, dann lautet der Dateiname der „alten“ Datei nun „Versuch“. Wenn Sie dagegen die Datei mit der Methode `SaveCopyAs` speichern, dann lautet der Name der Datei noch immer „Test“. Es wurde lediglich eine Kopie dieser Datei erstellt.

Um zu überprüfen, ob eine Datei bereits gespeichert wurde, kann die Eigenschaft `Saved` verwendet werden:

```
If ActiveWorkbook.Saved = False Then

   ActiveWorkbook.Save

End If
```

**Hinweis**

Erstaunlicherweise kann die Eigenschaft „Saved“ gesetzt werden, ohne die Datei zu speichern. Damit würde beim Schließen allerdings nicht mehr nachgefragt werden, ob die Datei gespeichert werden soll, was ziemlich „gefährlich“ ist.

**Hinweis**

Die Sammlung Workbooks verfügt über keine Methode Save – wenn Sie also alle offenen Dateien speichern möchten, dann müssen Sie die Dateien mit einer Schleife durchlaufen.

Sie können sich – auch ohne zu speichern – den Speichern-Dialog von Excel anzeigen lassen. Hierzu steht die Methode

```
Application.GetSaveAsFilename
```

zur Verfügung. Bricht der Benutzer den Dialog an, dann wird dieser Methode der Wert `False` übergeben. Speichert er sie, so zeigt sie den Dateinamen. Der Methode `GetSaveAsFilename` kann als Parameter der Wert des

Textfilters, ein Vorgabedateiname, eine Voreinstellung des Filters und eine Beschriftung der Titelzeile übergeben werden.

```
Dim strfileSaveName As String


strfileSaveName = Application.GetSaveAsFilename _
   (InitialFileName:=ActiveWorkbook.Name, _
   FileFilter:="Excel File ohne Makros (*.xlsx),*.xlsx," & _
   "Excel File mit Makros (*.xlsm),*.xlsm,Text Files (*.txt),*.txt", _
   Title:="Mein Speicherrn-Dialog")
If strfileSaveName <> "Falsch" Then
    MsgBox "Die Datei wurde gespeichert unter: " & vbCr & _
         strfileSaveName
End If
```

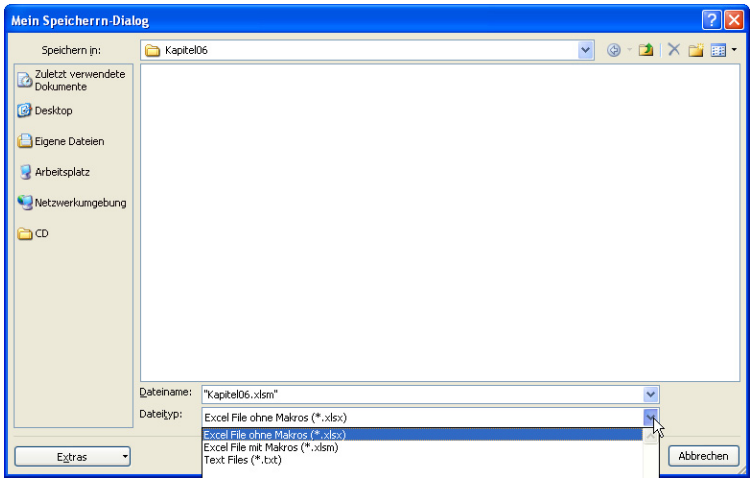



**Abbildung 5.1** Der Speichern-Dialog

**Hinweis**

Erstaunlicherweise zeigt der Dialog den Dateinamen in Anführungszeichen. Auch ein Ersetzen der Hochkommata mit der Funktion Replace nützt nichts – sie werden weiterhin angezeigt. Jedoch wird der Dateiname ohne Anführungszeichen gespeichert.

**Achtung**

Dieses Beispiel ist nicht ganz elegant, da GetSaveAsFilename entweder den booleschen Wert False oder einen Text zurückgibt. VBA besitzt allerdings keine Funktion IsBool. Dieses Beispiel würde auf einem Rechner mit englischer Oberfläche nicht funktionieren.

**Hinweis**

Vergessen Sie nicht den Befehl Dir, mit dessen Hilfe überprüft werden kann, ob die Datei bereits existiert.

### 5.1.4 Drucken

Für das Ausdrucken einer Datei steht der Befehl `PrintOut` mit einer Reihe an Parametern zur Verfügung:

```
Workbooks(1).PrintOut(From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate,
PrToFileName)
```

Sämtliche Einstellungen der Seite gehören nicht zur Datei und können folglich nicht an das Workbook-Objekt übergeben werden. Sie müssen mit der Eigenschaft `PageSetup` einem Blatt (oder mit einer Schleife allen Blättern) die Einstellungen für den Drucker übergeben:

```
Set xlBlatt = ActiveWorkbook.Worksheets(1)

xlBlatt.PageSetup.RightFooter = ActiveWorkbook.FullName
```

**Hinweis**

Die Bemerkung ist trivial: Wenn kein Drucker installiert ist, dann führt die Methode Printout zu einem Fehler.

**Achtung**

Wenn Sie per Programmierung auf einen bestimmten Drucker ausdrucken möchten, dann benötigen Sie den Namen Ihres Druckers. Um ihn zu ermitteln, müssen Sie den Makrorekorder verwenden.

Mein Drucker „Epson Color Stylus 600“ heißt:

```
Epson Stylus COLOR 600 ESC/P 2 auf LPT1:
```

mein Dateiexport in ein pdf-Dokument mithilfe des Adobe Acrobat:

```
Adobe PDF auf Ne03:
```

**Hinweis**

Mit dem Befehl Application.Dialogs(xlDialogPrint).Show kann der Drucken-Dialog angezeigt werden. Über die Argumente können Voreinstellungen vorgenommen werden. Das nächste Beispiel zeigt den Drucken-Dialog und hat als Vorauswahl die Zahl 5 als „Anzahl der Exemplare“ voreingestellt.

```
Application.Dialogs(xlDialogPrint).Show Arg4:=5
```

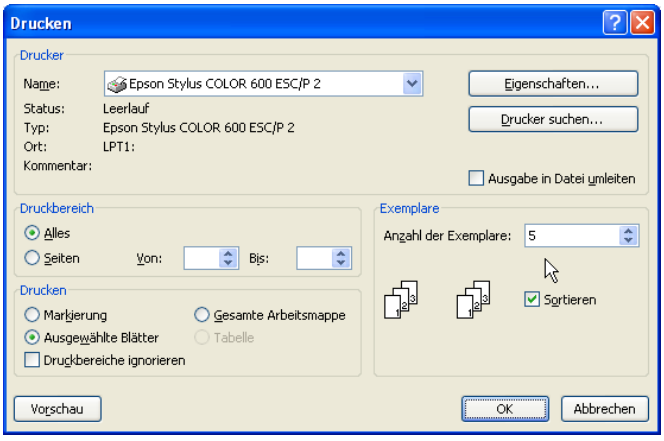



**Abbildung 5.2** Der Drucken-Dialog

### 5.1.5 Öffnen

Während bei `Close` die Sammlung Workbooks sämtliche offenen Dateien bezeichnet, nennt dagegen die Sammlung Workbooks bei `Open` sämtliche (noch) nicht offenen Dateien. Die Syntax sieht folgendermaßen aus:

```
Workbooks.Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, Ignore-
ReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMRU)

Workbooks.OpenText Filename, Origin, StartRow, DataType, TextQualifier, Consecutive-
Delimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo, TextVisualLayout,
DecimalSeperator, ThousandsSeperator, TrailingMinusNumbers, Local
```

Seit Excel 2003 kann auch eine Datenbank als Objekt geöffnet werden:

```
OpenDatabase(FileName, CommandText, CommandType, BackgroundQuery, ImportDataAs)
```

Oder eine XML-Datei kann geöffnet werden:

```
OpenXML(Filename, Stylesheets, LoadOption)
```

**Hinweis**

Vor dem Öffnen sollten Sie immer überprüfen, ob die Datei vorhanden ist. Und: bitte wechseln Sie nicht in das Verzeichnis, sondern geben immer den kompletten Pfad an:

```
Const DATEINAME As String = "C:\175\Test.xlsx"

Dim xlDatei As Workbook



If Dir(DATEINAME, vbNormal) = "" Then

   MsgBox "Datei konnte nicht gefunden werden."

Else

   Set xlDatei = Application.Workbooks.Open(Filename:=DATEINAME)

End If
```

**Hinweis**

Wenn Sie mit der Datei weiterarbeiten, dann sollten Sie das Objekt an eine Objektvariable vom Typ Workbook übergeben und damit weiterarbeiten.

Analog zur Methode `GetSaveAsFilename` stellt das Objekt Application die Methode

```
Application.GetOpenFilename
```

zur Verfügung. Damit kann der Öffnen-Dialog angezeigt werden, ohne dass eine Datei geöffnet wird. Klickt der Benutzer auf die Schaltfläche „Abbrechen“, dann gibt der Dialog den Wert `False` zurück – andernfalls den Dateinamen ohne dass die Datei geöffnet wird:

```
Dim strAuswahl As String

strAuswahl = Application.GetOpenFilename _

   (FileFilter:="Excel Files, *.xls; *.xlsx; *.xlsm," & _

   "Text Files (*.txt), *.txt")

If strAuswahl = "Falsch" Then

   MsgBox "Sie haben nichts ausgewählt!"

Else

   MsgBox "Sie haben die Datei " & strAuswahl & " gewählt!"

End If
```

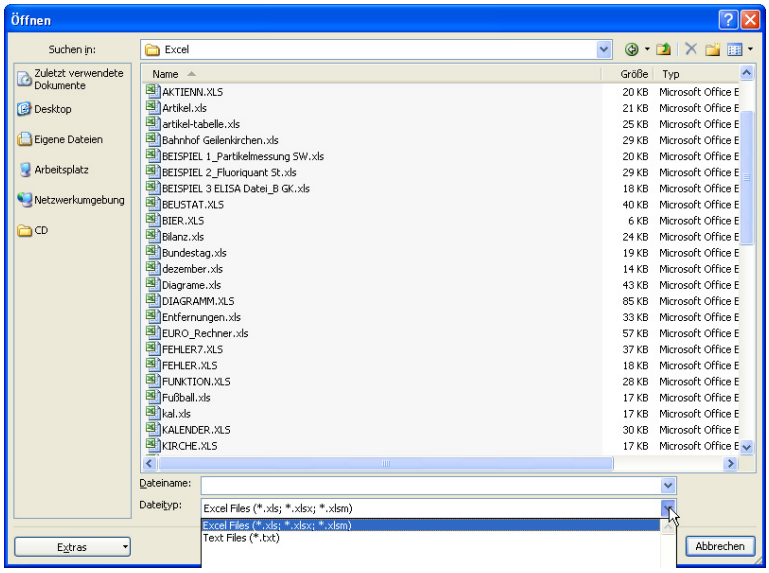



**Abbildung 5.3** Der Öffnen-Dialog

### 5.1.6 Neu

Ähnlich wie die Methode Open arbeitet die Methode `Add`. Mit ihrer Hilfe kann eine neue, leere Datei zu Excel hinzugefügt werden:

```
Workbooks.Add(Template)
```

Wird `Add` ohne Parameter verwendet, dann wird eine neue, leere Datei geöffnet, ansonsten eine Vorlage. Sie sollten beim Arbeiten mit Vorlagen – analog zu Arbeitsmappen – zuerst überprüfen, ob diese Datei offen ist.

**Beispiel**

Im folgenden Beispiel wird überprüft, ob eine bestimmte Datei schon geöffnet ist oder nicht. Falls ja, so wird sie nach vorne geholt, falls nein, dann wird sie geöffnet.

```
Sub DateiÖffnen()

   Dim xlDatei As Workbook

   Dim strDatName As String


   On Error Resume Next


   For Each xlDatei In Workbooks

      If xlDatei.Name = "Rechnung.xls" Then

         xlDatei.Activate

          Exit Sub

      End If

   Next
```

```
        Workbooks.Open Filename:= _

           "C:\Eigene Dateien\Uebungsdateien\Excel\Rechnung.xls"

End Sub
```

Wird mit dieser Datei weitergearbeitet, dann kann ein Verweis auf sie gesetzt werden:

```
Set xlDatei = Workbooks.Open Filename:= _

   "C:\Eigene Dateien\Uebungsdateien\Excel\Rechnung.xls"
```

Das Angenehme an den Microsoft-Produkten ist, dass nicht nach Groß- und Kleinschreibung beim Öffnen von Dateien unterschieden wird. Sollten Sie sich unsicher sein, ob nicht doch unterschieden wird, dann können Sie über

```
Option Compare Text
```

diese Funktion für das Modul, die Klasse oder das Formular deaktivieren.

**Tabelle 5.1 Die wichtigsten Methoden für ein Workbook-Objekt:**

| Methode | Erläuterung |
|---|---|
| Close | Schließen |
| PrintOut | Drucken |
| PrintPreview | Seitenansicht |
| Save | Speichern |
| SaveAs | Speichern unter |
| SaveCopyAs | Speichern als Kopie |
| Activate | aktiviert die Arbeitsmappe, das heißt, bringt sie nach vorne. |
| RefreshAll | aktualisiert externe Datenbereiche. |
| Protect | legt einen Schutz auf die Datei. |
| Unprotect | hebt den Schutz auf. |

**Tabelle 5.2 Die wichtigsten Sammlungen für ein Workbook-Objekt:**

| Sammlung | Erläuterung |
|---|---|
| Worksheets und Sheets | Tabellenblätter |
| BuiltinDocumentProperties | Dokumenteigenschaften der Arbeitsmappe |
| CustomDocumentProperties | benutzerdefinierte Dokumenteigenschaften |
| Charts | Diagramme |
| Colors | definierte Farben |
| CustomViews | benutzerdefinierte Arbeitsmappenansichten |

**Tabelle 5.3 Einige wichtige Eigenschaften des Workbook-Objekts**

| Eigenschaft | Erläuterung |
|---|---|
| Name | Dateiname |
| Path | Speicherort |
| Fullname | Speicherort und Dateiname |
| IsAddin | Ist die Datei ein Add-In? |
| Saved | Wurde die Datei schon gespeichert? |
| DisplayDrawingObjects | zeigt Zeichnungsobjekte an |

| Eigenschaft | Erläuterung |
|---|---|
| DisplayInkComments | zeigt Freihandkommentare an |
| HasPassword | Besitzt die Datei ein Passwort? |
| Password | gibt das Kennwort zurück oder legt es fest. |
| WritePassword | setzt ein Schreibkennwort. |
| Permission | die Berechtigungseinstellungen der Datei |
| ReadOnly | schreibgeschützt |
| RemovePersonalInformation | entfernt persönliche Informationen |
| VBProject | das VB-Projekt, das den VBA-Code enthält |
| Parent | das übergeordnete Objekt – hier: Application |

**Beispiel**

Mit diesem Wissen können Sie leicht eine Funktion schreiben, die überprüft, ob eine bestimmte Datei schon offen ist:

```
Function IstDateiSchonOffen(strDateiname As String, _
   Optional fUnterscheideGroßKleinschreibung As Boolean) As Boolean
   Dim fOffen As Boolean
   Dim xlDatei As Workbook
   fOffen = False

   For Each xlDatei In Application.Workbooks
      If xlDatei.Name = strDateiname Then
         fOffen = True
      End If
   Next

   If fUnterscheideGroßKleinschreibung = True And fOffen = False Then
      For Each xlDatei In Application.Workbooks
         If LCase(xlDatei.Name) = LCase(strDateiname) Then
            fOffen = True
         End If
      Next
   End If

   IstDateiSchonOffen = fOffen

End Function
```

**Achtung**

Beachten Sie, dass Sie dieser Funktion die Endung, also „.xls“, „.xlt“, „.xlsx“, „xlsm“, „xltx“ und so weiter hinzufügen müssen. Es wäre Unsinn, sämtliche Varianten an den Dateinamen ohne Endung zu hängen und dann ihre Existenz zu überprüfen.

**Beispiel**

Eine zweite Funktion listet sämtliche offenen Dateien auf und gibt ihre Dateinamen als Datenfeld zurück:

```
Function AlleOffenenDateien() As Variant

   Dim strDatei() As String

   Dim xlDatei As Workbook

   Dim i As Integer

   i = 0


   For Each xlDatei In Application.Workbooks

      ReDim Preserve strDatei(i)

      strDatei(i) = xlDatei.Name

      i = i + 1

   Next


   AlleOffenenDateien = strDatei


End Function
```

**Beispiel**

Ein Programm zählt so lange hoch, bis eine Zahl gefunden wird, die noch nicht vergeben wurde und speichert die Datei unter diesem Namen:

```
Sub NeuerDateiname()

   Dim i As Integer

   Const PFAD As String = "C:\175\"

   Const DATEINAME As String = "Rechnung"

   Dim strSpeichername As String

   i = 1

   Do

      strSpeichername = PFAD & DATEINAME & i & ".xlsx"

   Loop Until Dir(strSpeichername, vbNormal) = ""

   ActiveWorkbook.SaveAs Filename:=strSpeichername

End Sub
```

### 5.1.7 Löschen

Sie müssen nicht die Methode `Kill` verwenden um eine Datei zu löschen. Sie können eine beliebige Datei auch vom Benutzer löschen lassen. Dies funktioniert, indem Sie den Dialog

```
Application.Dialogs(xlDialogFileDelete)
```

öffnen. Hat ihn der Benutzer abgebrochen, wird der Wert `False` zurückgegeben. Man könnte die Warnmeldung ausschalten mit:

```
Application.DisplayAlerts = False
```

## 5.2 Fazit

Gerade wenn Sie mit mehreren Dateien gleichzeitig arbeiten oder auch, wenn Sie nicht wissen, worauf sich bestimmte Excel-VBA-Befehle beziehen, dann sollten Sie das Workbook-Objekt exakt definieren. Sie können auf eine beliebige offene Datei verweisen, auf eine Datei, die neu erstellt oder geöffnet wird, auf die „oben liegende“ Arbeitsmappe oder auf die Datei, an der der VBA-Code hängt, der gerade ausgeführt wird.

# 6 Das Excel-Objektmodell: Worksheet

So wie Sie exakt auf eine bestimmte Datei mit dem Objekt Workbook zugreifen sollten, sollten Sie auch auf das Blatt, aus dem Sie Werte holen oder in das Sie Werte schreiben zugreifen. Ebenso wie bei dem Objekt Workbook gibt es auch bei Worksheet einiges zu beachten – die fängt schon damit an, dass „Tabellenblatt“ in den beiden Formen Sheet und Worksheet vorliegt.

## 6.1 Zugriff auf Tabellenblätter: Worksheet und Sheet

Jede Excel-Datei hat ein oder mehrere Tabellenblätter. Dabei ist das Objekt von „Workbook“ entweder „Sheet“ oder „WorkSheet“. „Sheet“ ist allgemeiner, da Tabellenblätter auch Diagramme beinhalten können. Deklariert wird es allerdings vom Objekttyp „Worksheet“. Folgendes Beispiel durchläuft alle Tabellenblätter und meldet die Blattnamen:

```
Sub TabellenBlätterDurchlaufen()

   Dim xlTabBlatt As Worksheet

   Dim strBlattName As String

   On Error Resume Next


   For Each xlTabBlatt In Sheets

      strBlattName = strBlattName & vbCr & xlTabBlatt.Name

   Next


   MsgBox strBlattName

End Sub
```

### 6.1.1 Blätter: Wesen mit mehreren Namen

**Achtung**
Beachten Sie folgende Inkonsistenz in Excel:

- Ein Blatt kann nur vom Typ `Worksheet` deklariert werden.
- `For Each For Each xlTabBlatt In Sheets`

  und

  `For Each xlTabBlatt In Worksheets`

  wird nur die Tabellenblätter durchlaufen – nicht jedoch die Blätter, auf denen sich Diagramme befinden.
- `ActiveWorkbook.Worksheets.Count`

  gibt die Anzahl der Tabellenblätter zurück, jedoch

  `ActiveWorkbook.Sheets.Count`

  die Anzahl aller Blätter, auch wenn sich auf ihnen Diagramme befinden.
- `ActiveWorkbook.Worksheets(1).Name`

  liefert den Namen des ersten Tabellenblattes.

  `ActiveWorkbook.Sheets(1).Name`

  liefert den Namen des ersten Blattes – unabhängig davon, ob dies eine Tabelle darstellt oder ein Diagramm. Folglich können `ActiveWorkbook.Worksheets(1).Name` und `ActiveWorkbook.Sheets(1).Name` das gleiche Objekt bezeichnen oder ein unterschiedliches.

Sie sollten also sicher sein oder per Programmierung überprüfen, ob die Datei Diagramme als eigene Blätter enthält. Eine neu erzeugte Datei kann niemals ein Diagramm enthalten – jedoch wenn der Benutzer auf `Active-Workbook` zugreifen kann, dann könnte er ein Diagramm erzeugt haben:

```
    For intBlattZähler = 1 To ActiveWorkbook.Sheets.Count
       strBlattName = strBlattName & vbCr & _
       TypeName(ActiveWorkbook.Sheets(intBlattZähler)) & _
       vbTab & IIf(TypeName(ActiveWorkbook.Sheets(intBlattZähler)) = _
          "Chart", vbTab, "") & _
       ActiveWorkbook.Sheets(intBlattZähler).Name
    Next
```

**Hinweis**

Bis Excel 4.0 konnten auf Blättern auch Makrocode stehen. Selbst in der Version 2007 – also fast 15 Jahre danach wird dies immer noch unterstützt. Allerdings ist mir niemand bekannt, der heute noch Excel-Makros der Version 4.0 in Blattform verwendet oder gar neu schreibt.

### 6.1.2 Wichtige Methoden der Tabellenblätter

Ein Blatt wird mit der Methode `Activate` aktiviert. Ist es verborgen, so kann dies mit der Eigenschaft `Visible` überprüft werden. Gelöscht wird ein Blatt mit der Methode `Delete`, hinzugefügt mit `Add`. Die Eigenschaft `Name` übergibt den Namen.

**Beispiel**

Der Benutzer wird nach dem Namen eines Tabellenblatts gefragt. Existiert es, so wird es angesprungen, existiert es nicht, so erhält der Benutzer eine Fehlermeldung.

```
Sub BlattSprung1()
   Dim strBlattName As String
   On Error Resume Next
   strBlattName = InputBox("Wie lautet das gesuchte Blatt?")
   ActiveWorkbook.Sheets(strBlattName).Activate
   If Err.Number = 9 Then
      MsgBox "Das Blatt " & strBlattName & " existiert nicht."
   End If
End Sub
```

Wie im obigen Code wird der Benutzer nach einem Tabellenblattnamen gefragt. Er hat die Möglichkeit, statt des gesamten Namens einen Teil des Namens einzugeben. Existiert das Blatt, dann wird es angesprungen, falls nicht, so erhält der Benutzer eine Fehlermeldung. Eleganter als die obige Lösung ist sicherlich der Code der Prozedur `BlattSprung2`:

```
Sub BlattSprung2()
   Dim xlsTabBlatt As Worksheet
   Dim strBlattName As String
   On Error Resume Next
   strBlattName = InputBox("Wie lautet das gesuchte Blatt?")
```

```
    For Each xlsTabBlatt In Sheets
       If InStr(LCase(xlsTabBlatt.Name), LCase(strBlattName)) > 0 Then
          xlsTabBlatt.Activate
          Exit Sub
       End If
    Next

    MsgBox "Das Blatt " & strBlattName & " existiert nicht."
End Sub
```

Übrigens könnte man auch mit Platzhaltern arbeiten lassen. Dann würde die Bedingung lauten:

```
If LCase(xlsTabBlatt.Name) Like LCase(strBlattName) Then
```

Und statt den Funktionen `LCase`, welche die Schreibweise in Kleinbuchstaben verwandeln, könnte man ebenfalls am Anfang des Moduls ein

```
Option Compare Text
```

schreiben.

**Tabelle 6.1 Einige wichtige Methoden des Tabellenblattes**

| Methode | Erläuterung |
|---|---|
| SaveAs | speichert das Tabellenblatt – dient zum Exportieren. |
| PrintOut | druckt das Tabellenblatt. |
| PrintPreview | Datei \| Seitenansicht |
| Copy, Paste, PasteSpecial, Move | kopieren und einfügen |
| Delete | löscht das Tabellenblatt. |
| Calculate | berechnet die Formeln neu. |
| Activate, Select | aktiviert das Blatt. |
| ClearArrows | löscht die Spurpfeile des Detektiven. |
| ClearCircles | löscht Kreise von ungültigen Einträgen. |
| DisplayPageBreaks | zeigt die Seitenumbrüche an. |
| Protect, Unprotect | schützt das Blatt und hebt den Schutz auf. |

**Tabelle 6.2 Einige wichtige Eigenschaften**

| Eigenschaft | Beschreibung |
|---|---|
| Name | der Name des Tabellenblattes |
| PageSetup | Datei \| Seite einrichten |
| Visible | sichtbar oder unsichtbar (xlSheetVisible, xlSheetHidden und xlSheet-VeryHidden) |
| Parent | das übergeordnete Objekt – die Arbeitsmappe |
| ProtectionMode | Art des Schutzes |
| FilterMode, AutoFilter, AutoFilterMode, ShowAll-Data, EnableAutoFilter | der Autofilter |
| EnableOutlining | Gliederungssymbole |

| Eigenschaft | Beschreibung |
|---|---|
| CircleInvalid | kreist ungültige Einträge ein. |

**Tabelle 6.3 Die wichtigsten Sammlungen des Tabellenblattes**

| Sammlung | Erläuterung |
|---|---|
| Columns und Rows | Spalten und Zeilen |
| Cells, Range, UsedRange | Zellen |
| Comments | Kommentare |
| ChartObjects | Diagramme |
| PivotTables | Pivot-Tabelen |
| Hyperlinks | Hyperlinks |
| Scenarios | Szenarien (Extras \| Szenarien) |
| Shapes | Zeichnungsobjekte |

### 6.1.3 Blätter löschen und neu erzeugen

Hierzu einige Beispiele:

**Beispiel**

Das folgende Beispiel erzeugt eine neue Datei, löscht sämtliche Blätter bis auf eines und fügt elf neue Blätter hinzu. Diese werden anschließend Januar, Februar, März, … Dezember benannt.

```
Sub NeueDateiMitMonatsblättern()

   Dim xlDateiNeu As Workbook

   Dim intBlattZähler As Integer


   Set xlDateiNeu = Application.Workbooks.Add

   Application.DisplayAlerts = False

   For intBlattZähler = xlDateiNeu.Worksheets.Count To 2 Step -1

      xlDateiNeu.Worksheets(intBlattZähler).Delete

   Next

   Application.DisplayAlerts = True

   For intBlattZähler = 2 To 12

      xlDateiNeu.Worksheets.Add

   Next

   For intBlattZähler = 1 To 12

      xlDateiNeu.Worksheets(intBlattZähler).Name = _

         Format(DateSerial(2007, intBlattZähler, 1), "MMMM")

   Next

End Sub
```


**Abbildung 6.1** Die Tabellenblätter der neuen Datei

Einige Erläuterungen zum Code:

- Das Ergebnis der Methode `Application.Workbooks.Add` wird an die Objektvariable `xlDateiNeu` übergeben, weil mit ihr weitergearbeitet wird.

  Die Zeile

  ```
  Application.DisplayAlerts = False
  ```

  schaltet die Warnmeldungen aus. Würde dies unterbleiben, würde der Benutzer beim Löschen jedes Blattes gefragt werden, ob er das Blatt wirklich löschen möchte. Danach sollten Sie die Warnmeldungen wieder einschalten.
- Die Zählung der Indices muss von oben nach vorgenommen werden, da sich sonst die Nummerierung verschiebt: Sie löschen Blatt Nr. zwei – also ist das dritte Blatt die Nummer zwei. Wenn Sie nun das Blatt Nummer drei löschen, hätten Sie ein Blatt übersprungen und der Index würde am Ende des Programms eine Fehlermeldung liefern.
- Zum Herunterzählen benötigen Sie das Schlüsselwort `Step -1`.
- Statt mit einer Schleife elf neue Blätter hinzuzufügen hätte man auch mit einem Befehl elf neue Blätter erzeugen können:

  ```
  xlDateiNeu.Worksheets.Add Count:=11
  ```

- Noch eleganter hätte man anstelle des Löschens mit einem Befehl neue Blätter erzeugen können:

  ```
  Set xlDateiNeu = Application.Workbooks.Add

  If xlDateiNeu.Worksheets.Count < 12 Then

     xlDateiNeu.Worksheets.Add Count:=12 - xlDateiNeu.Worksheets.Count

  Else

     Application.DisplayAlerts = False

     For intBlattZähler = xlDateiNeu.Worksheets.Count To 13 Step -1

        xlDateiNeu.Worksheets(intBlattZähler).Delete

     Next

     Application.DisplayAlerts = True

  End If
  ```

- Die Funktion `DateSerial(2007, intBlattZähler, 1)` baut ein Datum zusammen: 01.01.2007, 01.02.2007, 01.03.2007, … und formatiert diese Datumsangabe als Text: Januar, Februar, März, … So werden die zwölf Blätter benannt. Damit spart man sich das Anlegen einer benutzerdefinierten Liste zum Abspeichern der zwölf Monatsnamen.

**Beispiel**

Im nächsten Beispiel wird hinter das letzte Blatt ein neues Blatt eingefügt:

```
Sub NeuesBlatt()

   Dim xlBlätter As Sheets

   Dim intBlattAnzahl As Integer


   Set xlBlätter = ActiveWorkbook.Sheets
```

```
    intBlattAnzahl = xlBlätter.Count

    xlBlätter.Add After:=xlBlätter(intBlattAnzahl)

    xlBlätter(intBlattAnzahl + 1).Name = "gesamt"


End Sub
```

Natürlich geht das Erzeugen eines neuen Blattes auch als Einzeiler – was allerdings nicht die Lesbarkeit erhöht:

```
ActiveWorkbook.Sheets.Add _

    After:=ActiveWorkbook.Sheets(ActiveWorkbook.Sheets.Count)
```

**Achtung**

Die Parameter After und Before verlangen nicht die Nummer des Blattes vor dem oder hinter dem das neue Blatt eingefügt werden soll, sondern das Blattobjekt. Man kann mit einem Objektverweis darauf zugreifen oder mit einem Index der Sammlung der Worksheets.

Wenn Sie mit diesem Blatt weiterarbeiten, sollten Sie selbstverständlich mit einer Objektvariablen arbeiten:

```
Set xlNeuesBlatt = ActiveWorkbook.Sheets.Add(After:= ...
```

### 6.1.4 Blätter umbenennen

Um ein Blatt umzubenennen, genügt es, der Eigenschaft Name eines Blattes einen Wert zuzuweisen.

**Achtung**

Ein Blatt umzubenennen birgt zwei gewisse Risiken: Zum einen müssen Sie sicherstellen, dass der Blattname korrekt ist (Blattnamen in Excel dürfen maximal 31 Zeichen lang sein und kein „/“ oder „\“ enthalten). Außerdem sind Blattnamen in Excel eindeutig. Das bedeutet, dass Sie überprüfen müssen, ob der Blattname bereits existiert. Das oben stehende Makro NeuesBlatt würde folglich nur einmal funktionieren. Fangen Sie die Fehler ab, beziehungsweise überprüfen Sie, ob der neu zu vergebende Name bereits existiert. Wie dies funktioniert wird im nächsten Abschnitt in der Funktion BlattExistiert gezeigt.

### 6.1.5 Blätter kopieren und verschieben

Während die Methode `Add` ein Objekt, also ein `Sheet` oder `Worksheet`, zurückgibt, geben die beiden Methoden `Copy` und `Move` kein Objekt zurück. Man muss also genau wissen, wohin man das Blatt verschiebt, beziehungsweise kopiert (oder es mit einer Schleife wieder suchen), wenn damit weitergearbeitet werden soll.

