Bernd Held

Know-how ist blau.

Studienausgabe

VBA-Programmierung

für Word, Excel und Access

> Ergänzen Sie Microsoft Office durch selbst programmierte Funktionen

> Erstellen Sie Meldungen und Dialoge, Menü- und Symbolleisten

> Tauschen Sie Daten zwischen den Office-Applikationen aus

Das Praxisbuch für Microsoft-Office-Entwickler

FRANZIS

Vorwort

Eine der Stärken des Office-Pakets von Microsoft ist, dass Sie es über den Einsatz von Visual Basic for Applications (VBA) erweitern und sogar noch verbessern können. Überall entstehen somit Office-Lösungen in Eigenregie, die den großen, bereits existierenden und oft sehr teuren Standardsoftwareprogrammen trotzen. Diese programmierten Office-Anwendungen können flexibler erstellt und auch jederzeit gepflegt werden. Dabei ist den Anwendern die Office-Oberfläche schon bekannt und die Einführung solcher Lösungen wird dadurch erleichtert.

In diesem Buch werden die wichtigsten Office-Applikationen Excel, Word und Access ausführlich behandelt. Zusätzlich werden Lösungen mit Outlook und PowerPoint präsentiert.

Das Buch ist in drei Teile gegliedert:

- Teil I: Die allgemeinen Office-Themen
- Teil II: Die wichtigsten Office-Komponenten und der Datenaustausch zwischen den einzelnen Office-Programmen
- Teil III: Fehlersuche, Tuning und Office-VBA-FAQ (Frequently Asked Questions, also häufig gestellte Fragen)

Im ersten Teil werden Office umfassende Themen behandelt, die Sie mit kleinen Anpassungen für nahezu alle Office-Komponenten einsetzen können. So lernen Sie im ersten Kapitel, wie Sie die Entwicklungsumgebung in den Office-Komponenten einsetzen können. Im zweiten Kapitel lernen Sie die wichtigsten Sprachelemente wie beispielsweise Schleifen, Verzweigungen und Abfragen kennen. In Kapitel 3 werden VBA-Standardfunktionen des Office-Pakets anhand von Aufgaben aus der täglichen Praxis vorgestellt. Im folgenden Kapitel erfahren Sie, wie Sie Office durch die Programmierung eigener Funktionen weiter bereichern können. In Kapitel 5 erstellen und programmieren Sie benutzerfreundliche UserForms und erfahren, wie Sie bereits integrierte Dialoge im Office-Paket für Ihre Programme einsetzen können. Die Programmierung von Kontextmenüs, Menü- und Symbolleisten ist Thema von Kapitel 6. Dort erfahren Sie, wie Sie diese Objekte ansprechen und programmieren können, um Ihre VBA-Projekte noch benutzerfreundlicher zu machen. In Kapitel 7 dreht sich alles um das Internet. Unter anderem erfahren Sie dort, wie Sie Hyperlinks programmieren, E-Mails verschicken und Web-Abfragen ausführen können. In Kapitel 8 greifen Sie mithilfe von VBA-Makros auf VBA-Projekte zu. Durch diese so genannten VBE-Zugriffe können Sie Quellcode sichern oder entfernen. Des Weiteren besteht die Möglichkeit, Quellcode zu importieren bzw. zu exportieren. Selbstverständlich können Sie über ein Makro auch weitere Makros Zeile für Zeile automatisch erstellen.

Im zweiten Teil wird der Fokus auf die wichtigsten Office-Komponenten gerichtet. In Kapitel 9 lernen Sie die wichtigsten Objekte in Excel wie Zellen, Zeilen, Spalten, Tabellen und Arbeitsmappen anhand zahlreicher Beispiele aus der täglichen Praxis kennen. In Kapitel 10 erstellen Sie Makros in Ihrer Textverarbeitung Word. Unter anderem fügen Sie Bilder in Dokumente ein, erstellen und füllen Tabellen und vieles mehr. In Kapitel 11 lernen Sie die Programmierung der wichtigsten Elemente von Access kennen. Sie erstellen unter anderem

Tabellen und führen Abfragen durch. Eines der wichtigsten Kapitel dieses Buchs stellt das Kapitel 12 dar. Es behandelt den Datenaustausch zwischen den einzelnen Office-Komponenten. In diesem Kapitel steht die Zusammenarbeit der einzelnen Office-Komponenten im Vordergrund. Unter anderem greifen Sie ausgehend von Word auf eine Access-Tabelle zu und übertragen Adressdaten in Ihre Textverarbeitung. Des Weiteren füllen Sie den Kontaktordner von Outlook mit Daten, die Sie aus einer Excel-Tabelle beziehen.

