

# Dies klingt wie BASIC, warum sieht es nicht vertraut aus?

## In diesem Kapitel:

- Unterschiede zwischen VBA und BASIC kennenlernen
- Die Elemente der VBA-Sprache verstehen
- Erkennen, dass das Erlernen von VBA nicht wirklich schwer ist
- Aufgezeichneten Code mit dem VB-Editor und der Hilfe untersuchen
- Mit den Debugging-Werkzeugen untersuchen, was der aufgezeichnete Code genau macht
- Den Objektkatalog kennenlernen
- Sieben Tipps für das Aufräumen des aufgezeichneten Codes

Wie in Kapitel 1, »Die Leistungsfähigkeit von Excel mit VBA nutzen«, erwähnt, kann es sein, dass, wenn Sie bereits eine prozedurale Programmiersprache wie BASIC oder COBOL kennen, Sie verwirrt sind, wenn Sie sich VBA-Code ansehen. Obwohl VBA die Abkürzung für *Visual Basic for Applications* ist, handelt es sich bei VBA um eine *objektorientierte* Version von BASIC.

Hier ein Stück VBA-Code:

```
Selection.End(xlDown).Select
Range("A11").Select
ActiveCell.FormulaR1C1 = "Summe"
Range("E11").Select
Selection.FormulaR1C1 = _
    "=SUM(R[-9]C:R[-1]C)"
Selection.AutoFill _
    Destination:=Range("E11:G11"), _
    Type:=xlFillDefault
```

Dieser Code macht wahrscheinlich für jemanden, der nur prozedurale Sprachen kennt, wenig Sinn. Leider war Ihre erste Einführung in die Programmierung in der Schulzeit (vorausgesetzt, Sie sind älter als 40 Jahre) vermutlich eine prozedurale Sprache.

Hier ist ein Codefragment, das in der Programmiersprache BASIC geschrieben wurde:

```
For x = 1 to 10
    Print Rpt$(" ",x);
    Print "*"
Next x
```

Wenn Sie diesen Code ausführen, erhalten Sie auf Ihrem Bildschirm eine diagonale Linie aus Sternchen:

```
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
```

Wenn Sie jemals eine prozedurale Programmiersprache erlernt haben, können Sie sich wahrscheinlich den Code ansehen und erkennen, was vor sich geht, weil prozedurale Programmiersprachen eher dem Englischen ähneln, als dies bei objektorientierten Sprachen der Fall ist. Die Anweisung `Print "Hello World"` folgt dem Format Verb-Objekt, so wie Sie normalerweise auch sprechen. Wir wollen uns für einen Moment von der Programmierung entfernen und uns ein konkretes Beispiel anschauen.

## Die Elemente der »VBA-Sprechweise«

---

Wenn Sie BASIC verwenden, um damit beispielsweise Code für ein Fußballspiel zu schreiben, würde die Anweisung für das Treten des Balls ungefähr so aussehen:

```
"Trete den Ball"
```

Prima, das entspricht dem, wie wir sprechen. Diese Anweisung macht sofort Sinn. Sie haben ein Verb (*trete*) und dann ein Substantiv (*Ball*). Auch der BASIC-Code im vorigen Abschnitt verwendet ein Verb (`Print`) und ein Substantiv (das Sternchen `*`). Das Leben meint es gut mit uns.

Es gibt jedoch ein Problem: VBA funktioniert so nicht. Tatsächlich funktioniert keine objektorientierte Sprache so. In einer objektorientierten Sprache sind die Objekte (Substantive) am wichtigsten; daher auch der Begriff »objektorientiert«. Wenn Sie mit VBA Anweisungen für ein Fußballspiel schreiben würden, wäre die Grundstruktur wie folgt:

```
Ball.Treten
```

Sie haben ein Substantiv (`Ball`), das zuerst angegeben wird. In VBA ist dies ein Objekt. Dann haben Sie ein Verb (`Treten`), das anschließend angegeben wird. In VBA ist dies eine Methode.

Die grundlegende Struktur von VBA bilden eine ganze Menge von Codezeilen mit dieser Syntax:

```
Objekt.Methode
```

Das ist natürlich kein Englisch. Wenn Sie in der Schule eine romanische Sprache erlernt haben, werden Sie sich daran erinnern, dass diese Sprachen ein Substantiv-Adjektiv-Konstrukt verwenden. Allerdings benutzt niemand Substantiv-Verb, um jemandem zu sagen, er solle etwas tun:

Wasser.Trinken  
Nahrung.Essen  
Mädchen.Küssen

Dies ist der Grund, warum VBA für diejenigen, die eine prozedurale Programmiersprache erlernt haben, verwirrend ist.

Lassen Sie uns die Analogie ein Stück weiterführen. Stellen Sie sich vor, Sie gehen auf ein grasbewachsenes Spielfeld und es liegen fünf Bälle vor Ihnen: ein Fußball, ein Basketball, ein Baseball, eine Bowlingkugel und ein Tennisball. Sie wollen ein Kind Ihrer Fußballmannschaft anweisen, »den Fußball zu treten«.

Wenn Sie das Kind anweisen, den Ball zu treten (Ball.Treten) dann wissen Sie nicht genau, welchen der Bälle das Kind treten wird. Vielleicht tritt es den Ball, der am nächsten liegt, was zu Problemen führen kann, wenn das Kind vor der Bowlingkugel steht.

Für jedes Substantiv, beziehungsweise Objekt in VBA, gibt es eine Sammlung (Auflistung) dieser Objekte. Stellen Sie sich Excel vor. Wenn es eine Zeile gibt, kann es auch eine ganze Menge dieser Zeilen geben. Wenn es ein Arbeitsblatt gibt, dann kann es auch eine Reihe von Arbeitsblättern geben. Der einzige Unterschied zwischen dem Namen eines Objekts und einer Sammlung besteht darin, dass Sie für die Auflistung an den Namen des Objekts ein s anhängen und so den Plural bilden.



**Beachten Sie, dass VBA für die Namen von Objekten, Methoden und so weiter die englische Sprache verwendet. Aus diesem Grunde verwenden wir ab jetzt auch für die Analogie mit dem Ball die englischen Begriffe.**

**Aus Row wird Rows.**

**Aus Cell wird Cells.**

**Aus Ball wird Balls.**

---

Wenn Sie auf eine Sammlung verweisen, müssen Sie die Programmiersprache darüber informieren, welches Element der Auflistung Sie meinen. Dies können Sie auf unterschiedliche Weise tun. Sie können auf ein bestimmtes Element verweisen, indem Sie eine Zahl angeben. Wenn der Fußball beispielsweise der zweite Ball ist, könnten Sie das so ausdrücken:

```
Balls(2).Kick
```

Dies funktioniert zwar, stellt aber eine gefährliche Art dar, dies zu programmieren. Diese Anweisung könnte beispielsweise an einem Dienstag funktionieren. Wenn Sie am Mittwoch zum Spielfeld zurückkehren und jemand die Reihenfolge der Bälle geändert hat, könnte Balls(2).Kick zu einer schmerzvollen Erfahrung werden.

Ein viel sicherer Ansatz besteht darin, einen Namen für das Objekt in der Sammlung zu verwenden:

```
Balls("Soccer").Kick
```

Bei diesem Ansatz wissen Sie immer, dass der Fußball getreten werden wird.

So weit, so gut. Sie wissen, dass ein Ball getreten wird, und Sie wissen, dass es der Fußball sein wird. Bei den meisten der Verben, oder Methoden in VBA, gibt es Parameter, die angeben, wie eine Aktion ausgeführt wird. Diese Parameter dienen als Adverbien. Angenommen, Sie wollen, dass der Fußball nach links getreten wird, und zwar mit voller Kraft. In diesem Fall würden Sie mehrere Parameter verwenden, um dem Programm zu sagen, wie die Methode ausgeführt werden soll:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard
```

Wenn Sie sich VBA-Code ansehen, können Sie an der Kombination aus Doppelpunkt und Gleichheitszeichen (:=) erkennen, dass es sich um Parameter darüber handelt, wie das Verb ausgeführt werden soll.

Manchmal besitzt eine Methode eine Liste mit zehn Parametern, von denen einige optional sind. Wenn die Methode `Kick` beispielsweise den Parameter `Elevation` besitzt, dann hätten Sie eine Codezeile wie die folgende:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard, Elevation:=High
```

Jetzt kommt der verwirrende Teil: Jede Methode besitzt eine Standardreihenfolge ihrer Parameter. Wenn Sie kein gewissenhafter Programmierer sind und zufällig die Reihenfolge der Parameter kennen, können Sie die Parameternamen weglassen. Der folgende Code entspricht der vorherigen Zeile des Codes

```
Balls("Soccer").Kick Left, Hard, High
```

Dies führt beim Verstehen des Codes zu Sand im Getriebe. Ohne := wird nicht ersichtlich, dass es sich um Parameter handelt. Nur dann, wenn Sie die exakte Parameterreihenfolge kennen, können Sie verstehen, was gesagt wird. Bei Parameterwerten wie `Left`, `Hard` und `High` ist es noch recht einfach. Was aber, wenn Sie Parameter wie die folgenden haben:

```
ActiveSheet.Shapes.AddShape Type:=1, Left:=10, Top:=20, _  
Width:=100, Height:=200
```

Dann wird es verwirrend, wenn Sie stattdessen diese Anweisung sehen:

```
ActiveSheet.Shapes.AddShape 1, 10, 20, 100, 200
```

Bei der obigen Zeile handelt es sich um gültigen Code. Jedoch macht der Code nur dann Sinn, wenn Sie die Standardreihenfolge dieser `Add`-Methode kennen: `Type`, `Left`, `Top`, `Width`, `Height`. Die Standardreihenfolge für eine Methode ist die Reihenfolge der Parameter, wie sie im Hilfefthema dieser Methode angezeigt wird.