**Beispiel**

Das folgende Beispiel kopiert das erste Blatt vor sich selbst, also vor das erste Blatt. Nun wird in die Zelle A1 das aktuelle Datum geschrieben. Dazu muss zuvor erneut auf das kopierte Blatt verwiesen werden.

```
    Dim xlBlatt As Worksheet

    Set xlBlatt = ActiveWorkbook.Worksheets(1)

    xlBlatt.Copy Before:=xlBlatt

    Set xlBlatt = ActiveWorkbook.Worksheets(1)

    xlBlatt.Range("A1").Value = Date
```

### 6.1.6 Blätter verbergen und schützen

Wenn Sie folgende Zeilen schreiben:

```
Sub BlattVerbergen()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet


   Set xlDatei = ThisWorkbook

   Set xlBlatt = xlDatei.Worksheets(1)

   xlBlatt.Visible = xlSheetVeryHidden

End Sub
```

dann fällt auf, dass bei den automatisch aufgelisteten Elementen drei Konstanten für die Eigenschaften Visible zur Verfügung stehen: `xlSheetVisible`, `xlSheetHidden` und `xlSheetVeryHidden`. Und wo ist `xlSheetVeryHidden`? Wenn Sie auf die oben beschrieben Variante ein Blatt ausblenden, kann der Benutzer es nicht mehr einblenden – es erscheint nicht in der Liste der ausgeblendeten Blätter.

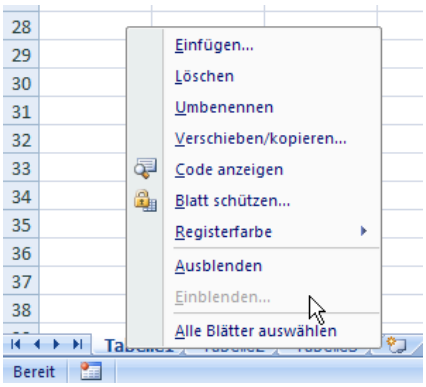



**Abbildung 6.2** Das Tabellenblatt, das mit Visible = xlSheetVeryHidden ausgeblendet wurde, ist für den Benutzer nicht mehr sichtbar.

Erstaunlicherweise kann man auf ein ausgeblendetes Blatt zugreifen und Werte aus Zellen auslesen, beziehungsweise in sie eintragen. Damit steht eine einfache Möglichkeit zur Verfügung, Daten vor dem Anwender zu verbergen. Wenn Sie noch einen Schutz auf das VBA-Projekt legen, hat der Anwender keine Möglichkeit mehr, diese Inhalte auszulesen und zu manipulieren.

**Hinweis**

Sie können diese Eigenschaft auch im Eigenschaftenfenster eintragen. Dort stehen ebenfalls alle drei Varianten.

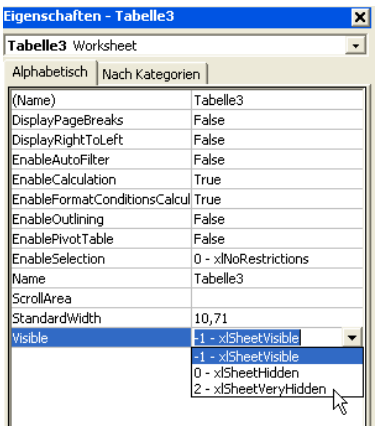



**Abbildung 6.3** Über die Eigenschaften kann das Blatt ebenfalls ausgeblendet werden.

### 6.1.7 Blätter schützen

Wenn Sie ein Blatt schützen möchten, dann steht Ihnen die Methode `Protect` zur Verfügung. Sie verwendet eine Reihe optionaler Parameter:

**Tabelle 6.4 die Parameter der Methode Protect**

| Parameter | Bedeutung |
|---|---|
| Password | Eine Zeichenfolge, mit der für das Arbeitsblatt oder die Arbeitsmappe ein Kennwort mit Unterscheidung der Groß-/Kleinschreibung festlegt wird. Wenn Sie dieses Argument weglassen, kann der Schutz des Arbeitsblatts oder der Arbeitsmappe ohne Angabe eines Kennworts aufgehoben werden. Wenn dies nicht möglich sein soll, müssen Sie ein Kennwort festlegen. Wenn Sie das Kennwort vergessen, können Sie den Schutz des Arbeitsblatts oder der Arbeitsmappe nicht wieder aufheben. |
| DrawingObjects | True, um Formen zu schützen. Der Standardwert lautet True. |
| Contents | True, um den Inhalt zu schützen. Bei einem Diagramm wird das gesamte Diagramm geschützt. Bei einem Arbeitsblatt werden die gesperrten Zellen geschützt. Der Standardwert ist True. |
| Scenarios | True, um Szenarios zu schützen. Das Argument ist nur für Arbeitsblätter gültig. Der Standardwert ist True. |
| UserInterfaceOnly | True, um die Benutzeroberfläche, jedoch keine Makros zu schützen. Ohne Angabe dieses Arguments wird der Schutz auf Makros und die Benutzeroberfläche angewendet. |
| AllowFormattingCells | Mit True können Benutzer jede Zelle eines geschützten Arbeitsblatts formatieren. Der Standardwert ist False. |
| AllowFormattingColumns | Mit Truekönnen Benutzer jede Spalte eines geschützten Arbeitsblatts formatieren. Der Standardwert ist False. |
| AllowFormattingRows | Mit True können Benutzer jede Zeile eines geschützten Arbeitsblatts formatieren. Der Standardwert ist False. |
| AllowInsertingColumns | Mit True können Benutzer Spalten in ein geschütztes Arbeitsblatt einfügen. Der Standardwert ist False. |
| AllowInsertingRows | Mit True können Benutzer Zeilen in ein geschütztes Arbeitsblatt einfügen. Der Standardwert ist False. |
| AllowInsertingHyperlinks | Mit True können Benutzer Hyperlinks auf dem Arbeitsblatt einfügen. Der Standardwert ist False. |
| AllowDeletingColumns | Mit True können Benutzer Spalten im geschützten Arbeitsblatt löschen, wobei keine Zelle in der zu löschenden Spalte gesperrt ist. Der Standardwert ist False. |
| AllowDeletingRows | Mit True können Benutzer Zeilen im geschützten Arbeitsblatt löschen, wobei keine Zelle in der zu löschenden Zeile gesperrt ist. Der Standardwert ist False. |
| AllowSorting | Mit True können Benutzer für das geschützte Arbeitsblatt eine Sortierung vornehmen. Für jede Zelle im Sortierbereich muss die Sperre oder der Schutz aufgehoben werden. Der Standardwert ist False. |
| AllowFiltering | Mit True Benutzer Filter für das geschützte Arbeitsblatt festlegen. Die Benutzer können Filterkriterien ändern, jedoch keinen AutoFilter aktivieren oder deaktivieren. Die Benutzer können Filter für einen vorhandenen AutoFilter festlegen. Der Standardwert ist False. |
| AllowUsingPivotTables | Mit True können Benutzer PivotTable-Berichte für das geschützte Arbeitsblatt verwenden. Der Standardwert ist False. |

### 6.1.8 Seite einrichten

Der bekannte Dialog aus dem Menüpunkt Layout (bis Excel 2003: Datei) Seite einrichten kann mithilfe des Makrorekorders aufgezeichnet werden. Er liefert zu dem zugehörigen Blatt das Objekt `PageSetup` mit einer Reihe Eigenschaften.

**Hinweis**
Damit Sie dieses Objekt verwenden können, muss ein Drucker installiert sein. Sonst liefert es einen Fehler.

**Beispiel**
Das folgende Beispiel schaltet in der Fußzeile den Dateinamen mit Pfad ein und schaltet ihn wieder aus:

```
Sub Fußzeile()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet

   Dim xlSeiteEinrichten As PageSetup


   Set xlDatei = ThisWorkbook

   Set xlBlatt = xlDatei.Worksheets(2)

   Set xlSeiteEinrichten = xlBlatt.PageSetup


   With xlSeiteEinrichten

      If .RightFooter = "" Then

         .RightFooter = xlDatei.FullName

      Else

         .RightFooter = ""

      End If

   End With

End Sub
```

**Tabelle 6.5 die Eigenschaften des Objektes PageSetup**

| Name | Beschreibung |
|---|---|
| **Papierformat** | |
| Orientation | Gibt einen XlPageOrientation-Wert zurück, der den Druckmodus im Hochformat oder im Querformat darstellt, oder legt diesen fest. |
| Zoom | Gibt einen Variant-Wert zurück, der den Prozentsatz (zwischen 10 % und 400 %) darstellt, um den das Arbeitsblatt beim Drucken von Microsoft Excel vergrößert oder verkleinert wird, oder legt diesen fest. |
| FitToPagesWide | Gibt die Anzahl der Seiten zurück (oder legt diese fest), auf die das Arbeitsblatt während des Druckens in der Breite skaliert wird. Nur gültig für Arbeitsblätter. |

| Name | Beschreibung |
|---|---|
| FitToPagesTall | Gibt die Anzahl der Seiten zurück (oder legt diese fest), auf die das Arbeitsblatt während des Druckens in der Höhe skaliert wird. Nur gültig für Arbeitsblätter. |
| Draft | True, wenn das Blatt ohne Grafiken gedruckt wird. |
| PaperSize | Gibt die Größe des Papiers zurück oder legt sie fest. |
| PrintQuality | Gibt die Druckqualität zurück oder legt diese fest. |
| **Seitenränder** | |
| TopMargin | Gibt die Breite des oberen Randes in Punkt zurück oder legt diese fest. Double-Wert mit Lese-/Schreibzugriff. |
| BottomMargin | Gibt die Breite des unteren Rands in Punkt zurück oder legt diese fest. Double-Wert mit Lese-/Schreibzugriff. |
| LeftMargin | Gibt die Breite des linken Randes in Punkt zurück oder legt sie fest. Double-Wert mit Lese-/Schreibzugriff. |
| RightMargin | Gibt die Breite des rechten Rands in Punkt zurück oder legt sie fest. Double-Wert mit Lese-/Schreibzugriff. |
| CenterHorizontally | True, falls das Arbeitsblatt horizontal auf der Druckseite zentriert ist. |
| CenterVertically | True, falls das Arbeitsblatt vertikal auf der Druckseite zentriert ist. |
| FooterMargin | Gibt den Abstand der Fußzeile vom unteren Rand der Seite in Punkt zurück oder legt ihn fest. |
| HeaderMargin | Gibt den Abstand der Kopfzeile vom oberen Rand der Seite in Punkt zurück oder legt ihn fest. |
| **Kopfzeile/Fußzeile** | |
| LeftHeader | Gibt die Ausrichtung von Text in der linken Kopfzeile einer Arbeitsmappe oder eines Abschnitts zurück oder legt diese fest. |
| LeftHeaderPicture | Gibt ein Graphic-Objekt zurück, das die Grafik für den linken Abschnitt der Kopfzeile darstellt. Wird dazu verwendet, Attribute zur Grafik festzulegen. |
| CenterHeader | Zentriert die Kopfzeileninformationen im PageSetup-Objekt. |
| CenterHeaderPicture | Gibt ein Graphic-Objekt zurück, das die Grafik für den mittleren Abschnitt der Kopfzeile darstellt. Wird verwendet, um Attribute zur Grafik festzulegen. |
| RightHeader | Gibt den rechten Teil der Kopfzeile zurück oder legt ihn fest. |
| RightHeaderPicture | Gibt die Grafik an, die in der rechten Kopfzeile angezeigt werden soll. Schreibgeschützt. |
| LeftFooter | Gibt die Ausrichtung von Text in der linken Fußzeile einer Arbeitsmappe oder eines Abschnitts zurück oder legt diese fest. |
| LeftFooterPicture | Gibt ein Graphic-Objekt zurück, das die Grafik für den linken Abschnitt der Fußzeile darstellt. Wird dazu verwendet, Attribute zur Grafik festzulegen. |
| CenterFooter | Zentriert die Fußzeileninformationen im PageSetup-Objekt. |
| CenterFooterPicture | Gibt ein Graphic-Objekt zurück, das die Grafik für den mittleren Abschnitt der Fußzeile darstellt. Wird verwendet, um Attribute zur Grafik festzulegen. |
| RightFooter | Gibt den Abstand (in Punkt) zwischen dem rechten Rand der Seite und dem rechten Rand der Fußnote zurück oder legt den Abstand fest. |
| RightFooterPicture | Gibt ein Graphic-Objekt zurück, das die Grafik für den rechten Abschnitt der Fußzeile darstellt. Wird dazu verwendet, Attribute zur Grafik festzulegen. |

| Name | Beschreibung |
|---|---|
| AlignMarginsHeaderFooter (ab Excel 2007) | Gibt True zurück, wenn die Kopf- und Fußzeile in Excel an den Rändern ausgerichtet werden soll, die in den Seiteneinrichtungsoptionen festgelegt sind. |
| DifferentFirstPageHeaderFooter (ab Excel 2007) | True, wenn auf der ersten Seite eine andere Kopf- oder Fußzeile verwendet wird. |
| EvenPage (ab Excel 2007) | Gibt die Ausrichtung von Text auf den geraden Seiten einer Arbeitsmappe oder eines Abschnitts zurück oder legt diese fest. |
| FirstPage (ab Excel 2007) | Gibt die Ausrichtung von Text auf der ersten Seite einer Arbeitsmappe oder eines Abschnitts zurück oder legt diese fest. |
| FirstPageNumber (ab Excel 2007) | Gibt die Seitenzahl für die erste Seite zurück, die beim Drucken dieses Blatts verwendet wird, oder legt diese fest. Wenn die Eigenschaft auf xlAutomatic festgelegt ist, wird die erste Seitenzahl von Microsoft Excel ausgewählt. |
| OddAndEvenPagesHeaderFooter (ab Excel 2007) | True, wenn das angegebene PageSetup-Objekt für Seiten mit geraden und ungeraden Seitenzahlen unterschiedliche Kopf- und Fußzeilen verwendet. |
| ScaleWithDocHeaderFooter (ab Excel 2007) | Gibt zurück oder legt fest, ob Kopf- und Fußzeile mit dem Dokument skaliert werden sollen, wenn die Größe des Dokuments geändert wird. |
| **Tabelle** | |
| BlackAndWhite | True, wenn die Elemente des Dokuments in schwarzweiß gedruckt werden. Boolean-Wert mit Lese-/Schreibzugriff. |
| Order | Gibt einen XlOrder-Wert zurück, der die von Microsoft Excel beim Drucken eines umfangreichen Arbeitsblatts zum Nummerieren von Seiten verwendete Reihenfolge darstellt, oder legt diesen fest. |
| Pages | Gibt die Anzahl der Seiten in der Pages-Auflistung zurück oder legt diese fest. |
| PrintArea | Gibt den Druckbereich als Zeichenfolge in der Sprache des Makros zurück oder legt ihn fest. Die Zeichenfolge verwendet den Bezug in der A1-Schreibweise. |
| PrintComments | Gibt die Art zurück, in der Kommentare zusammen mit dem Blatt gedruckt werden, oder legt sie fest. |
| PrintErrors | Gibt eine XlPrintErrors-Konstante zurück oder legt diese fest, die den Typ des angezeigten Druckfehlers angibt. Diese Funktion ermöglicht es den Benutzern, beim Drucken eines Arbeitsblatts die Anzeige von Fehlerwerten zu unterdrücken. |
| PrintGridlines | True, wenn Gitternetzlinien gedruckt werden sollen. Gilt nur für Arbeitsblätter. |
| PrintHeadings | True, falls Zeilen- und Spaltenköpfe mit dieser Seite gedruckt werden sollen. Gilt nur für Arbeitsblätter. |
| PrintNotes | True, wenn Notizen als Endnoten zusammen mit dem Blatt gedruckt werden. Gilt nur für Arbeitsblätter. |
| PrintTitleColumns | Gibt die Spalten, die links auf jeder Seite als Beschriftung wiederholt werden sollen, als Zeichenfolge in der Sprache des Makros zurück oder legt sie fest. Die Zeichenfolge verwendet den Bezug in der A1-Schreibweise. |
| PrintTitleRows | Gibt die Zeilen, die oben auf jeder Seite als Beschriftung wiederholt werden sollen, als Zeichenfolge in der Sprache des Makros zurück oder legt sie fest. Die Zeichenfolge verwendet den Bezug in der A1-Schreibweise. |

### 6.1.9 Existiert ein Blatt?

**Beispiel**

Leider stellt Excel-VBA keine Funktion zur Verfügung, mit deren Hilfe überprüft werden kann, ob ein Blatt vorhanden ist. Dies kann programmiert werden:

```
Function BlattExistiert(strBlattname As String) As Boolean

   Dim blnBlattEx As Boolean

   Dim xlBlatt As Worksheet

   blnBlattEx = False

   For Each xlBlatt In ActiveWorkbook.Worksheets

      If xlBlatt.Name = strBlattname Then

         blnBlattEx = True

         Exit For

      End If

   Next

   BlattExistiert = blnBlattEx

End Function
```

Oder – wenn Sie nicht zwischen Groß- und Kleinschreibung unterscheiden möchten:

```
If LCase(xlBlatt.Name) = LCase(strBlattname) Then
```

Oder – wenn ein Teil des Blattnamens zulässig ist:

```
If InStr(1, xlBlatt.Name, strBlattname, vbTextCompare) > 0 Then
```

**Beispiel**

Mit diesem Wissen können Sie nun leicht einen Assistenten erstellen, mit dessen Hilfe der Benutzer seine Blätter verschieben oder sortieren kann.

Beim Laden des Assistenten werden sämtliche Blattnamen in einem Listenfeld angezeigt:

```
Private Sub UserForm_Initialize()

   Dim intBlattAnzahl As Integer

   Dim intBlattZähler As Integer


   intBlattAnzahl = ActiveWorkbook.Worksheets.Count


   If intBlattAnzahl = 1 Then

      MsgBox "Der Assistent benötigt eine Datei mit mindestens " & _

         "zwei Blättern", vbInformation

   Else

      For intBlattZähler = 1 To intBlattAnzahl
```

```
            Me.lstTabellenblätter.AddItem _
               ActiveWorkbook.Worksheets(intBlattZähler).Name
         Next


         Me.cmdNachOben.Enabled = False
         Me.cmdNachUnten.Enabled = False
         blnSortiervorgang = False
      End If
   End Sub
```

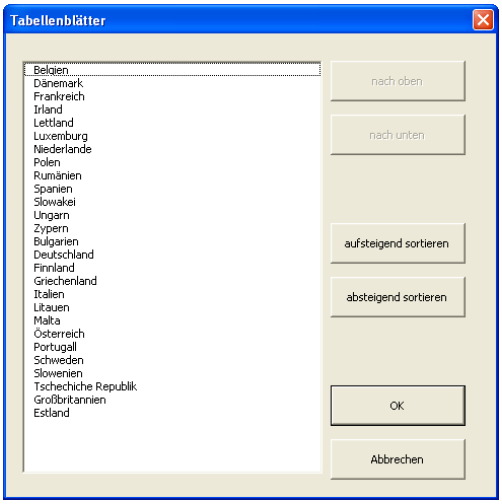



**Abbildung 6.4** Die Blattnamen werden angezeigt.

Wird ein Eintrag ausgewählt, dann wird überprüft, ob es der erste, der letze oder ein anderer ist. Dementsprechende werden die beiden Schaltflächen cmdNachOben oder cmdNachUnten inaktiv.

```
   Private Sub lstTabellenblätter_Change()
      If blnSortiervorgang = False Then
         If Me.lstTabellenblätter.Selected(0) = True Then
            Me.cmdNachOben.Enabled = False
            Me.cmdNachUnten.Enabled = True
         ElseIf Me.lstTabellenblätter. _
            Selected(Me.lstTabellenblätter.ListCount - 1) = True Then
            Me.cmdNachOben.Enabled = True
            Me.cmdNachUnten.Enabled = False
         Else
            Me.cmdNachOben.Enabled = True
            Me.cmdNachUnten.Enabled = True
         End If
```

```
    End If
End Sub
```

Nun kann ein ausgewählter Eintrag nach oben, beziehungsweise nach unten verschoben werden. Er wird hierzu aus der Liste gelöscht und eine Position darüber wieder eingefügt. Danach wird er markiert.

```
Private Sub cmdNachOben_Click()
    Dim intBlattZähler As Integer
    Dim intAusgewähltesBlatt As Integer
    Dim strBlattName As String

    For intBlattZähler = 0 To Me.lstTabellenblätter.ListCount - 1
        If Me.lstTabellenblätter.Selected(intBlattZähler) = True Then
            intAusgewähltesBlatt = intBlattZähler
            strBlattName = Me.lstTabellenblätter.Value
        End If
    Next

    Me.lstTabellenblätter.RemoveItem intAusgewähltesBlatt
    Me.lstTabellenblätter.AddItem strBlattName, intAusgewähltesBlatt - 1
    Me.lstTabellenblätter.ListIndex = intAusgewähltesBlatt - 1
End Sub
```


**Abbildung 6.5** Ein Eintrag wird nach oben (oder unten) verschoben.

Soll die Liste sortiert werden, dann wird sie in eine Array eingelesen. Dieses kann mit dem Bubblesort-Algorithmus sortiert werden. Anschließend wird die Liste geleert und die Daten wieder zurück geschrieben.

```
Private Sub cmdSortAufsteigend_Click()
    Dim strTemp As String
```

```
    Dim intBlattZähler1 As Integer
    Dim intBlattZähler2 As Integer
    Dim strListe()
    ReDim strListe(1 To Me.lstTabellenblätter.ListCount)

    blnSortiervorgang = True

    For intBlattZähler1 = 1 To Me.lstTabellenblätter.ListCount
       strListe(intBlattZähler1) = _
          Me.lstTabellenblätter.List(intBlattZähler1 - 1)
    Next

    For intBlattZähler1 = 1 To UBound(strListe)
       For intBlattZähler2 = intBlattZähler1 To UBound(strListe)
          If strListe(intBlattZähler2) < strListe(intBlattZähler1) Then
             strTemp = strListe(intBlattZähler2)
             strListe(intBlattZähler2) = strListe(intBlattZähler1)
             strListe(intBlattZähler1) = strTemp
          End If
       Next intBlattZähler2
    Next intBlattZähler1

    Me.lstTabellenblätter.Clear

    For intBlattZähler1 = 1 To UBound(strListe)
       Me.lstTabellenblätter.AddItem strListe(intBlattZähler1)
    Next

    blnSortiervorgang = False
End Sub
```

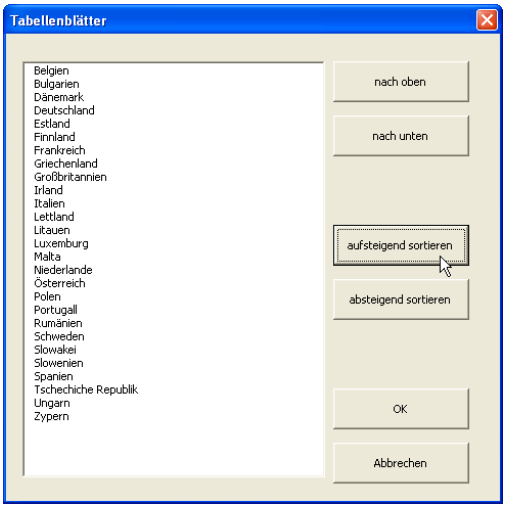



**Abbildung 6.6** Die Liste wird sortiert.

**Achtung**

Beim Sortieren wird das Ereignis lstTabellenblätter_Change ausgelöst. Dies führt zu einem Fehler, wenn die Liste leer ist. Deshalb wird mit einer globalen Variable dieses Ereignis „temporär“ ausgeschaltet.

**Hinweis**

Damit „Österreich“ nicht nach Zypern einsortiert wird, sondern zwischen „Niederlande“ und „Polen“ muss am Anfang der Userform der folgende Befehl stehen:

```
Option Compare Text
```

Der Benutzer kann nun die Tabellenblätter der Datei so verschieben, wie er es in der Listesortiert oder verschoben hat. In dieser Reihenfolge werden die Tabellenblätter der Datei verschoben. Das erste Blatt wird ausgewählt und der Dialog geschlossen.

```
Private Sub cmdOK_Click()

   Dim intBlattZähler As Integer

   Dim xlBlatt As Worksheet


   For intBlattZähler = 1 To Me.lstTabellenblätter.ListCount - 1

      Set xlBlatt = _

         ActiveWorkbook.Worksheets(Me.lstTabellenblätter. _

         List(intBlattZähler - 1))

      xlBlatt.Move Before:=ActiveWorkbook.Worksheets(intBlattZähler)

   Next


   ActiveWorkbook.Worksheets(1).Activate


   Unload Me

End Sub
```


**Abbildung 6.7** Die neu sortierte Liste

Diese Userform könnte an ein Add-In gebunden werden, das bei Start in Excel zur Verfügung steht.

## 6.2 Fazit

Ebenso wie beim Objekt Workbook sind auch bei den Objekten Sheet und Worksheet einige Dinge zu beachten. Am wichtigsten ist sicherlich die Unterscheidung zwischen beiden – vor allem dann, wenn die Datei Tabellenblätter und Diagrammblätter enthält.

Die Methoden Add, Move, Copy, Delete, Activate können sicherlich erraten werden; interessant ist in diesem Zusammenhang, dass ein einzelnes Blatt gespeichert und gedruckt werden kann. Für die Eigenschaft Visible ist bemerkenswert, dass es neben sichtbar oder unsichtbar (xlSheetVisible und xlSheetHidden) noch über eine dritte Variante vefügt: xlSheetVeryHidden, mit deren Hilfe das Blatt so unsichtbar gemacht werden kann, dass der Anwender von Excel aus es nicht mehr einblenden kann.

# 7 Das Excel-Objektmodell: Range

Die wohl häufigste Zugriffsart ist sicherlich der Zellzugriff. Dabei kann der Cursor auf eine Zelle gesetzt werden, und diese kann dann modifiziert werden, oder der Zugriff kann indirekt über einen Objektzugriff erfolgen. Der Objektzugriff über Objektvariable hat den Vorteil, dass Sie flexibler und schneller sind, wenn Sie viele Tausend Daten von einem Blatt auf ein anderes „schieben“ oder vergleichen müssen. Dieser Variante sollten Sie den Vorzug geben.

Leider lese ich häufig ich in Fachbüchern und –zeitschriften Code, wie beispielsweise:

```
Range("C7") = 104
```

oder auch:

```
Range("C7").Select

ActiveCell.FormulaR1C1 = "104"
```

Beide Varianten funktionieren zwar, haben jedoch einige gravierende Nachteile.

Im ersten Beispiel werden Objekt und Eigenschaft des Objektes vermischt – es wird stillschweigend die Standardeigenschaft des Objektes verwendet, was die Lesbarkeit erschwert. Im Sinne der Objektorientierung wird außerdem nicht geklärt, von welchen Objekten die Zelle C7 abgeleitet wird, das heißt konkret: zu welchem Tabellenblatt und zu welcher Datei sie gehört.

Im zweiten Beispiel wird der Cursor auf die Zelle C7 gesetzt. Das ist zeitaufwendig und ist bei sehr großen Programmen unflexibel. Davon sollte Abstand genommen werden.

Außerdem wäre die Eigenschaft `Value` besser geeignet als `FormulaR1C1`, die vom Makrorekorder aufgezeichnet wird.

## 7.1 Zellen: Rows und Columns, Range und Cell

Angenommen, Sie möchten auf die Zelle B9 des Tabellenblatts „Filmliste“ der Datei Sammlung.xls zugreifen. Ist diese Datei offen, dann kann auf sie verwiesen werden. Danach wird auf das Tabellenblatt Bezug genommen und schließlich auf die Zelle. Der Zugriff darüber erfolgt mit dem Objekt `Range`:

```
Sub AufB9Zugreifen3()

   Dim xlDatei As Workbook

   Dim xlTabelle As Worksheet

   Dim xlZelle As Range

   Set xlDatei = Application.Workbooks("Sammlung.xls")

   Set xlTabelle = xlDatei.Sheets("Filmliste")

   Set xlZelle = xlTabelle.Range("B9")
```

```
    MsgBox xlZelle.Value

End Sub
```

Zellinhalte können abgefragt (wie oben) oder auch gesetzt werden. Der Befehl

```
xlZelle.Value = "Titanic"
```

schreibt den Wert „Titanic“ in die Zelle `xlZelle`. `Activate` oder `Select` sind zwei Methoden, mit denen Zellen aktiviert werden. Statt des Objekts `Range` kann auch `Cells` als Sammlung verwendet werden:

```
Set xlZelle = xlTabelle.Cells(9, 2)
```

**Achtung**

Dabei wird zuerst die Zeile (Rows) und dann die Spalte (Columns) angegeben. Dies kann für uns europäische Exceluser verwirrend sein, weil wir den Zellnamen B7 lesen: Spalte B, Zeile 7 und nicht Rows(7), Columns(2).

### 7.1.1 Cell, Range, Selection, ActiveCell & co

Wenn Sie mit dem Makrorekorder etwas aufzeichnen, dann erstaunt Sie vielleicht, dass er manchmal `ActiveCell` manchmal `Selection` aufzeichnet. Excel unterscheidet nicht zwischen einer Zelle und einem Zellbereich. Deshalb muss der Objektverweis mit der Objektvariablen Range durchgeführt werden – Sie können schreiben:

```
Set xlZellBereich = xlBlatt.Range("A1")

Set xlZellBereich = xlBlatt.Range("A1:C17")
```

Interessant wird es jedoch, wenn Sie per Programmierung markieren.

```
xlBlatt.Range("A1:C17").Select

xlBlatt.Range("B5").Activate
```

Erstaunlicherweise funktionieren auch folgende Zeilen:

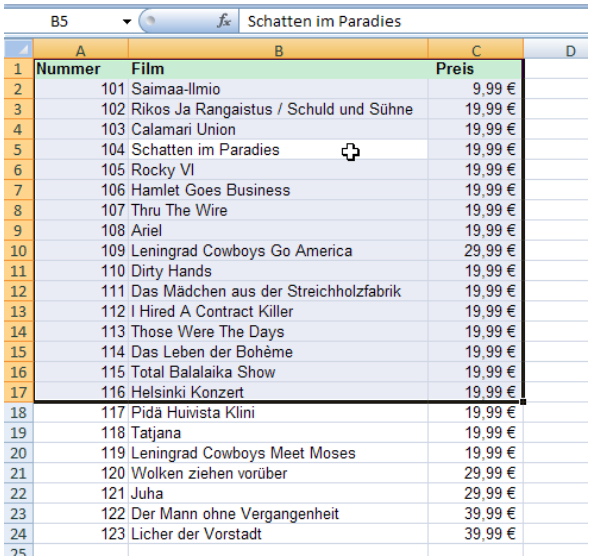
```
xlBlatt.Range("A1:C17").Activate

xlBlatt.Range("B5").Activate
```

Allerdings würde

```
xlBlatt.Range("A1:C17").Select

xlBlatt.Range("B5").Select
```

nur die Zelle B5 auswählen.