Im dritten Teil des Buchs erfahren Sie in Kapitel 13, wie Sie Fehler in Makros aufspüren und beseitigen können. Allgemeine Ratschläge für das Verhalten im Fehlerfall ergänzen den Inhalt dieses Kapitels. In Kapitel 14 messen Sie die Geschwindigkeit von einzelnen Makros. Schritt für Schritt erfahren Sie, wie Sie Makros noch schneller und benutzerfreundlicher machen können. Im letzten Kapitel dieses Buchs präsentiere ich Ihnen eine Office-FAQ mit Fragen, die in den letzten Jahren an mich gestellt wurden und für die ich eine Lösung zur Verfügung stellen konnte.

In Anhang A des Buchs finden Sie eine Auflistung der Dateien, die Sie auf der Website *www.buch.cd* finden können. Des Weiteren werden im Anhang B alle Listingbeschriftungen in einer Liste zum schnellen Nachschlagen angeboten. Im Index sind alle im Buch verwendeten Methoden, Funktion, Eigenschaften und Anweisungen aufgenommen worden. Sie können somit schnell herausfinden, auf welchen Seiten des Buchs die Befehle verwendet und beschrieben wurden.

In diesem Buch wurden Praxislösugnen in ca. 360 Makros beschrieben. Zögern Sie nicht, mich bei Fragen zum Buch über meine E-Mail-Adresse `held-office@t-online.de` zu kontaktieren.

Ich wünsche Ihnen beim Lesen des Buchs viel Spass und hoffe, dass ich damit einen Beitrag leisten konnte, der Ihnen hilft, erfolgreich Office-Lösungen zu entwickeln.

Bernd Held

Über den Autor:

Bernd Held ist gelernter Informatiker und programmierte drei Jahre lang bei einer Firma der Automobilbranche Warenwirtschafts- und Suchsysteme für den Kfz-Bereich. Danach arbeitete er sechs Jahre beim debis Systemhaus im Controlling. Dort war er verantwortlich für das Berichtswesen, die Leistungsverrechnung, das Erstellen von betrieblichen Auswertungen und Wirtschaftlichkeitsrechnungen sowie für die Entwicklung neuer Controlling-Tools auf der Basis von Microsoft Office. Seit dem 1. Januar 2002 ist Herr Held selbstständig. Er schreibt Fachartikel in renommierten Zeitschriften, verfasst Computerbücher, führt Software-Schulungen durch und programmiert im Auftrag von Kunden. Sein Spezialgebiet ist Microsoft Office. Dort hat er sich auf den Bereich Excel und die Office-VBA-Programmierung spezialisiert. Aber auch über Microsoft Works, FrontPage, Windows und diverse anderer Themen hat er schon viele Bücher geschrieben.

Vor drei Jahren wurde Bernd Held als MVP (Most Valuable Professional) von der Firma Microsoft ausgezeichnet. Dieser Titel wird für besondere fachliche Kompetenz, überdurchschnittlichen Einsatz in den Diskussionsforen und für außergewöhnliches Kommunikationstalent verliehen. Im deutschsprachigen Excel-Forum von Microsoft (news:microsoft.public.de.excel) können Sie Bernd Held des öfternen antreffen, wenn Sie Fragen zu Excel oder zur VBA-Programmierung haben. Dort hilft er Ihnen gerne weiter.

Inhaltsverzeichnis

1 Die Entwicklungsumgebung von VBA

Egal, in welcher Office-Komponente Sie sich gerade befinden – über die Tastenkombination Alt+F11 gelangen Sie direkt in die Entwicklungsumgebung. Da diese Entwicklungsumgebung in nahezu allen Office-Programmen identisch ist, werde ich Ihnen diese Umgebung am Beispiel von Microsoft Word 2002 beschreiben und auf die Besonderheiten der jeweiligen Office-Komponenten eingehen.