Um das Leben noch unübersichtlicher zu gestalten, ist es Ihnen erlaubt, die Parameter in ihrer Standardreihenfolge anzugeben, ohne sie zu benennen, und dann können Sie zu benannten Parametern wechseln, wenn Sie einen verwenden wollen, der nicht der Standardreihenfolge entspricht. Wenn Sie den Ball nach links und hoch treten wollen, sich aber nicht um die Kraft kümmern (das heißt, Sie sind bereit, die Standardkraft zu akzeptieren), sind die folgenden beiden Anweisungen gleichwertig:

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High  
Balls("Soccer").Kick Left, Elevation:=High
```

Beachten Sie jedoch, dass Sie ab dem Moment, ab dem Sie benannte Parameter verwenden, für den Rest der Codezeile ebenfalls benannte Parameter verwenden müssen.

Manche Methoden funktionieren einfach ohne Parameter. Um das Drücken der Taste `F9` zu simulieren (neu berechnen), verwenden Sie diesen Code:

```
Application.Calculate
```

Andere Methoden führen eine Aktion durch und erstellen etwas. Sie können beispielsweise ein Arbeitsblatt einfügen, indem Sie diesen Code verwenden:

```
Worksheets.Add Before:=Worksheets(1)
```

Da `Worksheets.Add` ein neues Objekt erzeugt, können Sie das Ergebnis dieser Methode einer Variablen zuweisen. In diesem Fall müssen Sie die Parameter in runde Klammern einfassen:

```
Set MyWorksheet = Worksheets.Add(Before:=Worksheets(1))
```

Ein letztes Element der Grammatik fehlt noch, die Adjektive. Genauso wie Adjektive ein Substantiv beschreiben, beschreiben Eigenschaften ein Objekt. Da Sie ein Excel-Fan sind, wollen wir von der Fußball-Analogie zur Excel-Analogie wechseln. Es gibt ein Objekt, das die aktive Zelle bezeichnet. Glücklicherweise besitzt es einen sehr intuitiven Namen:

```
ActiveCell
```

Angenommen, Sie wollen die Farbe der aktiven Zelle auf Rot setzen. Es gibt eine Eigenschaft mit dem Namen `Interior.Color` für eine Zelle, die eine komplexe Folge von Codes verwendet. Sie können jedoch eine Zelle rot einfärben, indem Sie einfach diesen Code verwenden:

```
ActiveCell.Interior.Color = 255
```

Sie sehen, wie verwirrend das sein kann. Auch hier gibt es das Konstrukt *Substantiv-Punkt-Etwas*, aber in diesem Fall handelt es sich nicht um `Objekt.Methode`, sondern um `Objekt.Eigenschaft`. Wie Sie beide unterscheiden können, ist recht subtil: Vor dem Gleichheitszeichen steht kein Doppelpunkt. Eine Eigenschaft wird fast immer auf irgendeinen Wert gesetzt oder der Wert einer Eigenschaft wird etwas anderem zugewiesen.

Damit die aktuelle Zelle die gleiche Farbe erhält wie Zelle A1, können Sie diesen Code verwenden:

```
ActiveCell.Interior.Color = Range("A1").Interior.Color
```

`Interior.Color` ist eine Eigenschaft. Indem Sie den Wert einer Eigenschaft ändern, können Sie Dinge anders aussehen lassen. Es ist irgendwie bizarr: Sie ändern ein Adjektiv und machen damit tatsächlich etwas mit der Zelle.

Menschen würden sagen: »Färbe die Zelle rot«, während VBA sagt:

```
ActiveCell.Interior.Color = 255
```

Tabelle 2-1 fasst die Elemente der VBA-Sprache zusammen.

| VBA-Komponenten         | Analog zu            | Notizen   |
|-------------------------|----------------------|---|
| Objekt                  | Substantiv           | Zu den Beispielen gehören Zelle oder Arbeitsblatt.  |
| Auflistung (Collection) | Substantiv im Plural | Gibt normalerweise ein bestimmtes Objekt an: Worksheets(1).   |
| Methode                 | Verb                 | Werden im Format Objekt.Methode verwendet.  |
| Parameter               | Adverb               | Gibt die Parameter nach der Methode an. Trennen Sie den Parameternamen von seinem Wert mit :=.  |
| Eigenschaft             | Adjektiv             | Sie können den Wert einer Eigenschaft festlegen (beispielsweise ActiveCell.Height=10) oder den Wert einer Eigenschaft abrufen und speichern (beispielsweise x = ActiveCell.Height). |

**Tabelle 2-1** Elemente der Programmiersprache VBA

## VBA ist nicht wirklich schwierig

Wenn Sie wissen, ob Sie es mit Eigenschaften oder Methoden zu tun haben, können Sie in Ihrem Code die richtige Syntax verwenden. Machen Sie sich keine Sorgen, wenn es im Moment alles verwirrend erscheint. Wenn Sie VBA-Code von Grund auf selbst schreiben, ist es schwierig zu wissen, ob der Vorgang, die Farbe einer Zelle auf Gelb zu setzen, ein Verb oder ein Adjektiv erfordert. Ist es eine Methode oder eine Eigenschaft?

Hier ist der Makrorekorder besonders hilfreich. Wenn Sie nicht wissen, wie man etwas codiert, nehmen Sie ein einfaches kleines Makro auf, schauen sich den aufgezeichneten Code an und finden so heraus, was Sie brauchen.

## VBA-Hilfdateien: Verwenden Sie F1, um alles zu finden

Die Excel VBA-Hilfe ist ein erstaunliches Feature, vorausgesetzt, Sie sind mit dem Internet verbunden. Wenn Sie VBA-Makros schreiben wollen, müssen Sie unbedingt Zugriff auf die VBA-Hilfethemen haben. Folgen Sie diesen Schritten, um zu sehen, wie einfach es ist, Hilfe in VBA zu bekommen:

1. Öffnen Sie Excel und wechseln Sie zum VB-Editor, indem Sie **[Alt] + [F11]** drücken. Wählen Sie im Menü *Einfügen* den Befehl *Modul*.
2. Geben Sie diese drei Codezeilen ein:

```
Sub Test()  
    MsgBox "Hello World!"  
End Sub
```

3. Klicken Sie innerhalb des Wortes MsgBox.
4. Während sich die Einfügemarke im Word MsgBox befindet, drücken Sie **[F1]**. Wenn Sie mit dem Internet verbunden sind, sehen Sie das Hilfethema für die Funktion MsgBox.

## Verwenden der Hilfethemen

Wenn Sie Hilfe für eine Funktion oder Methode anfordern, zeigt das Hilfethema die verschiedenen verfügbaren Argumente an. Wenn Sie an das Ende eines Hilfethemas blättern, sehen Sie eine großartige Ressource: Codebeispiele unter der Überschrift »Beispiel« (siehe Abbildung 2-1). Es ist möglich, den Code auszuwählen, ihn in die Zwischenablage zu kopieren, indem man **[Strg] + [C]** drückt, und ihn dann in ein Modul einzufügen, indem man **[Strg] + [V]** drückt. Nachdem Sie ein Makro aufgezeichnet haben, können Sie Hilfe erhalten, indem Sie den Cursor in ein beliebiges Schlüsselwort setzen und **[F1]** drücken.

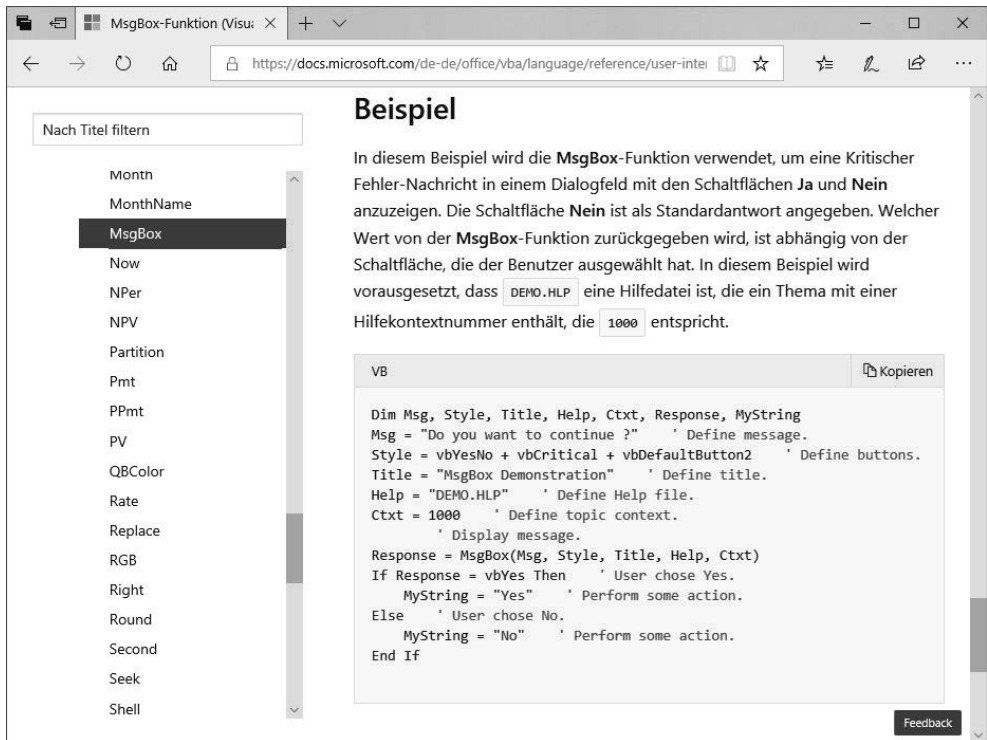


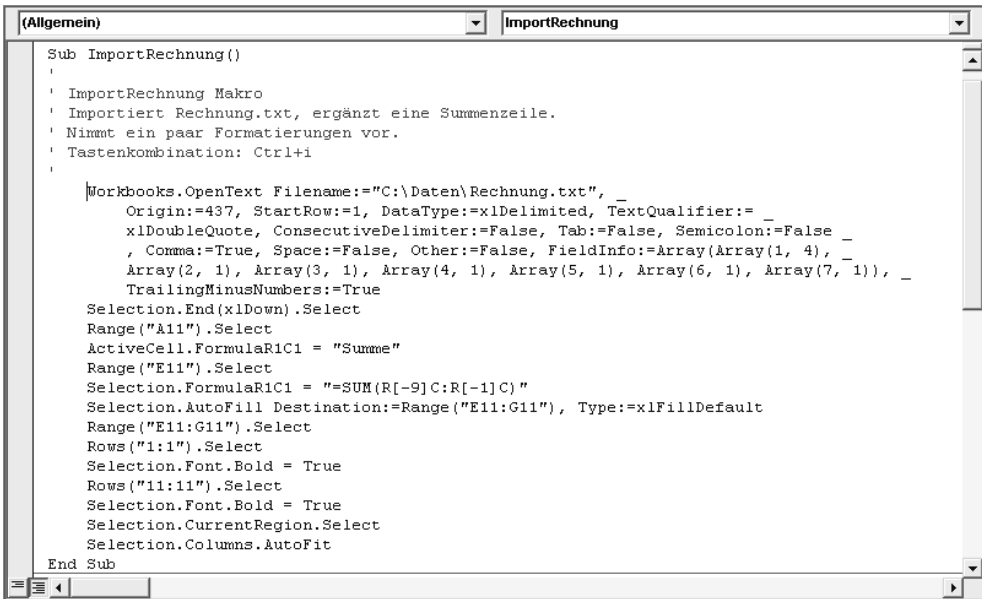
Abbildung 2-1 Die meisten Hilfethemen enthalten Codebeispiele.

# Untersuchen des aufgezeichneten Makrocodes: VB-Editor und die Hilfe verwenden

Werfen wir einen Blick auf den Code, den Sie in Kapitel 1 aufgezeichnet haben, um zu sehen, ob er jetzt mehr Sinn macht, nachdem Sie über Objekte, Eigenschaften und Methoden Bescheid wissen. Sie können auch sehen, ob es möglich ist, die Fehler zu korrigieren, die durch den Makrorekorder entstehen.

Schauen Sie sich jetzt, da Sie das Konzept Substantiv.Verb oder Objekt.Methode verstehen, noch einmal die erste Codezeile an, die mit `Workbooks.OpenText` beginnt. In diesem Fall ist `Workbooks` ein Objekt, und `OpenText` ist eine Methode. Klicken Sie mit der Maus in das Wort `OpenText` und drücken Sie `[F1]`, um eine Erklärung für die `OpenText`-Methode zu erhalten (siehe Abbildung 2-3).

Die Hilfedatei bestätigt, dass `OpenText` eine Methode oder ein Aktionswort ist. Die Standardreihenfolge für alle Argumente, die mit `OpenText` verwendet werden können, erscheint in der grauen Box. Beachten Sie, dass nur ein Argument erforderlich ist: der Dateiname. Alle anderen Argumente sind optional.