Damit wird auch klar, dass sich `Selection` auf A1:C17 bezieht, `ActiveCell` jedoch auf die Zelle B5 innerhalb dieses Bereichs.




| | A | B | C | D |
|---|---|---|---|---|
| 1 | Nummer | Film | Preis | |
| 2 | 101 | Saimaa-Ilmio | 9,99 € | |
| 3 | 102 | Rikos Ja Rangaistus / Schuld und Sühne | 19,99 € | |
| 4 | 103 | Calamari Union | 19,99 € | |
| 5 | 104 | Schatten im Paradies | 19,99 € | |
| 6 | 105 | Rocky VI | 19,99 € | |
| 7 | 106 | Hamlet Goes Business | 19,99 € | |
| 8 | 107 | Thru The Wire | 19,99 € | |
| 9 | 108 | Ariel | 19,99 € | |
| 10 | 109 | Leningrad Cowboys Go America | 29,99 € | |
| 11 | 110 | Dirty Hands | 19,99 € | |
| 12 | 111 | Das Mädchen aus der Streichholzfabrik | 19,99 € | |
| 13 | 112 | I Hired A Contract Killer | 19,99 € | |
| 14 | 113 | Those Were The Days | 19,99 € | |
| 15 | 114 | Das Leben der Bohème | 19,99 € | |
| 16 | 115 | Total Balalaika Show | 19,99 € | |
| 17 | 116 | Helsinki Konzert | 19,99 € | |
| 18 | 117 | Pidä Huivista Kiini | 19,99 € | |
| 19 | 118 | Tatjana | 19,99 € | |
| 20 | 119 | Leningrad Cowboys Meet Moses | 19,99 € | |
| 21 | 120 | Wolken ziehen vorüber | 29,99 € | |
| 22 | 121 | Juha | 29,99 € | |
| 23 | 122 | Der Mann ohne Vergangenheit | 39,99 € | |
| 24 | 123 | Licher der Vorstadt | 39,99 € | |

**Abbildung 7.1** `xlBlatt.Range("A1:C17").Select: xlBlatt.Range("B5").Activate`

**Hinweis**

Aus der Datenbankprogrammierung stammt die etwas veraltete Befehlsschreibweise [B5]. Wenn Sie diese Schreibweise bevorzugen, dann können Sie es gerne tun:

```
xlBlatt.[B5]

xlBlatt.[A1:C17]
```

Auf den ersten Blick umständlicher funktioniert der Zugriff über den Bereich:

```
Set xlBereich = xlBlatt.Range(xlBlatt.Cells(1, 1), xlBlatt.Cells(17, 3))
```

Der Vorteil davon ist allerdings, dass die Eckkoordinaten getrennt berechnet werden können:

```
x1 = 1

x2 = 17

y1 = 1

y2 = 3

Set xlBereich = xlBlatt.Range(xlBlatt.Cells(x1, y1), xlBlatt.Cells(x2, y2))
```

So kann ein Bereich per indirektem Zellzugriff formatiert werden:

```
Dim x1 As Integer, x2 As Integer

Dim y1 As Integer, y2 As Integer

Dim xlsTab As Worksheet

Dim xlsZelle1 As Range, xlsZelle2 As Range

Dim xlsBereich As Range


Set xlsTab = Application.ActiveWorkbook.Worksheets(3)

x1 = 1

x2 = 17
```

```
y1 = 1

y2 = 3

Set xlsZelle1 = xlsTab.Cells(x1, y1)

Set xlsZelle2 = xlsTab.Cells(x2, y2)

Set xlsBereich = xlsTab.Range(xlsZelle1, xlsZelle2)


xlsBereich.Font.Bold = False
```

### 7.1.2 „Bewegen“: Bereich verschieben

Mit einem ähnlichen Objekt wie `Cells` kann man sich „bewegen“. Verweist `xlZelle` auf die Zelle B9, so wird auf die Zelle B10 mit dem Befehl

```
Set xlZelle = xlZelle.Offset(1, 0)
```

verwiesen. Von B9 geht es zurück auf B8 mit

```
Set xlZelle = xlZelle.Offset(-1, 0)
```

auf C9 mit:

```
Set xlZelle = xlZelle.Offset(0, 1)
```

Soll dagegen nur ein Wert überprüft oder gesetzt werden, so genügt:

```
MsgBox xlZelle.Offset(1, 0).Value
```

Die Variable verweist noch immer auf die „alte“ Zelle, während das Meldungsfenster den Wert der darunter liegenden Zelle anzeigt. Erstaunlichweise geschieht das gleich mit dem Befehl:

```
Set xlZelle = xlZelle.Range("A2")
```

Allerdings erscheint Range("A2") als Anweisung „eine Zeile nach unten“ schwer verständlich lesbar. Analog dazu wäre auch die Schreibweise

```
Set xlZelle = xlZelle.Cells(2,0)
```

möglich. Der Zugriff über `Cells` eignet sich sehr gut, wenn mit Variablen gearbeitet wird.

Beachten Sie, dass der Makrorekorder an einigen Stellen überflüssigen Code aufzeichnet. Wenn Sie sich zwei Spalten nach links und vier Zeilen nach oben bewegen, dann zeichnet der Makrorekorder relativ auf:

```
    ActiveCell.Offset(-2, -4).Range("A1").Select
```

Die Eigenschaft Range("A1") st dabei überflüssig.

Bezüge auf Bereiche können selbstverständlich wie Zellbezüge verschoben werden:

```
Set xlBereich = xlBereich.Offset(0, 3)
```

verschiebt den Bereich um drei Spalten nach rechts.

Mit der Eigenschaft `Resize` kann ein Bereich in seiner Größe verändert werden:

```
Set xlBereich = xlBlatt.Range("A1")

Set xlBereich = xlBereich.Resize(2, 2)

Set xlBereich = xlBlatt.Range("A1:C7")

Set xlBereich = xlBereich.Resize(2, 2)
```

liefert jeweils den Bereich A1:B2. Leider gibt es keine Eigenschaft oder Methode, einen vorhandenen Bereich um eine bestimmte Anzahl an Spalten oder Zeilen zu verkleinern oder zu vergrößern.

Allerdings stellt Application die beiden Methoden `Union` und `Intersect` zur Verfügung, mit denen Bereiche vereinigt oder beschnitten werden können:

```
Set xlBereich = xlBlatt.Range("A1:F7")

Set xlBereich = xlApp.Union(xlBereich, xlBlatt.Range("D1:G7"))

Set xlBereich = xlApp.Intersect(xlBereich, xlBlatt.Range("E3:J10"))
```

Diese drei Anweisungen geben am Ende den Bereich E3:G7 für das Objekt `xlBereich` zurück.

### 7.1.3 Der Zellbezug verschwindet

Anfänger haben manchmal Schwierigkeiten das Problem zu verstehen, das sich hinter folgenden Zeilen verbirgt:

```
Set xlZellBereich = xlBlatt.Range("A1:A3")


xlZellBereich.EntireRow.Delete

xlZellBereich.Interior.Color = vbBlue
```

Die Lösung des Problems ist relativ simpel: Durch das Löschen der ersten drei Zeilen „verschwindet“ auch der Bezug auf die Zellen A1:A3. Damit Sie mit diesen Zellen weiterarbeiten können, muss der Bezug erneut gesetzt werden. Das Beispiel funktioniert so:

```
Set xlZellBereich = xlBlatt.Range("A1:A3")

xlZellBereich.EntireRow.Delete


Set xlZellBereich = xlBlatt.Range("A1:A3")

xlZellBereich.Interior.Color = vbBlue
```

### 7.1.4 Kopieren, Ausschneiden und Einfügen

Die Methode `Copy` wird auf einen Range angewandt. Sie kann entweder den Bereich in den Zwischenspeicher kopieren oder gleich an eine andere Stelle kopieren. Im zweiten Fall geben Sie den Parameter `Destination` ein:

```
Set xlDatei = ActiveWorkbook

Set xlBlatt = xlDatei.Worksheets(3)

Set xlQuellBereich = xlBlatt.Range("D1:D11")

Set xlZielBereich = xlBlatt.Range("J1")


xlQuellBereich.Copy Destination:=xlZielBereich
```

Analog können Sie die Methode `Cut` verwenden:

```
xlQuellBereich.Cut Destination:=xlZielBereich
```

**Tipp**

Sie sollten nicht den Code verwenden, den der Makrorekorder aufzeichnet beim Kopieren und Einfügen. Sollten Sie – warum auch immer – dennoch verwenden müssen, dann beachten Sie zwei Dinge:

```
Selection.Copy
```

```
Range("H1").Select

ActiveSheet.Paste

Application.CutCopyMode = False
```

Die Methode `Paste` wird nicht auf die (Ziel-)Zelle angewandt, sondern auf ein bestimmtes Blatt. Und danach sollten Sie die laufende Linie ausschalten. Dafür stellt Ihnen Application die Eigenschaft `CutCopyMode` zur Verfügung.

```
Application.CutCopyMode = False
```

Zugegeben: An einer Stelle bei der Programmierung kann das Kopieren und Einfügen Sinn machen: beim Inhalte einfügen. Angenommen, Sie haben in einer Tabelle eine Spalte, in der sich Formeln und Funktionen befinden. Dann können Sie diese kopieren und mithilfe des Assistenten Inhalte einfügen wieder so über die (gleichen oder anderen) Zellen schreiben, dass er nur noch die Werte enthält. Die beiden notwendigen Zeilen hierfür lauten:

```
xlBereich.Copy

xlBereich.PasteSpecial Paste:=xlPasteValues, _

   Operation:=xlNone, SkipBlanks :=False, Transpose:=False
```

Anschließend sollten Sie die gestrichelte Linie ausschalten:

```
Application.CutCopyMode = False
```

**Tabelle 7.1 Für den Parameter Paste der Methode PasteSpecial stehen Ihnen folgende Konstanten zur Verfügung**

| Konstante | Wert | Beschreibung |
|---|---|---|
| xlPasteAll | -4104 | Alles wird eingefügt. |
| xlPasteAllExceptBorders | 7 | Alles außer den Rahmen wird eingefügt. |
| xlPasteAllUsingSourceTheme | 13 | Alles wird unter Verwendung des Quelldesigns eingefügt. |
| xlPasteColumnWidths | 8 | Die kopierte Spaltenbreite wird eingefügt. |
| xlPasteComments | -4144 | Kommentare werden eingefügt. |
| xlPasteFormats | -4122 | Das kopierte Quellformat wird eingefügt. |
| xlPasteFormulas | -4123 | Formeln werden eingefügt. |
| xlPasteFormulasAndNumberFormats | 11 | Formeln und Zahlenformate werden eingefügt. |
| xlPasteValidation | 6 | Überprüfungen werden eingefügt. |
| xlPasteValues | -4163 | Werte werden eingefügt. |
| xlPasteValuesAndNumberFormats | 12 | Werte und Zahlenformate werden eingefügt. |

**Tabelle 7.2 Die Liste der Konstanten des Parameters Operation**

| Konstante | Wert | Beschreibung |
|---|---|---|
| xlPasteSpecialOperationAdd | 2 | Die kopierten Daten werden zum Wert in der Zielzelle addiert. |
| xlPasteSpecialOperationDivide | 5 | Die kopierten Daten werden durch den Wert in der Zielzelle dividiert. |
| xlPasteSpecialOperationMultiply | 4 | Die kopierten Daten werden mit dem Wert in der Zielzelle multipliziert. |
| xlPasteSpecialOperationNone | -4142 | Beim Einfügen wird keine Berechnung ausgeführt. |
| xlPasteSpecialOperationSubtract | 3 | Die kopierten Daten werden vom Wert in der Zielzelle subtrahiert. |

### 7.1.5 Zeilen und Spalten

Wenn Sie eine Zeile vergrößern (verkleinern) möchten, dann müssen Sie ausgehend von einer Zelle oder einem Zellbereich eine gesamte Zeile definieren:

```
Dim xlZeile As Range

Set xlZeile = ActiveSheet.Range("A1").EntireRow
```

Erstaunlicherweise kann man ohne Fehlermeldung von einer Zeile die Zeile bilden:

```
Set xlZeile = xlZeile.EntireRow

Set xlZeile = xlZeile.EntireRow
```

Beachten Sie, dass Sie von diesem Zellbereich nicht die `Height`, sondern die `RowHeight` festlegen müssen:

```
xlZeile.EntireRow.RowHeight = 66
```

**Hinweis**
Die Methode AutoFit passt eine ganze Zeile auf die optimale Höhe an.

Analog funktioniert die Spaltenbreite:

```
Dim xlSpalte As Range

   Set xlSpalte = ActiveSheet.Range("D1").EntireColumn

   xlSpalte.ColumnWidth = 20

   If MsgBox("Ist das breit genug?", vbYesNo) = vbNo Then

      xlSpalte.AutoFit

   End If
```

**Hinweis**
Um eine Spalte oder eine Zeile auszublenden, können Sie die EntireRow, beziehungsweise EntireColumn auf False setzen. Da Sie beim Einblenden natürlich nicht wissen, wie breit die Spalte oder Zeile wird, sollten Sie die Eigenschaften ColumnWidth oder RowHeight auf einen Zahlenwert setzen.

### 7.1.6 Wie viele Zeilen (Spalten)?

Häufig muss in Excel geprüft werden, in wie vielen Zeilen Daten stehen. Hierbei leistet das Objekt CurrentRegion wertvolle Dienste. Es beschreibt den zusammenhängenden Bereich. Er kann vom Anwender mit <Strg>+<Shift>+<*> markiert werden. Mit seiner Hilfe können leicht die Anzahl der Zeilen (oder Spalten) ermittelt werden:

```
xlZelle.CurrentRegion.Rows.Count
```

Ist jedoch nicht sicher, ob die Daten sich in einem zusammenhängenden Bereich befinden, dann kann mit `SpecialCells` gearbeitet werden:

```
Set xlZelleRechtsUnten = xlZelle.SpecialCells(xlLastCell)
```

Der Parameter `xlLastCell` liefert die rechte untere Zelle, unabhängig davon, ob sich in dem Bereich leere Zellen, Zeilen oder Spalten befinden. Selbstverständlich kann über diese Zelle die Adresse, die Zeilen- und Spaltennummer ausgelesen werden und somit die exakte Position bestimmt werden.

**Tabelle 7.3** Die Methode SpecialCells

| Konstante | Wert | Bedeutung |
|---|---|---|
| xlCellTypeAllFormatConditions | -4172 | Zellen mit beliebigem Format |
| xlCellTypeAllValidation | -4174 | Zellen mit Gültigkeitskriterien |
| xlCellTypeBlanks | 4 | Leere Zellen |
| xlCellTypeComments | -4144 | Zellen mit Kommentaren |
| xlCellTypeConstants | 2 | Zellen mit Konstanten |
| xlCellTypeFormulas | -4123 | Zellen mit Formeln |
| xlCellTypeLastCell | 11 | Die letzte Zelle im verwendeten Bereich |
| xlCellTypeSameFormatConditions | -4173 | Zellen mit gleichem Format |
| xlCellTypeSameValidation | -4175 | Zellen mit gleichen Gültigkeitskriterien |
| xlCellTypeVisible | 12 | Alle sichtbaren Zellen |

**Tabelle 7.4** Die Konstanten bei xlCellTypeFormulas oder xlCellTypeConstants

| XlSpecialCellsValue-Konstanten | Wert | Bedeutung |
|---|---|---|
| xlErrors | 16 | Fehler |
| xlLogical | 4 | logischer Wert |
| xlNumbers | 1 | Zahl |
| xlTextValues | 2 | Text |

Und welche Zelle ist das? Angenommen, Sie ermitteln die letzte Zelle rechts unten eines Bereichs, dann liefert Ihnen:

```
Set xlZelleRechtsUnten = xlZellBereich.SpecialCells(xlCellTypeLastCell)

MsgBox xlZelleRechtsUnten.Address & " (" & _

   xlZelleRechtsUnten.AddressLocal & ")" & vbCr & _

   "liegt in Zeile: " & xlZelleRechtsUnten.Row & vbCr & _

   "und in Spalte: " & xlZelleRechtsUnten.Column
```

Die beiden Eigenschaften `Row` und `Column` geben Ihnen die Nummer der Zeile und die Nummer der Spalte zurück. Damit kann weiter gearbeitet werden.

Analog dazu können Sie das Objekt End verwenden:

```
MsgBox xlZellBereich.End(xlToRight).Address
```

Dabei werden die vier Richtungen `xlToRight`, `xlToLeft`, `xlUp` und `xlDown` unterstützt. Sie entsprechen den vier Tastenkombinationen <Strg>+<→>, <Strg>+<←>, <Strg>+<↑> und <Strg>+<↓>.

### 7.1.7 Zeilen oder Spalten ausfüllen

Wenn Sie sehr viele Zellen beschriften möchten,dann müssen Sie nicht jede einzelne Zelle mit einem Wert füllen, sondern können die Werte mit der Methode `AutoFill` hineinschreiben. Diese Methode besitzt zwei Parameter: `Destination` – der Bereich, in den die Werte geschrieben werden und `Type`.

**Tabelle 7.5 Folgende Konstanten stehen für Type bei der Methode AutoFill zur Verfügung:**

| Konstante | Wert | Bedeutung |
|---|---|---|
| xlFillCopy | 1 | Die Werte und Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |

| Konstante | Wert | Bedeutung |
|---|---|---|
| xlFillDays | 5 | Die Namen der Wochentage im Quellbereich werden auf den Zielbereich ausgeweitet. Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillDefault | 0 | Die Werte und Formate, die zum Füllen des Zielbereichs verwendet werden, werden in Excel bestimmt. |
| xlFillFormats | 3 | Es werden nur die Formate aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillMonths | 7 | Die Namen der Monate im Quellbereich werden auf den Zielbereich ausgeweitet. Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillSeries | 2 | Die Werte im Quellbereich werden als Folge auf den Zielbereich ausgeweitet (z. B. wird "1, 2" durch "3, 4, 5" ergänzt). Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillValues | 4 | Es werden nur die Werte aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillWeekdays | 6 | Die Namen der Arbeitswochentage im Quellbereich werden auf den Zielbereich ausgeweitet. Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlFillYears | 8 | Die Jahre im Quellbereich werden auf den Zielbereich ausgeweitet. Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlGrowthTrend | 10 | Die numerischen Werte aus dem Quellbereich werden auf den Zielbereich ausgeweitet. Hierbei wird davon ausgegangen, dass die Beziehungen zwischen den Zahlen im Quellbereich multiplikativ sind (z. B. wird "1, 2" durch "4, 8, 16" ergänzt, wenn davon ausgegangen wird, dass jede Zahl das Ergebnis einer Multiplikation der vorhergehenden Zahl mit einem bestimmten Wert ist). Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |
| xlLinearTrend | 9 | Die numerischen Werte aus dem Quellbereich werden auf den Zielbereich ausgeweitet. Hierbei wird davon ausgegangen, dass die Beziehungen zwischen den Zahlen additiv sind (z. B. wird "1, 2" durch "3, 4, 5" ergänzt, wenn davon ausgegangen wird, dass jede Zahl das Ergebnis der Addition eines Werts zur vorhergehenden Zahl ist). Die Formate werden aus dem Quellbereich in den Zielbereich kopiert, ggf. mit Wiederholung. |

**Beispiel**

Das folgende Beispiel schreibt in die Zelle A1 den ersten Tag des aktuellen Montags, danach werden sämtliche Tage des aktuellen Monats in diese Spalte geschrieben:

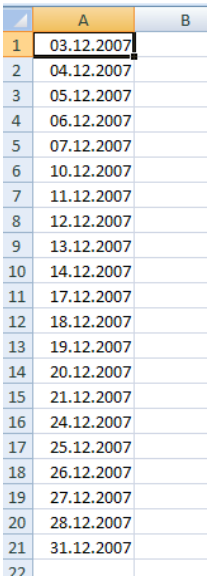
```
Sub NurMonatstage()

    Dim xlDatei As Workbook

    Dim xlBlatt As Worksheet

    Dim xlZelle As Range

    Dim intMonat As Integer

    Dim intJahr As Integer

    Dim datMonatsanfang As Date

    Dim datMonatsende As Date


    Set xlDatei = Application.Workbooks.Add
```

```
    Set xlBlatt = xlDatei.Sheets(1)
    Set xlZelle = xlBlatt.Range("A1")
    intMonat = Month(Date)
    intJahr = Year(Date)
    datMonatsanfang = DateSerial(intJahr, intMonat, 1)
    datMonatsende = DateSerial(intJahr, intMonat + 1, 1) - 1
    xlZelle.Value = datMonatsanfang
    'xlZelle.DataSeries Rowcol:=xlColumns, Type:=xlChronological, Date:= _
         xlWeekday, Step:=1, Stop:=datMonatsende, Trend:=False
    xlZelle.AutoFill xlBlatt.Range(xlZelle, _
    xlZelle.Offset(Day(datMonatsende) - 1, 0)), xlFillDays
End Sub
```

**Beispiel**

Mit ein wenig Rechnung kann man auch die Datumsangaben vom ersten Wochentag bis zum letzten Wochentag des aktuellen Montags hochzählen. Beachten Sie, dass hierfür die Methode DataSeries verwendet werden muss:

```
...
datMonatsanfang = DateSerial(intJahr, intMonat, 1)
   If Weekday(datMonatsanfang, vbMonday) > 5 Then
      datMonatsanfang = datMonatsanfang + 8 - _
         Weekday(datMonatsanfang, vbMonday)
   End If
   datMonatsende = DateSerial(intJahr, intMonat + 1, 1) - 1
   intLetzterMonatstag = Day(datMonatsende)
   intLetzterWochentag = Weekday(datMonatsende)
   If intLetzterWochentag > 5 Then
      intLetzterMonatstag = intLetzterMonatstag - _
         (intLetzterWochentag - 5)
   End If
   datMonatsende = DateSerial(intJahr, intMonat, intLetzterMonatstag)
   xlZelle.Value = datMonatsanfang
   xlZelle.DataSeries Rowcol:=xlColumns, Type:=xlChronological, Date:= _
        xlWeekday, Step:=1, Stop:=datMonatsende, Trend:=False
End Sub
```



| | A | B |
|---|---|---|
| 1 | 03.12.2007 | |
| 2 | 04.12.2007 | |
| 3 | 05.12.2007 | |
| 4 | 06.12.2007 | |
| 5 | 07.12.2007 | |
| 6 | 10.12.2007 | |
| 7 | 11.12.2007 | |
| 8 | 12.12.2007 | |
| 9 | 13.12.2007 | |
| 10 | 14.12.2007 | |
| 11 | 17.12.2007 | |
| 12 | 18.12.2007 | |
| 13 | 19.12.2007 | |
| 14 | 20.12.2007 | |
| 15 | 21.12.2007 | |
| 16 | 24.12.2007 | |
| 17 | 25.12.2007 | |
| 18 | 26.12.2007 | |
| 19 | 27.12.2007 | |
| 20 | 28.12.2007 | |
| 21 | 31.12.2007 | |
| 22 | | |

**Abbildung 7.2** Die Liste der Wochentage im Dezember 2007

### 7.1.8 Text, Value und Value2

In einer Excel-Zelle können Texte, Zahlen und Formeln stehen. Um zu ermitteln, ob in einer Zelle eine Formel steht, kann die Eigenschaft `HasFormula` verwendet werden.

Wenn Sie auf den Inhalt einer Zelle zugreifen möchten, stehen Ihnen drei Eigenschaften zur Verfügung: `Text`, `Value` und `Value2`. Steht in einer Zelle ein Text, dann .liefern alle drei Eigenschaften das gleiche Ergebnis – den Inhalt der Zelle. Steht dagegen eine Zahl in der Zelle, dann gibt `Text` die formatierte Zahl zurück, `Value` dagegen den eigentlichen Wert, der sich in der Zelle befindet. Also beispielsweise 200,00 € oder 200. Steht in einer Zelle eine Datumsangabe, dann liefert `Text` den (formatierten) Datumswert (beispielsweise „Dienstag, 16. Oktober 2007“), `Value` den (unformatierten) Datumswert (beispielsweise „16.10.2007“) und `Value2` die interne Zahl (beispielsweise 39371).

**Achtung**

Sie sollten von der Eigenschaft Text Abstand nehmen – damit kann nicht exakt bestimmt werden, was in der Zelle steht. Außerdem ist diese Eigenschaft schreibgeschützt – man kann in eine Zelle nicht über Text etwas hineinschreiben.

### 7.1.9 Zellen und Zellinhalte löschen

Wenn Sie eine Zeile löschen möchten, dann müssen Sie von dieser Zeile eine Zelle angeben und mit der Eigenschaft `EntireRow` die gesamte Zeile löschen:

```
Set xlZellBereich = xlBlatt.Range("A1")

xlZellBereich.EntireRow.Delete
```

Sollen mehrere Zeilen gelöscht werden, brauchen Sie nicht eine Schleife zu schreiben, sondern können das Objekt `EntireRow` auf den Zellbereich anwenden:

```
Set xlZellBereich = xlBlatt.Range("A1:A3")

xlZellBereich.EntireRow.Delete
```

Analog verhält es sich mit dem Objekt `EntireColumn`.

Wollen Sie dagegen die Inhalte löschen, dann muss unterschieden werden, ob Sie nur den Wert der Zelle oder auch die Formatierung löschen möchten:

```
xlZelle.ClearContents
```

löscht den Inhalt der Zelle xlZelle. Sollen die Formatierungen gelöscht werden, so ist

```
xlZelle.ClearFormats
```

zu verwenden. Weitere Löscheigenschaften finden sich mit:

```
xlZelle.ClearComments

xlZelle.ClearNotes
```

Beide löschen unterschiedslos die Kommentare der aktiven Zelle. Und alles wird gelöscht mit:

```
xlZelle.Clear
```

### 7.1.10 Zellfomate

#### Schrift

Schriftformatierungen können mit

```
xlZelle.Font
```

festgelegt werden. Beispielsweise:

```
xlZelle.Font.Bold = True

xlZelle.Font.Name = "Times"

xlZelle.Font.Size = 24
```

Eleganter natürlich:

```
With xlZelle.Font

    .Bold = True

    .Name = "Times"

    .Size = 24

End With
```

Oder Sie lagern das Font-Objekt in einer Variablen aus:

```
    Dim xlFont As Font

    Set xlFont = xlZelle.Font

    With xlFont

        .Bold = True

        .Italic = True

        .Underline = xlUnderlineStyleNone

        .Name = "Arial"

    End With
```

**Tabelle 7.6** Für das Font-Objekt stehen folgende wichtige Eigenschaften zur Verfügung

| Eigenschaft | Beschreibung |
|---|---|
| Bold | Fett |
| Italic | Kursiv |

| Eigenschaft | Beschreibung |
|---|---|
| Underline | unterstrichen. Dafür stehen die Systemkonstanten<br>xlUnderlineStyleNone<br>xlUnderlineStyleSingle<br>xlUnderlineStyleDouble<br>xlUnderlineStyleSingleAccounting<br>xlUnderlineStyleDoubleAccounting<br>zur Verfügung |
| Name | Schriftart |
| Size | Schriftgröße – die Angabe muss als Zahl erfolgen |
| Color, ColorIndex | Schriftfarbe |
| Strikethrough | Durchgestrichen |
| Subscript, Superscript | Hochgestellt, tiefgestellt |

### Ausrichtung

**Tabelle 7.7** Weitere wichtige Zellformatierungen

| Eigenschaft | Beschreibung |
|---|---|
| HorizontalAlignment | horizontale Ausrichtung<br>mit den Werten: xlCenter, xlDistributed, xlJustify, xlLeft, xlRight |
| VerticalAlignment | vertikale Ausrichtung<br>mit den Werten: xlBottom, xlCenter, xlDistributed, xlJustify, xlTop |
| WrapText | Zeilenumbruch |
| Orientation | Orientierung (in Grad) |
| AddIndent | Einzug |
| MergeCells | Zellen verbinden |

**Hinweis**
Während xlBlatt.Range("A1:C7").MergeCells = True einen Bereich erstellt, verbindet die Methode Range("A1:C7").MergeCells = True xlBlatt.Range("E1:G7").Merge True zu sieben Bereichen, die jeweils drei Zellen beinhalten.

### Ausfüllen

Wenn Sie das Innere einer Zelle formatieren möchten, dann gelangen Sie mit dem Interior-Objekt an die entsprechenden Eigenschaften:

```
Dim xlInnen As Interior

   Set xlInnen = xlZelle.Interior

    With xlInnen

        .Pattern = xlSolid

        .PatternColorIndex = xlAutomatic

        .Color = 14066153
```

```
        .TintAndShade = 0

        .PatternTintAndShade = 0

    End With
```

**Achtung**

Beachten Sie, dass die beiden Eigenschaften TintAndShade (aufdunkeln oder abhellen) und PatternTintAndShade (Farbton und Schattierungsmuster) erst seit Excel 2007 zur Verfügung stehen.

### Rahmen

Ein wenig mühsam gestaltet sich das Formatieren von Linien, da von einer Zelle jede der Linien einzeln angegeben werden muss. Jede Linie wiederum besitzt die Eigenschaften `LineStyle`, `ColorIndex`, `Weight` und seit Excel 2007 `TintAndShade`. Um in einem Bereich links, rechts, oben, unten und zwischen sämtlichen Zellen – nicht jedoch bei den diagonalen – eine dünne Linie einzuschalten, müssen folgende Zeilen abgearbeitet werden:

```
With xlBereich

    .Borders(xlDiagonalDown).LineStyle = xlNone

    .Borders(xlDiagonalUp).LineStyle = xlNone

    With .Borders(xlEdgeLeft)

        .LineStyle = xlContinuous

        .ColorIndex = 0

        .TintAndShade = 0

        .Weight = xlThin

    End With

    With .Borders(xlEdgeTop)

        .LineStyle = xlContinuous

        .ColorIndex = 0

        .TintAndShade = 0

        .Weight = xlThin

    End With

    With .Borders(xlEdgeBottom)

        .LineStyle = xlContinuous

        .ColorIndex = 0

        .TintAndShade = 0

        .Weight = xlThin

    End With

    With .Borders(xlEdgeRight)

        .LineStyle = xlContinuous

        .ColorIndex = 0

        .TintAndShade = 0
```

```
            .Weight = xlThin
        End With
        With .Borders(xlInsideVertical)
            .LineStyle = xlContinuous
            .ColorIndex = 0
            .TintAndShade = 0
            .Weight = xlThin
        End With
        With .Borders(xlInsideHorizontal)
            .LineStyle = xlContinuous
            .ColorIndex = 0
            .TintAndShade = 0
            .Weight = xlThin
        End With
    End With
```

Zugegeben: Es geht etwas kürzer, wenn auf die Standardeigenschaften verzichtet wird:

```
    With xlBereich
        .Borders(xlEdgeLeft).LineStyle = xlContinuous
        .Borders(xlEdgeTop).LineStyle = xlContinuous
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
        .Borders(xlEdgeRight).LineStyle = xlContinuous
        .Borders(xlInsideVertical).LineStyle = xlContinuous
        .Borders(xlInsideHorizontal).LineStyle = xlContinuous
    End With
```

**Tipp**

Wenn Sie per Programmierung an mehreren Stellen im Code formatieren müssen, dann sollten Sie diese Funktionalität in eine Prozedur oder in eine Klasse auslagern.

### Zahlenformate

Wenn Sie mit dem Makrorekorder Zahlenformate aufzeichnen, dann ergeben sich Codezeilen wie beispielsweise:

```
    Selection.NumberFormat = "General"
    Selection.Style = "Currency"
    Selection.Style = "Percent"
    Selection.Style = "Comma"
```

Noch relativ leicht zu verstehen sind Zahlenformate in der Form:

```
Selection.NumberFormat = "0.00"

Selection.NumberFormat = "#,##0.00 $"

Selection.NumberFormat = "m/d/yyyy"

Selection.NumberFormat = "0.00%"

Selection.NumberFormat = "# ?/?"

Selection.NumberFormat = "0.00E+00"
```

Dagegen nehmen schon fast kryptischen Charakter folgende Formate an:

```
Selection.NumberFormat = _

    "_($* #,##0.00_);_($* (#,##0.00);_($* ""-""??_);_(@_)"

Selection.NumberFormat = "[$-F800]dddd, mmmm dd, yyyy"

Selection.NumberFormat = "[$-F400]h:mm:ss AM/PM"
```

Deshalb sollten Sie auf die englische Schreibweise verzichten und nicht mit der Eigenschaft `NumberFormat` arbeiten, sondern die für uns Mitteleuropäer leichter zu lesende Formatanweisung `NumberFormatLocal` benutzen. Sie kann im Dialog Zellen formatieren | Zahlen eingesehen werden:

```
xlZelle.NumberFormatLocal = "#.##0,00 ""Taler"""

xlZelle.NumberFormatLocal = """München, den ""TT.MM.JJJJ"

xlZelle.NumberFormatLocal = "#..,# ""Mio"""
```

### 7.1.11 Kommentare

Ein Kommentar wird an eine Zelle gebunden. Er wird mit der Methode `AddComment` erzeugt und kann an eine Objektvariable übergeben werden:

```
Dim xlKommentar As Comment

Set xlKommentar = xlZelle.AddComment("Bitte nicht!")
```

Er beinhaltet einige erstaunliche Methoden:

**Tabelle 7.8 Die Methoden des Comment-Obejektes**

| Methode | Bedeutung |
|---|---|
| Delete | löscht den Kommentar |
| Next, Previous | greift auf den nächsten oder letzten Kommentar zu |
| Text | übergibt dem Kommentar einen Text. Diese Methode hat drei Parameter:<br>Text: der Text, er übergeben wird<br>Start: die Position ab der der neue Text geschrieben wird<br>Overwrite: mit True wird bereits vorhandener Text überschrieben |

```
Set xlKommentar = xlZelle.AddComment("Bitte nicht!")

xlKommentar.Text Text:=" die Kaffeetasse neben die Tastatur stellen", _

   Start:=12, Overwrite:=False
```

**Achtung**

Die Methode AddComment liefert einen Fehler, wenn die Zelle bereits einen Kommentar enthält. Deshalb muss vorher ein möglicher Kommentar gelöscht werden:

```
xlZelle.ClearComments
```

Man kann mit einer Schleife sämtliche Kommentare durchlaufen. Beachten Sie, dass die Sammlung der Kommentare zu Worksheet gehört; Workbook dagegen besitzt kein Comments-Objekt:

```
Sub WoSindDieKleinenKommentare()

   Dim xlKommentar As Comment

   Dim strKommListe As String


   For Each xlKommentar In ActiveSheet.Comments

      strKommListe = strKommListe & vbCr & _

         xlKommentar.Parent.Address & vbTab & _

         xlKommentar.Text

   Next


   MsgBox strKommListe

End Sub
```

Kommentare können formatiert werden. Das Objekt Shape ist dafür zuständig. Das folgende Beispiel zeigt sämtliche Kommentare auf dem Arbeitsblatt an und vergrößert die Schrift um 2 Punkt:

```
Sub GroßeKommentare()

   Dim xlKommentar As Comment


   Application.DisplayCommentIndicator = xlCommentAndIndicator

   ' -- zeigt alle Kommentare an

   For Each xlKommentar In ActiveSheet.Comments

      xlKommentar.Shape.TextFrame.Characters.Font.Size = _

         xlKommentar.Shape.TextFrame.Characters.Font.Size + 2

      xlKommentar.Shape.TextFrame.AutoSize = True

   Next

End Sub
```

**Hinweis**

Sie könnten auch die Methode NoteText verwenden. Allerdings ist sie nicht so komplex wie das Comment-Objekt.