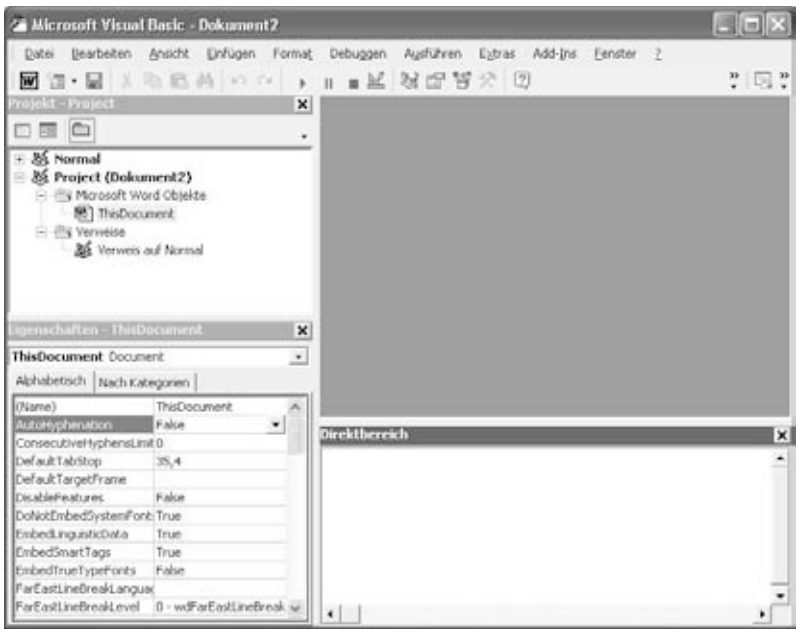


Abb. 1.1 Die Entwicklungsumgebung von Word 2002

1.1 Makros einfügen

Um ein Makro anzulegen, müssen Sie zuerst einmal ein neues Modul anlegen. Dazu wählen Sie in der Entwicklungsumgebung von Word 2002 aus dem Menü EINFÜGEN den Befehl MODUL. Auf der rechten Seite Ihrer Entwicklungsumgebung erscheint nun das so genannte Code-Fenster, in welches Sie Ihre Makros eintippen.

Legen Sie nun Ihr erstes Makro an, indem Sie in das Codefenster das Schlüsselwort `Sub` schreiben. Erfassen Sie danach nach einem Leerschritt einen beliebigen Namen für Ihr Makro, welcher allerdings keine Leer- und Sonderzeichen enthalten darf. Außerdem darf der erste Buchstaben keine Zahl sein. Nach der Benennung des Makros setzen Sie eine öffnenden sowie direkt im Anschluss eine schließende runde Klammer hinter den Namen und bestätigen mit ↵. Die erste Zeile Ihres Makros wird sogleich mit einer End-Sub-Zeile vervollständigt. Sie haben somit den Rumpf für Ihr erstes Makro.