```
Sub ImportRechnung()  
    ' ImportRechnung Makro  
    ' Importiert Rechnung.txt, ergänzt eine Summenzeile.  
    ' Nimmt ein paar Formatierungen vor.  
    ' Tastenkombination: Ctrl+i  
    '  
    Workbooks.OpenText Filename:="C:\Daten\Rechnung.txt", _  
        Origin:=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:= _  
        xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False _  
        , Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(Array(1, 4), _  
        Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _  
        TrailingMinusNumbers:=True  
    Selection.End(xlDown).Select  
    Range("A11").Select  
    ActiveCell.FormulaR1C1 = "Summe"  
    Range("E11").Select  
    Selection.FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"  
    Selection.AutoFill Destination:=Range("E11:G11"), Type:=xlFillDefault  
    Range("E11:G11").Select  
    Rows("1:1").Select  
    Selection.Font.Bold = True  
    Rows("11:11").Select  
    Selection.Font.Bold = True  
    Selection.CurrentRegion.Select  
    Selection.Columns.AutoFit  
End Sub
```

**Abbildung 2-2** Hier ist der aufgezeichnete Code des Beispiels aus Kapitel 1.



| Parameter                   |                       |                 |   |
|-----------------------------|-----------------------|-----------------|---|
| Name                        | Erforderlich/Optional | Datentyp        | Beschreibung  |
| <i>Filename</i>             | Erforderlich          | String          | Gibt den Dateinamen der zu öffnenden und analysierenden Textdatei an.   |
| <i>Okeanos</i>              | Optional              | Variant         | Gibt den Ursprung der Textdatei an. Kann eine der folgenden <b>XIPlatform</b> - Konstanten sein: <b>XIMacintosh</b> , <b>XIWindows</b> oder <b>XIMSDOS</b> . Darüber hinaus kann eine Ganzzahl für die Seitenzahl Code, der die gewünschte Codepage handeln. Beispielsweise würde "1256" angeben, dass die Codierung der Quelldatei Arabisch (Windows) ist. Wenn dieses Argument nicht angegeben ist, verwendet die Methode die aktuelle Einstellung der Option <b>Dateiursprung</b> im <b>Textimport-Assistent</b> . |
| <i>StartRow</i>             | Optional              | Variant         | Die Nummer der Zeile an dem mit dem Analysieren von Text. Der Standardwert ist 1.   |
| <i>DataType</i>             | Optional              | Variant         | Gibt das Spaltenformat der Daten in der Datei an. Kann eine der folgenden <b>XITextParsingType</b> -Konstanten sein: <b>XIDelimited</b> oder <b>XIFixedWidth</b> . Wenn dieses Argument nicht angegeben wird, versucht Microsoft Excel das Spaltenformat zu bestimmen, wann die Datei geöffnet wird.  |
| <i>TextQualifier</i>        | Optional              | xlTextQualifier | Gibt den Textbezeichner an.   |
| <i>ConsecutiveDelimiter</i> | Optional              | Variant         | <b>True</b> , wenn aufeinander folgende Trennzeichen als ein Trennzeichen betrachtet. Der Standardwert ist <b>False</b> .   |

**Abbildung 2-3** Dies ist ein Auszug aus dem Hilfetext für die Methode OpenText.

## Optionale Parameter

In der Hilfe können Sie erkennen, ob Sie einen optionalen Parameter auslassen können. Für StartRow gibt die Hilfe an, dass der Standardwert 1 ist. Wenn Sie den StartRow-Parameter auslassen, beginnt Excel mit dem Import in Zeile 1. Das ist ziemlich ungefährlich.

Schauen Sie sich nun die Hilfe zu Origin an. Wenn Sie dieses Argument weggelassen, erben Sie den Wert für Origin, der auf diesem Computer zuletzt von jemandem in Excel verwendet wurde. Das ist eine Garantie für eine Katastrophe. Ihr Code könnte beispielsweise 98% der Zeit arbeiten. Doch unmittelbar nachdem jemand eine arabische Datei importiert hat, merkt sich Excel die Einstellung für Arabisch und geht danach davon aus, dass Ihr Makro dies wünscht, wenn Sie diesen Parameter nicht explizit angeben.

## Definierte Konstanten

Schauen Sie sich in der Hilfedatei den Eintrag für `DataType` in Abbildung 2-3 an, der sagt, dass einer dieser Konstanten verwendet werden kann: `xlDelimited` oder `xlFixedWidth`. Die Hilfe sagt weiter, dass dies die gültigen, in Excel VBA vordefinierten `xlTextParsingType`-Konstanten sind. Drücken Sie im VB-Editor `Strg` + `G`, um das Direktfenster zu öffnen. Geben Sie im Direktfenster diese Zeile ein und drücken Sie `↵`:

```
Print xlFixedWidth
```

Die Antwort wird im Direktfenster ausgegeben. `xlFixedWidth` entspricht dem Wert 2 (siehe Abbildung 2-4). Geben Sie im Direktfenster **Print xlDelimited** ein, was im Prinzip der Eingabe von 1 entspricht. Microsoft geht richtigerweise davon aus, dass es einfacher ist, Code zu lesen, der den etwas Englisch klingenden Ausdruck `xlDelimited` enthält, als wenn dort einfach nur eine 1 stehen würde.



**Abbildung 2-4** Im Direktfenster vom VB-Editor können Sie den eigentlichen Wert von Konstanten wie beispielsweise `xlFixedWidth` abfragen.

Wenn Sie ein bössartiger Programmierer wären, könnten Sie sich sicherlich all diese Konstanten merken und Code mit den numerischen Entsprechungen der Konstanten schreiben. Die Programmiergötter (und der nächste Mensch, der sich Ihren Code ansehen muss) werden Sie jedoch dafür verfluchen.

In den meisten Fällen zeigt die Hilfe entweder gezielt die gültigen Werte der Konstanten an oder stellt einen Hyperlink zur Verfügung, der das Hilfethema öffnet, das die vollständige Aufzählung und die gültigen Werte für die Konstanten anzeigt (siehe Abbildung 2-5).

Ein Problem bei diesem ausgezeichneten Hilfesystem ist, dass es nicht angibt, welche Parameter in einer bestimmten Version neu sind. In diesem speziellen Fall wurde `TrailingMinusNumbers` in Excel 2002 eingeführt. Wenn Sie versuchen, dieses Programm jemandem zu geben, der noch Excel 2000 verwendet, läuft der Code nicht, weil Excel den Parameter `TrailingMinusNumbers` nicht versteht. Leider ist Versuch und Irrtum die einzige Möglichkeit, zu lernen, mit diesem frustrierenden Problem umzugehen.

Wenn Sie das Hilfethema zu `OpenText` lesen, können Sie vermuten, dass es im Grunde genommen dem Öffnen einer Datei mit dem Textkonvertierungs-Assistenten entspricht. In Schritt 1 des Assistenten wählen Sie in der Regel als Datentyp entweder *Getrennt* oder *Feste Breite* aus. Außerdem geben Sie den *Dateiursprung* an und ab welcher Zeile der Import beginnen soll. Dieser erste Schritt des Assistenten wird von den folgenden Parametern der Methode `OpenText` behandelt:

```
Origin:=437
StartRow:=1
DataType:=xlDelimited
```

# xlColumnDataType-Enumeration (Excel)

08.06.2017 · 2 Minuten Lesedauer · Beitragende 

Gibt an, wie eine Spalte analysiert werden soll.

| Name            | Wert | Beschreibung                 |
|-----------------|------|------------------------------|
| xlIDMYFormat    | 4    | TMJ-Datumsformat             |
| xlIDYMFormat    | 7    | TJM-Datumsformat             |
| xlEMDFormat     | 10   | ZMT-Datumsformat             |
| xlGeneralFormat | 1    | Allgemein                    |
| xlIMDYFormat    | 3    | MTJ-Datumsformat             |
| xlIMYDFormat    | 6    | MJT-Datumsformat             |
| xlSkipColumn    | 9    | Spalte wird nicht analysiert |
| xlTextFormat    | 2    | Text                         |
| xlYDMFormat     | 8    | JTM-Datumsformat             |
| xlYMDFormat     | 5    | JMT-Datumsformat             |

**Abbildung 2-5** Klicken Sie den Hyperlink an, um alle möglichen Werte der Konstante zu sehen. Hier werden die 10 möglichen Konstanten für xlColumnDataType in einem neuen Hilfefthema angezeigt.

Schritt 2 des Textkonvertierungs-Assistenten ermöglicht Ihnen, anzugeben, dass Ihre Felder durch Kommata getrennt sind. Da Sie zwei aufeinanderfolgende Kommata nicht wie ein Komma behandeln wollen, sollte das Kontrollkästchen *Aufeinanderfolgende Trennzeichen als ein Zeichen behandeln* nicht aktiviert sein. Manchmal kann ein Feld ein Komma enthalten, wie »XYZ, Inc«. In diesem Fall sollte der Wert des Felds durch Anführungszeichen eingeschlossen sein, wie es im Feld *Textqualifizierer* angegeben ist. Dieser zweite Schritt des Assistenten wird von den folgenden Parametern der Methode `OpenText` behandelt:

```
TextQualifier:=xlDoubleQuote  
ConsecutiveDelimiter:=False  
Tab:=False  
Semicolon:=False  
Comma:=True  
Space:=False  
Other:=False
```

In Schritt 3 des Assistenten geben Sie die Feldtypen an. In diesem Beispiel lassen Sie alle Felder auf *Standard* stehen, mit Ausnahme des ersten Felds, für das Sie das Datum im Format TMJ (Tag, Monat, Jahr) festlegen. Dies wird im Code durch den `FieldInfo`-Parameter dargestellt.

Der dritte Schritt des Textkonvertierungs-Assistenten ist ziemlich komplex. Der gesamte `FieldInfo`-Parameter der `OpenText`-Methode dupliziert die Einstellungen, die Sie in diesem Schritt treffen. Wenn Sie im dritten Schritt des Assistenten auf *Erweitert* klicken, können Sie das Dezimaltrennzeichen, das 1000er-Trennzeichen angeben und, ob bei negativen Zahlen das Minuszeichen nach der Zahl steht.



**Beachten Sie, dass der Makrorekorder nur dann Code für `DecimalSeparator` oder `ThousandsSeparator` erstellt, wenn Sie die Standardeinstellungen ändern. Der Parameter `TrailingMinusNumbers` hingegen wird immer aufgezeichnet.**

---

Denken Sie daran, dass alle Aktionen, die Sie in Excel ausführen, während Sie ein Makro aufzeichnen, in den VBA-Code übersetzt werden. Bei vielen Dialogfeldern werden die Einstellungen, die Sie nicht ändern, zusammen mit den Einstellungen, die geändert wurden, aufgezeichnet. Wenn Sie auf *OK* klicken, um das Dialogfeld zu schließen, zeichnet der Makrorekorder oft alle aktuellen Einstellungen aus dem Dialogfeld im Makro auf.

Hier ist ein weiteres Beispiel. Die nächste Codezeile des Makros lautet:

```
Selection.End(xlDown).Select
```

Sie können klicken, um zu den drei Elementen in dieser Anweisung Hilfe zu erhalten: `Selection`, `End` und `Select`. Ich gehe mal davon aus, dass `Selection` und `Select` irgendwie selbsterklärend sind. Klicken Sie daher in das Wort `End` und drücken Sie `F1`, um Hilfe zu erhalten. Wenn Sie ein Dialogfeld sehen, in dem Sie die Bibliothek des Hilfethemas auswählen können, wählen Sie den Eintrag *Excel* aus, und nicht *VBA*.

Das Hilfethema informiert Sie darüber, dass `End` eine Eigenschaft ist. Sie gibt ein Range-Objekt zurück, das dem Drücken von `Ende` + `↑` oder `Ende` + `↓` in der Excel-Benutzeroberfläche entspricht (siehe Abbildung 2-6). Wenn Sie den blauen Hyperlink für `xlDirection` anklicken, sehen Sie die gültigen Parameter, die Sie an die Eigenschaft `End` übergeben können.

# Range.End-Eigenschaft (Excel)

08.06.2017 · 2 Minuten Lesedauer · Beitragende  

Gibt ein **Range** -Objekt, das die Zelle am Ende des Bereichs darstellt, die den Quellbereich enthält. Entspricht dem Drücken von Ende + nach-oben, Ende + nach-unten-Pfeil, Ende + nach-links oder Ende + nach-rechts-Pfeil. Schreibgeschütztes **Range** -Objekt.

## Syntax

Ausdruck. End (   Direction\_ )

*expression* Eine Variable, die ein **Range**-Objekt darstellt.

## Parameter

| Name             | Erforderlich/Optional | Datentyp    | Beschreibung                                 |
|------------------|-----------------------|-------------|--|
| <i>Direction</i> | Erforderlich          | xIDirection | Die Richtung, in die gewechselt werden soll. |

## Beispiel

In diesem Beispiel wird die Zelle oben in Spalte B in dem Bereich ausgewählt, der Zelle B4 enthält.

```
VB Kopieren  
Range("B4").End(xlUp).Select
```

In diesem Beispiel wird die Zelle am Ende von Zeile 4 in dem Bereich ausgewählt, der Zelle B4 enthält.

```
VB Kopieren  
Range("B4").End(xlToRight).Select
```

**Abbildung 2-6** Die korrekte Hilfeseite für die Eigenschaft End

## Eigenschaften können Objekte zurückgeben

Weiter vorne in diesem Kapitel haben Sie gesehen, dass die grundlegende VBA-Syntax Objekt.Methode lautet. Schauen Sie sich diese Codezeile an, die wir derzeit untersuchen:

```
Selection.End(xlDown).Select
```

In dieser Codezeile ist die Methode Select. Das Schlüsselwort End ist eine Eigenschaft und in der Onlinehilfe haben Sie gesehen, dass diese Eigenschaft ein Range-Objekt zurückgibt. Da die Methode Select auf ein Range-Objekt angewendet werden kann, wird die Methode in diesem Fall an eine Eigenschaft angehängt.

Ausgehend von diesen Informationen können Sie vielleicht vermuten, dass in dieser Codezeile Selection das Objekt repräsentiert. Wenn Sie mit der Maus in das Wort Selection klicken und **F1** drücken, können Sie der Hilfe entnehmen, dass Selection genau genommen eine Eigenschaft ist und kein Objekt. In Wirklichkeit wäre der korrekte Code Application.Selection. Wenn Sie den Code innerhalb von Excel ausführen, geht VBA davon aus, dass Sie sich auf das Excel-Objektmodell beziehen. Sie können dann das Application-Objekt weglassen. Falls Sie jedoch in Word VBA ein Programm schreiben, um Excel zu automatisieren, müssen Sie vor der Selection-Eigenschaft ein Application-Objekt angeben, damit deutlich ist, auf welche Anwendung Sie verweisen.

In diesem Fall kann Application.Selection unterschiedliche Arten von Objekten zurückgeben. Wenn eine Zelle markiert ist, gibt die Eigenschaft ein Range-Objekt zurück.

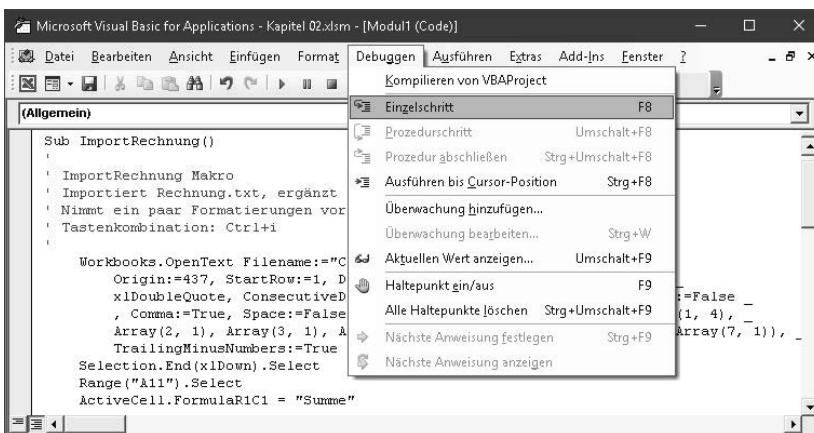
## Verwenden der Debugging-Werkzeuge, um aufgezeichneten Code zu untersuchen

Die folgenden Abschnitte stellen Ihnen einige tolle Debugging-Werkzeuge vor, die im VB-Editor verfügbar sind. Diese Tools sind hervorragend geeignet, um genau zu sehen, was der Code eines aufgezeichneten Makros macht.

### Schrittweise Codeausführung

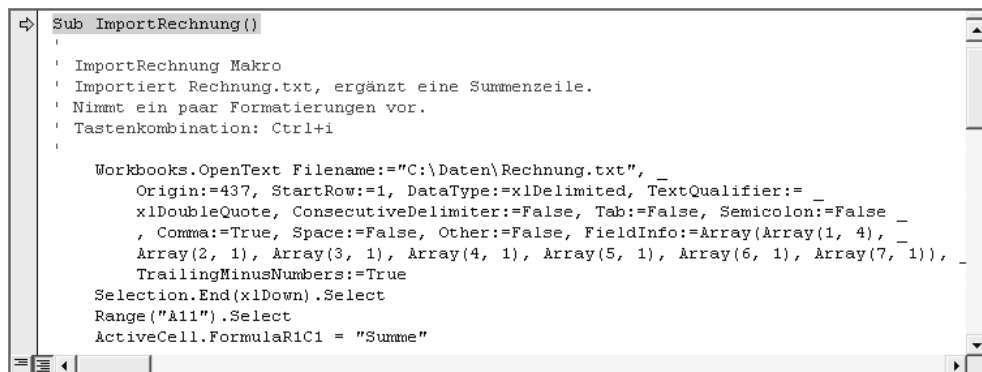
Generell wird ein Makro schnell ausgeführt: Sie starten es und in weniger als einer Sekunde ist es fertig. Wenn etwas schief geht, hat man keine Gelegenheit herauszufinden, was das Makro macht. Mit dem Feature *Einzelschritt* im VB-Editor ist es jedoch möglich, den Code zeilenweise ausführen zu lassen.

Um dieses Feature zu verwenden, stellen Sie sicher, dass sich die Einfügemarke in der Prozedur befindet, die Sie ausführen wollen, wie beispielsweise in ImportRechnung. Wählen Sie dann im Menü *Debuggen* | *Einzelschritt*, wie in Abbildung 2-7 zu sehen. Alternativ können Sie auch **F8** drücken.



**Abbildung 2-7** Sie können das Feature *Einzelschritt* verwenden, um den Code zeilenweise ausführen zu lassen.

Der VB-Editor befindet sich nun im Unterbrechungsmodus. Die nächste Codezeile, die ausgeführt wird, ist gelb markiert. Außerdem sehen Sie in der Spalte links vom Code einen gelben Pfeil (siehe Abbildung 2-8).