## 7.2 Namen

Wie Sie sicherlich wissen, können in Excel Namen für Zellen vergeben werden. Dabei kann einer Zelle oder einem Zellbereich ein Name hinzugefügt werden.

**Achtung**

Sowohl das Objekt Application, als auch Workbook und Worksheet besitzen die Sammlung Names. Allerdings kann Excel als Applikation kein Name vergeben werden – Namen sind immer global in der Mappe oder lokal für das Blatt definiert.

Die beiden Zeilen

```
Application.Names.Add "MwSt", "=Tabelle1!$J$1"

ActiveWorkbook.Names.Add "Mwst", "=Tabelle2!$J$1"
```

sind identisch. Sie sollten aufgrund besserer Lesbarkeit auf die erste Variante verzichten!

**Achtung**

Sie müssen das Dollarzeichen als Symbol für absoluten Bezug auf die Zelle verwenden!

Sie können allerdings auch einen Namen lokal für ein Blatt definieren:

```
ActiveSheet.Names.Add "Mwst", "=$J$1"
```

Da Excel innerhalb einer Datei globale und lokal definierte Namen verwenden kann, sollten Sie keinen Namen doppelt verwenden, da Sie möglicherweise in Konflikt kommen, weil Sie nicht mehr wissen, auf welchen Namen sich nun Excel bezieht.

Mit einer Schleife können Sie sämtliche Namen auslesen:

```
Dim xlName As Name

Dim strNamen As String


For Each xlName In ActiveWorkbook.Names

   strNamen = strNamen & vbCr & xlName.RefersTo & vbTab & xlName.Name

Next

MsgBox strNamen
```

**Tipp**

Sie können Daten in Namen speichern. Dabei verwenden Sie den Parameter RefersTo:

```
Dim strGeheim As String

strGeheim = "René Martin"


ActiveWorkbook.Names.Add Name:="Autor", RefersTo:=strGeheim
```

Zwar wird der Name nicht in der Dropdownliste neben den Namen angezeigt – jedoch im Namensmanager. Und er erscheint, wenn Sie mit einem Gleichheitszeichen den Anfang des Namens in eine Formel eingeben.

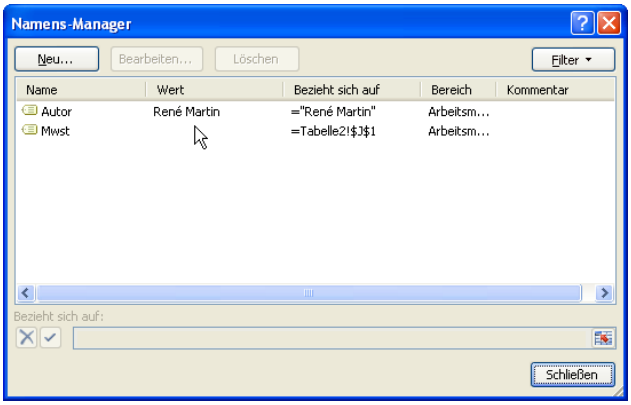



**Abbildung 7.3** Der Namensmanager mit dem „geheimen“ Namen

**Tipp**

Der Name wird nicht mehr im Namensmanager oder in der Liste der Funktionen angezeigt, wenn Sie den dritten Parameter Visible auf False setzen.

## 7.3 Beispiele

### 7.3.1 Daten suchen

**Beispiel**

In einer Tabelle sind drei Spalten ausgefüllt. In der ersten, die mit „Nummer“ überschrieben ist, stehen fortlaufende Nummern, beginnend mit 100. In der zweiten Spalte stehen die Bezeichnungen, beispielsweise die Filmnamen, in der dritten die (fiktiven) Preise (für die Videos oder DVDs).

Der Benutzer wird nach einer Nummer gefragt. Er trägt sie in eine InputBox ein und erhält den Namen des Films.

Zur Vorgehensweise: Der Benutzer wird nach einer Nummer gefragt. Im Tabellenblatt „Almodovar“ wird auf die Zelle A1 zugegriffen. Eine For ... Next-Schleife durchläuft die Tabelle von A1 bis zu der letzten gefüllten Zelle, die über die Eigenschaft `ActiveCell.CurrentRegion.Rows.Count` ermittelt wird. Jede der Zellen wird mit dem Inhalt der InputBox-Variablen verglichen (intNr). Sind sie gleich, bewegt sich der Zeiger eine Spalte nach rechts und zeigt den Inhalt dieser Zelle an. Wird die Schleife ohne Erfolg durchlaufen, dann wird die Meldung unterhalb der Schleife angezeigt.

```
Sub FilmAnzeigen2()

   Dim xlDatei As Workbook

   Dim xlTabelle As Worksheet

   Dim xlZelle As Range

   Dim intNr As Integer

   Dim intZähler As Integer


   On Error GoTo ende


   Set xlDatei = ActiveWorkbook
```

```
    Set xlTabelle = xlDatei.Sheets("Almodovar")
    Set xlZelle = xlTabelle.Range("A1")

    intNr = InputBox("Bitte eine Nummer eingeben")

    With xlZelle
       For intZähler = 1 To .CurrentRegion.Rows.Count

          If .Offset(intZähler, 0).Value = intNr Then
             MsgBox "Der Film mit der Nummer " & _
             intNr & " lautet: " & vbCr & Chr(187) & _
             .Offset(intZähler, 1).Value & _
             Chr(171) & vbCr & " und kostet " & _
             FormatCurrency(.Offset(intZähler, 2).Value)
             Exit Sub
          End If

       Next intZähler
    End With
    MsgBox "Schade, aber die Nummer " & intNr & _
       " wurde nicht gefunden!"
    Exit Sub
ende:
    MsgBox "Es trat ein Fehler auf: " & _
       Err.Description, vbCritical, "Fehler!"
End Sub
```

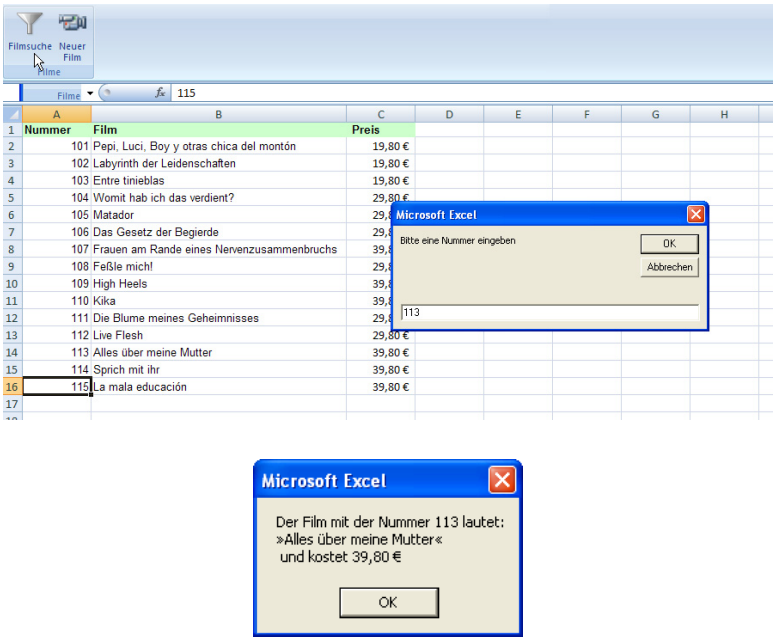



**Abbildung 7.4** Die Nummernsuche

## 7.3.2 Daten eintragen

**Beispiel**

Ein zweites Makro in der Filmliste soll dafür sorgen, dass der Benutzer einen neuen Artikel eintragen kann. Ihm wird automatisch die nächsthöhere Nummer vergeben, und der Artikel und sein Preis unten an die Liste angefügt.

Die Liste könnte mit einer Do Loop ... Until-Schleife durchlaufen werden (wie in Lösung 1) oder indem die Anzahl der vorhandenen Zellen bestimmt wird. Letzteres ist eleganter:

```
Sub NeuerFilm()

   Dim xlDatei As Workbook

   Dim xlTabelle As Worksheet

   Dim xlZelle As Range

   Dim xlsBereich As Range

   Dim intZeilen As Integer

   Dim strNeuTitel As String

   Dim curNeuPreis As Currency


   Set xlDatei = ActiveWorkbook

   Set xlTabelle = xlDatei.Sheets("Almodovar")

   Set xlZelle = xlTabelle.Range("A1")

   Set xlsBereich = xlZelle.CurrentRegion


   intZeilen = xlsBereich.Rows.Count
```

```
    strNeuTitel = InputBox("Wie lautet der Name " & "des neuen Films?")

    curNeuPreis = InputBox("Und was kostet " & strNeuTitel & "?")


    With xlZelle

       .Offset(intZeilen, 0).Value = 100 + intZeilen

       .Offset(intZeilen, 1).Value = strNeuTitel

       .Offset(intZeilen, 2).Value = curNeuPreis

    End With

End Sub
```

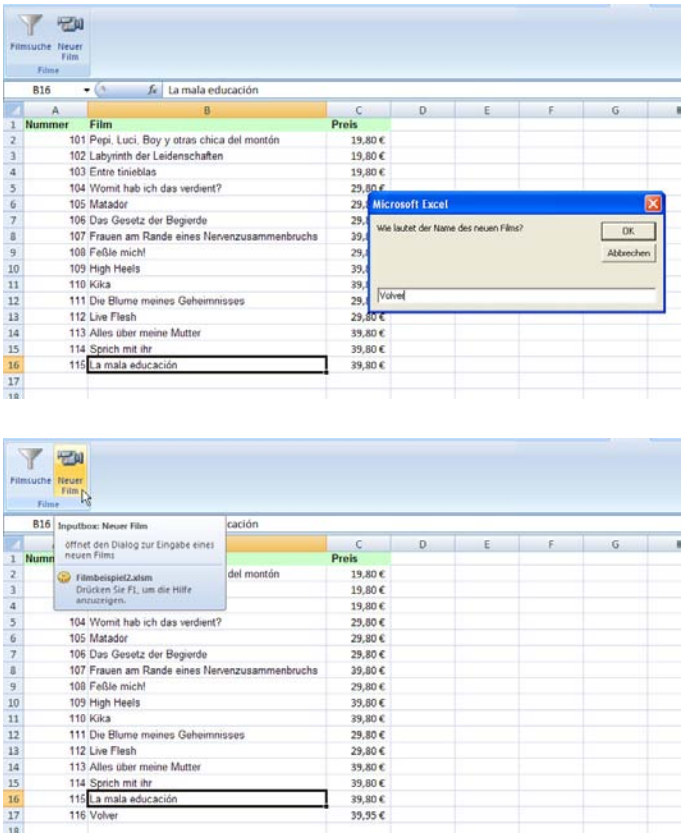



**Abbildung 7.5** Neue Filme können eingefügt werden.

### 7.3.3 Listen synchronisieren

Stellen Sie sich zwei Außendienstmitarbeiter vor, die auf ihrem Laptop jeweils eine Excel-Tabelle mit Namen haben. Nun ändern beide bestimmte Datensätze. Am Abend eines Tages oder am Ende eines Quartals sollen beide Listen miteinander verglichen werden. Nun gibt es verschiedene Möglichkeiten der Synchronisation: In jeder der beiden Tabellen werden die Datensätze rot formatiert, die nur in einer der beiden Tabellen stehen, damit sofort die Änderungen erkannt werden.

**Beispiel**

Die beiden Dateien heißen NAMENSLISTE1.XLS und NAMENSLISTE2.XLS. Es wird überprüft, ob beide offen sind. Wenn ja, dann werden sie nach der ersten Spalte sortiert, in der sich ein Zähler befindet. Nun benötigt man zwei Schleifen. In der äußeren Schleife durchläuft ein Zähler alle Werte. Jeder einzelne dieser Werte wird mit jedem Wert der zweiten Tabelle verglichen. Steht der Wert der ersten Tabelle in der zweiten, wird die Zelle rot formatiert, und der Zähler

wird um eins vergrößert. Wird er dagegen nicht gefunden, so wird weitergesucht. Nachdem die erste Tabelle durchlaufen wurde, wird auch die zweite Tabelle durchlaufen.

```
Sub ListenVergleichen()

   Dim xlDatei1 As Workbook

   Dim xlDatei2 As Workbook

   Dim xlFenster As Workbook

   Dim lngDatensätze1 As Long

   Dim lngDatensätze2 As Long

   Dim lngZähler1 As Long

   Dim lngZähler2 As Long


   For Each xlFenster In Workbooks

      If xlFenster.Name = "Namensliste1.xls" Then

         Set xlDatei1 = xlFenster

      ElseIf xlFenster.Name = "Namensliste2.xls" Then

         Set xlDatei2 = xlFenster

      End If

   Next


   If TypeName(xlDatei1) = "Nothing" Or _

      TypeName(xlDatei2) = "Nothing" Then

      MsgBox "Eine der beiden Dateien ist nicht geöffnet. " & _

      vbCr & "Bitte erst öffnen!", vbCritical, "Achtung!"

      Exit Sub

   End If



   xlDatei1.Worksheets(1).Range("A1").CurrentRegion.Sort _

      Key1:=xlDatei1.Worksheets(1).Range("A1"), _

      Order1:=xlAscending, _

      Header:=xlYes, OrderCustom:=1, MatchCase:=False, _

      Orientation:=xlTopToBottom

   xlDatei2.Worksheets(1).Range("A1").CurrentRegion.Sort _

      Key1:=xlDatei2.Worksheets(1).Range("A1"), _

      Order1:=xlAscending, _
```

```
      Header:=xlYes, OrderCustom:=1, MatchCase:=False, _
      Orientation:=xlTopToBottom
   lngDatensätze1 = xlDatei1.Sheets(1).Range("A1"). _
      CurrentRegion.Rows.Count
   lngDatensätze2 = xlDatei2.Sheets(1).Range("A1"). _
      CurrentRegion.Rows.Count

   For lngZähler1 = 1 To lngDatensätze1
      For lngZähler2 = 1 To lngDatensätze2
         If xlDatei1.Sheets(1).Range("A1"). _
            Offset(lngZähler1, 0).Value = xlDatei2.Sheets(1). _
               Range("A1").Offset(lngZähler2, 0).Value Then
                  xlDatei1.Sheets(1).Range("A1"). _
                     Offset(lngZähler1, 0).Font.ColorIndex = 3
               Exit For
         End If
      Next lngZähler2
   Next lngZähler1

   For lngZähler2 = 1 To lngDatensätze2
      For lngZähler1 = 1 To lngDatensätze1
         If xlDatei2.Sheets(1).Range("A1"). _
            Offset(lngZähler2, 0).Value = xlDatei1.Sheets(1). _
               Range("A1").Offset(lngZähler1, 0).Value Then
                  xlDatei2.Sheets(1).Range("A1"). _
                     Offset(lngZähler2, 0).Font.ColorIndex = 3
                  Exit For
          End If
      Next lngZähler2
   Next lngZähler1
End Sub
```

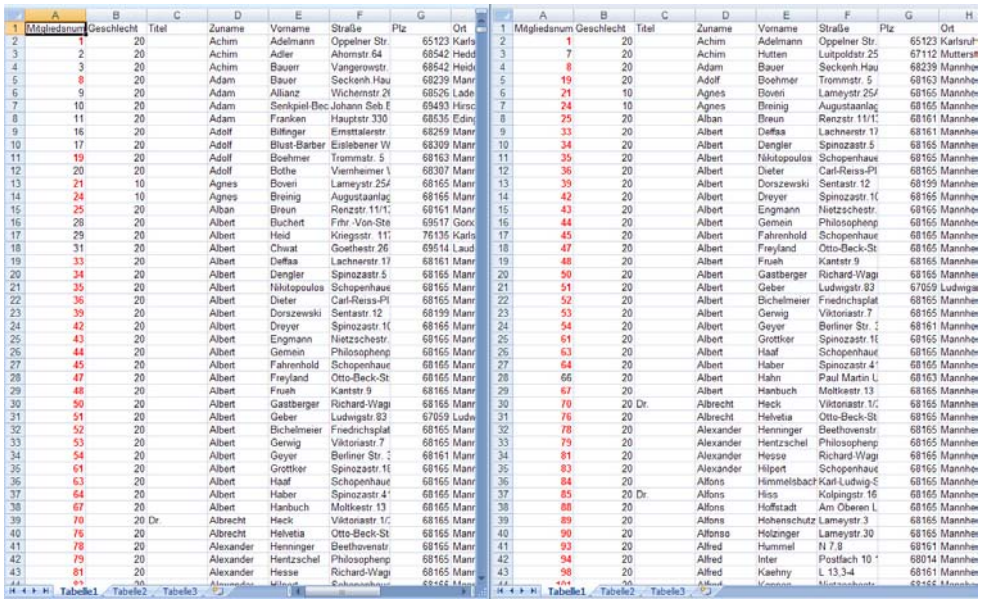


**Abbildung 7.6** Zwei Listen werden miteinander verglichen.

**Tabelle 7.9 Einige wichtige Methoden für eine Zelle oder einen Zellbereich**

| Methode | Erläuterung |
|---|---|
| Activate, Select | wählt Zelle aus. |
| Clear, ClearComments, ClearFormats, ClearNotes, ClearOutline, Delete | verschiedene Löschmethoden |
| Copy, Cut, PasteSpecial, Insert | kopieren, ausschneiden und einfügen |
| FillDown, FillLeft, FillUp, FillRight, AutoFill | Reihe ausfüllen (Bearbeiten \| Ausfüllen \| Reihe) |
| Calculate | berechnet die Zelle neu. |
| Merge | Zellen verbinden |
| Group | Zellen gruppieren |
| Sort | Bereiche sortieren |
| Subtotal | Teilergebnisse |
| AddComment | Kommentare einfügen |
| AutoFilter | Autofilter |
| AdvancedFilter | Spezialfilter |
| ApplyNames, CreateNames | Namen einfügen |
| FunctionWizard | Funktionsassistent |
| GoalSeek | Zielwertsuche |
| AutoFit | optimale Breite oder Höhe |

**Tabelle 7.10 Tabelle 7.11 Eigenschaften zur Zelladressierung (xlZelle verweist auf A2, xlBereich auf A1:C7)**

| Eigenschaft | liefert |
|---|---|
| xlZelle.Address | $A$2 |
| xlZelle.AddressLocal | $A$2 |
| xlZelle.Name | Name, falls festgelegt |
| xlZelle.Offset(1, 0).AddressLocal | $A$3 |
| xlZelle.Range("A2").AddressLocal | $A$3 |

| Eigenschaft | liefert |
|---|---|
| xlZelle.Cells(2, 1).AddressLocal | $A$3 |
| xlZelle.Column | 1 |
| xlZelle.Columns.AddressLocal | $A$2 |
| xlZelle.Row | 2 |
| xlZelle.Rows.AdressLocal | $A$2 |
| xlZelle.Count | 1 |
| xlZelle.CurrentRegion.Rows.Count | 1 |
| xlZelle.EntireColumn.AddressLocal | $A:$A |
| xlZelle.EntireRow.AddressLocal | $2:$2 |
| | |
| xlBereich.Address | $A$1:$C$7 |
| xlBereich.AddressLocal | $A$1:$C$7 |
| xlBereich.Name | Name, falls festgelegt |
| xlBereich.Offset(1, 0).AddressLocal | $A$2:$C$8 |
| xlBereich.Range("A2").AddressLocal | $A$2 |
| xlBereich.Cells(2, 1).AddressLocal | $A$2 |
| xlBereich.Column | 1 |
| xlBereich.Columns.AddressLocal | $A$1:$C$7 |
| xlBereich.Row | 1 |
| xlBereich.Columns.Count | 3 |
| xlBereich.Rows.AdressLocal | $A$1:$C$7 |
| xlBereich.Rows.Count | 7 |
| xlBereich.Count | 21 |
| xlBereich.CurrentRegion.Rows.Count | 1 |
| xlBereich.EntireColumn.AddressLocal | $A:$C |
| xlBereich.EntireRow.AddressLocal | $1:$7 |
| | |
| End | die letzte gefüllte Zelle (xlDown, xlUp, xlToLeft, xlToRight) |
| xlCellTypeLastCell | bestimmte Zellen (xlLastCell, xlCellTypeLastCell, xlCell-TypeFormulas ...) |

**Tabelle 7.12 Einige Zellformatierungen**

| Eigenschaft | Beschreibung |
|---|---|
| NumberFormat, NumberFormatLocal | Zahlenformat |
| Font | Schriftart, Größe, fett, kursiv ... |
| FormatConditions | bedingte Formatierung |
| Interior | Muster |
| Borders | Linien |
| Orientation | Ausrichtung |
| Height und Width | Breite und Höhe |
| Hidden | verborgene Zelle |
| WrapText | Zeilenumbruch |
| HorizontalAlignment und VerticalAlignment | Ausrichtung |
| Text, Value, Value2 | Inhalt der Zelle |

**Tabelle 7.13 Einige weitere Eigenschaften**

| Eigenschaft | Beschreibung |
|---|---|
| AllowEdit | schreibgeschützt |
| Comment | Kommentar |
| DataSeries | Datenreihe |
| Hyperlinks | Hyperlinks |
| Locked | gesperrte Zelle |

## 7.4 Rechnen in Excel

Der Makrorekorder leistet nicht nur beim Zellformatieren und Sortieren unschätzbare Dienste, sondern auch bei der Eingabe von Formeln und Funktionen. Angenommen, in der Spalte B stehen Einnahmen, in der Spalte C Ausgaben. In D soll nun die Differenz aus beiden, also der Gewinn stehen. Dazu kann aufgezeichnet werden:

```
ActiveCell.FormulaR1C1 = "=RC[-2]-RC[-1]"
```

Dies ist die amerikanische Schreibweise. Steht der Cursor in D2, so wird mit `RC[-2]` die Zelle zwei Spalten links davon bezeichnet, also B2. In der deutschen Excel-Schreibweise steht die Formel:

```
=B2-C2
```

Dies kann programmiertechnisch umgesetzt werden in:

```
xlZelle.Formula = "=B2-C2"
```

Angenommen, in der Zelle G1 stünde ein Wert, der absolut addiert werden soll, dann liefert die amerikanische Schreibweise:

```
ActiveCell.FormulaR1C1 = "=RC[-2]-RC[-1]+R1C7"
```

Dabei stehen relative Zellbezüge in eckigen Klammern und beziehen sich auf „ActiveCell“. Absolute Bezüge stehen ohne Klammern und beginnen ihre Zählung bei 1. `R1C7` entspricht folglich der Formel `$G$1`, oder in der deutschen Schreibweise:

```
ActiveCell.Formula = "=B2-C2+$G$7"
```

Soll dies nun auf mehrere Zeilen aufgefüllt werden, so könnte man es mit folgenden Befehlen erzeugen:

```
Sub Automatisch_Rechnen()

   Dim intZeilenzahl As Integer

   Dim intZähler As Integer

   Dim xlZelle As Range

   intZeilenzahl = _

      ActiveWorkbook.Sheets(1).Range("B2").CurrentRegion.Rows.Count

   Set xlZelle = ActiveWorkbook.Sheets(1).Range("D2")


   For intZähler = 1 To intZeilenzahl - 1

      xlZelle.FormulaR1C1 = "=RC[-2]-RC[-1]+R1C7"

      Set xlZelle = xlZelle.Offset(1, 0)

   Next
```

```
End Sub
```

Und nun soll unter der Tabelle die Summe gezogen werden. Hierfür kann wieder die US-amerikanische oder die deutsche Schreibweise verwendet werden:

```
ActiveCell.FormulaR1C1 = "=SUM(R[-11]C:R[-1]C)"

ActiveCell.FormulaLocal = "=SUMME(B2:B12)"
```

Dabei ist zu beachten, dass die US-amerikanische Schreibweise auch in der deutschen Excel-Version die englischsprachigen Funktionsnamen verwendet!

Am Ende des obigen Makros könnte also stehen:

```
Sub Automatisch_Rechnen()

   Dim intZeilenzahl As Integer

[...]

   Set xlZelle = _

      ActiveWorkbook.Sheets(1).Cells(iZeilenzahl + 1, 2).Activate

      xlZelle.FormulaR1C1 = "=SUM(R[-" & iZeilenzahl - 1 & "]C:R[-1]C)"

      Set xlZelle = xlZelle.Offset(0, 1).Activate

      xlZelle.FormulaR1C1 = "=SUM(R[-" & iZeilenzahl - 1 & "]C:R[-1]C)"

      Set xlZelle = xlZelle.Offset(0, 1)

      xlZelle.FormulaR1C1 = "=SUM(R[-" & iZeilenzahl - 1 & "]C:R[-1]C)"

End Sub
```

**Beispiel**

Eine Mappe besteht aus mehreren Blättern. Auf jedem Blatt befindet sich in Zelle R37 ein Gesamtergebnis. Über eine Verknüpfung soll in jedem Blatt (beginnend ab dem zweiten) in der Zelle A1 Bezug auf die Zelle R37 des vorhergehenden Blatts genommen werden.

```
Sub BezugAufVorherigesBlatt()

   Dim intZähler As Integer

   For intZähler = 2 To ActiveWorkbook.Sheets.Count

      ActiveWorkbook.Sheets(intZähler).Range("A1").Formula = _

         "=" & ActiveWorkbook.Sheets(intZähler - 1).Name & "!R37"

   Next

End Sub
```

Statt der Eigenschaft „Formula“ kann auch „FormulaLocal“ verwendet werden.

In Zelle C1 steht eine Formel. Dieselbe Formel wird in F7 noch einmal benötigt. Würde man sie kopieren, würden die Bezüge nicht mehr stimmen. Deshalb soll die Formel so nach F7 kopiert werden, dass dieselben relativen Bezügen in dieser neuen Zelle stehen.

```
Sub FormelKopieren1()

   Dim strFormel As String

   strFormel = ActiveSheet.Range("C1").Formula

   ActiveSheet.Range("F7").Value = strFormel
```

```
End Sub
```

Oder kürzer:

```
Sub FormelKopieren2()

   ActiveSheet.Range("F7").Value = ActiveSheet.Range("C1").Formula

End Sub
```

In einer Arbeitsmappe werden in allen Tabellenblättern in der Zelle C27 die Summe der darüber stehenden Werte der Spalte C benötigt.

```
Sub SummeErzeugen()

   Dim xlBlatt As Worksheet

   For Each xlBlatt In Worksheets

      xlBlatt.Range("C27").FormulaLocal = "=Summe(C1:C26)"

   Next

End Sub
```

**Tabelle 7.14 Liste der Funktionen einer Zelle oder eines Zellbereiches**

| Eigenschaft | Erklärung |
|---|---|
| Formula | eine Formel entsprechend der Ländereinstellung |
| FormulaR1C1 | eine Formel mit Z1S1-Bezügen |
| FormulaLocal | eine Formel mit A1-Bezügen |
| FormulaR1C1Local | ausgeblendete Formel |
| HasFormula | überprüft, ob eine Formel in der Zelle steht |
| Errors | Fehler in der Formel |

**Hinweis**

Häufig finde ich den folgenden Fehler, den Anfänger der Programmierung nicht auflösen können:

```
intZeilen = xlBlatt.Range("A1").CurrentRegion.Rows.Count

xlBlatt.Cells(intZeilen + 1, 1).Formula = _

   "=AVERAGE(R[-intZeilen]C:R[-1]C)"
```

Die Quelle des Fehlers liegt darin, dass die Variable `intZeile` direkt in den Formelstring eingefügt wurde. Dadurch versucht VBA die Funktion

```
=AVERAGE(R[-intZeilen]C:R[-1]C)
```

als Formel zu übergeben. Damit die Formel dagegen beispielsweise

```
=AVERAGE(R[-27]C:R[-1]C)
```

in die Zelle schreibt, muss die Zeichenkette der Formel verkettet werden:

```
intZeilen = xlBlatt.Range("A1").CurrentRegion.Rows.Count

xlBlatt.Cells(intZeilen + 1, 1).Formula = _

   "=AVERAGE(R[-" & intZeilen & "]C:R[-1]C)"
```

**Hinweis**

Eine weitere Schwierigkeit, die ich oft beobachte, ist das Umwandeln einer bestehenden Formel in die „VBA-Schreibweise“. Zugegeben: bei komplexen Funktionen ist dies nicht immer trivial:

```
xlZelleAuswertung.Offset(0, 13).FormulaR1C1 = _

   "=IF(RC[-6]="""","""",IF(AND(RC[-6]<-31%,RC[-2]<=0),0," & _

   "IF(AND(RC[-6]<-31%,RC[-2]>0),"""",IF(AND(RC[-6]<0,RC[-8]<0," & _

   "RC[-8]>-31%),IF((-31%-RC[-6])*RC[-8]<RC[-2],"" & _

   """""",(-31%-RC[-6])*RC[-8]),IF(RC[-8]*31%+RC[-7]+RC[-2]<0," & _

   "-(RC[-8]*31%+RC[-7]),""""))))"
```

**Tipp**

Wenn Sie keine komplexe Funktion nachbilden müssen, sondern lediglich eine Summe, einen Durchschnitt, die Anzahl und so weiter berechnen möchten, dann können Sie auch mit dem Objekt End arbeiten:

```
xlZelle.Formula = _

   "=sum(" & _

   xlZelle.End(xlUp).Address & _

   ":" & _

   xlZelle.Offset(-1, 0).Address & _

   ")"
```

Es funktioniert auch, wenn Sie den Zellbereich mit dem Range-Objekt zu einem Bereich zusammenbauen und davon die Zelladresse auslesen:

```
xlZelle.Formula = _

   "=sum(" & _

   xlBlatt.Range(xlZelle.End(xlUp), xlZelle.Offset(-1, 0)).Address & _

   ")"
```

Auf der CD befindet sich eine Datei „liste der Funktionen.xlsx“ die sämtliche Funktionsnamen auf Englisch und ihr Pendant auf Deutsch auflistet.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 41 | CHIINV | CHIINV | | CODE | CODE |
| 42 | CHITEST | CHITEST | | COLUMN | SPALTE |
| 43 | CODE | CODE | | COMBIN | KOMBINATIONEN |
| 44 | COS | COS | | CONCATENA | VERKETTEN |
| 45 | COSHYP | COSH | | CONFIDENCI | KONFIDENZ |
| 46 | | **D** | | CORREL | KORREL |
| 47 | DATUM | DATE | | COS | COS |
| 48 | DATWERT | DATEVALUE | | COSH | COSHYP |
| 49 | DBANZAHL | DCOUNT | | COUNT | ANZAHL |
| 50 | DBANZAHL2 | DCOUNTA | | COUNTA | ANZAHL2 |
| 51 | DBAUSZUG | DGET | | COUNTBLAN | ANZAHLLEERZELLEN |
| 52 | DBMAX | DMAX | | COUNTIF | ZÄHLENWENN |
| 53 | DBMIN | DMIN | | COVAR | KOVAR |
| 54 | DBMITTELWERT | DAVERAGE | | CRITBINOM | KRITBINOM |
| 55 | DBPRODUKT | DPRODUCT | | | **D** |
| 56 | DBSTDABW | DSTDEV | | DATE | DATUM |
| 57 | DBSTDABWN | DSTDEVP | | DATEVALUE | DATWERT |
| 58 | DBSUMME | DSUM | | DAVERAGE | DBMITTELWERT |
| 59 | DBVARIANZ | DVAR | | DAY | TAG |
| 60 | DBVARIANZEN | DVARP | | DAYS360 | TAGE360 |
| 61 | DELTA | DELTA | | DB | GDA2 |
| 62 | DEZINBIN | DEZINBIN | | DCOUNT | DBANZAHL |
| 63 | DEZINHEX | DEZINHEX | | DCOUNTA | DBANZAHL2 |
| 64 | DEZINOKT | DEZINOKT | | DDB | GDA |
| 65 | DIA | SYD | | DEGREES | GRAD |
| 66 | DISAGIO | DISAGIO | | DELTA | DELTA |
| 67 | DM | DOLLAR | | DEVSQ | SUMQUADABW |
| 68 | DURATION | DURATION | | DEZINBIN | DEZINBIN |
| 69 | | **E** | | DEZINHEX | DEZINHEX |
| 70 | EDATUM | EDATUM | | DEZINOKT | DEZINOKT |
| 71 | EFFEKTIV | EFFEKTIV | | DGET | DBAUSZUG |
| 72 | ERSETZEN | REPLACE | | DISAGIO | DISAGIO |
| 73 | EXP | EXP | | DMAX | DBMAX |
| 74 | EXPONVERT | EXPONDIST | | DMIN | DBMIN |
| 75 | | **F** | | DOLLAR | DM |
| 76 | FAKULTÄT | FACT | | DPRODUCT | DBPRODUKT |
| 77 | FALSCH | FALSE | | DSTDEV | DBSTDABW |
| 78 | FEHLER.TYP | ERROR.TYPE | | DSTDEVP | DBSTDABWN |
| 79 | FEST | FIXED | | DSUM | DBSUMME |
| 80 | FINDEN | FIND | | DURATION | DURATION |
| 81 | FINV | FINV | | DVAR | DBVARIANZ |
| 82 | FISHER | FISHER | | DVARP | DBVARIANZEN |
| 83 | FISHERINV | FISHERINV | | | **E** |

**Abbildung 7.7** Die Liste sämtlicher Funktionsnamen

## 7.5 Zugriff auf Zeichen innerhalb einer Zelle

Zugegeben: Man kommt mit den vier Objekten – Application, Workbook, Worksheet und Range (Cell) – aus. Es geht aber noch eine Ebene tiefer!