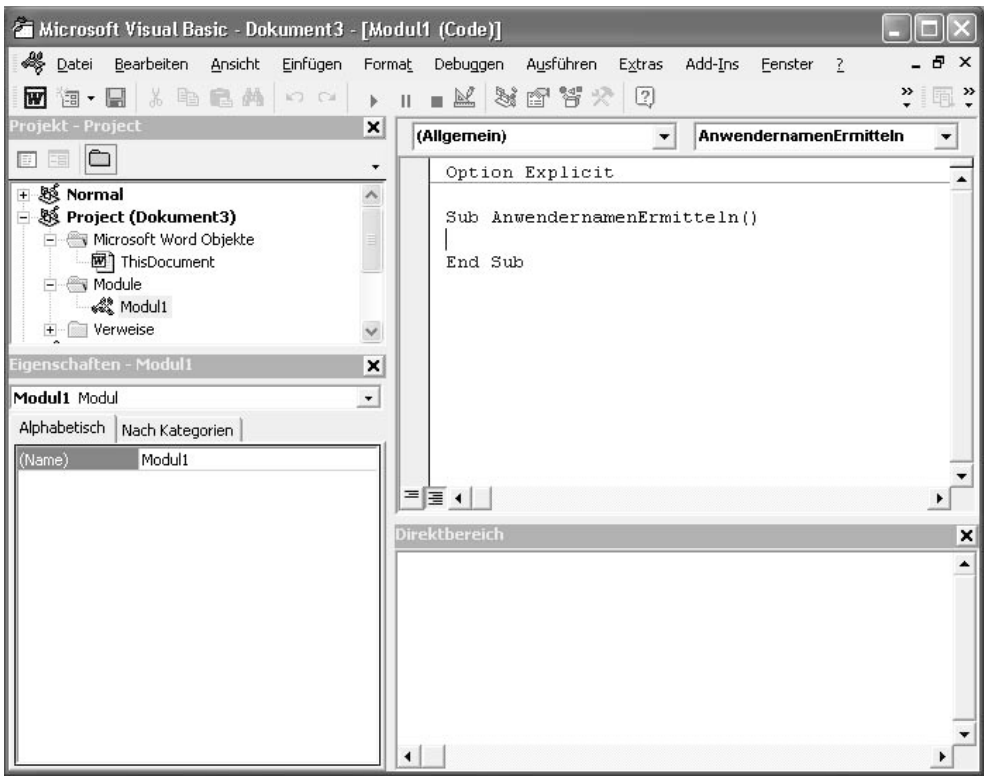


Abb. 1.2 Das erste Makro in Word

Sie können jetzt im ersten Makro ein paar Leerzeilen einfügen, wenn Sie möchten. Dazu setzen Sie den Mauszeiger ans Ende der ersten Zeile des Makros und drücken ein paar Mal die Taste ↵. Das Makro macht im Moment noch nichts. Als Einstiegsaufgabe werden Sie nun den Namen des Anwenders, den Sie übrigens im Menü EXTRAS unter dem Befehl OPTIONEN auf der Registerkarte BENUTZERINFORMATIONEN wiederfinden, auf dem Bildschirm aus. Der Code für diese Aufgabe lautet:

```
Sub AnwendernamenErmitteln()
```

```
 MsgBox Application.UserName
End Sub
```

Mithilfe der Funktion `Msgbox` können Sie eine Meldung auf dem Bildschirm anzeigen. Die Eigenschaft `Application` gibt Ihnen Zugriff auf Ihre Word-Anwendung. Sie haben somit die Möglichkeit, auf die Eigenschaft `UserName` zurückzugreifen, die Ihnen den Namen des Anwenders ermittelt.

1.2 Makros starten

Zum Starten eines Makros haben Sie mehrere Möglichkeiten:

1. In der Entwicklungsumgebung in der Symbolleiste VOREINSTELLUNG mit einem Klick auf das Symbol SUB/USERFORM AUSFÜHREN.
2. Starten eines Makros vom Dokument aus über das Menü EXTRAS und den Befehl MAKRO/MAKROS und die Auswahl des Makros im Listenfeld mit abschließendem Klick auf die Schaltfläche AUSFÜHREN.
3. Starten eines Makros direkt aus der Entwicklungsumgebung im Codefenster, indem Sie den Mauszeiger auf die erste Zeile des Makros setzen und die Taste F5 drücken.

1.3 Den Makrorekorder einsetzen

Haben Sie bisher noch nicht viel mit VBA gemacht, wird es am Anfang nicht einfach sein, den Aufbau der einzelnen Befehle zu erkennen. Um den Einstieg in die VBA-Programmierung zu erleichtern, stehen Ihnen für die Anwendungsprogramme Word und Excel so genannte Makrorekorder zur Verfügung. Mithilfe dieser Makrorekorder können Sie automatisch Quellcode erzeugen und anhand dieser Umsetzung schnell Fortschritte in der Programmierung mit VBA machen. Dabei führen Sie einige Arbeitsschritte manuell aus und lassen den Makrorekorder im Hintergrund mitlaufen. Nach der Aufzeichnung springen Sie in den Quellcode und passen diesen an.

1.3.1 Makro aufzeichnen

Als kleine Vorarbeit legen Sie zunächst einmal ein neues, noch leeres Dokument an und erfassen einen kleinen Beispieltext. Möchten Sie diesen Beispieltext automatisch von Word selbst erzeugen lassen, dann schreiben Sie die Formel =Rand(3,2) und bestätigen mit ↵.