```
Sub ImportRechnung()  
    ' ImportRechnung Makro  
    ' Importiert Rechnung.txt, ergänzt eine Summenzeile.  
    ' Nimmt ein paar Formatierungen vor.  
    ' Tastenkombination: Ctrl+i  
    '  
    Workbooks.OpenText Filename:="C:\Daten\Rechnung.txt", _  
        Origin:=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:= _  
        xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False _  
        , Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(Array(1, 4), _  
        Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _  
        TrailingMinusNumbers:=True  
    Selection.End(xlDown).Select  
    Range("A11").Select  
    ActiveCell.FormulaR1C1 = "Summe"
```

**Abbildung 2-8** Die erste Codezeile wartet auf die Ausführung.

In diesem Fall ist die nächste Codezeile, die ausgeführt wird, die Zeile `Sub ImportRechnung`. Diese Zeile bedeutet im Prinzip: Die Prozedur wird gestartet. Drücken Sie **F8**, um die gelb markierte Zeile auszuführen und zur nächsten Zeile vorzurücken. Nun wird der lange Code für `OpenText` hervorgehoben. Drücken Sie **F8**, um diese Codezeile auszuführen. Wenn Sie sehen, dass `Selection.End(xlDown).Select` markiert wird, wissen Sie, dass Visual Basic die Ausführung von `OpenText` beendet hat. An diesem Punkt können Sie mit **Alt** + **↵** zu Excel wechseln und sehen, dass die Datei `Rechnung.txt` in Excel eingelesen wurde. Beachten Sie, dass Zelle A1 markiert ist.



Wenn Sie einen breiten Monitor verwenden, können Sie die Schaltfläche *Wiederherstellen* oben rechts im Fenster vom VB-Editor verwenden und dann die Fenster so anordnen, dass Sie sowohl das VBA- als auch das Excel-Fenster sehen können. (Die Schaltfläche *Wiederherstellen* ist mit zwei überlappenden Fenstern versehen und befindet sich neben der Schaltfläche *Minimieren* (mit dem Minuszeichen) und dem X der Schaltfläche *Schließen* oben an jedem Fenster.)

Dieser Trick eignet sich auch gut, wenn Sie neuen Code aufzeichnen. Sie können dann, während Sie in Excel etwas machen, gleichzeitig den Code sehen.

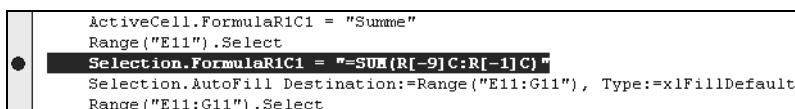
Kehren Sie zum VB-Editor zurück, indem Sie **Alt** + **↵** drücken. Die nächste Zeile, die ausgeführt wird, ist `Selection.End(xlDown).Select`. Drücken Sie **F8**, um diesen Code auszuführen. Kehren Sie zurück zu Excel, um zu sehen, dass nun die letzte Zelle in Ihrem Datensatz markiert ist. Drücken Sie erneut **F8**, um die Zeile `Range("A11").Select` auszuführen. Anstatt zur ersten leeren Zeile zu gehen, wird die Zellmarkierung im Arbeitsblatt in die falsche Zeile verschoben.

Da Sie nun den Problembereich identifiziert haben, können Sie die Codeausführung mit dem Befehl *Zurücksetzen* beenden. Sie können hierfür entweder den Menübefehl *Ausführen | Zurücksetzen* oder die Schaltfläche *Zurücksetzen* in der Symbolleiste verwenden. (Diese Schaltfläche ist mit einem kleinen Quadrat versehen und befindet sich neben den Schaltflächen *Makro ausführen* und *Zurücksetzen*.) Nachdem Sie auf *Zurücksetzen* geklickt haben, sollten Sie zu Excel zurückkehren und alles, was das teilweise ausgeführte Makro getan hat, rückgängig machen. In diesem Fall müssen Sie die Datei *Rechnung.txt* schließen, ohne sie zu speichern.

## Weitere Debugging-Optionen: Haltepunkte

Wenn Sie Hunderte von Codezeilen haben, möchten Sie vielleicht nicht jede Zeile einzeln durchlaufen. Wenn Sie eine allgemeine Idee davon haben, dass in einem bestimmten Abschnitt des Programms ein Problem auftritt, können Sie einen Haltepunkt setzen. Sie können dann den Code starten und die Makroausführung wird unterbrochen, bevor die Zeile mit dem Haltepunkt ausgeführt wird.

Um einen Haltepunkt zu setzen, klicken Sie in den grauen Bereich links von der Zeile, bei der die Codeausführung angehalten werden soll. Neben diesem Code wird ein rötlich brauner Punkt angezeigt und diese Zeile wird rötlich braun hervorgehoben (siehe Abbildung 2-9). (Falls Sie den grauen Randbereich nicht sehen, wählen Sie *Extras | Optionen | Editorformat* und schalten Sie das Kontrollkästchen *Kennzeichenleiste* ein.) Oder markieren Sie eine Codezeile und drücken Sie **[F9]**, um einen Haltepunkt zu setzen bzw. einen zuvor gesetzten zu deaktivieren.



**Abbildung 2-9** Der große rötlich braune Punkt symbolisiert einen Haltepunkt.

Wählen Sie anschließend im Visual Basic-Menü *Ausführen | Sub/UserForm ausführen* oder drücken Sie **[F5]**. Das Programm wird ausgeführt und die Ausführung unterbrochen, sobald die Zeile mit dem Haltepunkt erreicht wird. Der VB-Editor hebt die Zeile mit dem Haltepunkt in Gelb hervor. Sie können nun **[F8]** drücken, um schrittweise durch den Code zu gehen.

Nachdem Sie mit dem Debugging Ihres Codes fertig sind, entfernen Sie einzelne Haltepunkte, indem Sie den rötlich braunen Punkt in der Randspalte anklicken. Alternativ können Sie den Befehl *Debuggen | Alle Haltepunkte löschen* verwenden oder **[Strg] + [↕] + [F9]** drücken, um alle Haltepunkte, die Sie in Ihrem Projekt gesetzt haben, zu entfernen.

## Codeausführung an anderer Stelle fortsetzen

Wenn Sie schrittweise durch den Code gehen, wollen Sie vielleicht manche Codezeilen überspringen oder Codezeilen, die Sie korrigiert haben, erneut ausführen lassen. Dies ist ganz einfach, wenn Sie im Unterbrechungsmodus arbeiten. Eine beliebte Methode besteht darin, den gelben Pfeil im Randbereich zu verschieben. Wenn Sie den Pfeil ziehen, wird der Mauszeiger zu einem Dreifachpfeil, mit dem Sie eine Zeile nach oben oder unten gehen können. Ziehen



Sie den Pfeil bis zu der Zeile, die Sie ausführen wollen; diese Zeile wird dann gelb markiert. Die andere Option besteht darin, die Zeile, zu der Sie springen wollen, mit der rechten Maustaste anzuklicken und dann den Befehl *Nächste Anweisung festlegen* zu verwenden.

## Nicht jede Codezeile einzeln ausführen lassen

Wenn Sie durch den Code gehen, möchten Sie vielleicht einen Teil des Codes ausführen, ohne dabei jede Zeile einzeln ausführen zu lassen, beispielsweise wenn Sie zu einer Schleife gelangen. Sie möchten vielleicht, dass VBA die Schleife hundertmal durchläuft, sodass Sie nach der Schleife den Code wieder zeilenweise ausführen lassen können. Besonders eintönig ist es, die **F8**-Taste Hunderte Male zu drücken, um schrittweise durch eine Schleife zu gehen. Klicken Sie stattdessen auf die Zeile, zu der Sie gehen wollen, und drücken Sie dann **Strg** + **F8** oder wählen Sie *Debuggen | Ausführen bis Cursor-Position*. Dieser Befehl ist auch im Kontextmenü verfügbar.

## Werte abfragen, während Sie schrittweise durch den Code gehen

Obwohl wir das Thema Variablen noch nicht erörtert haben, können Sie den Wert von fast allem abfragen, während der Unterbrechungsmodus aktiv ist. Behalten Sie jedoch im Hinterkopf, dass der Makrorekorder nie Variablen aufzeichnet.

### Verwenden des Direktfensters

Drücken Sie **Strg** + **G**, um das Direktfenster des VB-Editors anzuzeigen. Während sich das Makro im Unterbrechungsmodus befindet, können Sie den VB-Editor fragen, Ihnen die aktuell ausgewählte Zelle, den Namen des aktiven Blattes oder den Wert einer Variablen anzuzeigen. Abbildung 2-10 zeigt mehrere Beispiele von Abfragen, die in das Direktfenster eingegeben wurden.



**Abbildung 2-10** Abfragen, die in das Direktfenster eingegeben werden können, während sich das Makro im Unterbrechungsmodus befindet, mitsamt der Antworten

Anstelle Print einzutippen, können Sie auch ein Fragezeichen eingeben: **? Selection.Address**. Lesen Sie das Fragezeichen als »Was ist«.

Wenn das Direktfenster mit **Strg** + **G** geöffnet wird, wird es in der Regel am unteren Rand des Code-Fensters angezeigt. Sie können den Größenanfasser, der sich oberhalb der blauen Titelleiste befindet, verwenden, um den Direktbereich zu vergrößern oder zu verkleinern.

An der rechten Seite des Direktfensters gibt es eine Bildlaufleiste, mit der Sie durch die angezeigten Einträge rückwärts- oder vorwärtsblättern können.

Es ist nicht erforderlich, Anweisungen nur am unteren Rand des Sofortfensters auszuführen. Wenn Sie beispielsweise gerade eine Codezeile ausgeführt haben, können Sie im Direktfenster nach `Selection.Address` fragen, um zu prüfen, ob diese Codezeile funktioniert.

Drücken Sie die `F8`-Taste, um die nächste Codezeile auszuführen. Anstatt die gleiche Abfrage erneut einzutippen, klicken Sie im Direktfenster irgendwo in der Zeile, die die letzte Abfrage enthält, und drücken `↵`.

Das Direktfenster führt die Abfrage erneut aus, zeigt die Ergebnisse auf der nächsten Zeile an und schiebt ältere Ergebnisse im Fenster weiter nach unten. In diesem Beispiel ist die ausgewählte Adresse `$E$11:$G$11`. Die vorige Antwort, `$A$6`, wird im Fenster nach unten verschoben.

Sie können diese Methode auch verwenden, um die Abfrage zu ändern, indem Sie im Direktfenster rechts neben das Wort `Address` klicken. Drücken Sie die `Rück`-Taste, um das Wort `Address` zu löschen, und geben Sie stattdessen **Columns.Count** ein. Drücken Sie `↵`, und das Direktfenster zeigt die Anzahl der Spalten in der Auswahl an.

Dies ist eine hervorragende Technik, wenn Sie versuchen zu verstehen, welches Stück Code nicht funktioniert. Zum Beispiel können Sie den Namen des aktiven Blattes (**Print ActiveSheet.Name**), die aktuelle Auswahl (**Print Selection.Address**), die aktive Zelle (**Print ActiveCell.Address**), die Formel in der aktiven Zelle (**Print ActiveCell.Formula**), den Wert der aktiven Zelle (**Print ActiveCell.Value** oder **Print ActiveCell** eingeben, weil der Wert die Standard-eigenschaft einer Zelle ist) und so weiter abfragen.

Um das Direktfenster zu schließen, klicken Sie in seiner oberen rechten Ecke auf das X.



`Strg` + `G` können Sie nicht verwenden, um das geöffnete Direktfenster zu schließen. Verwenden Sie das X an der oberen rechten Ecke des Fensters.

---

## Werte durch Zeigen mit der Maus abfragen

In vielen Fällen können Sie mit der Maus auf einen Ausdruck im Codefenster zeigen, dann eine Sekunde warten, bis eine QuickInfo den aktuellen Wert des Ausdrucks anzeigt. Das ist unglaublich hilfreich, wenn Sie sich in Kapitel 4 mit den Themen Schleifen, Verzweigungen und Ablaufsteuerung befassen. Auch bei aufgezeichnetem Code ist dieses Feature nützlich. Beachten Sie, dass sich der Ausdruck, auf den Sie zeigen, nicht in der gerade ausgeführten Codezeile befinden muss. In Abbildung 2-11 hat Visual Basic `E11` ausgewählt, wodurch `E11` zur aktiven Zelle wurde. Wenn Sie mit der Maus auf `ActiveCell.FormulaR1C1` zeigen, sehen Sie eine QuickInfo, der Sie entnehmen können, dass die Formel in der aktiven Zelle `"= SUM(R[-9]C:R[-1]C)"` ist.