Nach dem Objekt der Zelle folgen die Zeichen innerhalb einer Zelle. Angenommen, für die Darstellung chemischer Formeln werden tiefgestellte Ziffern benötigt. Dazu wird die betreffende Zahl in der editierten Zelle markiert und dann über Start | Schriftart (bis Excel 2003: im Menü Format | Zellen | Schrift) die Option „Tiefgestellt“ eingestellt.

Diese Funktion kann aufgezeichnet werden, beispielsweise für H2SO4. Sofort wird klar, dass die Zelle selbst eine weitere Eigenschaft besitzt: „Characters“.

Sollen nun in einer Zelle alle darin befindlichen Ziffern tiefer gestellt werden, dann geht das folgendermaßen:

```
Sub Zeichentieferstellen()

   Dim intZeichenAnzahl As Integer

   Dim intZähler As Integer

   intZeichenAnzahl = ActiveCell.Characters.Count

   For intZähler = 1 To intZeichenAnzahl

      If IsNumeric(ActiveCell.Characters _

         (Start:=i, Length:=1).Caption) Then

            ActiveCell.Characters(Start:=i, _

               Length:=1).Font.Subscript = True

      End If
```

```
    Next

End Sub
```

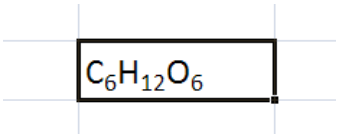



**Abbildung 7.8** Die Ziffern werden tiefgestellt – hier: meine Lieblingsformel

Damit wird klar, wie Sie einzelne Zeichen in einer Zellen formatieren können, beispielsweise mit der Schriftart Wingdings:

```
With ActiveCell
  .Value = "F vorwärts immer, rückwärts nimmer."
  .Characters(Start:=1, Length:=1).Font.Name = "Wingdings"
End With
```

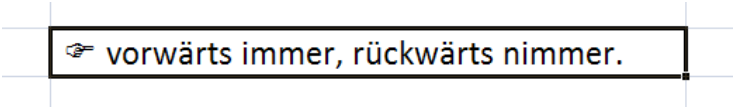



**Abbildung 7.9** Einzelne Zellen können formatiert werden.

## 7.6 Hilfsmittel in Excel

Hilfsmittel – oder wie sollte man sonst die Hilfen nennen, die Excel rund ums rechnen zur Verfügung stellt. Angenommen, Sie exportieren Daten in eine neue Tabelle nd möchten dort bestimmte Einstellungen vornehmen oder Sie möchten per Programmierung im aktuellen Blatt einen dieser Hilfsmittel verwenden, dann müssen Sie die Objekte kennen, die Excel verwendet.

### 7.6.1 Datenüberprüfung (Gültigkeit)

Seit Excel 97 hat sich an der Gültigkeit nichts geändert – lediglich der Name ist ein anderer: Datenüberprüfung. Sie wenden auf eine Zelle oder einen Zellbereich eine Datenüberprüfung mit dem Objekt `Validation` an. Es wird mit der Methode `Add` erzeugt:

```
xlZelle.Validation.Add Type:=xlValidateWholeNumber, _
   AlertStyle:=xlValidAlertStop, _
   Operator:=xlBetween, Formula1:=0, Formula2:=100
Set xlGültigkeit = xlzelle.Validation
With xlGültigkeit
   .InputTitle = "Achtung"
   .ErrorTitle = "Hinweis"
   .InputMessage = _
      "Bitte geben Sie nur ganze Zahlen zwischen 1 und 100 ein!"
     .ErrorMessage = _
```

```
        "Bitte korrigieren Sie die Eingaben - erlaubt sind nur Zahlen!"

    End With
```

**Achtung**
Beachten Sie, dass die Methode Add des Objektes Validation kein Objekt zurück gibt. Der Objektverweis muss erneut gesetzt werden.

Dabei bedeuten:

**Tabelle 7.15 Der Type des Objektes Validation**

| Konstante | Wert | Erklärung |
|---|---|---|
| xlValidateInputOnly | 0 | jeder Wert |
| xlValidateWholeNumber | 1 | ganze Zahl |
| xlValidateDecimal | 2 | Dezimalzahl |
| xlValidateDate | 4 | Datumsangabe |
| xlValidateTime | 5 | Uhrzeit |
| xlValidateTextLength | 6 | Text |
| xlValidateCustom | 7 | benutzerdefiniert |

**Tabelle 7.16** Für den Operator kann verwendet werden:

| Konstante | Wert | Erklärung |
|---|---|---|
| xlBetween | 1 | zwischen |
| xlEqual | 3 | gleich |
| xlGreater | 5 | größer als |
| xlGreaterEqual | 7 | größer oder gleich |
| xlLess | 6 | kleiner |
| xlLessEqual | 8 | kleiner oder gleich |
| xlNotBetween | 2 | nicht zwischen |
| xlNotEqual | 4 | ungleich |

Bei `xlEqual`, `xlGreater`, `xlGreaterEqual`, `xlLess`, `xlLessEqual` und `xlNotEqual` muss `Formula1` angegeben werden, bei `xlBetween` und `xlNotBetween` müssen `Formula1` und `Formula2` verwendet werden.

Für den `AlertStyle` stehen Ihnen die drei Konstanten `xlValidAlertInformation`, `xlValidAlertStop` und `xlValidAlertWarning` zur Verfügung. Sie können die beiden Eigenschaften `ShowInput` und `ShowError` auf True setzen.

Selbstverständlich können Sie auch Formeln per Programmierung eingeben.

**Achtung**
Beachten Sie dabei, dass einfache Anführungszeichen als doppelte Anführungszeichen geschrieben werden müssen. Auch wenn der Makrorekorder die deutschen Funktionsnamen aufzeichnet, muss in den VBA-Code der englische Funktionsname eingegeben werden. Und: in der deutschen Schreibweise steht das Semikolon als Trennzeichen – in der US-amerikanischen das Komma

**Beispiel**
Das folgende Beispiel verbietet dem Benutzer am Ende einer Zelle in Leerzeichen einzugeben:

```
Sub KeineLeerZeichenAmEnde()

   ActiveSheet.Cells.Validation.Delete
```

```
    ActiveSheet.Cells(1, 1).Validation.Add _
       Type:=xlValidateCustom, AlertStyle:=xlValidAlertStop, _
       Formula1:="=Right(A1,1)<>"" """
End Sub
```

**Hinweis**

Beachten Sie, dass die Eigenschaft Formula1 des Objekts Validation schreibgeschützt ist. Um eine Formel zu ändern müssen Sie die Methode Modify verwenden.

**Tipp**

Wenn Sie mit der Methode Add eine neue Gültigkeit einschalten, darf nicht bereits eine Gültigkeit vorhanden sein. Sie müssen die Gültigkeit entweder überprüfen – oder – viel einfacher: zuvor löschen-

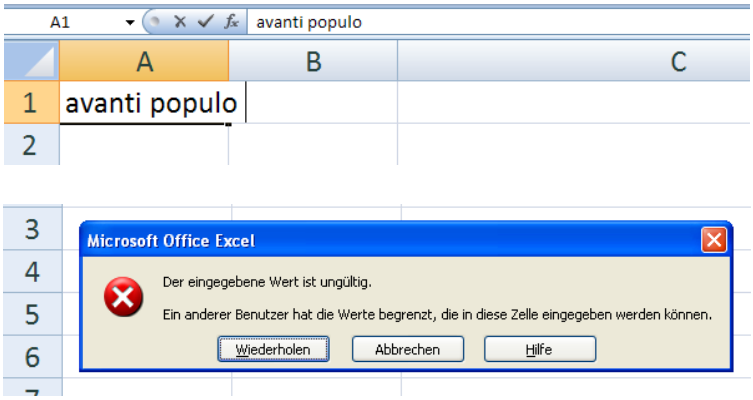



**Abbildung 7.10** Die Gültigkeit (Datenüberprüfung) kann per Programmierung gesetzt werden.

## 7.6.2 Bedingte Formatierung

Schon seit der Version 97 besitzt Excel eine bedingte Formatierung. Diese wurde in der aktuellen Version 2007 noch einmal gründlich überarbeitet. Und so stellt auch das Objekt nun weitere Methoden und Eigenschaften zur Verfügung.

Um einer Zelle oder einem Zellbereich eine bedingte Formatierung hinzufügen zu können, muss die Methode `FormatConditions.Add` verwendet werden. Sie gibt ein Objekt zurück, das als FormatCondition-Objekt weiter verarbeitet werden kann. Die Methode hat folgende optionale Parameter:

**Tabelle 7.17 Der Type der Methode FormatConditions.Add**

| Konstante | Wert | Beschreibung | seit Excel 2007 |
|---|---|---|---|
| xlCellValue | 1 | Zellwert | |
| xlExpression | 2 | Ausdruck | |
| xlAboveAverageCondition | 12 | Bedingung Über dem Durchschnitt | ✓ |
| xlBlanksCondition | 10 | Bedingung Leerzellen | ✓ |
| xlColorScale | 3 | Farbskala | ✓ |
| xlCompareColumns | 18 | Spalten vergleichen | ✓ |
| xlDatabar | 4 | Datenbalken | ✓ |
| xlErrorsCondition | 16 | Bedingung Fehler | ✓ |
| XlIconSet | 6 | Symbolsatz | ✓ |
| xlNoBlanksCondition | 13 | Bedingung Keine Leerzeichen | ✓ |

| Konstante | Wert | Beschreibung | seit Excel 2007 |
|---|---|---|---|
| xlNoErrorsCondition | 17 | Bedingung Keine Fehler | ✓ |
| xlTextString | 9 | Textzeichenfolge | ✓ |
| xlTimePeriod | 11 | Zeitraum | ✓ |
| xlTop10 | 5 | Top-10-Werte | ✓ |
| xlUniqueValues | 8 | Eindeutige Werte | ✓ |

**Tabelle 7.18** Für den Operator kann verwendet werden:

| Konstante | Wert | Erklärung |
|---|---|---|
| xlBetween | 1 | zwischen |
| xlEqual | 3 | gleich |
| xlGreater | 5 | größer als |
| xlGreaterEqual | 7 | größer oder gleich |
| xlLess | 6 | kleiner |
| xlLessEqual | 8 | kleiner oder gleich |
| xlNotBetween | 2 | nicht zwischen |
| xlNotEqual | 4 | ungleich |

Bei `xlEqual`, `xlGreater`, `xlGreaterEqual`, `xlLess`, `xlLessEqual` und `xlNotEqual` muss `Formula1` angegeben werden, bei Bei `xlBetween` und `xlNotBetween` müssen `Formula1` und `Formula2` verwendet werden, beispielsweise:

```
xlBereich.FormatConditions.Add Type:=xlCellValue, _
Operator:=xlGreater, Formula1:=100
```

Da mehrere Bedingungen verwendet werden können, kann man sie (seit Excel 2007) nach oben oder unten „schieben“:

```
xlBereich.FormatConditions(3).SetFirstPriority
xlBereich.FormatConditions(3).SetLastPriority
```

Auf eine einzelne bedingte Formatierung wird über die Indexnummer zugegriffen:

```
With xlBereich.FormatConditions(1).Interior
   .PatternColorIndex = xlAutomatic
   .Color = vbYellow
End With
```

**Tabelle 7.19 Folgende Methoden stehen Ihnen für FormatConditions zur Verfügung**

| Methode | Parameter | Beschreibung | ab Excel 2007 |
|---|---|---|---|
| Add | Type, Operator, Formula1, Formula2 | fügt eine neue bedingte Formatierung hinzu | |
| Delete | | löscht die bedingten Formatgierungen | |
| Item | | das einzelne Objekt | |
| AddAboveAverage | | formatiert Werte über oder unter dem Durchschnitt | ✓ |

| Methode | Parameter | Beschreibung | ab Excel 2007 |
|---|---|---|---|
| AddColorScale | ColorScale | fügt Farben-Skane hinzu | ✓ |
| AddDatabar | | fügt Datenbalken hinzu | ✓ |
| AddIconSetCondition | | Zellen basierend auf ihren Werten formatieren | ✓ |
| AddTop10 | | obere oder untere Werte formatieren | ✓ |
| AddUniqueValues | | eindeutige oder doppelte Werte formatieren | ✓ |

Zur Formatierung stehen Ihnen folgende Objekte zur Verfügung:

- Borders
- Font
- Interior
- NumberFormat

Damit wird klar, wie man auch Formeln über die bedingte Formatierung hinzufügen kann:

```
Dim xlSpalte As Range

Set xlSpalte = ActiveSheet.Range("A:A")


 xlSpalte.FormatConditions.Add Type:=xlExpression, Formula1:= _

     "=GANZZAHL(ZEILE()/2)=ZEILE()/2"

With xlSpalte.FormatConditions(1)

   .Interior.PatternColorIndex = xlAutomatic

   .Interior.Color = vbBlack

   .Font.Color = vbWhite

End With
```

**Hinweis**

Beachten Sie, dass ein zweimaliger Aufruf bis Excel 2003 die bereits vorhandene bedingte Formatierung überschreibt, ab Excel 2007 – bei der beliebig viele bedingte Formatierungen eingeschaltet werden können – wird sie erneut angefügt. Deshalb sollten Sie – unabhängig von der Version – mögliche bedingte Formatierungen am Beginn löschen:

```
xlSpalte.FormatConditions.Delete
```

**Hinweis**

Beachten Sie auch, dass Sie – anders als bei der Gültigkeit – die Formeln in der deutschen Schreibweise eingeben dürfen, also:

```
xlSpalte.FormatConditions.Add Type:=xlExpression, Formula1:= _

   "=GANZZAHL(ZEILE()/2)=ZEILE()/2"
```

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Gemüse | | | | | | | | | | | | | |
| 2 | Die Angaben beziehen sich auf je 100 Gramm verzehrfertiges Nahrungsmittel | roh oder gekocht | kcal | kJ | % Kohlehydrate | Eiweiß | % Gesamtfett | % Wasser | % Ballaststoffe | mg Vitamin C | mg Calcium | mg Kalium | mg Phopsphor | mg Magnesium |
| 3 | | | | | | | | | | | | | | |
| 4 | Artischocken | roh | 57 | 239 | 11 | + | 3 | 86 | 3 | 9 | 52 | 410 | 110 | 26 |
| 5 | " | gek | 57 | 239 | 11 | + | 3 | 86 | 3 | 6 | 50 | 315 | 90 | |
| 6 | Auberginen | roh | 25 | 105 | 5 | + | 1 | 93 | 3 | 5 | 16 | 210 | 26 | 11 |
| 7 | " | gek | 16 | 67 | 3 | + | 1 | 93 | 3 | 5 | 16 | 210 | 26 | |
| 8 | Blattsellerie | roh | 20 | 84 | 4 | + | 1 | 95 | 2 | 9 | 45 | 290 | 40 | |
| 9 | Blumenkohl | roh | 28 | 117 | 8 | + | 2 | 92 | 2 | 76 | 24 | 380 | 60 | 7 |
| 10 | " | gek | 20 | 84 | 3 | + | 2 | 94 | 2 | 45 | 18 | 250 | 52 | |
| 11 | Bohnen, grün | gek | 33 | 138 | 6 | + | 2 | 92 | 2 | 13 | 50 | 150 | 40 | 26 |
| 12 | Bohnen in Dosen | gek | 24 | 100 | 4 | + | 1 | 93 | 3 | 4 | 35 | 145 | 25 | 20 |
| 13 | Bohnenkerne | gek | 137 | 574 | 25 | - | 1 | 6 | 67 | | 2 | 48 | 400 | 130 |
| 14 | Broccoli | roh | 32 | 134 | 4 | + | 4 | 91 | 92 | 4 | 110 | 110 | 405 | 78 |
| 15 | " | gek | 32 | 134 | 4 | + | 4 | 92 | | 4 | 50 | 80 | 315 | 54 |

**Abbildung 7.11** Die Zeilen werden mit einer bedingten Formatierung gestaltet – zugegeben: Excell 2007 hat dafür einige Designs („als Tabelle formatieren“) vorbereitet.

## 7.7 Listen in Excel

Die Strategie von Excel seit der aktuellen Version 2007 zeigt es deutlich: Excel wird mehr und mehr zu einem Datenbankinstrument ohne natürlich die komplette Funktionalität einer Datenbank wie beispielsweise Access oder SQL-Server übernehmen zu können. Auch wenn Sie tausende von Datensätzen sortieren und filtern müssen, um die Ergebnisse der entsprechenden „Abfrage“ zu erhalten, so ist Excel dennoch schnell und kann immer noch Datenmengen, die sich in mehreren Tausend Datensätzen bewegen, problemlos analysieren. Diese Methoden werden im Folgenden vorgestellt:

### 7.7.1 Daten sortieren

Von Excel 2003 nach 2007 wurde das Sortieren überarbeitet. Waren bis Excel 2003 nur drei Sortierkriterien möglich, sind es nun beliebig viele. Die Sortiermethode wurde nicht erweitert, sondern es wurde eine neue hinzugefügt.

Bis Excel 2003 haben Sie sortiert, indem Sie die Methode beispielsweise folgendermaßen verwenden:

```
xlZelle.Sort Key1:=xlBlatt.Range("E1"), Order1:=xlAscending, _

   Key2:=xlBlatt.Range("D1"), Order2:=xlAscending, Header:=xlYes
```

Excel „erkennt“ den zusammenhängenden Bereich und sortiert automatisch den korrekten Bereich. Leichter zu lesen ist sicherlich folgende Schreibweise:

```
xlZelle.CurrentRegion.Sort _

   Key1:=xlBlatt.Range("E1"), Order1:=xlAscending, _

   Key2:=xlBlatt.Range("D1"), Order2:=xlAscending, Header:=xlYes
```

**Hinweis**

Beachten Sie, dass dies Liste der Parameter über einen Parameter Header verfügt. Er besitzt drei Werte: xlYes, xlNo und xlGuess. Sie werden in der Praxis sicherlich meistens mit Header:=xlYes sortieren.

Die Sortiermethode, die Excel seit der Version 2007 verwendet, ist nicht mehr eine Methode einer Zelle oder eines Bereiches, sondern gehört zum Blatt. Es wird dabei in zwei Schritten vorgegangen. Im ersten Schritt werden die Sortierkriterien festgelegt:

```
xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("E1:E3817"), _
```

```
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
    xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("D1:D3817"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
    xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("J1:J3817"), _
        SortOn:=xlSortOnValues, Order:=xlDescending, _
        DataOption:=xlSortNormal
    xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("I1:I3817"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
```

Es ist natürlich albern die Anzahl der zu sortierenden Zeilen zu bestimmen. Deshalb können Sie die Spaltenreihenfolge auch wie folgt bestimmen:

```
    xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("E1").EntireColumn, _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
```

Es geht sogar noch kürzer, indem Sie lediglich auf eine Zelle nach der zu sortierenden Spalte verweisen und nicht die komplette Spalte angeben:

```
    xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("E1"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:= xlSortNormal
```

Der Parameter `SortOn` erwartet dabei einen der vier Werte: `SortOnValues`, `SortOnCellColor`, `SortOnIcon` oder `SortOnFontColor`. `SortOnValue` „erkennt“ dabei automatisch, ob in der Spalte Texte, Zahlen oder Datumsangaben stehen und sortiert sie korrekt.

Anschließend wird die eigentliche Sortierung mithilfe der Voreinstellungen durchgeführt. Auch sie wird wieder auf das Tabellenblatt angewandt:

```
With xlBlatt.Sort
        .SetRange xlZelle.CurrentRegion
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
```

Die eigentliche Sortierung geschieht erst bei der Methode Apply.

**Tipp**

Wenn Sie Sortierkriterien mit der Methode Add hinzufügen, werden bereits vorhandene nicht gelöscht. Da doppelte Sortierkriterien zu einem Fehler führen, sollten Sie zu Beginn der Sortieraktion mögliche, bereits vorhandene Sortierkriterien ausschalten:

```
xlBlatt.Sort.SortFields.Clear
```

| Kundennum | Geschlecht | Titel | Name | Straße | Plz | Ort | Jahresbeitrag | Geburtsdatum | Eintrittsdatum | Konto-Nr | Bankleitzahl | Bankbezei |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4598 | 10 | Dr. Med. | Kathrin Waligura | Ursulahof 10 | 52511 | Geilenkirchen | 148 | 03.08.1921 | 01.04.1986 | | | |
| 4726 | 10 | | Sieglinde Geidel | Kappesgaerten 22 | 68535 | Edingen-Neckar | 148 | 23.11.1923 | 01.01.1984 | | | |
| 6266 | 10 | | Martha Kleinschmitt | Tilsiter Weg 7-9 | 68723 | Schwetzingen | 148 | 03.02.1924 | 29.01.1942 | | | |
| 810 | 10 | Dr. | Ann Weg | Lameystr.34 | 68165 | Mannheim | 148 | 17.02.1926 | 13.02.1944 | | | |
| 2009 | 20 | | Walter Kocher | Rathausstr.34 | 82008 | Unterhaching | 148 | 23.04.1926 | 01.06.1985 | | | |
| 2802 | 20 | | Dieter Kiessling | Ahornhof 24 | 68305 | Mannheim | 148 | 15.03.1927 | 01.01.1977 | | | |
| 3275 | 10 | | Kaethe Lehmeyer | Muehlenweg 69 | 68549 | Ilvesheim | 148 | 07.01.1928 | 02.01.1946 | 23743101 | 67080050 | Dresdner E |
| 109 | 20 | | Willi Werner | Ulanenweg 12 | 68163 | Mannheim | 148 | 08.01.1928 | 01.08.1987 | | | |
| 7021 | 20 | | Kurt Bumiller | Rheintalbahnstr.33 | 68753 | Waghaeusel | 148 | 04.02.1928 | 01.07.1972 | | | |
| 1543 | 20 | | Karl Temme | Siemensstr.6 | 68549 | Ilvesheim | 148 | 16.02.1928 | 11.02.1946 | | | |
| 5042 | 10 | | Maria Wagner | Scheidegger Str. 2 | 81476 | Muenchen | 136 | 10.03.1928 | 01.05.1963 | 10119762 | 67020259 | Hypo-Mann |
| 563 | 20 | | Peter Seitz | Hauptstr.94 | 68535 | Edingen-Neckar | 148 | 14.03.1928 | 01.05.1981 | 11441301 | 67070010 | Dtbk Mann |
| 806 | 10 | Dr. | Roswitha Albig | Adlerstr.59 | 68199 | Mannheim | 148 | 27.03.1928 | 01.05.1983 | | | |
| 3196 | 10 | | Gertrud Leinweber | Drachenfelsstr.10 | 68163 | Mannheim | 148 | 28.03.1928 | 24.03.1946 | | | |

**Abbildung 7.12** Die Kunden werden nach dem Alter sortiert.

## 7.7.2 Daten filtern: Autofilter und Spezialfilter

Excel bietet seit vielen Versionen zwei verschiedene Filter an: der Autofilter und der Spezialfilter. Während ersterer großer Beliebtheit erfreut, weil er so leicht zu bedienen ist, ist der zweite nahezu unbekannt. Zugegeben: der Spezialfilter birgt einige Schwierigkeiten, die schnell zu Fehlern führen können, in sich, die man beim Programmieren beachten muss. Wenn Sie ihn allerdings einmal benutzt haben, dann werden Sie schnell feststellen, dass er flexibler ist als der Autofilter und mehr Verknüpfungsoptionen bietet als der Autofilter. In diesem Abschnitt werden beide Filter vorgestellt.

### Der Autofilter

Sie schalten den Autofilter mit der folgenden Methode ein:

```
ActiveCell.AutoFilter
```

**Achtung**

Diese Methode schaltet den Autofilter ein oder aus. Beim zweiten Durchlauf würde er folglich wieder ausgeschaltet werden. Um sicherzugehen, ob der Filter eingeschaltet ist, müssen Sie vorher überprüfen, dass er noch nicht eingeschaltet ist. Dafür stellt das Blatt die Eigenschaft AutoFilterMode bereit:

```
If ActiveSheet.AutoFilterMode Then
```

**Hinweis**

Besser lesbar – wenn auch für die Programmierung nicht unbedingt nötig – ist folgende Codezeile:

```
ActiveCell.CurrentRegion.AutoFilter
```

Damit ist er lediglich eingeschaltet. Da diese Methode keine Objekte zurückgibt, könnten Sie nun die Filterkriterien auf den gefilterten Bereich anwenden. Oder – sinnvoller – Sie schalten die Kriterien beim Filtern ein:

```
ActiveCell.CurrentRegion.AutoFilter Field:=7, Criteria1:= "Muenchen"
```

**Tabelle 7.20 Der Autofilter verfügt über folgende Parameter**

| Parameter | Beschreibung |
|---|---|
| Field | Der ganzzahlige Offset (Versatz) des Felds, auf dem der Filter basieren soll (links in der Liste beginnend, wobei das Feld ganz links als erstes Feld verwendet wird). |
| Criteria1 | Die Kriterien (eine Zeichenfolge; z. B. "101"). Verwenden Sie für die Suche nach leeren Feldern "=" und für die Suche nach nicht leeren Feldern "<>". Wenn dieses Argument nicht angegeben wird, lauten die Kriterien "All". Wenn Operator gleich xlTop10Items ist, gibt Criteria1 die Anzahl von Elementen an (z. B. "10"). |
| Operator | Eine der Konstanten von XlAutoFilterOperator, wodurch der Typ des Filters |

| Parameter | Beschreibung |
|---|---|
| | angegeben wird. |
| Criteria2 | Die zweiten Kriterien (eine Zeichenfolge). Wird mit Criteria1 und Operator zum Erstellen von zusammengesetzten Kriterien verwendet. |
| VisibleDropDown | Beim Wert True werden die Dropdownpfeile von AutoFilter für das gefilterte Feld angezeigt. Beim Wert False werden die Dropdownpfeile von AutoFilter für das gefilterte Feld ausgeblendet. Der Standardwert ist True. |

**Tabelle 7.21 Die Konstanten des Operators xlAutoFilterOperator**

| Operator | Wert | Beschreibung |
|---|---|---|
| xlAnd | 1 | Logisches UND zwischen Kriterium1 und Kriterium2 |
| xlOr | 2 | Logisches ODER zwischen Kriterium1 und Kriterium2 |
| xlBottom10Items | 4 | Die Einträge mit dem niedrigsten Wert werden angezeigt (die Anzahl der Einträge ist in Kriterium1 angegeben). |
| xlBottom10Percent | 6 | Die Einträge mit dem niedrigsten Wert werden angezeigt (der Prozentsatz ist in Kriterium1 angegeben). |
| xlTop10Items | 3 | Die Einträge mit dem höchsten Wert werden angezeigt (die Anzahl der Einträge ist in Kriterium1 angegeben). |
| xlTop10Percent | 5 | Die Einträge mit dem höchsten Wert werden angezeigt (der Prozentsatz ist in Kriterium1 angegeben). |
| xlFilterCellColor | 8 | Farbe der Zelle (erst seit Excel 2007) |
| xlFilterDynamic | 11 | Dynamischer Filter (erst seit Excel 2007) |
| xlFilterFontColor | 9 | Farbe der Schriftart (erst seit Excel 2007) |
| xlFilterIcon | 10 | Filtersymbol (erst seit Excel 2007) |
| xlFilterValues | 7 | Filterwerte (erst seit Excel 2007) |

Bis Excel 2003 war es möglich nur mit zwei Kriterien zu filtern, die mit einem logischen und beziehungsweise einem oder verknüpft sind. Filter alle Kunden, die in Hamburg oder Berlin wohnen, hatte folgenden Aufbau:

```
ActiveCell.CurrentRegion.AutoFilter Field:=7, _
Criteria1:="=Hamburg", Operator:=xlOr, Criteria2:="=Berlin"
```

In Excel 2007 ist es nun möglich ein Array aus beliebt vielen Werten anzugeben, die gefiltert werden können:

```
ActiveCell.CurrentRegion.AutoFilter Field:=7, _
   Criteria1:=Array("Altoetting", "Aschaffenburg", "Augsburg", _
   "Bamberg", "Muenchen", "Nuernberg", "Regensburg", "Wuerzburg"), _
   Operator:=xlFilterValues
```

**Hinweis**

Dabei spielt es keine Rolle, ob diese Kriterien wirklich in der Liste vorhanden sind. Taucht beispielsweise „Altoetting“ nicht auf, dann wird das mit oder verknüpfte Kriterium übergangen.