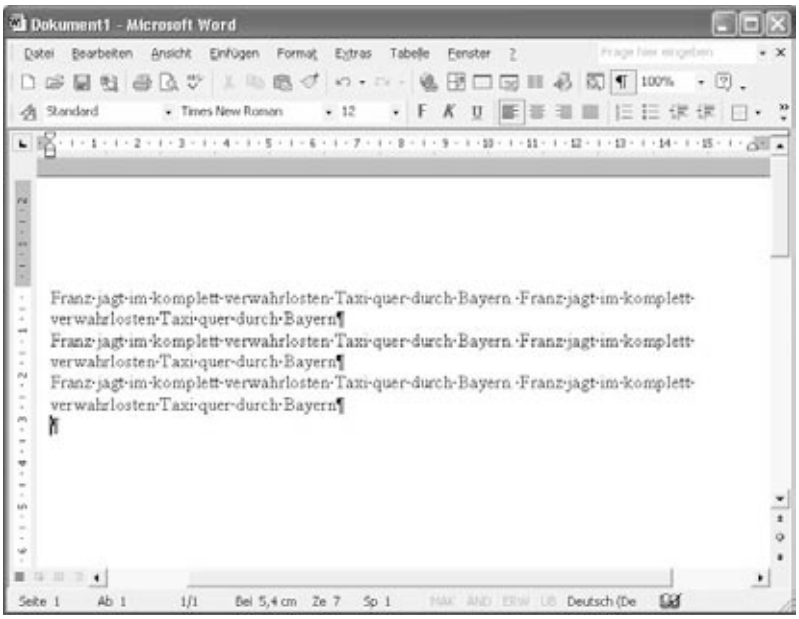


Abb. 1.3 Beispieltext automatisch erzeugen

Wenn Sie sich den Beispielsatz ansehen, dann werden Sie feststellen, dass jeder Buchstabe des Alphabets mindestens einmal darin enthalten ist. Dieser Text ist in diesem Beispiel in der Schriftart TIMES NEW ROMAN eingetragen worden. Ihre Aufgabe ist es nun, die Schriftart des Textes zu ändern. Dabei werden Sie diesen Vorgang mit dem Makrorekorder aufzeichnen lassen. Befolgen Sie dazu die nächsten Arbeitsschritte:

1. Wählen Sie aus dem Menü EXTRAS den Befehl MAKRO/AUFZEICHNEN.

Abb. 1.4 Makro aufzeichnen

2. Geben Sie dem Makro im Feld MAKRONAME einen beschreibenden Namen.
3. Sie haben die Möglichkeit, dieses Makro einem Symbol in einer Symbolleiste bzw. das Makro einer Tastenkombination zuzuweisen. Diese Möglichkeit sollten Sie jedoch beim Testen bzw. bei der Neuanlage des Makros noch nicht einsetzen.
4. Wählen Sie im Dropdownfeld MAKRO SPEICHERN Ihr aktuelles Dokument aus, sofern Sie das Makro nur für dieses Dokument einsetzen möchten. Entscheiden Sie sich hingegen für den Befehl ALLE DOKUMENTE (NORMAL.DOT), dann wird das Makro in der zentralen Dokumentdatei von Word gesichert. Damit kann das Makro für alle Dokumente eingesetzt werden. Sie sollten aber wirklich nur Makros dort hineinnehmen, die Sie wirklich für alle oder zumindest viele Dokumente gebrauchen können.
5. Starten Sie Ihre Aufzeichnung mit OK.
6. Weisen Sie jetzt dem Text im Dokument eine andere Schriftart zu, indem Sie die Tastenkombination Strg+A drücken, um den Text des ganzen Dokuments zu markieren. Wählen Sie danach aus dem Menü FORMAT den Befehl ZEICHEN. Auf der Registerkarte SCHRIFT entscheiden Sie sich für die Schrift ARIAL und bestätigen mit OK. Achtung: Wenn Sie übrigens die Schriftart über die Symbolleiste FORMAT und dem Dropdownfeld SCHRIFTART ändern, dann zeichnet der Makrorekorder diese Aktion nicht auf.
7. Beenden Sie die Aufzeichnung, indem Sie aus dem Menü EXTRAS den Befehl MAKRO/ AUFZEICHNUNG BEENDEN wählen.

Tipp:

Übrigens gibt es in Excel ebenso eine zentrale Arbeitsmappe, in der Sie Makros speichern können, die Sie auch für andere Arbeitsmappen und nicht nur für die aktuelle Mappe verwenden können. Diese Arbeitsmappe heißt PERSONL.XLS und befindet sich im Unterverzeichnis von Office im Ordner XLSTART. Standardmäßig wird diese Arbeitsmappe jedoch nicht automatisch nach der Installation von Office angelegt. Diese Arbeitsmappe können Sie aber recht schnell selbst anlegen, indem Sie ein Makro aufzeichnen und als Speicherort im Dialog MAKRO AUFZEICHNEN aus dem Dropdownfeld MAKRO SPEICHERN den Befehl PERSÖNLICHE MAKROARBEITSMAPPE auswählen.

1.3.2 Resultate des Makrorekorders ansehen

Nun aber zurück zu unserem Word-Beispiel. Sehen Sie sich jetzt den aufgezeichneten Quellcode an, indem Sie über die Tastenkombination Alt+F11 in die Entwicklungsumgebung wechseln.