```

Range("A11").Select
ActiveCell.FormulaR1C1 = "Summe"
ActiveCell.FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"
Selection.FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"
Selection.AutoFill Destination:=Range("E11:G11"), Type:=xlFillDefault
Range("E11:G11").Select
Rows("1:1").Select

```

**Abbildung 2-11** Zeigen Sie einen kurzen Moment mit dem Mauszeiger auf einen Ausdruck, und eine QuickInfo zeigt den aktuellen Wert des Ausdrucks an.

Manchmal scheint das VBA-Fenster, auf das Sie zeigen, nicht zu reagieren. Da einige Ausdrücke keine Werte anzeigen sollen, ist es schwierig zu sagen, ob VBA absichtlich keinen Wert anzeigt oder ob ein Fehler vorliegt und VBA nicht reagiert. Versuchen Sie, auf etwas zu zeigen, von dem Sie wissen, dass es reagieren sollte, wie zum Beispiel eine Variable. Wenn Sie keine Reaktion erhalten, klicken Sie in die Variable und zeigen Sie weiter auf ihren Namen. Manchmal löst dies die Starre auf, in der sich Excel befindet, und das Draufzeigen funktioniert wieder.

Sind Sie schon beeindruckt? Dieses Kapitel begann mit einer Klage darüber, dass VBA nicht so wie BASIC aussieht. Mittlerweile müssen Sie jedoch zugeben, dass die Visual Basic-Programmierungsumgebung großartig ist und dass die Debugging-Werkzeuge ausgezeichnet sind.

## Abfragen mit dem Überwachungsfenster

In Visual Basic dient eine Überwachung dazu, den Wert eines Ausdrucks anzuzeigen, während Sie den Code schrittweise ausführen lassen. Angenommen, Sie wollen im aktuellen Beispiel sehen, was gerade ausgewählt ist, während der Code läuft. Sie machen dies, indem Sie eine Überwachung für `Selection.Address` hinzufügen.

Wählen Sie im VB-Editor im Menü *Debuggen* den Befehl *Überwachung hinzufügen*. Geben Sie im Dialogfeld *Überwachung hinzufügen* in das Feld *Ausdruck* **Selection.Address** ein und klicken Sie auf *OK* (siehe Abbildung 2-12).



**Abbildung 2-12** Eine Überwachung einrichten, um die Adresse der jeweils aktuellen Auswahl zu sehen

Im Fenster von Visual Basic wird das Fenster *Überwachungsausdrücke* eingeblendet, und zwar normalerweise unterhalb des Codefensters. Wenn Sie das Makro starten, die Datei importieren und dann `Ende` + `↓` drücken, um zur letzten Datenzeile zu gehen, bestätigt das Fenster *Überwachungsausdrücke*, dass `Selection.Address` den Wert `$E$11` enthält (siehe Abbildung 2-13).

| Ausdruck          | Wert      | Typ            | Kontext               |
|-------------------|-----------|----------------|-----------------------|
| Selection.Address | "\$E\$11" | Variant/String | Modul1.ImportRechnung |

**Abbildung 2-13** Ohne mit der Maus zu zeigen oder in das Direktfenster etwas einzutippen, können Sie immer den Wert von überwachten Ausdrücken sehen.

Drücken Sie die **[F8]**-Taste, um den Code Rows("1:1").Select auszuführen. Das Fenster *Überwachungsausdrücke* wird aktualisiert und zeigt an, dass die aktuelle Adresse der Auswahl jetzt \$1:\$1 beträgt.

Im Fenster *Überwachungsausdrücke* ist die Spalte *Wert* lesbar/beschreibbar (wenn möglich), Sie können hier einen neuen Wert eintippen und beobachten, wie sich der Wert auf dem Arbeitsblatt ändert.

## Eine Überwachung verwenden, um einen Haltepunkt zu setzen

Klicken Sie mit der rechten Maustaste einen Ausdruck im Fenster *Überwachungsausdrücke* an und wählen Sie *Überwachung bearbeiten*. Wählen Sie im Bereich *Art der Überwachung* die Option *Unterbrechen, wenn Wert geändert wurde* aus und klicken Sie auf **OK**.

Das Symbol mit der Brille wird in eine Hand mit einem Dreieck geändert. Sie können nun **[F5]** drücken, um den Code auszuführen. Das Makro wird ausgeführt, bis etwas Neues ausgewählt wird. Dies ist ein sehr mächtiges Feature. Anstatt zeilenweise durch das gesamte Makro zu gehen, können Sie es nun automatisch anhalten lassen, wenn etwas Wichtiges geändert wurde. Sie können eine Überwachung auch so konfigurieren, dass die Codeausführung anhält, wenn sich der Wert einer bestimmten Variablen ändert.

## Überwachung für ein Objekt verwenden

Im vorhergehenden Beispiel haben Sie eine bestimmte Eigenschaft überwacht: *Selection.Address*. Es ist auch möglich, ein Objekt wie beispielsweise *Selection* zu überwachen. In *Abbildung 2-14* wurde eine Überwachung für *Selection* hinzugefügt; vor dem Namen sehen Sie das Brillensymbol und ein Pluszeichen.

| Ausdruck          | Wert      | Typ            | Kontext               |
|-------------------|-----------|----------------|-----------------------|
| Selection         | 3306900   | Object/Range   | Modul1.ImportRechnung |
| Selection.Address | "\$E\$11" | Variant/String | Modul1.ImportRechnung |

**Abbildung 2-14** Wenn Sie eine Überwachung für ein Objekt hinzufügen, sehen Sie das Brillensymbol und ein Pluszeichen.

Wenn Sie auf das Pluszeichen klicken, sehen Sie alle Eigenschaften des Objekts *Selection*. Werfen Sie einen Blick auf *Abbildung 2-15* und Sie sehen dort vermutlich mehr, als Sie jemals über *Selection* wissen wollten. Es gibt dort Eigenschaften, von denen Sie wahrscheinlich niemals dachten, dass sie überhaupt existieren. Sie können dort erkennen, dass die Eigenschaft *AddIndent* auf *Falsch* gesetzt ist und die Eigenschaft *AllowEdit* auf *Wahr*. Weiter unten in der Liste stehen nützliche Eigenschaften, wie beispielsweise die Formel (*Formula*) der Auswahl.

In diesem Fenster *Überwachungsausdrücke* können einige Einträge erweitert werden. Beispielsweise befindet sich neben der Sammlung *Borders* ein Pluszeichen, was bedeutet, dass Sie das Pluszeichen anklicken können, um weitere Details zu sehen.

| Ausdruck         | Wert                                  | Typ                                     | Kontext               |
|------------------|---------------------------------------|---|-----------------------|
| Selection        |                                       | Object/Range                            | Modul1.ImportRechnung |
| AddIndent        | Falsch                                | Variant/Boolean                         | Modul1.ImportRechnung |
| AllowEdit        | Wehr                                  | Boolean                                 | Modul1.ImportRechnung |
| Application      |                                       | Application/Application                 | Modul1.ImportRechnung |
| Areas            |                                       | Areas/Areas                             | Modul1.ImportRechnung |
| Borders          |                                       | Borders/Borders                         | Modul1.ImportRechnung |
| Cells            |                                       | Range/Range                             | Modul1.ImportRechnung |
| Column           | 5                                     | Long                                    | Modul1.ImportRechnung |
| ColumnWidth      | 10,71                                 | Variant/Double                          | Modul1.ImportRechnung |
| Comment          | Nothing                               | Comment                                 | Modul1.ImportRechnung |
| CommentThreaded  | <Anwendungs- oder objektdefinierter F | CommentThreaded                         | Modul1.ImportRechnung |
| Count            | 1                                     | Long                                    | Modul1.ImportRechnung |
| CountLarge       | 1                                     | Variant/<Nichtunterstützter Variant-Typ | Modul1.ImportRechnung |
| Creator          | xlCreatorCode                         | XICreator                               | Modul1.ImportRechnung |
| CurrentArray     | <Keine Zellen gefunden.>              | Range                                   | Modul1.ImportRechnung |
| CurrentRegion    |                                       | Range/Range                             | Modul1.ImportRechnung |
| Dependents       | <Keine Zellen gefunden.>              | Range                                   | Modul1.ImportRechnung |
| DirectDependents | <Keine Zellen gefunden.>              | Range                                   | Modul1.ImportRechnung |
| DirectPrecedents |                                       | Range/Range                             | Modul1.ImportRechnung |
| DisplayFormat    |                                       | DisplayFormat/DisplayFormat             | Modul1.ImportRechnung |
| Errors           |                                       | Errors/Errors                           | Modul1.ImportRechnung |
| Font             |                                       | Font/Font                               | Modul1.ImportRechnung |
| FormatConditions |                                       | FormatConditions/FormatConditions       | Modul1.ImportRechnung |
| Formula          | "=SUM(E2:E10)"                        | Variant/String                          | Modul1.ImportRechnung |
| FormulaArray     | "=SUMME(E2:E10)"                      | Variant/String                          | Modul1.ImportRechnung |
| FormulaHidden    | Falsch                                | Variant/Boolean                         | Modul1.ImportRechnung |
| FormulaLocal     | <Anwendungs- oder objektdefinierter F | XIFormulaLocal                          | Modul1.ImportRechnung |

**Abbildung 2-15** Ein Klick auf das Pluszeichen zeigt eine Fülle von Eigenschaften und deren aktuelle Werte.

## Objektkatalog: Die ultimative Referenz

Drücken Sie im VB-Editor **[F2]**, um den Objektkatalog zu öffnen, in dem Sie die gesamte Excel-Objektbibliothek durchstöbern können. Früher besaß ich umfangreiche Excel-Bücher, die mehr als 400 Seiten dafür verwendeten, jedes Objekt im Objektkatalog aufzuführen. Sie können einen Baum retten, indem Sie lernen, den viel mächtigeren Objektkatalog zu verwenden. Der integrierte Objektkatalog steht immer zur Verfügung; drücken Sie einfach **[F2]**. Die nächsten Seiten zeigen, wie Sie den Objektkatalog nutzen.

Wenn Sie **[F2]** drücken, erscheint der Objektkatalog dort, wo sich normalerweise das Codefenster befindet. Im obersten Dropdownmenü steht derzeit *<Alle Bibliotheken>*. Es gibt in diesem Dropdownmenü Einträge für Excel, Office, VBA und jede Arbeitsmappe, die Sie geöffnet haben, sowie zusätzliche Einträge für alles, was Sie in *Extras | Verweise* aktiviert haben. Öffnen Sie dieses Dropdownmenü und wählen Sie nur die Option *Excel* aus.

Im linken Fenster des Objektkatalogs sehen Sie eine Liste aller Klassen, die für Excel verfügbar sind. Klicken Sie im linken Fenster auf die Klasse *Application*. Das rechte Fenster wird geändert und zeigt alle Eigenschaften und Methoden des Objekts *Application* an. Klicken Sie auf etwas im rechten Fenster, wie zum Beispiel *ActiveCell*. Das untere Fenster des Objektkatalogs zeigt Ihnen, dass *ActiveCell* eine Eigenschaft (Property) ist, die einen Range (Bereich)

zurückgibt. Außerdem werden Sie informiert, dass ActiveCell schreibgeschützt ist. (Dies ist eine Warnung, dass Sie ActiveCell keine Adresse zuweisen können, um die Zellmarkierung zu bewegen.)

Sie haben im Objektkatalog gesehen, dass ActiveCell einen Bereich zurückgibt. Wenn Sie im unteren Fenster auf den blauen Hyperlink für Range klicken, sehen Sie alle Eigenschaften und Methoden, die für Range-Objekte und damit für die ActiveCell-Eigenschaft gelten. Klicken Sie auf eine beliebige Eigenschaft oder Methode und klicken Sie dann auf die *Hilfe*-Schaltfläche (das ist die mit dem gelben Fragezeichen oben im Objektkatalog), um das Hilfethema für diese Eigenschaft oder Methode zu öffnen.

Geben Sie einen beliebigen Begriff in das Textfeld neben dem Fernglas ein und klicken Sie auf das Fernglas, um alle passenden Mitglieder der Excel-Bibliothek zu finden. Methoden erscheinen als grüne Bücher mit Geschwindigkeitslinien. Eigenschaften erscheinen als Karteikarten, auf die eine Hand zeigt.

Die Suchmöglichkeiten und Hyperlinks, die im Objektkatalog zur Verfügung stehen, machen ihn zu einer viel nützlicheren Referenz, als es eine alphabetisch sortierte und gedruckte Auflistung aller Informationen jemals sein kann. Lernen Sie, den Objektkatalog im VBA-Fenster zu nutzen, indem Sie **F2** drücken. Um den Objektkatalog zu schließen und zu Ihrem Codefenster zurückzukehren, klicken Sie auf das X in der oberen rechten Ecke.

## Sieben Tipps für das Aufräumen aufgezeichneten Codes

---

In Kapitel 1 haben Sie vier Tipps für die Aufzeichnung von Code erhalten. Bisher hat sich dieses Kapitel damit befasst, wie man den aufgezeichneten Code versteht, wie man auf die VBA-Hilfe für jedes Schlüsselwort zugreifen kann und wie man die ausgezeichneten VBA-Debugging-Werkzeuge benutzt, um Code schrittweise auszuführen und zu untersuchen. Im Rest dieses Kapitels finden Sie sieben Tipps, die Sie beim Aufräumen des aufgezeichneten Codes verwenden können.

### Tipp 1: Wählen Sie nichts aus

Nichts schreit mehr nach aufgezeichnetem Code als Code, der zuerst etwas auswählt, bevor daran eine Aktion vorgenommen wird. Irgendwie macht dies Sinn: In der Excel-Benutzeroberfläche müssen Sie Zeile 1 auswählen, bevor Sie sie fett formatieren können. In VBA wird jedoch nur selten so vorgegangen. Zu dieser Regel gibt es jedoch auch Ausnahmen. Sie müssen zum Beispiel eine Zelle auswählen, wenn Sie eine Formel für die bedingte Formatierung einrichten. Es ist jedoch möglich, Zeile 1 fett zu formatieren, ohne sie zuerst auszuwählen.

Um den Code zu straffen und zu beschleunigen, den der Makrorekorder aufgezeichnet hat, können Sie in vielen Fällen den Teil des Codes, der die Auswahl durchführt, entfernen. Die folgenden beiden Zeilen sind Makrorekorder-Code, bevor er gestrafft wurde:

```
Cells.Select  
Selection.Columns.AutoFit
```

Sie können diese beiden Anweisungen zu einer kombinieren und dabei die Auswahl entfernen. Der Code sieht dann so aus;

```
Cells.Columns.AutoFit
```

Den Code so zu straffen bietet mehrere Vorteile. Erstens besteht Ihr Code dann aus deutlich weniger Zeilen. Zweitens läuft das Programm schneller.

Um den Code so zu straffen, nachdem er aufgezeichnet wurde, markieren Sie ab dem `Select` in der ersten Zeile alles bis zum Punkt nach dem Wort `Select` in der nächsten Zeile und drücken dann `Entf` (siehe Abbildung 2-16 und Abbildung 2-17).

```
Range("E11:G11").Select
Rows("1:1").Select
Selection.Font.Bold = True
Rows("11:11").Select
Selection.Font.Bold = True
Selection.CurrentRegion.Select
Selection.Columns.AutoFit
End Sub
```

**Abbildung 2-16** Markieren Sie den Teil des Codes, der hier hervorgehoben ist ...

```
Range("E11:G11").Select
Rows("1:1").Select
Selection.Font.Bold = True
Rows("11:11").Font.Bold = True
Selection.CurrentRegion.Select
Selection.Columns.AutoFit
End Sub
```

**Abbildung 2-17** ... und drücken Sie die Taste `Entf`. Dies ist das Einmaleins für das Aufräumen aufgezeichneten Codes.

## Tipp 2: Verwenden Sie `Cells(2,5)`, weil es bequemer ist als `Range("E2")`

Der Makrorekorder nutzt häufig die `Range()`-Eigenschaft. Wenn Sie dem Beispiel des Makrorekorders folgen, dann erstellen Sie eine Menge komplizierten Code. Wenn Sie zum Beispiel die Zeilennummer für die Summenzeile in der Variablen `TotalRow` gespeichert haben, können Sie versucht sein, diesen Code zu schreiben:

```
Range("E" & TotalRow).Formula = "=SUM(E2:E" & TotalRow-1 & ")"
```

In diesem Code verwenden Sie Verkettung, um den Buchstaben `E` mit dem aktuellen Wert der Variablen `TotalRow` zu verbinden. Das funktioniert, aber eventuell müssen Sie sich auf einen Bereich beziehen, in dem die Spalte in einer Variablen gespeichert wird. Angenommen, `FinalCol` hat den Wert 10, was Spalte `J` anzeigt. Um auf diese Spalte in einem `Range`-Befehl zu verweisen, müssen Sie etwas Ähnliches wie das Folgende tun:

```
FinalColLetter = MID("ABCDEFGHIJKLMNOPQRSTUVWXYZ", FinalCol, 1)
Range(FinalColLetter & "2").Select
```

Alternativ könnten Sie Code ähnlich wie diesen verwenden:

```
FinalColLetter = CHR(64 + FinalCol)
Range(FinalColLetter & "2").Select
```

Beide Ansätze funktionieren bei den ersten 26 Spalten, bei den verbleibenden 99,85% der Spalten schlagen sie fehl.

Sie könnten damit beginnen, Funktionen mit 10 oder mehr Zeilen zu schreiben, um zu berechnen, dass der Spaltenbuchstabe für Spalte 15896 WMJ lautet, aber das ist nicht notwendig. Statt `Range("WMJ17")` können Sie die Syntax `Cells(Zeile,Spalte)` verwenden. Kapitel 3, »Auf Bereiche verweisen«, behandelt dieses Thema ausführlich. Für den Augenblick reicht es aus, zu verstehen, dass sowohl `Range("E10")` als auch `Cells(10,5)` auf die Zelle am Schnittpunkt der fünften Spalte und der zehnten Reihe verweisen. Kapitel 3 zeigt Ihnen außerdem, wie Sie `.Resize` verwenden, um auf einen rechteckigen Bereich zu verweisen. `Cells(11,5).Resize(1,3)` entspricht `E11:G11`.

### Tipp 3: Verwenden Sie zuverlässigere Wege, um die letzte Zeile zu finden

Es ist schwierig, Daten zu trauen, die aus x-beliebigen Quellen stammen. Wenn Sie Daten in Excel analysieren, denken Sie daran, dass die Daten von irgendeinem System stammen und auch irgendwann in der nahen oder weit entfernten Vergangenheit geschrieben wurden. Die universelle Wahrheit ist, dass möglicherweise irgendwelche Mitarbeiter einen Weg finden werden, das System zu umgehen und einen Datensatz ohne Rechnungsnummer einzugeben. Möglicherweise kann auch ein Stromausfall die Ursache sein; grundsätzlich gilt, dass Sie nicht ausnahmslos darauf bauen können, dass jede Zelle ausgefüllt ist.

Dies ist ein Problem, wenn Sie die Tastenkombination `Ende` + `↓` verwenden. Diese Tastenkombination bringt Sie nicht in die letzte Zeile mit Daten im Arbeitsblatt, sondern in die letzte Zeile mit Daten im aktuellen Bereich. In Abbildung 2-18 würde das Drücken von `Ende` + `↓` den Cursor in Zelle A7 und nicht in die letzte Zeile mit Daten verschieben.

Eine bessere Lösung ist es, am unteren Ende des Arbeitsblattes zu beginnen und die erste nicht-leere Zelle zu suchen, indem man diese Anweisung verwendet:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```

|    | A           | B           | C           | D           |
|----|-------------|-------------|-------------|-------------|
| 1  | Überschrift | Überschrift | Überschrift | Überschrift |
| 2  | Daten       | Daten       | Daten       | Daten       |
| 3  | Daten       | Daten       | Daten       | Daten       |
| 4  | Daten       | Daten       | Daten       | Daten       |
| 5  | Daten       | Daten       | Daten       | Daten       |
| 6  | Daten       | Daten       | Daten       | Daten       |
| 7  | Daten       | Daten       | Daten       | Daten       |
| 8  |             | Daten       | Daten       | Daten       |
| 9  | Daten       | Daten       | Daten       | Daten       |
| 10 | Daten       | Daten       | Daten       | Daten       |
| 11 | Daten       | Daten       | Daten       | Daten       |

**Abbildung 2-18** `Ende` + `↓` funktioniert in der Benutzeroberfläche nicht, wenn in einem Datensatz ein Wert fehlt. Dementsprechend funktioniert in Excel VBA auch `End(xlDown)` nicht.



Diese Methode könnte fehlschlagen, wenn der allerletzte Datensatz die leere Zeile enthält. Wenn die Daten so dicht sind, dass es immer einen diagonalen Pfad von nicht-leeren Zellen bis zur letzten Zeile geben wird, können Sie Folgendes verwenden:

```
FinalRow = Cells(1,1).CurrentRegion.Rows.Count
```

Wenn Sie sicher sind, dass es keine Notizen oder mit anderen Inhalten versehene Zellen unter dem Datensatz gibt, können Sie dies versuchen:

```
FinalRow = Cells(1, 1).SpecialCells(xlLastCell).Row
```

Die `xlLastCell`-Eigenschaft ist oft falsch. Angenommen, Sie haben Daten im Bereich A1:F500. Wenn Sie versehentlich `[Strg] + [↓]` in Zelle 500 drücken, gelangen Sie zu A1048576. Wenn Sie dann *Fett* auf die leere Zelle anwenden, wird sie aktiviert. Oder, wenn Sie **Summe** eintippen und dann die Zelle löschen, wird sie aktiviert. An dieser Stelle wird `xlLastCell` auf F1048576 verweisen.

Ein anderer Weg ist die Verwendung der Methode `Find`:

```
FinalRow = Cells.Find("**", SearchOrder:=xlByRows, _  
    SearchDirection:=xlPrevious).Row
```

Sie müssen aus diesen verschiedenen Methoden diejenige wählen, die am besten der Art Ihres Datensatzes entspricht. Wenn Sie sich nicht sicher sind, können Sie in einer Schleife alle Spalten durchlaufen. Wenn Sie sieben Datenspalten erwarten, können Sie diesen Code verwenden:

```
FinalRow = 0  
For i = 1 to 7  
    ThisFinal = Cells(Rows.Count, i).End(xlUp).Row  
    If ThisFinal > FinalRow then FinalRow = ThisFinal  
Next i
```

## Tip 4: Verwenden Sie Variablen, um hartverdrahtete Zeilen und Formeln zu vermeiden

Der Makrorekorder zeichnet keine Variablen auf. Die Verwendung von Variablen ist einfach, aber genau wie in BASIC kann sich eine Variable einen Wert merken. Variablen werden in Kapitel 4 ausführlicher erörtert.

Es wird empfohlen, die Nummer der letzten Zeile, die Daten enthält, einer Variablen zuzuweisen. Achten Sie darauf, aussagekräftige Variablennamen wie `FinalRow` zu verwenden:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```

Wenn Sie die Zeilennummer des letzten Datensatzes kennen, schreiben Sie das Wort **Summe** in Spalte A der nächsten Zeile:

```
Cells(FinalRow + 1, 1).Value = "Summe"
```

Sie können die Variable sogar beim Erstellen der Formel verwenden. Diese Formel bildet die Summe der Zellen ab E2 bis zur `FinalRow` von E:

```
Cells(FinalRow + 1, 5).Formula = "=SUM(E2:E" & FinalRow & ")"
```

## Tipp 5: Verwenden Sie R1C1-Formeln, die Ihnen das Leben vereinfachen

Der Makrorekorder erstellt oft Formeln im undurchsichtigen R1C1-Format. Allerdings ändern die meisten Leute den Code ab und verwenden Formeln in der normalen A1-Notation. Nach der Lektüre von Kapitel 5, »Formeln und die R1C1-Bezugsart«, werden Sie verstehen, dass es Zeiten gibt, in denen Sie eine R1C1-Formel erstellen können, die viel einfacher ist als die entsprechende Formel in der A1-Schreibweise. Mit einer R1C1-Formel können Sie alle drei Zellen in der Zeile mit den Summen mit folgendem Code erstellen:

```
Cells(FinalRow+1, 5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

## Tipp 6: Kopieren und Einfügen in einer Anweisung

Aufgezeichneter Code ist dafür berüchtigt, dass zuerst ein Bereich kopiert, dann ein anderer Bereich ausgewählt und schließlich `ActiveSheet.Paste` aufgerufen wird. Die Methode `Copy`, die für Bereiche zur Verfügung steht, ist jedoch viel leistungsfähiger. Sie können in einer einzigen Anweisung angeben, was kopiert werden soll und wohin es kopiert werden soll.

Hier ist der aufgezeichnete Code:

```
Range("E14").Select  
Selection.Copy  
Range("F14:G14").Select  
ActiveSheet.Paste
```

Hier ist der bessere Code:

```
Range("E14").Copy Destination:=Range("F14:G14")
```

## Tipp 7: Verwenden Sie »With...End With«, um mehrere Aktionen durchzuführen

Wenn Sie die Zeile mit den Summen fett formatieren, doppelt unterstreichen sowie mit einem größeren Schriftgrad und einer besonders Farbe versehen, sieht der aufgezeichnete Code ähnlich wie dieser aus:

```
Range("A14:G14").Select  
Selection.Font.Bold = True  
Selection.Font.Size = 12  
Selection.Font.ColorIndex = 5  
Selection.Font.Underline = xlUnderlineStyleDoubleAccounting
```

Bei vier dieser Codezeilen muss VBA den Ausdruck `Selection.Font` auflösen. Da alle vier Zeilen auf das gleiche Objekt verweisen, können Sie das Objekt am Anfang eines `With`-Blocks angeben. Innerhalb des `With...End With`-Blocks verweist alles, was mit einem Punkt beginnt, auf das `With`-Objekt. Beachten Sie außerdem, dass die Auswahl mit `Select` entfallen ist (siehe Tipp 1 etwas weiter vorne).

```

With Range("A14:G14").Font
    .Bold = True
    .Size = 12
    .ColorIndex = 5
    .Underline = xlUnderlineStyleDoubleAccounting
End With

```

## Fallstudie: Die Einzelteile zusammenfügen – den aufgezeichneten Code verbessern

Mit den sieben Tipps aus dem vorigen Abschnitt können Sie den aufgezeichneten Code aus Kapitel 1 in effizienten, professionell aussehenden Code konvertieren. Hier ist der Code, wie er vom Makrorekorder am Ende von Kapitel 1 aufgezeichnet wird:

```

Sub FormatiereRechnung3()
    Workbooks.OpenText Filename:="C:\Data\Rechnung.txt", Origin:=437, _
        StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote, _
        ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, _
        Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(_
        Array(1, 4), Array(2, 1), Array(3, 1), Array(4, 1), _
        Array(5, 1), Array(6, 1), Array(7, 1))
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Summe"
    ActiveCell.Offset(0, 4).Range("A1").Select
    Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _
        xlFillDefault
    ActiveCell.Range("A1:C1").Select
    ActiveCell.Rows("1:1").EntireRow.Select
    ActiveCell.Activate
    Selection.Font.Bold = True
    Application.Goto Reference:="R1C1:R1C7"
    Selection.Font.Bold = True
    Selection.CurrentRegion.Select
    Selection.Columns.AutoFit
End Sub

```

Führen Sie diese Schritte durch, um den aufgezeichneten Makrocode aufzuräumen:

1. Lassen Sie die Zeilen mit `Workbook.OpenText` unverändert; sie sind so in Ordnung, wie sie aufgezeichnet wurden.
2. Die folgende Codezeile versucht, die letzte Zeile mit Daten zu finden, damit das Programm weiß, wo die Zeile mit den Summen erstellt werden soll:

```

Selection.End(xlDown).Select

```

3. Sie müssen nichts markieren, um die letzte Zeile zu finden. Außerdem ist es hilfreich, die Zeilennummern der letzten Datenzeile und der Summenzeile Variablen zuzuweisen, damit sie später verwendet werden können. Um den Fall zu berücksichtigen, in dem eine einzelne Zelle in Spalte A leer ist, beginnen Sie unten auf dem Arbeitsblatt und gehen Sie nach oben, um die letzte Zeile zu finden:

```
' Letzte Zeile mit Daten finden. Dies kann sich jeden Tag ändern.  
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row  
TotalRow = FinalRow + 1
```

Beachten Sie, dass diese Codezeilen das Wort *Summe* in Spalte A der Summenzeile einfügen:

```
ActiveCell.Offset(1,0).Select  
ActiveCell.FormulaR1C1 = "Summe"
```

Der Code wird besser, wenn die Variable *TotalRow* verwendet wird, um die Stelle zu ermitteln, an der das Wort *Summe* stehen soll. Auch hier ist es nicht erforderlich, die Zelle auszuwählen, bevor Sie die Beschriftung einfügen:

```
' Unterhalb dieser Zeile die Summenzeile erstellen  
Cells(TotalRow,1).Value = "Summe"
```

4. Beachten Sie, dass diese Codezeilen die Summenformel in Spalte E einfügt und diese dann in die beiden benachbarten Zellen kopiert:

```
ActiveCell.Offset(0, 4).Range("A1").Select  
Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"  
Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _  
xlFillDefault  
ActiveCell.Range("A1:C1").Select
```

Es ist nicht erforderlich, die ganzen Auswahlaktionen durchzuführen. Die folgende Zeile fügt die Formel in drei Zellen ein:

```
Cells(TotalRow,5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

(Die R1C1-Schreibweise für Formeln wird in Kapitel 5 erörtert.)

5. Beachten Sie, dass der Makrorekorder einen Bereich auswählt und dann die Formatierung zuweist:

```
ActiveCell.Rows("1:1").EntireRow.Select  
ActiveCell.Activate  
Selection.Font.Bold = True  
Application.Goto Reference:="R1C1:R1C7"  
Selection.Font.Bold = True
```



Es ist nicht erforderlich, vor der Zuweisung der Formatierung zuerst etwas auszuwählen. Die vorigen fünf Zeilen lassen sich in den beiden folgenden Zeilen zusammenfassen. Diese beiden Zeilen führen die gleichen Aktionen durch und machen dies auch noch schneller:

```
Cells(TotalRow, 1).Resize(1, 7).Font.Bold = True
Cells(1, 1).Resize(1, 7).Font.Bold = True
```

6. Beachten Sie, dass der Makrorekorder alle Zellen auswählt, bevor er den Befehl AutoFit verwendet:

```
Selection.CurrentRegion.Select
Selection.Columns.AutoFit
```

Es ist nicht erforderlich, die Zellen zu markieren, bevor AutoFit aufgerufen wird:

```
Cells(1, 1).Resize(TotalRow, 7).Columns.AutoFit
```

7. Beachten Sie, dass der Makrorekorder am Anfang jedes Makros eine kurze Beschreibung einfügt:

```
' ImportRechnung Makro
```

Sie haben den aufgezeichneten Makrocode in etwas geändert, das tatsächlich funktioniert. Daher können Sie Ihren Namen als Autor des Codes ergänzen und angeben, was das Makro macht:

```
' Erstellt von Bill Jelen. Importiere Rechnung.txt und ergänze Summen.
```

Hier ist das fertige Makro mit allen Änderungen:

```
Sub FormatiereRechnungGefixt()
' Erstellt von Bill Jelen. Importiere Rechnung.txt und ergänze Summen.
Workbooks.OpenText Filename="C:\Data\invoice.txt", Origin:=437, _
  StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote, _
  ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, _
  Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(_
  Array(1, 4), Array(2, 1), Array(3, 1), Array(4, 1), _
  Array(5, 1), Array(6, 1), Array(7, 1))
' Letzte Zeile mit Daten finden. Dies kann sich jeden Tag ändern.
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
TotalRow = FinalRow + 1
' Unterhalb dieser Zeile die Summenzeile erstellen
Cells(TotalRow, 1).Value = "Total"
Cells(TotalRow, 5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
Cells(TotalRow, 1).Resize(1, 7).Font.Bold = True
Cells(1, 1).Resize(1, 7).Font.Bold = True
Cells(1, 1).Resize(TotalRow, 7).Columns.AutoFit
End Sub
```

## Nächste Schritte

---

Mittlerweile sollten Sie wissen, wie man ein Makro aufzeichnet. Sie sollten auch in der Lage sein, die Hilfe sowie die Debugging-Werkzeuge zu verwenden, um herauszufinden, wie der Code funktioniert. In diesem Kapitel haben Sie sieben Werkzeuge kennengelernt, mit denen Sie den aufgezeichneten Code in professionellen Code verwandeln können.

Die nächsten Kapitel gehen detaillierter auf den Verweis auf Bereiche, auf Schleifen und die etwas verrückte, aber nützliche R1C1-Bezugsart für Formeln ein, die der Makrorekorder gerne verwendet.