| Kundennr. | Geschlec | Titel | Name | Straße | Plz | Ort | Jahresbeitra | Geburtsdatum | Eintrittsdatu | Konto-Nr | Bankleitz | Bankbeze |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 10 | | Alberta Winzer | Fischersteige 9 | 87435 | Kempten | 148 | 04.12.1967 | 01.01.2000 | | | |
| 151 | 10 | | Herta Kolb | Obere Waldstr.19 | 88709 | Meersburg | 136 | 14.04.1948 | 09.04.1964 | 6450253 | 59420088 | Bayvbk-S |
| 229 | 20 | | Lothar Lahm | Meisenweg 15 | 67269 | Gruenstadt | 148 | 21.04.1959 | 16.04.1977 | | | |
| 237 | 10 | Dr. | Lea Korstick | Adolf-von-Baeyer-V | 84508 | Burgkirchen | 148 | 17.11.1974 | 12.11.1992 | | | |
| 267 | 20 | | Ralf Schmidt | Prinzregentenstras | 80538 | Muenchen | 148 | 30.07.1975 | 25.07.1993 | | | |
| 327 | 20 | | Ralf Schmickal | Prinzregentenstras | 80538 | Muenchen | 148 | 18.03.1980 | 14.03.1998 | | | |
| 330 | 10 | | Annette Rohrmuelle | Sedelhofstr. 5 | 81247 | Muenchen | 142 | 07.10.1970 | 01.10.1990 | | | |
| 333 | 20 | Dr. | Antonie Kirsch | Rieneckerstr. 11 | 81249 | Muenchen | 142 | 06.10.1970 | 01.10.1990 | | | |
| 405 | 10 | | Doris Esser | Braeuhausstr. 4 B | 82152 | Planegg | 148 | 12.10.1985 | 08.10.2003 | | | |
| 416 | 20 | | Peter Sachau | Possartstrasse 11 | 81679 | Muenchen | 148 | 23.08.1972 | 19.08.1990 | | | |
| 443 | 20 | | Lothar Kropp | Hubertusstr.88 | 82131 | Gauting | 148 | 01.06.1930 | 27.05.1948 | 252312 | 67050101 | Stspka M |
| 453 | 20 | | Gerd Grosse-Allern | Europaplatz 1 | 81675 | Muenchen | 148 | 10.03.1959 | 05.03.1977 | 10453689 | 67020259 | Hypo-Mar |
| 513 | 20 | | Thomas Kirsch | Bellinzonastr. 1 | 81475 | Muenchen | 132 | 05.10.1960 | 01.10.1995 | | | |
| 519 | 20 | | Frank Geberzahn | Denningerstrasse | 81927 | Muenchen | 148 | 16.12.1971 | 11.12.1989 | 471300 | 67070010 | Dtbk Mar |
| 566 | 20 | | Michael Pischke | Muenchhausenstr. | 81247 | Muenchen | 148 | 31.12.1958 | 26.12.1976 | | | |
| 568 | 20 | | Peter Sacher | Possartstrasse 11 | 81679 | Muenchen | 148 | 21.03.1971 | 16.03.1989 | 3665275 | 66090800 | Bdbeabk- |
| 605 | 10 | | Stefanie Adrom | Beblostr. 32 | 81677 | Muenchen | 148 | 03.10.1970 | 01.10.1995 | | | |
| 612 | 10 | | Tina Dohm | Fuerstenriederstr. | 80686 | Muenchen | 148 | 01.10.1970 | 01.05.1992 | | | |
| 613 | 20 | Dr. | Ernesto Lecuona | Konventstr. 22 | 84503 | Altoetting | 152 | 07.06.1958 | 02.06.1976 | | | |
| 636 | 20 | | Ralf Schmelcher | Prinzregentenstras | 80538 | Muenchen | 136 | 12.12.1973 | 08.12.1991 | | | |
| 655 | 20 | | Oliver Ritthoff | Osterwaldstr. 10 | 80805 | Muenchen | 148 | 26.02.1975 | 21.02.1993 | 241513 | 67050101 | Stspka M |
| 750 | 20 | | Olaf Rid-Niebler | Oskar-von-Miller-Ri | 80333 | Muenchen | 148 | 25.06.1974 | 20.06.1992 | 9780842 | 67070010 | Dtbk Mar |
| 783 | 20 | | Frank-Peter GmbH | Dr.-Mack-Strasse 7 | 90762 | Fuerth | 148 | 11.08.1987 | 06.08.2005 | 275065 | 67050101 | Stspka M |
| 855 | 20 | | Ralph Schmidt | Prinzregentenstras | 80538 | Muenchen | 148 | 08.07.1971 | 03.07.1989 | 199232 | 67050101 | Stspka M |
| 928 | 20 | | Sergej Schumache | Schackstrasse 2 | 80539 | Muenchen | 148 | 08.03.1987 | 03.03.2005 | | | |
| 1015 | 20 | | Ob Rickert | Oskar-von-Miller-Ri | 80333 | Muenchen | 148 | 12.09.1964 | 08.09.1982 | | | |
| 1018 | 20 | Dr. | Manfred Mueller | Leopoldstrasse 57 | 80802 | Muenchen | 148 | 03.03.1983 | 26.02.2001 | | | |
| 1024 | 20 | | Michael Pechtl | Mies-van-der-Rohe | 80807 | Muenchen | 148 | 16.09.1983 | 11.09.2001 | 7578305 | 67050101 | Stspka M |
| 1034 | 20 | | John Axelrod | Josef-Stegmair-Str | 84489 | Burghausen | 136 | 09.11.1934 | 04.11.1952 | | | |
| 1040 | 20 | | Sebastian Schulz | Salvatorstrasse 13 | 80333 | Muenchen | 136 | 26.04.1966 | 21.04.1984 | 110130 | 67050101 | Stspka M |
| 1065 | 20 | | Thomas Stubbe | Stockdorfer Str. 58 | 81475 | Muenchen | 128 | 17.06.1987 | 12.06.2005 | 6005326 | 67051203 | Spk-Hock |
| 1076 | 10 | | Andrea Schliffenbac | Herzogstandstr. 24 | 81539 | Muenchen | 136 | 02.10.1955 | 01.04.1983 | | | |
| 1083 | 20 | | Albert Alarmbeigi | Adenauer Allee 9 | 81737 | Muenchen | 148 | 17.04.1956 | 13.04.1974 | 3201316 | 67040031 | Czbk-Mar |
| 1084 | 20 | | Ed Fedderke | Brienner Str. 9 | 80333 | Muenchen | 148 | 12.04.1975 | 07.04.1993 | | | |
| 1105 | 10 | | Sabine Schulenburg | Rundfunkplatz 1 | 80335 | Muenchen | 148 | 08.10.1984 | 04.10.2002 | | | |
| 1107 | 20 | | Matthias Ortlinghau | Maximilianstr. 27 | 80539 | Muenchen | 148 | 14.11.1980 | 10.11.1998 | | | |
| 1128 | 20 | | Norbert Ricke | Oskar-von-Miller-Ri | 80333 | Muenchen | 148 | 23.09.1955 | 18.09.1973 | | | |
| 1181 | 10 | | Ilka Klein | Siedlerstr. 7 | 83607 | Holzkirchen | 148 | 29.02.1964 | 24.02.1982 | | | |
| 1214 | 20 | | Willi Schuster | Trifelsstr.34 A | 67269 | Gruenstadt | 148 | 23.09.1932 | 01.08.1987 | | | |

**Abbildung 7.13** Der Autofilter

Um die gesetzten Kriterien wieder auszuschalten, können Sie entweder

```
ActiveCell.CurrentRegion.AutoFilter Field:=7
```

das eine Kriterium ausschalten oder mit einer Schleife sämtliche Kriterien ausschalten:

```
For i = 1 To ActiveCell.CurrentRegion.Columns.Count
   ActiveCell.CurrentRegion.AutoFilter Field:=i
Next
```

Oder Sie schalten den Filter aus und wieder ein:

```
ActiveCell.CurrentRegion.AutoFilter
ActiveCell.CurrentRegion.AutoFilter
```

Auch hier sollten Sie zuerst überprüfen, ob der Filter bereits gesetzt wurde:

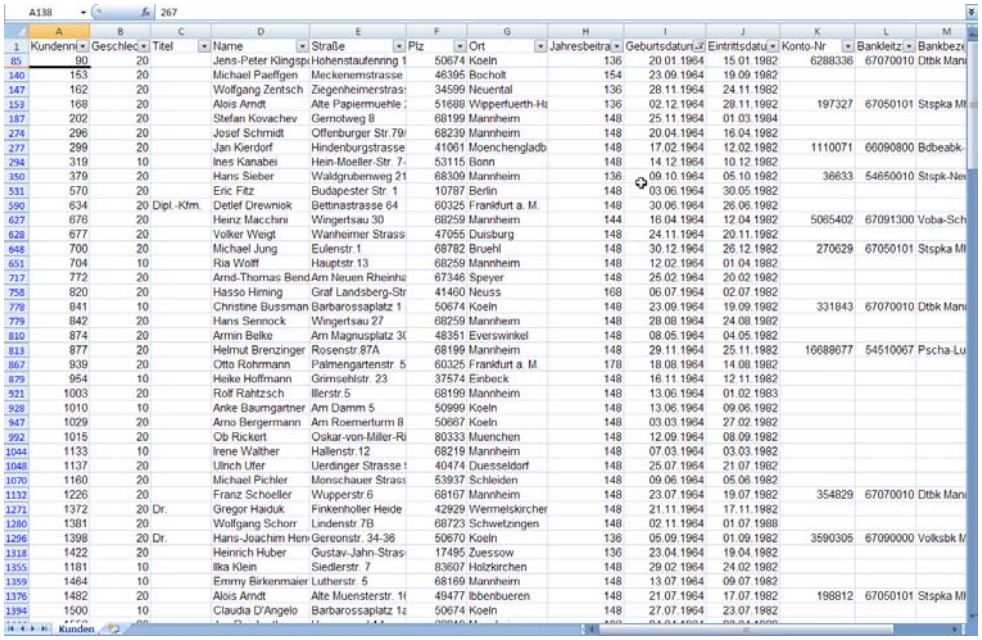
```
If ActiveSheet.AutoFilterMode Then
   Selection.AutoFilter
   Selection.AutoFilter
End If
```

**Beispiel**

Im nächsten Beispiel wird der Benutzer nach einer Jahreszahl gefragt, die in einer Spalte mit Datumsangaben filtert. Beachten Sie, dass der Makrorekorder zwar den 31.12.1964 aufzeichnet, VBA jedoch intern die US-amerikanische Schreibweise 12/31/1964 verlangt.

```
Sub Jahresfilter()
   Dim intJahr As Integer
   On Error GoTo ende
   intJahr = InputBox("Welches Jahr soll gefiltert werden?")
```

```
    ActiveCell.CurrentRegion.AutoFilter Field:=9, _
       Criteria1:= ">=01/01/" & intJahr, Operator:=xlAnd, _
       Criteria2:="<=12/31/" & intJahr


    Exit Sub
ende:
    MsgBox "Es trat ein Fehler auf: " & Err.Description
End Sub
```



| | Kundenni | Geschlec | Titel | Name | Straße | Plz | Ort | Jahresbeitra | Geburtsdatun | Eintrittsdatu | Konto-Nr | Bankleitz | Bankbez |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 85 | 90 | 20 | | Jens-Peter Klingsp | Hohenstaufenring 1 | 50674 | Koeln | 136 | 20.01.1964 | 15.01.1982 | 6288336 | 67070010 | Dtbk Mani |
| 140 | 153 | 20 | | Michael Paeffgen | Meckenemstrasse | 46395 | Bocholt | 154 | 23.09.1964 | 19.09.1982 | | | |
| 147 | 162 | 20 | | Wolfgang Zentsch | Ziegenheimerstras | 34599 | Neuental | 136 | 28.11.1964 | 24.11.1982 | | | |
| 153 | 168 | 20 | | Alois Arndt | Alte Papiermuehle | 51688 | Wipperfuerth-Ha | 136 | 02.12.1964 | 28.11.1982 | 197327 | 67050101 | Stspka Mi |
| 187 | 202 | 20 | | Stefan Kovachev | Gernotweg 8 | 68199 | Mannheim | 148 | 25.11.1964 | 01.03.1984 | | | |
| 274 | 296 | 20 | | Josef Schmidt | Offenburger Str.79 | 68239 | Mannheim | 148 | 20.04.1964 | 16.04.1982 | | | |
| 277 | 299 | 20 | | Jan Kierdorf | Hindenburgstrasse | 41061 | Moenchengladb | 148 | 17.02.1964 | 12.02.1982 | 1110071 | 66090800 | Bdbeabk- |
| 294 | 319 | 10 | | Ines Kanabei | Hein-Moeller-Str. 7- | 53115 | Bonn | 148 | 14.12.1964 | 10.12.1982 | | | |
| 350 | 379 | 20 | | Hans Sieber | Waldgrubenweg 21 | 68309 | Mannheim | 136 | 09.10.1964 | 05.10.1982 | 38633 | 54850010 | Stspk-Ne |
| 531 | 570 | 20 | | Eric Fitz | Budapester Str. 1 | 10787 | Berlin | 148 | 03.06.1964 | 30.05.1982 | | | |
| 590 | 634 | 20 | Dipl.-Kfm. | Detlef Drewniok | Bettinastrasse 64 | 60325 | Frankfurt a. M. | 148 | 30.06.1964 | 26.06.1982 | | | |
| 627 | 676 | 20 | | Heinz Macchini | Wingertsau 30 | 68259 | Mannheim | 144 | 16.04.1964 | 12.04.1982 | 5065402 | 67091300 | Voba-Sch |
| 628 | 677 | 20 | | Volker Weigt | Wanheimer Strass | 47055 | Duisburg | 148 | 24.11.1964 | 20.11.1982 | | | |
| 648 | 700 | 20 | | Michael Jung | Eulenstr.1 | 68782 | Bruehl | 148 | 30.12.1964 | 26.12.1982 | 270629 | 67050101 | Stspka Mi |
| 651 | 704 | 10 | | Ria Wolff | Hauptstr.13 | 68259 | Mannheim | 148 | 12.02.1964 | 01.04.1982 | | | |
| 717 | 772 | 20 | | Arnd-Thomas Bend | Am Neuen Rheinha | 67346 | Speyer | 148 | 25.02.1964 | 20.02.1982 | | | |
| 758 | 820 | 20 | | Hasso Himing | Graf Landsberg-Str | 41460 | Neuss | 168 | 06.07.1964 | 02.07.1982 | | | |
| 778 | 841 | 10 | | Christine Bussman | Barbarossaplatz 1 | 50674 | Koeln | 148 | 23.09.1964 | 19.09.1982 | 331843 | 67070010 | Dtbk Mani |
| 779 | 842 | 20 | | Hans Sennock | Wingertsau 27 | 68259 | Mannheim | 148 | 28.08.1964 | 24.08.1982 | | | |
| 810 | 874 | 20 | | Armin Belke | Am Magnusplatz 3( | 48351 | Everswinkel | 148 | 08.05.1964 | 04.05.1982 | | | |
| 813 | 877 | 20 | | Helmut Brenzinger | Rosenstr.87A | 68199 | Mannheim | 148 | 29.11.1964 | 25.11.1982 | 16688677 | 54510067 | Pscha-Lu |
| 867 | 939 | 20 | | Otto Rohrmann | Palmengartenstr. 5 | 60325 | Frankfurt a. M. | 178 | 18.08.1964 | 14.08.1982 | | | |
| 879 | 954 | 10 | | Heike Hoffmann | Grimsehlstr. 23 | 37574 | Einbeck | 148 | 16.11.1964 | 12.11.1982 | | | |
| 921 | 1003 | 20 | | Rolf Rahtzsch | Illerstr.5 | 68199 | Mannheim | 148 | 13.06.1964 | 01.02.1983 | | | |
| 928 | 1010 | 10 | | Anke Baumgartner | Am Damm 5 | 50999 | Koeln | 148 | 13.06.1964 | 09.06.1982 | | | |
| 947 | 1029 | 20 | | Arno Bergermann | Am Roemerturm 8 | 50667 | Koeln | 148 | 03.03.1964 | 27.02.1982 | | | |
| 992 | 1015 | 20 | | Ob Rickert | Oskar-von-Miller-Ri | 80333 | Muenchen | 148 | 12.09.1964 | 08.09.1982 | | | |
| 1044 | 1133 | 10 | | Irene Walther | Hallenstr.12 | 68219 | Mannheim | 148 | 07.03.1964 | 03.03.1982 | | | |
| 1048 | 1137 | 20 | | Ulrich Ufer | Uerdinger Strasse | 40474 | Duesseldorf | 148 | 25.07.1964 | 21.07.1982 | | | |
| 1070 | 1160 | 20 | | Michael Pichler | Monschauer Strass | 53937 | Schleiden | 148 | 09.06.1964 | 05.06.1982 | | | |
| 1132 | 1226 | 20 | | Franz Schoeller | Wupperstr.6 | 68167 | Mannheim | 148 | 23.07.1964 | 19.07.1982 | 354829 | 67070010 | Dtbk Mani |
| 1271 | 1372 | 20 | Dr. | Gregor Haiduk | Finkenholler Heide | 42929 | Wermelskircher | 148 | 21.11.1964 | 17.11.1982 | | | |
| 1280 | 1381 | 20 | | Wolfgang Schorr | Lindenstr 7B | 68723 | Schwetzingen | 148 | 02.11.1964 | 01.07.1988 | | | |
| 1296 | 1398 | 20 | Dr. | Hans-Joachim Her | Gereonstr. 34-36 | 50670 | Koeln | 136 | 05.09.1964 | 01.09.1982 | 3590305 | 67090000 | Volksbk M |
| 1318 | 1422 | 20 | | Heinrich Huber | Gustav-Jahn-Stras | 17495 | Zuessow | 136 | 23.04.1964 | 19.04.1982 | | | |
| 1355 | 1181 | 10 | | Ilka Klein | Siedlerstr. 7 | 83607 | Holzkirchen | 148 | 29.02.1964 | 24.02.1982 | | | |
| 1359 | 1464 | 10 | | Emmy Birkenmaier | Lutherstr. 5 | 68169 | Mannheim | 148 | 13.07.1964 | 09.07.1982 | | | |
| 1376 | 1482 | 20 | | Alois Arndt | Alte Muensterstr. 1( | 49477 | Ibbenbueren | 148 | 21.07.1964 | 17.07.1982 | 198812 | 67050101 | Stspka Mi |
| 1394 | 1500 | 10 | | Claudia D'Angelo | Barbarossaplatz 1a | 50674 | Koeln | 148 | 27.07.1964 | 23.07.1982 | | | |

**Abbildung 7.14** Alle Kunden, die im Jahre 1964 geboren sind.

Wenn Sie mehrere Filter einschalten möchten, die Excel mit einem logischen `und` verknüpft, aktivieren Sie die Filter nacheinander. Da die und-Verknüpfung kommutativ ist, spielt die Reihenfolge keine Rolle:

```
ActiveCell.CurrentRegion.AutoFilter Field:=7, Criteria1:= "=Mannheim"
    ActiveCell.CurrentRegion.AutoFilter Field:=3, Criteria1:="<>"
    ActiveCell.CurrentRegion.AutoFilter Field:=2, Criteria1:="10"
```

### Der Spezialfilter

Da der Autofilter in Excel 2007 überarbeitet wurde, spielt der Spezialfilter in seiner ursprünglichen Mächtigkeit nicht mehr die größte Rolle, die er bis Excel 2003 innehatte (in Excel 2007: Daten | Erweitert)

Die Technik des Spezialfilters besteht darin, dass die Kriterien in Zellen definiert und für die Filterung werden. Das Ergebnis wird auf einem anderen Blatt oder in einer anderen Datei angezeigt.

**Beispiel**

Angenommen, Sie möchten alle Kunden filtern, die entweder in München wohnen oder mehr als 180 € Jahresbeitrag bezahlen. dann können Sie die Kriterien in ein anderes Blatt eintragen. Kriterien, die nebeneinander stehen werden mit einem logischen und verknüpft, untereinander bedeutet für Excel oder:

```
Sub Spezialfilter1()

   Dim xlDatei As Workbook

   Dim xlBlattDaten As Worksheet

   Dim xlBlattZiel As Worksheet


   Set xlDatei = ActiveWorkbook

   Set xlBlattDaten = xlDatei.Worksheets(1)

   Set xlBlattZiel = xlDatei.Worksheets(3)


    xlBlattDaten.Range("A1").CurrentRegion.AdvancedFilter _

      Action:=xlFilterCopy, _

      CriteriaRange:=xlBlattZiel.Range("A1:B3"), _

      CopyToRange:=xlBlattZiel.Range("A5"), _

      Unique:=False

End Sub
```

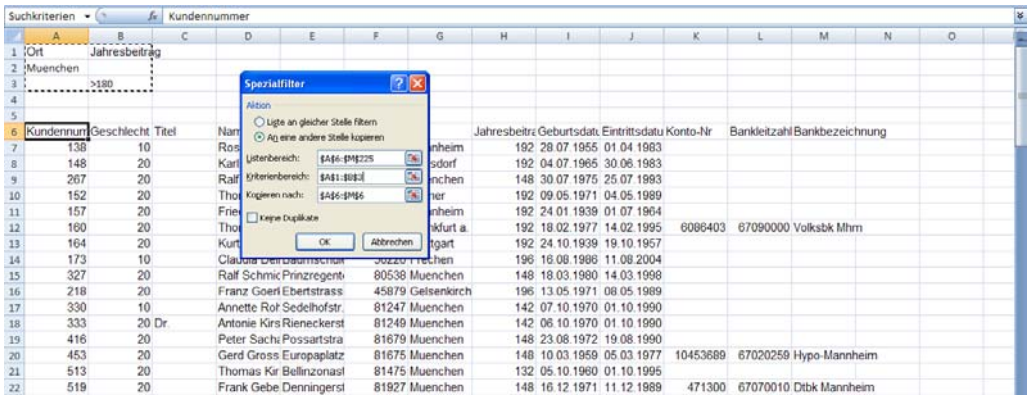



**Abbildung 7.15** Der Spezialfilter – hier: mit dem Ergebnis an der gleichen Stelle

Der Spezialfilter (`AdvancedFilter` ) verfügt über die vier obenstehenden Parameter, die Sie eingeben sollten:

- Action
- CriteriaRange
- CopyToRange
- Unique

Da obere Beispiel kann nun modifiziert werden, so dass die gefilterten Daten in einer neuen Datei stehen:

```
Sub Spezialfilter2()

   Dim xlDateiDaten As Workbook

   Dim xlDateiZiel As Workbook

   Dim xlBlattDaten As Worksheet

   Dim xlBlattZiel As Worksheet

   On Error GoTo ende
```

```
    Set xlDateiDaten = ActiveWorkbook
    Set xlDateiZiel = Application.Workbooks.Add
    Set xlBlattDaten = xlDateiDaten.Worksheets(1)
    Set xlBlattZiel = xlDateiZiel.Worksheets(1)

    xlBlattZiel.Range("A1").Value = "Jahresbeitrag"
    xlBlattZiel.Range("B1").Value = "Ort"
    xlBlattZiel.Range("A2").Value = ">" & _
    InputBox("Ab welchem Jahresbeitrag möchten Sie die Daten filtern?...")
    xlBlattZiel.Range("B3").Value = _
       InputBox("... oder welchen Ort möchten Sie filtern?")
    ' -- die Kriterien werden definiert und in die Zellen geschrieben

     xlBlattDaten.Range("A1").CurrentRegion.AdvancedFilter _
       Action:=xlFilterCopy, _
       CriteriaRange:=xlBlattZiel.Range("A1:B3"), _
       CopyToRange:=xlBlattZiel.Range("A5"), _
       Unique:=False
     ' -- der Filter wird gestartet

     xlBlattZiel.Range("A1:B4").EntireRow.Delete
     ' -- lösche die Zellen, die als Kriterien benutzt wurden
Exit Sub
ende:
   MsgBox "Es trat ein Fehler auf: " & Err.Description
End Sub
```

**Hinweis**

Der Spezialfilter ist ein schnelles und effektives Werkzeug, da Sie mit einem Befehl aus Tausenden von Daten genau die Daten filtern können, die Sie benötigen. Allerdings setzt er ein wenig Übung voraus um die korrekten Kriterien in den richtigen Zellen zu definieren, die dann als Filterkriterien verwendet werden.

### 7.7.3 Teilsummen

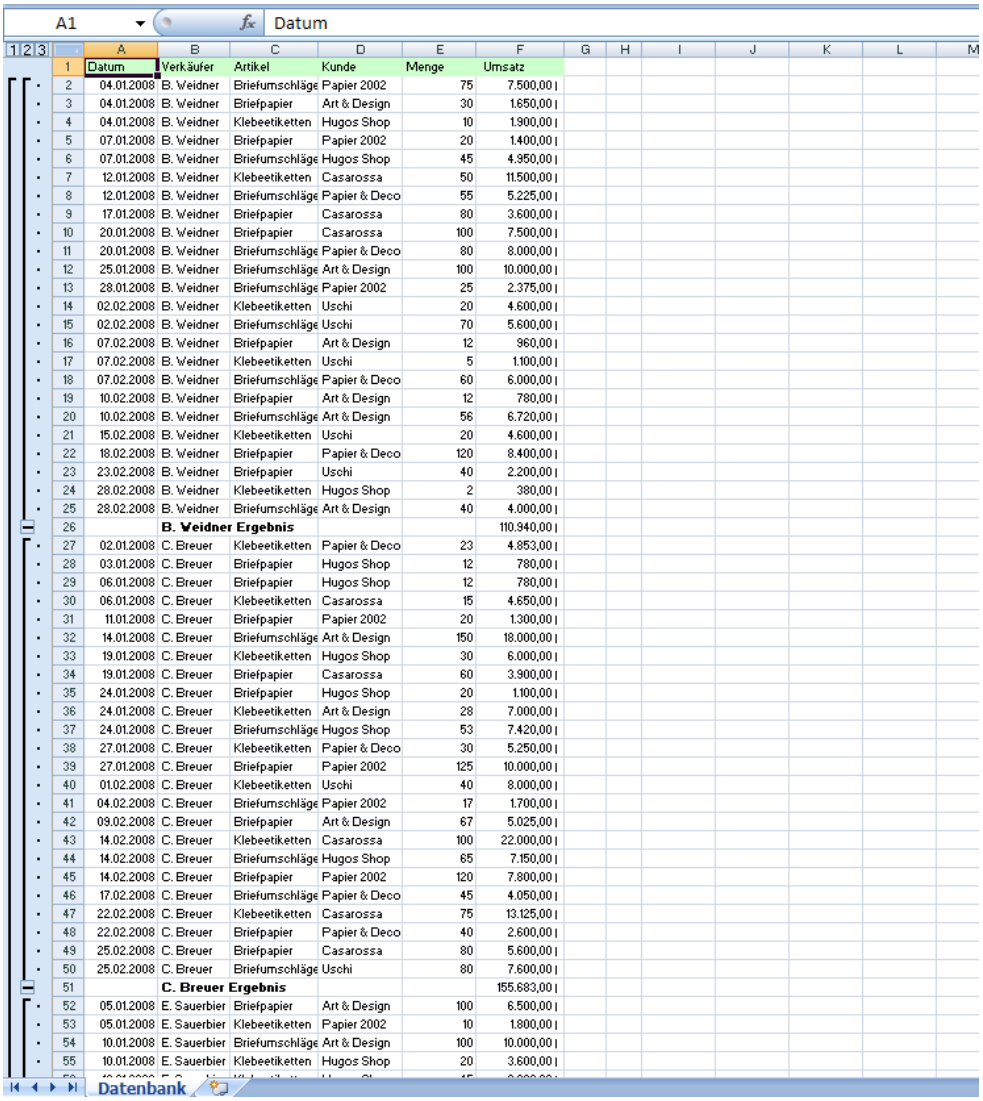
Zugegeben: auch wenn die Teilergebnisse nicht so mächtig sind wie die Pivottabellen, können Sie sie dennoch für kleinere Zusammenfassungen und Analysen verwenden.

Im ersten Schritt sollten Sie die Daten sortieren, damit gleiche, untereinander stehende Daten zusammengefasst werden können:

```
Sub Teilergebnisse()
   Dim xlDatei As Workbook
   Dim xlBlatt As Worksheet

   Set xlDatei = ActiveWorkbook
   Set xlBlatt = xlDatei.Worksheets(1)

   xlBlatt.Sort.SortFields.Clear
   xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("B11"), _
      SortOn:=xlSortOnValues, Order:=xlAscending, _
    DataOption:=xlSortNormal
   With xlBlatt.Sort
      .SetRange xlBlatt.Range("A1").CurrentRegion
      .Header = xlYes
      .MatchCase = False
      .Orientation = xlTopToBottom
      .Apply
   End With
```

Die Methode `Subtotal` wird nun auf den Bereich (oder eine Zelle) angewendet:

```
xlBlatt.Range("A1").CurrentRegion.Subtotal GroupBy:=2, _
   Function:=xlSum, TotalList:=Array(6), _
   Replace:=True, PageBreaks:=False, SummaryBelowData:=True
```



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Datum | Verkäufer | Artikel | Kunde | Menge | Umsatz |
| 2 | 04.01.2008 | B. Weidner | Briefumschläge | Papier 2002 | 75 | 7.500,00 |
| 3 | 04.01.2008 | B. Weidner | Briefpapier | Art & Design | 30 | 1.650,00 |
| 4 | 04.01.2008 | B. Weidner | Klebeetiketten | Hugos Shop | 10 | 1.900,00 |
| 5 | 07.01.2008 | B. Weidner | Briefpapier | Papier 2002 | 20 | 1.400,00 |
| 6 | 07.01.2008 | B. Weidner | Briefumschläge | Hugos Shop | 45 | 4.950,00 |
| 7 | 12.01.2008 | B. Weidner | Klebeetiketten | Casarossa | 50 | 11.500,00 |
| 8 | 12.01.2008 | B. Weidner | Briefumschläge | Papier & Deco | 55 | 5.225,00 |
| 9 | 17.01.2008 | B. Weidner | Briefpapier | Casarossa | 80 | 3.600,00 |
| 10 | 20.01.2008 | B. Weidner | Briefpapier | Casarossa | 100 | 7.500,00 |
| 11 | 20.01.2008 | B. Weidner | Briefumschläge | Papier & Deco | 80 | 8.000,00 |
| 12 | 25.01.2008 | B. Weidner | Briefumschläge | Art & Design | 100 | 10.000,00 |
| 13 | 28.01.2008 | B. Weidner | Briefumschläge | Papier 2002 | 25 | 2.375,00 |
| 14 | 02.02.2008 | B. Weidner | Klebeetiketten | Uschi | 20 | 4.600,00 |
| 15 | 02.02.2008 | B. Weidner | Briefumschläge | Uschi | 70 | 5.600,00 |
| 16 | 07.02.2008 | B. Weidner | Briefpapier | Art & Design | 12 | 960,00 |
| 17 | 07.02.2008 | B. Weidner | Klebeetiketten | Uschi | 5 | 1.100,00 |
| 18 | 07.02.2008 | B. Weidner | Briefumschläge | Papier & Deco | 60 | 6.000,00 |
| 19 | 10.02.2008 | B. Weidner | Briefpapier | Art & Design | 12 | 780,00 |
| 20 | 10.02.2008 | B. Weidner | Briefumschläge | Art & Design | 56 | 6.720,00 |
| 21 | 15.02.2008 | B. Weidner | Klebeetiketten | Uschi | 20 | 4.600,00 |
| 22 | 18.02.2008 | B. Weidner | Briefpapier | Papier & Deco | 120 | 8.400,00 |
| 23 | 23.02.2008 | B. Weidner | Briefpapier | Uschi | 40 | 2.200,00 |
| 24 | 28.02.2008 | B. Weidner | Klebeetiketten | Hugos Shop | 2 | 380,00 |
| 25 | 28.02.2008 | B. Weidner | Briefumschläge | Art & Design | 40 | 4.000,00 |
| 26 | | **B. Weidner Ergebnis** | | | | 110.940,00 |
| 27 | 02.01.2008 | C. Breuer | Klebeetiketten | Papier & Deco | 23 | 4.853,00 |
| 28 | 03.01.2008 | C. Breuer | Briefpapier | Hugos Shop | 12 | 780,00 |
| 29 | 06.01.2008 | C. Breuer | Briefpapier | Hugos Shop | 12 | 780,00 |
| 30 | 06.01.2008 | C. Breuer | Klebeetiketten | Casarossa | 15 | 4.650,00 |
| 31 | 11.01.2008 | C. Breuer | Briefpapier | Papier 2002 | 20 | 1.300,00 |
| 32 | 14.01.2008 | C. Breuer | Briefumschläge | Art & Design | 150 | 18.000,00 |
| 33 | 19.01.2008 | C. Breuer | Klebeetiketten | Hugos Shop | 30 | 6.000,00 |
| 34 | 19.01.2008 | C. Breuer | Briefpapier | Casarossa | 60 | 3.900,00 |
| 35 | 24.01.2008 | C. Breuer | Briefpapier | Hugos Shop | 20 | 1.100,00 |
| 36 | 24.01.2008 | C. Breuer | Klebeetiketten | Art & Design | 28 | 7.000,00 |
| 37 | 24.01.2008 | C. Breuer | Briefumschläge | Hugos Shop | 53 | 7.420,00 |
| 38 | 27.01.2008 | C. Breuer | Klebeetiketten | Papier & Deco | 30 | 5.250,00 |
| 39 | 27.01.2008 | C. Breuer | Briefpapier | Papier 2002 | 125 | 10.000,00 |
| 40 | 01.02.2008 | C. Breuer | Klebeetiketten | Uschi | 40 | 8.000,00 |
| 41 | 04.02.2008 | C. Breuer | Briefumschläge | Papier 2002 | 17 | 1.700,00 |
| 42 | 09.02.2008 | C. Breuer | Briefpapier | Art & Design | 67 | 5.025,00 |
| 43 | 14.02.2008 | C. Breuer | Klebeetiketten | Casarossa | 100 | 22.000,00 |
| 44 | 14.02.2008 | C. Breuer | Briefumschläge | Hugos Shop | 65 | 7.150,00 |
| 45 | 14.02.2008 | C. Breuer | Briefpapier | Papier 2002 | 120 | 7.800,00 |
| 46 | 17.02.2008 | C. Breuer | Briefumschläge | Papier & Deco | 45 | 4.050,00 |
| 47 | 22.02.2008 | C. Breuer | Klebeetiketten | Casarossa | 75 | 13.125,00 |
| 48 | 22.02.2008 | C. Breuer | Briefpapier | Papier & Deco | 40 | 2.600,00 |
| 49 | 25.02.2008 | C. Breuer | Briefpapier | Casarossa | 80 | 5.600,00 |
| 50 | 25.02.2008 | C. Breuer | Briefumschläge | Uschi | 80 | 7.600,00 |
| 51 | | **C. Breuer Ergebnis** | | | | 155.683,00 |
| 52 | 05.01.2008 | E. Sauerbier | Briefpapier | Art & Design | 100 | 6.500,00 |
| 53 | 05.01.2008 | E. Sauerbier | Klebeetiketten | Papier 2002 | 10 | 1.800,00 |
| 54 | 10.01.2008 | E. Sauerbier | Briefumschläge | Art & Design | 100 | 10.000,00 |
| 55 | 10.01.2008 | E. Sauerbier | Klebeetiketten | Hugos Shop | 20 | 3.600,00 |

**Abbildung 7.16** Die Teilergebnisse

**Tabelle 7.22 Die Methode Subtotal verwendet folgende Parameter:**

| Parameter | Beschreibung |
|---|---|
| GroupBy | Das Feld, nach dem gruppiert werden soll, als ganzzahliger, bei 1 beginnender Versatz (Offset) |
| Function | Die Subtotal-Funktionen: xlAverage, xlCount, xlCountNums, xlMax, xlMin, xlProduct, xlStDev, xlStDevP, xlSum, xlUnknown, xlVar, xlVarP, |
| TotalList | Ein Array mit bei 1 beginnenden Feldversätzen. Diese geben die Felder an, denen die Teilergebnisse hinzugefügt werden |
| Replace | Mit True werden bestehende Teilergebnisse ersetzt. Der Standardwert ist True. |
| PageBreaks | Mit True wird nach jeder Gruppe ein Seitenumbruch eingefügt. Der Standardwert ist False. |
| SummaryBelowData | Platziert die Zusammenfassungsdaten relativ zum Teilergebnis. |

### 7.7.4 Text in Spalten trennen

Erstaunlicherweise werden in vielen Zeitschriften und Bücher Assistenten angeboten, die importierte Daten in ihre Bestandteile trennen. Dies ist umso erstaunlicher, weil Excel seit vielen Versionen einen Assistenten zur Verfügung stellt, der dies erledigt.