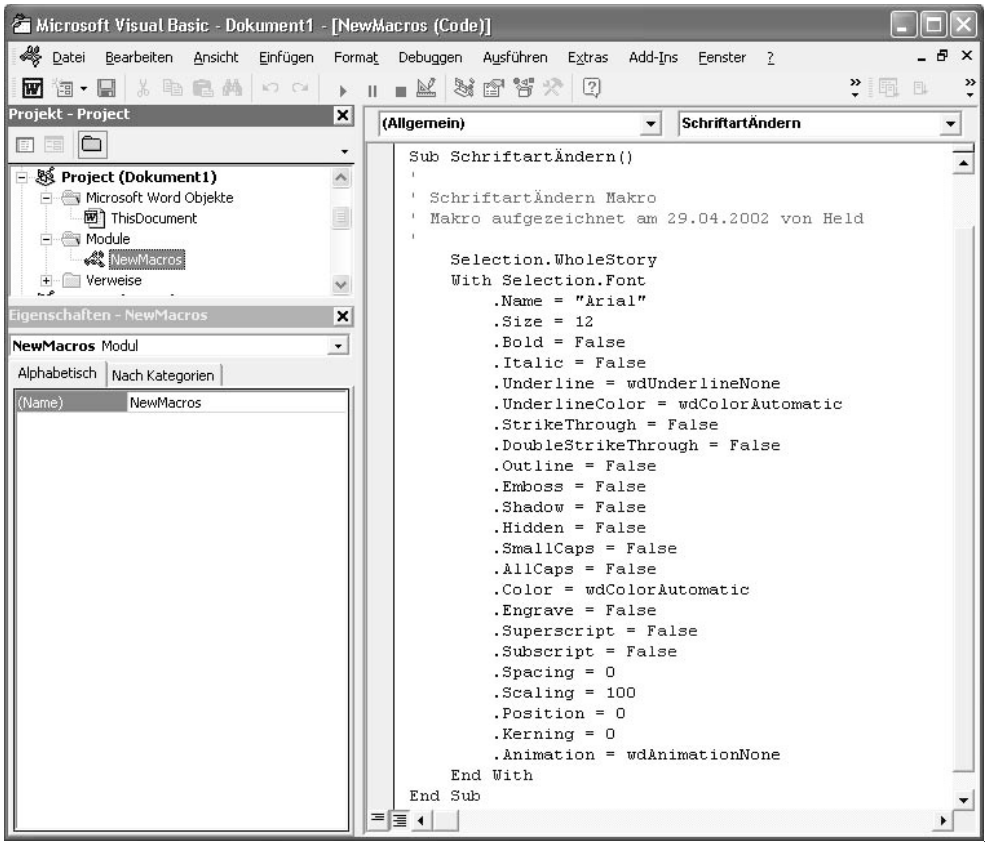


Abb. 1.5 Die Arbeit des Makrorekorders begutachten

Der Makrorekorder hat eine ganze Menge Befehle aufgezeichnet, die eigentlich gar nicht unbedingt gebraucht werden, um die gestellte Aufgabe auszuführen. Standardmäßig werden in den ersten paar Zeilen Informationen über den Anwender eingetragen, der das Makro aufgezeichnet hat sowie der Zeitpunkt der Aufnahme. Diese Informationen beginnen mit einem einfachen Apostroph und werden somit von Word als Kommentar interpretiert. Diese Kommentarzeilen werden beim Makroablauf ignoriert.

Danach wird der Text des gesamten Dokuments mithilfe der Eigenschaft `Selection` markiert, welches Ihnen das Objekt `Selection` (= Markierung) liefert. Auf dieses Objekt werden dann einige Eigenschaften gesetzt, die das Objekt näher beschreiben. So wird beispielsweise die Eigenschaft `Name` abgefragt, um die gewünschte Schriftart des Textes in der Markierung zu bestimmen. Neben der Schriftgröße, die über die Eigenschaft `Size` festgelegt wird, werden Informationen zum Schriftschnitt über die Eigenschaften `Italic` und `Bold` (kursiv und fett) vorgenommen, die für diese Aufgabe nicht benötigt und zu diesem Zweck mit dem Wert `False` ausgestattet werden. Weitere Eigenschaften wie `Underline` und `UnderlineColor` (Unterstreichung und Farbe für die Unterstreichung) werden mit so

genannten Konstanten belegt. Diese Konstanten sind in VBA festgelegt und müssen daher genauso angegeben werden.

Mithilfe der Anweisung `With` können Sie sich eine ganze Menge Schreibarbeit sparen. Über diese Anweisung können Sie eine Reihe von Anweisungen für ein nachfolgendes Objekt ausführen, ohne dieses immer wieder zu benennen. Sie benennen das Objekt `Font` im Beispiel einmal zu Beginn und setzen danach anstatt des Objekts einfach einen Punkt. Schließen Sie die Gültigkeit von `with` mit dem Befehl `End With` ab.

1.3.3 Zusatzinformationen anzeigen

Nachdem Sie den aufgezeichneten Quellcode einmal näher angesehen haben, würde ich Ihnen empfehlen, die Online-Hilfe heranzuziehen, um noch mehr Informationen über die verwendeten Befehle, Objekte und Eigenschaften zu bekommen. Setzen Sie zu diesem Zweck den Mauszeiger auf das Wort, zu dem Sie nähere Informationen wünschen und drücken die Taste F1.