**Beispiel**

Angenommen, Sie öffnen regelmäßig Dateien in Excel, die aus einem anderen System exportiert werden,beispielsweise aus SAP, DATEV, oder einem anderen System. Wenn dort nun Informationen in einer Zelle zusammengefasst sind, die eigentlich getrennt werden müssen, können Sie die Methode TextToColumns verwenden:

Im ersten Schritt fügen Sie eine Spalte ein:

```
Sub DatenTextInSpalten()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet


   Set xlDatei = ActiveWorkbook

   Set xlBlatt = xlDatei.Worksheets(1)

   xlBlatt.Range("E1").EntireColumn.Insert
```

Nun werden die Daten getrennt:

```
   xlBlatt.Range("D1").EntireColumn.TextToColumns _

      Destination:=xlBlatt.Range("D1"), DataType:=xlDelimited, _

        TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=True, _

        Tab:=False, Semicolon:=False, Comma:=False, Space:=True, _

        Other:=False, FieldInfo:=Array(Array(1, 1), Array(2, 1)), _

        TrailingMinusNumbers:=True

End Sub
```

| B | C | D | E | |
|---|---|---|---|---|
| Geschlecht | Titel | Name | Straße | P |
| 10 | | Helma Unangst | Schuetzenstr.28 | |
| 20 | Dr. | Horst Junghaus | Havixbecker Str. 62 | |
| 10 | | Liselotte Kaeser | Schwetzingerstr.37 | |
| 20 | | Helmut Illemann | Hansaallee 177 | |
| 20 | | Andreas Bauer | Am Bonneshof 35 | |
| 10 | | Helga Scherer | Schuetzenstr.29 | |
| 20 | | Franz Schaeffner | Stamitzstr.1 | |
| 20 | | Andreas Bagusch | Am Bonneshof 35 | |
| 20 | | Kurt Grimminger | Schwarzwaldstr.67 | |
| 20 | | Rolf Bruder | Schwarzwaldstr.50 | |
| 20 | | Ernst Ritter | Am Schwimmbad 4 | |
| 20 | | Gerd Vogelmann | Schwetzinger Str. 5 | |
| 20 | | Otti Kober | Schwetzinger Str.1 | |
| 10 | | Emmy Boeckh | Stamitzstr. 8 | |
| 20 | | Georg Schaefer | Schwetzinger Str. 1 | |
| 20 | | Wolfgang Kiefer | Schwarzwaldstr.18 | |

**Abbildung 7.17** Ein schönes Beispiel von zu trennenden Daten

**Tabelle 7.23 Die Methode TextToColumns stellt folgende Parameter zur Verfügung:**

| Parameter | Bedeutung |
|---|---|
| Destination | Ein Range-Objekt, das angibt, an welcher Stelle das Ergebnis ausgegeben werden soll |
| DataType | Das Format des Texts, der in Spalten aufgeteilt werden soll. |
| TextQualifier | Gibt an, ob als Texterkennungszeichen einfache, doppelte oder keine Anführungszeichen verwendet werden sollen |
| ConsecutiveDelimiter | True, wenn Microsoft Excel aufeinanderfolgende Trennzeichen als ein Trennzeichen interpretieren soll. Der Standardwert ist False. |
| Tab | das Tabstoppzeichen wird als Trennzeichen verwendet |
| Semicolon | das Semikolon wird als Trennzeichen verwendet |
| Comma | das Komma wird als Trennzeichen verwendet |
| Space | das Leerzeichen wird als Trennzeichen verwendet |
| Other | ein anderes Zeichen wird als Trennzeichen verwendet. Es wird im Argument OtherChar festgelegt |
| OtherChar | Das Trennzeichen, das benutzt wird, wenn Other auf True festgelegt ist |
| FieldInfo | Ein Array mit Informationen zur Analyse der einzelnen Datenspalten |
| DecimalSeparator | Das Dezimaltrennzeichen, das von Microsoft Excel beim Erkennen von Zahlen verwendet wird. Als Standardeinstellung wird die Systemeinstellung verwendet. |
| ThousandsSeparator | Das 1.000er-Trennzeichen, das von Excel beim Erkennen von Zahlen verwendet wird. Als Standardeinstellung wird die Systemeinstellung verwendet. |
| TrailingMinusNumbers | Zahlen, denen ein Minuszeichen vorangestellt ist. |

## 7.8 Duplikate entfernen?

Ein häufiges Problem, mit dem sich auch Datenbankprogrammierer herumschlagen müssen, ist das mehrfache Vorkommen von Datensätzen. Es gibt viele Beispiele hierfür:

- Mehrere Sekretärinnen oder Buchhalter erfassen Daten. Dabei kann es vorkommen, dass ein Posten, ein Auftrag, eine Rechnung etc. mehrfach eingegeben wurde
- Zwei Außendienstmitarbeiter erfassen getrennt voneinander Daten. Dabei sollen die Daten „zusammengefasst“, das heißt identische Daten gelöscht oder addiert werden.
- Aus zwei verschiedenen Listen werden Informationen importiert, Nun kann es passieren, dass bestimmte eindeutige Informationen, beispielsweise E-Mailadressen nur einmal auftauchen sollen.

Sicherlich finden sich noch weitere Beispiele. Um solche Duplikate zu finden und zu löschen stehen mehrere Varianten zur Verfügung.

**Hinweis**
Wenn sich die Daten in zwei getrennten Dateien befinden, dann können Sie sie in eine Datei untereinander kopieren. Achten Sie darauf, dass die Spalten korrekt untereinander stehen. Falls dies nicht sichergestellt ist, müssen Sie es überprüfen.

Der Einfachheit halber stehen die Daten bereits untereinander in einer Datei. Der Schlüsselwert, der mehrmals vorkommen kann, steht in Spalte A.

### 7.8.1 Sortieren und Zeilen löschen

Die erste Methode besteht darin die Liste zu sortieren, die sortierte Spalte zu durchlaufen und Duplikate zu löschen

```
Sub DuplikateLöschen1()
   Dim xlDatei As Workbook
   Dim xlBlatt As Worksheet
   Dim i As Long

   Set xlDatei = ActiveWorkbook
   Set xlBlatt = xlDatei.Worksheets(1)

   xlBlatt.Sort.SortFields.Clear
   xlBlatt.Sort.SortFields.Add Key:=xlBlatt.Range("A1"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
    With xlBlatt.Sort
        .SetRange xlBlatt.Range("A1").CurrentRegion
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .Apply  ' -- sortiere
    End With

    For i = xlBlatt.Range("A1").CurrentRegion.Rows.Count To 2 Step -1
      If xlBlatt.Cells(i, 1).Value = xlBlatt.Cells(i - 1, 1).Value Then
         xlBlatt.Cells(i, 1).EntireRow.Delete
      End If
    Next
    ' -- lösche die Zeile, falls die Nummer identisch ist
    ' -- mit der darüberstehenden Nummer
End Sub
```

**Hinweis**

Diese Methode eignet sich zwar sehr gut für kleinere Listen – für Dateien, in denen sich jedoch mehrere Zehntausend oder gar Hunderttausend Daten stehen, ist sie sicherlich nicht effektiv.

### 7.8.2 Filtern ohne Duplikate

Da der Spezialfilter die Möglichkeit beinhaltet ohne Duplikate zu filtern, können Sie ihn für das oben beschriebene Problem auch verwenden. Wenn Sie die Daten „über sich selbst“ filtern, werden die Duplikate lediglich ausgeblendet. Da sie allerdings gelöscht werden sollen, sollten die neuen Daten ohne Duplikate an eine andere

Stelle, beispielsweise vier Zeilen unterhalb des aktuellen Bereichs eingefügt werden. Anschließend kann der Bereich gelöscht werden:

```
Sub DuplikateLöschen2()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet

   Dim xlBereich As Range


   Set xlDatei = ActiveWorkbook

   Set xlBlatt = xlDatei.Worksheets(1)

   Set xlBereich = xlBlatt.Range("A1").CurrentRegion


   xlBereich.AdvancedFilter _

      Action:=xlFilterCopy, _

      CopyToRange:=xlBlatt.Cells(xlBereich.Rows.Count + 5, 1), _

      Unique:=True

   xlBereich.EntireRow.Delete

End Sub
```

### 7.8.3 Anzahl berechnen, filtern und löschen

Vielleicht ist die folgende Variante etwas umständlich, aber sie kann sicherlich als „Werkzeugkasten“ für weitere Probleme verwendet werden, die eine ähnlich gelagerte Aufgabenstellung vorweisen.

Im ersten Schritt wird nach der ersten Spalte sortiert, wie bereits oben gezeigt:

```
Sub DuplikateLöschen3()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet

   Dim xlBereich As Range

   Dim intSpalten As Integer


   Set xlDatei = ActiveWorkbook

   Set xlBlatt = xlDatei.Worksheets(1)

   Set xlBereich = xlBlatt.Range("A1").CurrentRegion

   intSpalten = xlBereich.Columns.Count


   xlBlatt.Sort.SortFields.Clear

   xlBlatt.Sort.SortFields.Add Key:=Range("A1"), _

      SortOn:=xlSortOnValues, Order:=xlAscending, _
```

```
    DataOption:=xlSortNormal
    With xlBlatt.Sort
       .SetRange xlBereich
       .Header = xlNo
       .MatchCase = False
       .Orientation = xlTopToBottom
       .SortMethod = xlPinYin
       .Apply
    End With
```

Anschließend wird direkt hinter die Tabelle die Formel eingefügt:

```
=WENN(UND(ZÄHLENWENN(A:A;A1)>1;A1=A2);"lösche";"")
```

Sie lautet als VBA-Befehl folgendermaßen:

```
    xlBlatt.Cells(1, intSpalten + 1).FormulaR1C1 = _
       "=IF(AND(COUNTIF(C[-" & _
       intSpalten & "],RC[-" & intSpalten & _
       "])>1,RC[-" & intSpalten & "]=R[1]C[-" & _
       intSpalten & "]),""lösche"","""")"
```

Diese Formel wird nun über alle Zeilen heruntergezogen:

```
    xlBlatt.Cells(1, intSpalten + 1).AutoFill _
       Destination:=xlBlatt.Range(xlBlatt.Cells(1, intSpalten + 1), _
       xlBlatt.Cells(xlBereich.Rows.Count, intSpalten + 1))
```

Nun werden die nicht-leeren Zellen gefiltert:

```
    xlBlatt.Range("A1").CurrentRegion.AutoFilter Field:=7, _
       Criteria1:="<>"
```

und gelöscht:

```
    xlBlatt.Range("A1").CurrentRegion.EntireRow.Delete
```

Anschließend wird die Spalte, in der die nun überflüssige Formel steht, gelöscht:

```
    xlBlatt.Columns(intSpalten + 1).EntireColumn.Delete
End Sub
```

### 7.8.4 Pivottabelle

Sie können ebenso eine Pivottabelle erstellen. In ihr wird die Anzahl der Vorkommen bestimmt. Nun kann absteigend nach der Anzahl sortiert warden und die entsprechenden Einträge gelöscht warden.

### 7.8.5 RemoveDuplicates

Am schnellsten lässt sich dieses Problem sicherlich in Excel 2007 lösen. Dort seht Ihnen die Methode `RemoveDuplicates` zur Verfügung:

```
Sub DuplikateLöschen4()

   Dim xlDatei As Workbook

   Dim xlBlatt As Worksheet

   Dim xlBereich As Range


   Set xlDatei = ActiveWorkbook

   Set xlBlatt = xlDatei.Worksheets(1)

   Set xlBereich = xlBlatt.Range("A1").CurrentRegion


    xlBereich.RemoveDuplicates Columns:=1, Header:=xlNo

End Sub
```

### 7.8.6 Bedingte Formatierung

Excel 2007 stellt in der bedingten Formatierung die Option DupeUnique = xlDuplicate zur Verfügung. Damit können alle Duplikate farblich gekennzeichnet warden. Da Excel 2007 nach Farben sortieren kann, könnte man die beiden Werkzeuge zusammen fassen:

- schalte die bedingte Formatierung für Duplikate ein
- sortiere die farblich gekennzeichneten zellen nach oben

Im Code wird die Spalte, in welcher sich die Duplikate befinden in einer Konstanten ausgelagert:

```
Sub DuplikateHervorheben()


   Const DUPLIKATENSPALTE As Integer = 11

   Dim xlBlatt As Worksheet


   Set xlBlatt = ActiveSheet
```

Falls noch bedingte Formatierungen vorhanden sind, warden sie ausgeschaltet:

```
   xlBlatt.Cells.FormatConditions.Delete
```

Die bedingte Formatierung wird aktiviert:

```
   With xlBlatt.Columns(DUPLIKATENSPALTE)
      .FormatConditions.AddUniqueValues
      .FormatConditions(1).DupeUnique = xlDuplicate
      .FormatConditions(1).Font.Color = vbWhite
      .FormatConditions(1).Interior.Color = vbRed
      .FormatConditions(1).StopIfTrue = False
    End With
```

Und anschließend nach den Farbwerten sortiert:

```
With xlBlatt.Sort
   .SortFields.Clear
   .SortFields.Add _
      Key:=xlBlatt.Columns(DUPLIKATENSPALTE), _
      SortOn:=xlSortOnCellColor, _
      Order:=xlDescending, _
      DataOption:=xlSortNormal
   .SetRange xlBlatt.Range("A1").CurrentRegion
   .Header = xlYes
   .MatchCase = False
   .Orientation = xlTopToBottom
   .SortMethod = xlPinYin
   .Apply
End With
```

Das Ergebnis sieht folgendermaßen aus:

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Kundennum | Geschlecht | Titel | Name | Straße | Plz | Ort | Jahresbeitrag | Geburtsdatum | Eintrittsdatum | E-Mail | Konto-Nr |
| 2 | 150 | 20 | | Heinz Knapp | Forlenweg 3 | 68804 | Altlussheim | 148 | 09.11.1955 | 04.11.1973 | Heinz@google.de | 10837 |
| 3 | 519 | 20 | | Frank Geberzahn | Denningerstrasse | 81927 | Muenchen | 148 | 16.12.1971 | 11.12.1989 | Frank@hotmail.com | 47130 |
| 4 | 800 | 20 | | Heinz Hiemenz | Wingertsau 9 | 68259 | Mannheim | 148 | 04.12.1944 | 30.11.1962 | Heinz@web.de | |
| 5 | 956 | 20 | | Heinrich Duschner | Wingertsau 4 | 68259 | Mannheim | 148 | 27.08.1942 | 22.08.1960 | Heinrich@google.de | 32950 |
| 6 | 1282 | 20 | | Heinz Huelsemann | Hagenauer Str. 53 | 65203 | Wiesbaden | 148 | 15.06.1956 | 11.06.1974 | Heinz@web.de | |
| 7 | 1284 | 20 | | Rolf Brandes | Eichelsheimer Str. | 68163 | Mannheim | 120 | 03.03.1963 | 01.02.1983 | RolfB@excite.de | |
| 8 | 1401 | 20 | | Hans Klein | Lindenstr. 7 | 68309 | Mannheim | 148 | 24.06.1939 | 19.06.1957 | Hans@web.de | |
| 9 | 1441 | 20 | | Heinz Kloetzer | Andreas-Hofer-Str. | 68259 | Mannheim | 136 | 02.04.1946 | 28.03.1964 | Heinz@web.de | |
| 10 | 1463 | 20 | | Peter Schaefer | Postfach 11 20 | 66688 | Mettlach | 148 | 08.09.1958 | 03.09.1976 | Peter@pop.de | 74303 |
| 11 | 1507 | 20 | | Paul Roth | Paulinenstr. 15 | 65189 | Wiesbaden | 136 | 05.09.1987 | 31.08.2005 | Paul@t-online.de | 2187 |
| 12 | 1687 | 20 | | Heinz Huber | Gutleutstr. 32 | 60329 | Frankfurt a. M. | 130 | 06.09.1970 | 01.09.1988 | Heinz@google.de | |
| 13 | 1800 | 20 | | Robert Schneider | Reeser Landstr. 21 | 46487 | Wesel | 148 | 27.11.1968 | 23.11.1986 | RobertS@strato.de | 758419 |
| 14 | 1820 | 10 | | Ingrid Bauer | Relaisstr.116 | 68219 | Mannheim | 148 | 12.02.1961 | 08.02.1979 | Bauer@gmx.de | |
| 15 | 1874 | 20 | | Ludwig Mueller | Dronkestr.7 | 36039 | Fulda | 148 | 17.04.1958 | 12.04.1976 | LudwigM@everymail.de | |
| 16 | 1925 | 20 | | Peter Wasser | Koernerstr.13 | 68775 | Ketsch | 148 | 20.08.1958 | 01.09.1981 | Peter@pop.de | |
| 17 | 1951 | 20 | | Ulrich Vetter | Unter der Au 4 | 74889 | Sinsheim-Steinf | 148 | 11.01.1959 | 06.01.1977 | Ulrich@dnv.de | |
| 18 | 1980 | 20 | | Hartmut Bauer | Obere Riedstr 4 | 68309 | Mannheim | 136 | 30.12.1946 | 25.12.1964 | Bauer@gmx.de | 9691690 |
| 19 | 2009 | 20 | | Walter Kocher | Rathausstr.34 | 82008 | Unterhaching | 148 | 23.04.1926 | 01.06.1985 | Walter@sofort-mail.de | |
| 20 | 2178 | 20 | | Klaus Lindholm | Kekulestrasse 30 | 44579 | Castrop-Rauxel | 148 | 11.06.1982 | 06.06.2000 | KlausL@abacho.de | |
| 21 | 2193 | 20 | | Michael Ricker | Lemaitrestr.7 | 68309 | Mannheim | 148 | 14.11.1947 | 09.11.1965 | Michael@abacho.de | |
| 22 | 2281 | 20 | Dr. | Frieda Trautwein | Chamissostr.6 | 68167 | Mannheim | 148 | 06.04.1961 | 02.04.1979 | Trautwein@dnv.de | |
| 23 | 2307 | 20 | | Werner Wilhelm | Wiesenau 36 | 60323 | Frankfurt a. M. | 148 | 14.10.1971 | 09.10.1989 | Wilhelm@wreckedmail.de | |
| 24 | 2318 | 20 | | Rolf Becker | Feldbergstr.62 | 68163 | Mannheim | 148 | 27.03.1962 | 01.02.1983 | RolfB@excite.de | 70187 |
| 25 | 2385 | 20 | | Gerhard Karle | Waldparkdamm 2 | 68163 | Mannheim | 148 | 12.09.1968 | 07.09.1984 | Karle@wreckedmail.de | 36560 |
| 26 | 2501 | 20 | | Wolfgang Zimmerm | Zum Frenser Feld | 50127 | Bergheim | 148 | 09.11.1965 | 05.11.1983 | Wolfgang@redSeven.de | 18880 |
| 27 | 2554 | 20 | | Harald Anton | Wormser Str. 33 | 68309 | Mannheim | 148 | 15.11.1974 | 10.11.1992 | Harald@abacho.de | 1697 |
| 28 | 2725 | 20 | | Wolfgang Matriciani | Richthofenstr.10 | 68723 | Oftersheim | 148 | 23.11.1951 | 01.06.1988 | Wolfgang@hotmail.com | |
| 29 | 2772 | 10 | | Irmgard Kirchgessr | Beim Johannkirchh | 68219 | Mannheim | 148 | 03.02.1952 | 01.07.1970 | Irmgard@Nixneues.de | |
| 30 | 2834 | 20 | | Klaus Loon | Kirchweg 28 | 51143 | Koeln | 148 | 27.09.1984 | 23.09.2002 | Klaus@wreckedmail.de | |
| 31 | 3030 | 20 | | Werner Wieckmann | Westendstr. 199 | 80686 | Muenchen | 148 | 17.10.1972 | 13.10.1990 | Werner@pop.de | |
| 32 | 3100 | 20 | | Frank Gier | Domstrasse 20 | 50668 | Koeln | 148 | 10.04.1955 | 05.04.1973 | Frank@hotmail.com | |
| 33 | 3279 | 20 | | Werner Werner | Stettiner Str.4 | 68549 | Ilvesheim | 148 | 04.02.1959 | 01.01.1987 | Werner@pop.de | |
| 34 | 3434 | 20 | | Wolfgang Langer | Friedensstr.33 | 67059 | Ludwigshafen | 148 | 24.06.1947 | 01.05.1988 | Wolfgang@hotmail.com | |
| 35 | 3646 | 20 | | Kurt Schneider | Kastellweg 17 | 69120 | Heidelberg | 148 | 13.04.1939 | 08.04.1957 | Schneider@google.de | |
| 36 | 3704 | 20 | | Wolfgang Wolff | Willy-Brandt-Allee | 45891 | Gelsenkirchen | 148 | 03.10.1977 | 29.09.1995 | Wolfgang@redSeven.de | |
| 37 | 3747 | 20 | | Volker Weber | Waddenhauser Str. | 32791 | Lage/Lippe | 148 | 31.07.1983 | 26.07.2001 | VolkerW@Nixneues.de | |
| 38 | 3824 | 20 | | Karl Englert | Robert-Koch-Str.6 | 68723 | Oftersheim | 124 | 03.10.1958 | 28.09.1976 | KarlE@wreckedmail.de | |
| 39 | 3938 | 20 | | Hans-Georg Vetter | Roxheimer Str. 8 | 68219 | Mannheim | 136 | 19.07.1945 | 15.07.1963 | Hans-Georg@sofort-mail.de | 1311440 |
| 40 | 4162 | 20 | | Werner Wiessmeie | Widenmayerstrass | 80538 | Muenchen | 148 | 08.12.1976 | 04.12.1994 | Werner@Unicum.de | |

**Abbildung 7.18** Die Duplikate werden farblich gekennzeichnet.

### Noch mehr Varianten?

Es ist nun Ihrer Phantasie überlassen, weiltere Varainten für die Lösung dieses Problems zu finden. Eine Technik wäre beispielsweise die Erstellung einer Pivottabelle, in der in der Spalte sämtliche Daten (einmal) aufgelistet sind. Anschließend kann – beispielsweise mit der Funktion SVERWEIS die Information der alten Tabelle geholt werden. Diese kann nun im letzten Schritt gelöscht werden.

## 7.9 Fazit

Excel stellt für eine Zelle, für Zellbereiche, mehrere Zellbereiche, Spalten und Zeilen das gleiche Objekt Range zur Verfügung. Das erleichtert zwar die Programmierarbeit, andererseits muss zum Teile überprüft warden, ob der Bereich mehrere Zeilen hat, wie viele Zeilen er hat, ob er Leerzeilen hat, ob der Bereich zusammenhängt und so weiter. Und auf diese Zelle oder diesen Bereich kann zugegriffen warden – man kann Daten auslesen und hineinschreiben, per Programmierung Formeln setzen, Datenmengen sortieren, filtern, formatieren, löschen und so weiter.

Das alles macht dan Sinn, wenn solche Aufgaben in einem Workflow zu sehen sind. Jeden Tag kommt beispielsweise von SAP, vom Großrechner oder von einem anderen System eine Excel-Datei, die auf die immer gleiche Art bearbeitet warden muss. Hierfür hilft Ihnen der Makrorekorder, mit dem Sie an die entsprechenden Befehle kommen, aber auch die Überprüfung, wie viele Zeilen und Spalten enthalten die Daten, in welcher Form kommen die Daten, sind sie korrekt und so weiter.

# 8 Symbolleisten, Menüleisten und Tastenkombinationen

Bis Excel 2003 gab es Symbolleisten und Menüleisten, die per Programmierung angepasst werden konnten. Zwar stehen sie als Objekte in der aktuellen Excel-Version 2007 nicht mehr zur Verfügung (erstaunlicherweise aber noch als Objekte im Objektmodell), sollen aber dennoch an dieser Stelle beschrieben werden, da möglicherweise nicht jeder Leser schon auf Office 2007 umgestiegen ist.

**Hinweis**

Erstaunlicherweise besitzen in der aktuellen Version MS Visio und MS Project keine Ribbons. Das in diesem Kapitel beschrieben Objektmodell findet in diesen Applikationen noch seine Verwendung. Auch in Outlook wurden die Ribbons nur „zur Hälfte“ umgesetzt.

## 8.1 Symbolleisten

Das Objekt, mit dem auf alle Symbolleisten zugegriffen werden kann, heißt `Commandbars`. Es handelt sich dabei um eine Sammlung.

**Beispiel**

Folgendes Makro durchläuft alle Symbolleisten und listet sie namentlich auf:

```
Sub AlleSymbolLeisten()

   Dim i As Integer

   Dim strSymbLeiste As String

   For i = 1 To Application.CommandBars.Count

      strSymbLeiste = strSymbLeiste & ", " & CommandBars(i).Name

   Next

   MsgBox strSymbLeiste

End Sub
```

Jede einzelne Symbolleiste hat eine Reihe Eigenschaften. Mit `Visible` kann überprüft werden, ob die Symbolleiste sichtbar ist:

```
Sub AlleSichtbarenSymbolLeisten()

   Dim i As Integer

   Dim strSymbLeiste As String

   For i = 1 To Application.CommandBars.Count
```

```
        If CommandBars(i).Visible = True Then
           strSymbLeiste = strSymbLeiste & ", " & _
              CommandBars(i).Name
        End If
     Next
     MsgBox strSymbLeiste
  End Sub
```

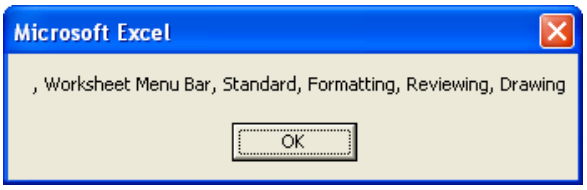



**Abbildung 8.1** Alle sichtbaren Symbolleisten

Daraus wird erkennbar, dass auch die Menüleiste eine Sonderform der Symbolleiste ist. Sie trägt den Namen „Worksheet Menu Bar“. Mit VBA können Sie auf die Symbolleisten über deren englische Bezeichnungen zugreifen.

**Achtung**
Beachten Sie, dass Sie beim Zugriff das Parent-Objekt Application benötigen. Der Befehl MsgBox CommandBars("Drawing").BuiltIn liefert einen Fehler. Korrekt arbeitet hingegen:

```
MsgBox Application.CommandBars("Drawing").BuiltIn
```

Jede Symbolleiste hat einen `Type`: Die Menüleiste ist vom Typ `msoBarTypeMenuBar`, die übrigen besitzen die Eigenschaft `msoBarTypeNormal`. Daneben steht noch der Typ `msoBarTypePopUp` zur Verfügung.

Jede Symbolleiste hat Symbole (Controls). Auch sie können durchlaufen werden:

```
Sub AlleSymbole()
   Dim intZähler As Integer
   Dim strSymbole As String
   With Application.CommandBars("Formatting")
      For intZähler = 1 To .Controls.Count
         strSymbole = strSymbole & vbCr & _
            .Controls(intZähler).Caption
      Next
   End With
MsgBox strSymbole, vbInformation, "Menü Format"
End Sub
```

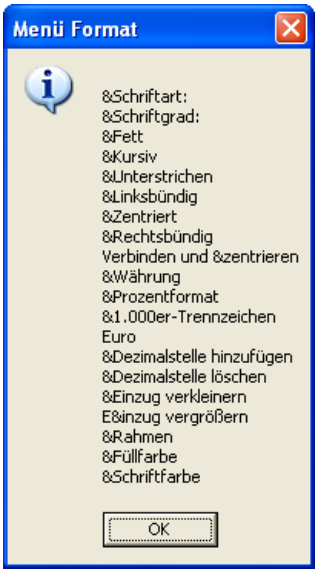



**Abbildung 8.2** Die Symbole der Symbolleiste „Format“

Analog kann die Menüleiste durchlaufen und alle Menübefehle angezeigt werden:

```
With Application.CommandBars("Worksheet Menu Bar")

[...]
```

Auch die Symbole haben verschiedene Präfigurationen. Sie können über die Eigenschaft „Type“ abgefragt werden:

```
strSymbole = strSymbole & vbCr & .Controls(intZähler).Type _

      & vbTab & .Controls(intZähler).Caption
```

**Tabelle 8.1 Folgende Symbolarten stehen Ihnen zur Verfügung:**

| Typ | Wert |
|---|---|
| msoControlActiveX | 22 |
| msoControlButton | 1 |
| msoControlButtonDropDown | 5 |
| msoControlButtonPopup | 12 |
| msoControlComboBox | 4 |
| msoControlCustom | 0 |
| msoControlDropdown | 3 |
| msoControlEdit | 2 |
| msoControlExpandingGrid | 16 |
| msoControlGauge | 19 |
| msoControlGenericDropdown | 8 |
| msoControlGraphicCombo | 20 |
| msoControlGraphicDropdown | 9 |
| msoControlGraphicPopup | 11 |
| msoControlGrid | 18 |
| msoControlLabel | 15 |
| msoControlOCXDropdown | 7 |
| msoControlPane | 21 |

| Typ | Wert |
|---|---|
| msoControlPopup | 10 |
| msoControlSplitButtonMRUPopup | 14 |
| msoControlSplitButtonPopup | 13 |
| msoControlSplitDropdown | 6 |
| msoControlSplitExpandingGrid | 17 |

### 8.1.1 Alle Menüpunkte

Und ebenso können auch alle Menübefehle durchlaufen werden:

```
Sub AlleMenüPunkte()
   Dim intZähler As Integer
   Dim strMenüs As String
   With Application.CommandBars("Worksheet Menu Bar").Controls("E&xtras")
      For intZähler = 1 To .Controls.Count
          strMenüs = strMenüs & vbCr & _
            .Controls(intZähler).Caption
      Next
   End With
MsgBox strMenüs
End Sub
```

Da manche Menüs weitere Untereinträge besitzen, können auch diese mittels einer Schleife durchlaufen werden. Das folgende Beispiel zeigt alle Menüeinträge im Menü Extras | Formelüberwachung an:

```
Sub MenüExtrasSprache()
   Dim intZähler As Integer
   Dim strMenüs As String
   With Application.CommandBars("Worksheet Menu Bar"). _
      Controls("E&xtras").Controls("For&melüberwachung")
         For intZähler = 1 To .Controls.Count
            strMenüs = strMenüs & vbCr & _
               .Controls(intZähler).Caption
         Next
   End With
   MsgBox strMenüs
End Sub
```

Im folgenden Beispiel wird eine vorhandene Symbolleiste am unteren Rand der Anwendung angezeigt:

```
With Application.CommandBars("Drawing")
   .Visible = True
   .Position = msoBarBottom
End With
```

Oder sie wird ein- oder ausgeschaltet:

```
With Application.CommandBars("Drawing")
   .Visible = Not (.Visible)
   .Position = msoBarBottom
End With
```

### 8.1.2 Vorhandene Menüs und Symbolleisten ändern

Vorhandene Symbolleisten und Menüleisten können geschützt werden. Dies erledigt die Eigenschaft `Protection`:

```
Sub SymbolleistenVerändern()
   With Application.CommandBars("Worksheet Menu Bar")
      .Protection = msoBarNoChangeVisible
   End With
End Sub
```

**Tabelle 8.2** Für Protection stehen folgende Konstanten zur Verfügung:

| Konstante | Bedeutung |
|---|---|
| msoBarNoChangeDock | Die Symbolleiste kann aus ihrer Verankerung nicht mehr herausgerissen werden. |
| msoBarNoChangeVisible | Die Symbolleiste taucht weder im Kontextmenü noch im Menü Anpassen auf. |
| msoBarNoCustomize | Symbole können vom Benutzer nicht mehr gelöscht oder hinzugefügt werden. |
| msoBarNoHorizontalDock | Die Symbolleiste kann weder oben noch unten angedocked werden. Diese Eigenschaft steht nur dann zur Verfügung, wenn die Position auf msoBarFloating gesetzt wurde |
| msoBarNoVerticalDock | Die Symbolleiste kann weder links noch rechts angedocked werden. Diese Eigenschaft steht nur dann zur Verfügung, wenn die Position auf msoBarFloating gesetzt wurde |
| msoBarNoMove | Die Symbolleiste kann nicht aus ihrer Verankerung gelöst werden oder – falls sie frei schwebt – vom Anwender verschoben werden. |
| msoBarNoProtection | hebt den Schutz auf |
| msoBarNoResize | Die Symbolleiste kann nicht in ihrer Form verändert werden. |

**Hinweis**

Übrigens schlägt die Eigenschaft Position = msoBarTop fehl, wenn sie zuvor auf Protection = msoBarNoHorizontalDock gesetzt wurde.