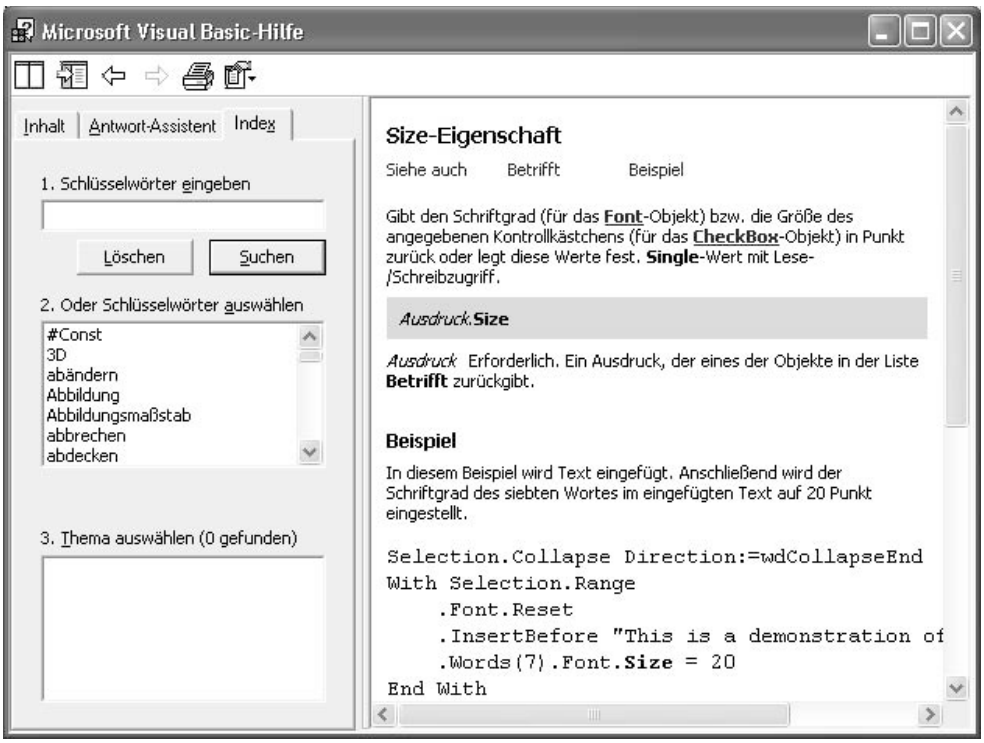


Abb. 1.6 Hilfetexte zur Eigenschaft Font

Die so aufgerufene Hilfe bietet Ihnen eine detaillierte Erklärung des Befehls sowie auch sehr oft ein Beispielmakro an, welches Sie aus der Hilfe kopieren und in Ihr Modul einfügen kön-

4 Eigene Funktionen programmieren

Lernen Sie in diesem Kapitel, wie Sie zusätzliche Funktionen in Ihre Office-Anwendungen bringen können. Eigene Funktionen sind dann sinnvoll, wenn beispielsweise immer wieder gleiche Codeteile geschrieben werden müssen. Anstatt diese immer wieder neu zu schreiben, packen Sie diese Programmteile in eine einzige Funktion und rufen diese bei Bedarf aus Ihren Modulen auf. Der Pflegeaufwand dieser Funktion ist dabei leichter, da eine Änderung immer nur an einer Stelle, nämlich in der Funktion selbst, erfolgt. Sie ersparen sich die lästige Suche und Anpassungen im gesamten Quellcode.

4.1 Farbige Zellen addieren

Im ersten Beispiel zu diesem Kapitel erstellen Sie eine Funktion, über die Sie die Inhalte farbiger Zellen in Excel-Tabellen summieren können.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.xls finden.