### 8.1.3 Neue Menüs und Symbole erzeugen

Wichtiger als das Durchlaufen vorhandener Menüeinträge und Symbole ist das Erzeugen und das Löschen von neuen Menüs und Symbolen.

**Beispiel**

Das folgende Beispiel erzeugt einen neuen Menüpunkt und fügt zwei Menüeinträge hinzu.

```
Sub NeuesMenü()
   Dim mnuMenü As CommandBar
   Dim mnuMenüeintrag As CommandBarControl
   Dim mnuMenüPunkt As CommandBarControl

   Set mnuMenü = Application.CommandBars.ActiveMenuBar
   Set mnuMenüeintrag = mnuMenü.Controls.Add _
      (Type:=10, temporary:=True)
   With mnuMenüeintrag
      .Caption = "&Datenbank"
      .Enabled = True
      .Visible = True
      .DescriptionText = "Datenbank"
   End With
   Set mnuMenüPunkt = mnuMenüeintrag.Controls.Add(Type:=1)
   With mnuMenüPunkt
      .Caption = "&Datenbankexport"
      .Enabled = True
      .Visible = True
      .OnAction = "Datenbankexport"
   End With

   Set mnuMenüPunkt = mnuMenüeintrag.Controls.Add(Type:=1)
   With mnuMenüPunkt
      .Caption = "&Datenbank&import"
      .Enabled = True
      .Visible = True
      .OnAction = "Datenbankimport"
   End With

End Sub
```


**Abbildung 8.3** Ein neuer Menüpunkt wird generiert.

Und schließlich kann der Menüpunkt wieder gelöscht werden:

```
Sub MenüDatenbankLöschen()
   Application.CommandBars.ActiveMenuBar.Controls _
      ("&Datenbank").Delete
End Sub
```

**Beispiel**

Das folgende Beispiel erzeugt eine neue Symbolleiste und fügt ihr drei Symbole hinzu:

```
Sub MeineNeuenSymbis()
   Dim mnuSymbLeiste As CommandBar
   Dim mnuSymbol As CommandBarControl
   Application.CommandBars("Meine_Symbole").Delete
   Set mnuSymbLeiste = Application.CommandBars.Add
   With mnuSymbLeiste
      .Name = "Meine_Symbole"
      .Position = msoBarFloating
      .Visible = True
   End With

   Set mnuSymbol = mnuSymbLeiste.Controls.Add(Type:=1, ID:=480, _
      temporary:=True)
   With mnuSymbol
      .Caption = "Icon1"
      .DescriptionText = "Mein erstes Icon"
      .TooltipText = "Mein erstes Icon"
      .OnAction = "MeinMakro"
      .Visible = True
      .Enabled = True
   End With

   Set mnuSymbol = mnuSymbLeiste.Controls.Add(Type:=1, ID:=1741, _
      temporary:=True)
   With mnuSymbol
      .Caption = "Icon2"
      .DescriptionText = "Mein zweites Icon"
      .TooltipText = "Mein zweites Icon"
      .OnAction = "MeinMakro2"
      .Visible = True
      .Enabled = True
      .BeginGroup = True
   End With

   Set mnuSymbol = mnuSymbLeiste.Controls.Add(Type:=1, ID:=51, _
      temporary:=True)
```

```
    With mnuSymbol
        .Caption = "Icon3"
        .DescriptionText = "Mein drittes Icon"
        .TooltipText = "Mein drittes Icon"
        .OnAction = "MeinMakro3"
        .Visible = True
        .Enabled = True
    End With
End Sub
```


**Abbildung 8.4** Die neue Symbolleiste mit den drei neuen Symbolen

Dieses Beispiel wurde deshalb so ausführlich ausgeschrieben, um einige der Eigenschaften (`Visible`, `Enabled`, `BeginGroup` und `OnAction`) zu zeigen. Die Eigenschaft `OnAction` ist besonders wichtig, da mit ihr dem Symbol ein Makro zugewiesen wird.

**Tabelle 8.3** Die wichtigsten Eigenschaften des Objekts CommandBar

| Eigenschaft | Beschreibung |
|---|---|
| BuitIn | True, wenn die angegebene Befehlsleiste oder das angegebene Befehlsleisten-steuerelement eine integrierte Befehlsleiste bzw. ein integriertes Steuerelement der Containeranwendung ist. False, wenn es sich um eine benutzerdefinierte Befehlsleiste bzw. ein benutzerdefiniertes Steuerelement oder um ein integriertes Steuerelement handelt, dessen OnAction-Eigenschaft festgelegt wurde. |
| Enabled | True, wenn die angegebene Befehlsleiste aktiviert ist |
| Height, Width | Die Höhe und die Breite |
| Left, Top | Gibt die linke und obere Entfernung in Punkten zurück |
| Type | die verschiedenen Typen: msoBarTypeMenuBar (Menüleiste), msoBarTypeNor-mal (Symbolleiste) und msoBarTypePopup (Kontextmenü) |
| Visible | Ist die Symbolleiste sichtbar? Achtung: Der Standardwert ist False – Sie müssen folglich die Symbolleiste sichtbar machen, also den Wert explizit auf True setzen. |

**Beispiel**

Das folgende Makro löscht alle Symbole von der (oben erzeugten) Symbolleiste und schließlich die Symbolleiste selbst:

```
Sub MeineNeuenSymbisWerdenGelöscht()
    Dim mnuSymbLeiste As CommandBar
    Dim i As Integer
    Set mnuSymbLeiste = Application.CommandBars("Meine Symbole")
    For i = mnuSymbLeiste.Controls.Count To 1 Step -1
        mnuSymbLeiste.Controls(i).Delete
    Next
    mnuSymbLeiste.Delete
End Sub
```

Oder, etwas eleganter, auf folgende Weise:

```
Sub MeineNeuenSymbisWerdenGelöscht()

   Dim mnuSymbLeiste As CommandBar

   Dim ctlSymbol As CommandBarControl

   Set mnuSymbLeiste = Application.CommandBars("Meine Symbole")

   For Each ctlSymbol In mnuSymbLeiste.Controls

      ctlSymbol.Delete

   Next

   mnuSymbLeiste.Delete

End Sub
```

Das Löschen der einzelnen Symbole ist selbstverständlich überflüssig. An dieser Stelle sollte lediglich die Methode gezeigt werden.

**Hinweis**

Jedem vorhandenen Symbol ist eine ID zugeordnet. Aber nicht hinter jeder Zahl steckt ein Symbol. Lassen Sie sich per Programmierung eine neue Symbolleiste generieren, die für die Werte zwischen 1 und 1000 neue Symbole erzeugt. Lassen Sie die Werte im „ToolTip-Text“ anzeigen.

```
Sub VieleSymbole()

   Dim mnuSymbLeiste As CommandBar

   Dim mnuSymbol As CommandBarButton

   Dim intZähler As Integer

   On Error Resume Next

   Application.CommandBars("Neue_Symbole").Delete

   Set mnuSymbLeiste = Application.CommandBars.Add

   With mnuSymbLeiste

      .Name = "Neue_Symbole"

      .Position = msoBarFloating

      .Visible = True

   End With

   For intZähler = 1 To 1000

      Set mnuSymbol = mnuSymbLeiste.Controls.Add(ID:=intZähler)

      If Err.Number = 0 Then

         mnuSymbol.TooltipText = intZähler

      End If

      Err.Clear

   Next

End Sub
```

**Hinweis**

Die Zeile

```
Application.CommandBars("Neue_Symbole").Delete
```

würde einen Fehler verursachen, wenn der Benutzer die Symbolleiste bereits geöscht hat. Damit sie ohne Schleife trotzdem gelöscht werden kann, wurde der Befehl

```
On Error Resume Next
```

eingefügt. Dies ist zugegebenermaßen kein guter Programmierstil, aber in unserem Falle völlig zweckdienlich.

**Achtung**

Würde man auf die Verzweigung verzichten, bei der die Err.Number überprüft wird, dann würde dem jeweils letzten Symbol die jeweils höchste mögliche Zahl vor dem nächsten Symbol zugewiesen werden, was falsch ist. Beispiel: Wenn die Symbole mit der Nummer 51, 52 und 59 existieren, dann wird dem ersten Symbol die Nummer 51 zugewiesen. Dem zweiten Symbol nacheinander die Nummern 52, 53, 54, ... und schließlich 58. Dann wird für 59 ein neues Symbol generiert.

Wenn Sie die ID-Nummern nicht als Tooltiptext angezeigt bekommen möchten, können Sie die Nummer auch als Beschriftung auf dem Symbol sichtbar machen:

```
If Err.Number = 0 Then

   mnuSymbol.Style = msoButtonIconAndCaption

   mnuSymbol.TooltipText = intZähler

   mnuSymbol.Caption = intZähler

End If
```

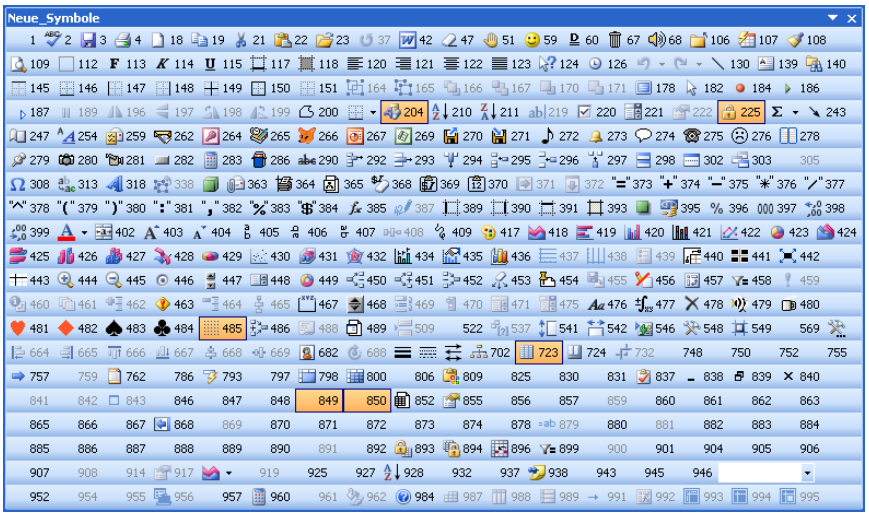



**Abbildung 8.5** Die Symbole mit ihren Nummern

**Hinweis**

Selbstverständlich können Sie die ID und den Namen in eine Exceltabelle schreiben.

**Tabelle 8.4** Folgende Eigenschaften stehen für die Controls zur Verfügung

| Eigenschaft | Beschreibung |
|---|---|
| BeginGroup | True, wenn sich das angegebene Befehlsleistensteuerelement am Anfang einer Gruppe von Steuerelementen auf der Befehlsleiste befindet. |

| Eigenschaft | Beschreibung |
|---|---|
| BuitIn | True, wenn die angegebene Befehlsleiste oder das angegebene Befehlsleistensteuerelement eine integrierte Befehlsleiste bzw. ein integriertes Steuerelement der Containeranwendung ist. False, wenn es sich um eine benutzerdefinierte Befehlsleiste bzw. ein benutzerdefiniertes Steuerelement oder um ein integriertes Steuerelement handelt, dessen OnAction-Eigenschaft festgelegt wurde. |
| ID | Das integrierte Symbol |
| Caption | Die Beschriftung |
| ToolTipText | Der Text des QuickInfos |
| DescriptionText | Die Beschreibung – sie wird nicht angezeigt! |
| Tag | Eine Interne Information |
| Enabled | True, wenn das Befehlsleistensteuerelement aktiviert ist |
| Height, Width | Höhe und Breite |
| OnAction | Der Name des Makros |
| Type | Der Typ. Folgende Konstanten stehen zur Verfügung:<br>msoControlActiveX, msoControlAutoCompleteCombo, msoControlButton, msoControlButtonDropdown, msoControlButtonPopup, msoControlComboBox, msoControlCustom, msoControlDropdown, msoControlEdit, msoControlExpandingGrid, msoControlGauge, msoControlGenericDropdown, msoControlGraphicCombo, msoControlGraphicDropdown, msoControlGraphicPopup, msoControlGrid, msoControlLabel, msoControlLabelEx, msoControlOCXDropdown, msoControlPane, msoControlPopup, msoControlSpinner, msoControlSplitButtonMRUPopup, msoControlSplitButtonPopup, msoControlSplitDropdown, msoControlSplitExpandingGrid, msoControlWorkPane |
| Style | Das Aussehen. Folgende Konstanten stehen Ihnen zur Verfügung:<br>msoButtonAutomatic, msoButtonIconAndCaption, msoButtonCaption, msoButtonIcon, msoButtonIconAndCaptionBelow, msoButtonIconAndWrapCaption, msoButtonIconAndWrapCaptionBelow und msoButtonWrapCaption |
| Visible | Ist das Symbol sichtbar? |

Mit der Methode `AddItem` kann ein Eintrag zu einem Dropdownsymbol hinzugefügt werden. Die Werte können fest im Code eingetragen sein oder dynamisch aus einer Liste gelesen werden:

```
Sub NochMehrNeueSymbis()

[...]

   Set mnuSymbol = mnuSymbLeiste.Controls.Add(Type:=msoControlDropdown)

   With mnuSymbol

      .Width = 300

      For i = 1 To _

         ThisWorkbook.Worksheets(1).Range("A1").CurrentRegion.Rows.Count

      .AddItem ThisWorkbook.Worksheets(1).Cells(i, 1).Value & " " & _

         ThisWorkbook.Worksheets(1).Cells(i, 2).Value

      Next

      .TooltipText = "Währungsliste"

   End With

End Sub
```

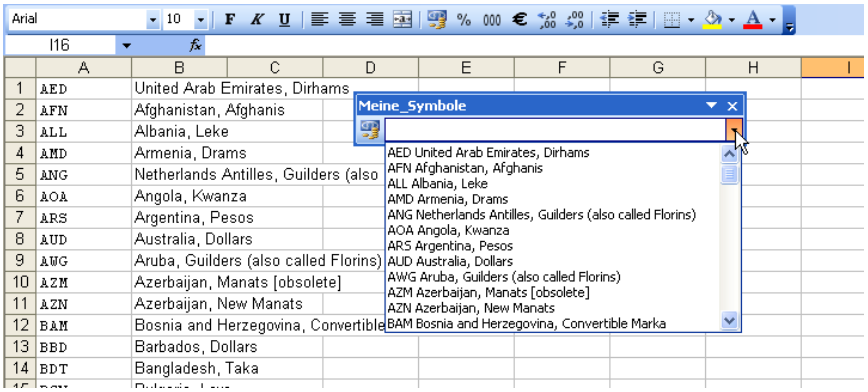



**Abbildung 8.6** Das neue Dropdown-Symbol

Mit diesem Wissen ist es nun nicht mehr schwierig sämtliche Popupmenüs aufzulisten:

```
Sub PopUp_Auflisten()

   Dim cbr As CommandBar

   Dim ctl As CommandBarControl

   Dim strListe As String


   For Each cbr In Application.CommandBars

      If cbr.Type = msoBarTypePopup Then

         strListe = strListe & vbCr & cbr.Name & vbCr

         For Each ctl In cbr.Controls

            strListe = strListe & ctl.Caption & "/"

         Next

      End If

   Next


MsgBox strListe

End Sub
```

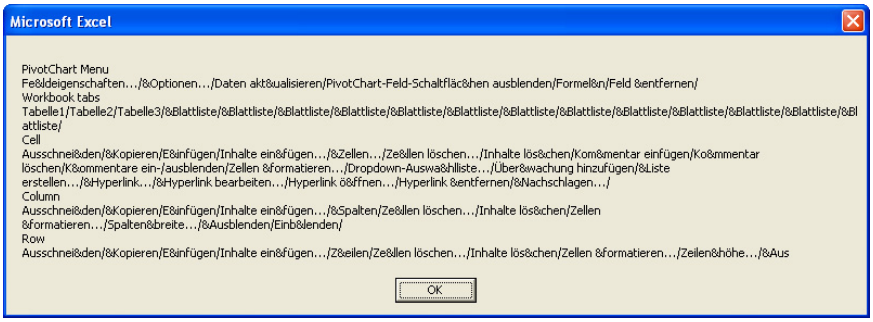



**Abbildung 8.7** Alle Popupmenüs

Oder – da es sich um sehr viele Menüs, beziehungsweise Symbole handelt, können sie besser in eine Exceltabelle eingetragen werden:

```
Sub PopUp_Eintragen()


   Dim cbr As CommandBar
   Dim ctl As CommandBarControl
   Dim xlDatei As Workbook
   Dim xlBlatt As Worksheet
   Dim i As Integer


   On Error Resume Next


   Set xlDatei = Application.Workbooks.Add
   Set xlBlatt = xlDatei.Worksheets(1)


   i = 2
   xlBlatt.Cells(1, 1).Value = "PopupMenüname"
   xlBlatt.Cells(1, 2).Value = "Symbolname"
   xlBlatt.Cells(1, 3).Value = "Symbolname"
   xlBlatt.Cells(1, 3).Value = "Symbol"


   For Each cbr In Application.CommandBars
      If cbr.Type = msoBarTypePopup Then
         xlBlatt.Cells(i, 1).Value = cbr.Name
         For Each ctl In cbr.Controls
             xlBlatt.Cells(i, 2).Value = ctl.Caption
             xlBlatt.Cells(i, 3).Value = ctl.FaceId
             ctl.CopyFace
             xlBlatt.Paste xlBlatt.Cells(i, 4)
             i = i + 1
         Next
      End If
   Next
   xlBlatt.Range("A1:D1").EntireColumn.AutoFit
End Sub
```

| | A | B | C | D |
|---|---|---|---|---|
| 1 | PopupMenüname | Symbolname | Symbolname | Symbol |
| 47 | | &Spalten | 297 | |
| 48 | | Ze&llen löschen... | 294 | |
| 49 | | Inhalte lös&chen | 3125 | |
| 50 | | Zellen &formatieren... | 855 | |
| 51 | | Spalten&breite... | 542 | |
| 52 | | &Ausblenden | 886 | |
| 53 | | Einb&lenden | 887 | |
| 54 | Row | Ausschnei&den | 21 | |
| 55 | | &Kopieren | 19 | |
| 56 | | E&infügen | 22 | |
| 57 | | Inhalte ein&fügen... | 755 | |
| 58 | | Z&eilen | 296 | |
| 59 | | Ze&llen löschen... | 293 | |
| 60 | | Inhalte lös&chen | 3125 | |
| 61 | | Zellen &formatieren... | 855 | |
| 62 | | Zeilen&höhe... | 541 | |
| 63 | | &Ausblenden | 883 | |
| 64 | | Einb&lenden | 884 | |
| 65 | Cell | Ausschnei&den | 21 | |
| 66 | | &Kopieren | 19 | |
| 67 | | E&infügen | 22 | |
| 68 | | Inhalte ein&fügen... | 755 | |
| 69 | | &Zellen... | 295 | |
| 70 | | Ze&llen löschen... | 292 | |
| 71 | | Inhalte lös&chen | 3125 | |
| 72 | | Kom&mentar einfügen | 2031 | |
| 73 | | Ko&mmentar löschen | 1592 | |
| 74 | | K&ommentare ein-/ausblenden | 1593 | |
| 75 | | Zellen &formatieren... | 855 | |
| 76 | | Seitenumbruch einf&ügen | 1588 | |
| 77 | | &Alle Seitenumbrüche zurücksetzen | 1585 | |
| 78 | | Druckbereich &festlegen | 364 | |
| 79 | | &Zum Druckbereich hinzufügen | 1583 | |
| 80 | | &Aus Druckbereich ausschließen | 1586 | |

**Abbildung 8.8** Die Popupmenüs mit ihren Controls (Ausschnitt)

Und damit ist auch klar, wie ein neues Popupmenü erstellt werden kann oder wie Sie vorhandene Popup-Menüs anpassen können.

**Beispiel**

Das folgende Beispiel fügt in das Popup-Menü der Spalten den neuen Eintrag Daten | Text in Spalten ein. Wird er angeklickt, dann wird der entsprechende Assistent aktiviert:

```
Sub NeuesPopUpMenü()

   Dim cbr As CommandBar

   Dim ctl As CommandBarControl

   Set cbr = Application.CommandBars("Column")

   cbr.Reset

   Set ctl = cbr.Controls.Add(msoControlButton, 107, , 8, True)

   ctl.Caption = "Daten | Text in Spalten"

   ctl.OnAction = "DatenTextInSpalten"

End Sub


Sub DatenTextInSpalten()

   Dim dlg As Dialog

   Set dlg = Application.Dialogs(xlDialogTextToColumns)

   dlg.Show

End Sub
```

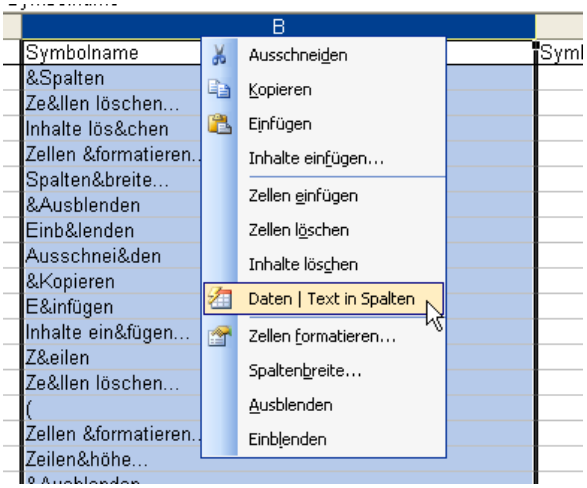



**Abbildung 8.9** Das Kontextmenü wird erweitert.

**Hinweis**

Selbstverständlich können und sollten bei sehr vielen Einträgen die neuen Einträge in Exceltabellen ausgelagert werden und von dort dynamisch eingelesen werden wie im Beispiel des Dropdownsymbols gezeigt wurde(Siehe **Abbildung 8.6**).

**Beispiel**

In Excel werden für Symbole drei (vier) Makros benötigt. Das erste Makro überprüft, ob ein Symbol mit dem Namen „Abrechnung“ vorhanden ist. Falls ja, so wird es sichtbar gemacht, falls nein, so wird es erzeugt. Wechselt der Benutzer nun in eine andere Datei, dann wird dieses Symbol unsichtbar geschaltet. Wechselt der Benutzer zurück in seine Datei, so wird wie beim Start überprüft, ob das Symbol existiert. Entweder wird es sichtbar gemacht oder erzeugt. Beendet der Benutzer seine Datei, dann wird das Symbol gelöscht.

Die drei Makros sind sehr ähnlich aufgebaut. Alle drei überprüfen eine bestimmte Symbolleiste, ob das spezielle Symbol schon (noch) vorhanden ist. Falls dies der Fall ist, wird es sichtbar oder unsichtbar gemacht oder ganz gelöscht. Im ersten Makro SymbAbrechnungSymbolErzeugen wird es bei Nichtexistenz erzeugt. Der Zähler (intZähler) läuft dabei von der letzten bis zur ersten Zahl. Der Grund: Das Symbol wird am rechten Rand der Symbolleiste Standard erzeugt. Also werden die Symbole von rechts nach links überprüft, damit die Schleife nicht unnötig viele Durchläufe haben muss.

```
Sub SymbAbrechnungSymbolErzeugen()

   Dim mnuSymbLeiste As CommandBar

   Dim mnuSymbol As CommandBarButton

   Dim intZähler As Integer

   Dim intSymbAnzahl As Integer


   Set mnuSymbLeiste = Application.CommandBars("Standard")

   intSymbAnzahl = mnuSymbLeiste.Controls.Count

   For intZähler = intSymbAnzahl To 1 Step -1

      If mnuSymbLeiste.Controls(intZähler).Caption = _

      "Abrechnung" Then

         mnuSymbLeiste.Controls(intZähler).Visible = True
```

```
            Exit Sub
         End If
      Next

      Set mnuSymbol = mnuSymbLeiste.Controls.Add(ID:=279, _
         Before:=mnuSymbLeiste.Controls.Count, temporary:=True)

      mnuSymbol.Style = msoButtonIconAndCaption
      mnuSymbol.DescriptionText = "Abrechnung"
      mnuSymbol.Caption = "Abrechnung"
      mnuSymbol.TooltipText = "Abrechnung"
   End Sub

   Sub SymbAbrechnungSymbolUnsichtbar()
      Dim mnuSymbLeiste As CommandBar
      Dim mnuSymbol As CommandBarButton
      Dim intZähler As Integer
      Dim intSymbAnzahl As Integer

      Set mnuSymbLeiste = Application.CommandBars("Standard")
      intSymbAnzahl = mnuSymbLeiste.Controls.Count
      For intZähler = intSymbAnzahl To 1 Step -1
         If mnuSymbLeiste.Controls(intZähler).Caption = _
            "Abrechnung" Then
            mnuSymbLeiste.Controls(intZähler).Visible = False
            Exit Sub
         End If
      Next
   End Sub

   Sub SymbAbrechnungSymbolLöschen()
      Dim mnuSymbLeiste As CommandBar
      Dim mnuSymbol As CommandBarButton
      Dim intZähler As Integer
      Dim intSymbAnzahl As Integer
```

```
    Set mnuSymbLeiste = Application.CommandBars("Standard")
    intSymbAnzahl = mnuSymbLeiste.Controls.Count
    For intZähler = intSymbAnzahl To 1 Step -1
       If mnuSymbLeiste.Controls(intZähler).Caption = _
       "Abrechnung" Then
          mnuSymbLeiste.Controls(intZähler).Delete
          Exit Sub
       End If
    Next
End Sub
```

**Hinweis**
Selbstverständlich können Sie im Ereignis Workbook_Deactivate auch die Symbole oder Symbolleiste löschen.

## 8.2 Tastenkombinationen

Wenn Sie eine Tastenkombination per Programmierung vergeben möchten, dann verwenden Sie die Methode `OnKey` des Objekts `Application`. Es verlangt zwei Parameter: eine Taste und den Namen eines Makros:

```
Sub NeueTastenkombi()
   Application.OnKey "{F1}", "nichtwdh"
End Sub
Sub nichtwdh()
   MsgBox "Der Helpdesk macht heute einen Betriebsausflug!" & vbCr & _
      vbTab & "Keine Hilfe verfügbar.", vbInformation
End Sub
```

Sie können in Excel jede Taste umbelegen: Ziffern, Buchstaben, Sonderzeichen, … Sie müssen die Buchstaben oder Ziffern lediglich in Anführungszeichen schreiben:

```
Application.OnKey "x", "x_Ändern"
```

Wenn Sie [Alt], [Strg] oder [Umschalt] + Buchstabe oder Ziffer umbelegen möchten, dann schreiben Sie % für [Alt], ^ für [Strg] und + für [Umschalt] gefolgt von dem entsprechenden Zeichen:

```
Application.OnKey "^s", "heuteNichtSpeichern"
```

Verwenden Sie für Zeichen, die beim Drücken der betreffenden Taste nicht angezeigt werden (z.B. EINGABE oder TAB), die Codes aus der folgenden Tabelle. Jeder Code in der Tabelle steht für eine Taste auf der Tastatur.

**Tabelle 8.5 Die Tasten können mit folgenden Zeichen umbelegt werden:**

| Taste | Code |
|---|---|
| BILD-AB | {PGDN} |
| BILD-AUF | {PGUP} |
| EINFG | {INSERT} |
| EINGABETASTE | ~ (Tilde) |

| Taste | Code |
|---|---|
| EINGABETASTE | {RETURN} |
| EINGABETASTE (Zehnertastatur) | {ENTER} |
| ENDE | {END} |
| ENTF | {CLEAR} |
| ENTFERNEN oder ENTF | {DELETE} oder {DEL} |
| ESC | {ESCAPE} oder {ESC} |
| F1 bis F15 | {F1} bis {F15} |
| FESTSTELLTASTE | {CAPSLOCK} |
| HILFE | {HELP} |
| NACH-LINKS-TASTE | {LEFT} |
| NACH-OBEN-TASTE | {UP} |
| NACH-RECHTS-TASTE | {RIGHT} |
| NACH-UNTEN-TASTE | {DOWN} |
| NUM | {NUMLOCK} |
| POS1 | {HOME} |
| ROLLEN | {SCROLLLOCK} |
| RÜCKTASTE | {BACKSPACE} oder {BS} |
| TAB | {TAB} |
| UNTBR | {BREAK} |

Das folgende Makro belegt die Tastenkombination [Strg]+[F1] um:

```
Application.OnKey "^{F1}", "ZeichenFormatieren"
```

**Hinweis**

Sie können eine Standardtastenkombination deaktivieren, indem Sie eine leere Zeichenkette als Makronamen übergeben. Das folgende Makro deaktiviert [Strg]+[#]:

```
Application.OnKey "^#", ""
```

Wenn Sie eine benutzerdefinierte Tastenkombination zurücksetzen möchten, rufen Sie nur die Tastenkombination auf:

```
Application.OnKey "^#"

Application.OnKey "^{F1}"

Application.OnKey "^s"

Application.OnKey "x"
```

**Hinweis**

Beim Beenden von Excel verlieren die benutzerdefinierten Tastenkombinationen ihre Bedeutung. Sie müssen also beim Neustart von Excel neu gesetzt werden, wenn die Tastenkombinationen für die Makros global in Excel zur Verfügung stehen sollen. Wenn die Tastenkombination nur in einer Datei zur Verfügung stehen soll, sollten Sie beim Öffnen und Aktivieren der Datei die Tastenkombination aktivieren, beim Schließen und Deaktivieren der Datei die Tastenkombination entfernen.

# 9 Index