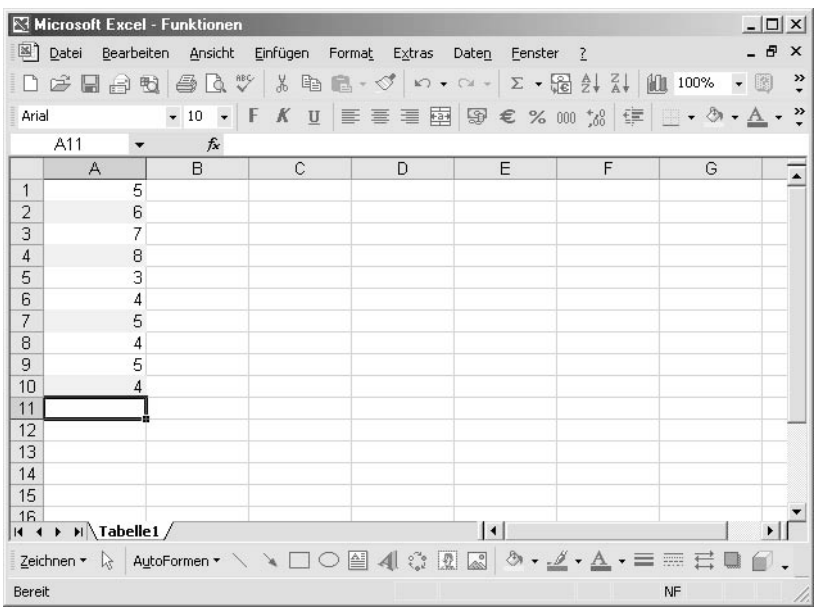


Abb. 4.1 Alle hellgrauen Zellen sollen summiert werden

Erfassen Sie zunächst die Funktion, die die farbigen Zellen summieren soll.

```
Function FarbenSumme(Bereich As Range, CI As Integer) _
                     As Double
Dim zelle As Range

Application.Volatile
 For Each zelle In Bereich
   If zelle.Interior.ColorIndex = CI Then
       FarbenSumme = FarbenSumme + zelle.Value
   End If
 Next
End Function
```

Listing 4.1: Farbige Zellen summieren

Jede Funktion beginnt mit dem Schlüsselwort `Function`. Danach folgt der Name der Funktion, gefolgt von den erwarteten Argument(en), die der Funktion übergeben werden müssen. In unserem Beispiel wird ein Zellenbereich an die Funktion übergeben, in dem die Summierung stattfinden soll. Zusätzlich wird noch ein Farbindex erwartet, der die Farbe bekannt gibt. Alle Zellen, die sich in dem angegebenen Bereich befinden und als Hintergrundfarbe dem übergebenen Farbindex entsprechen, werden somit summiert. Die Summierung wird innerhalb der Funktion durchgeführt.

Was nun noch fehlt, ist das Makro, welches die soeben erstellte Funktion aufruft.

```
Sub ZahlenInGefärbtenZellenAddieren()
 MsgBox FarbenSumme(Range("A1:A10"), 35)
End Sub
```

Listing 4.2: Die Funktion wird aufgerufen

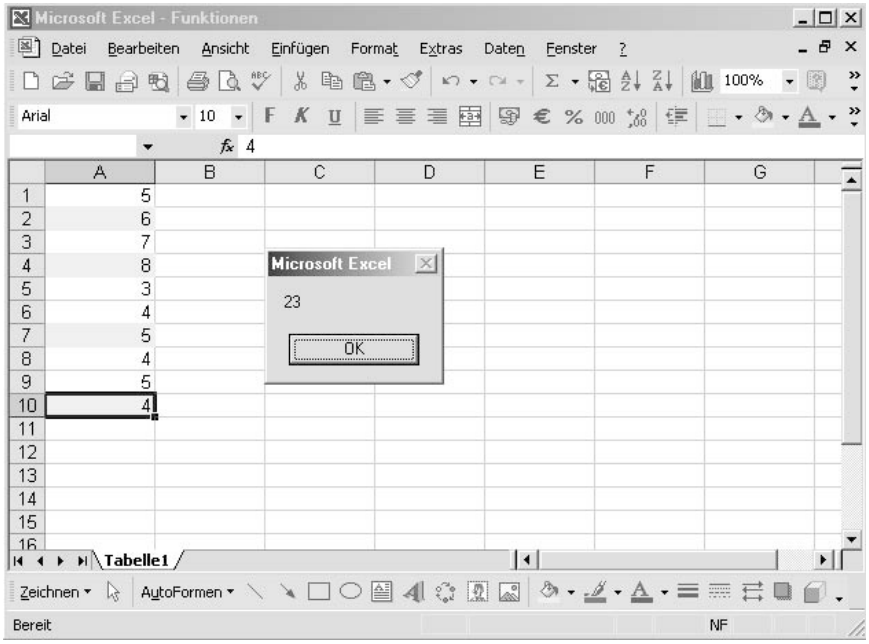


Abb. 4.2 Alle Zellwerte mit hellgrünem Hintergrund werden summiert

4.2 Dateiprüfung

Bevor Sie eine Datei per VBA öffnen, sollten Sie prüfen, ob diese auch vorhanden ist. Im nächsten Beispiel prüfen Sie, ob das Dokument Dok2.doc in einem bestimmten Verzeichnis existiert. Wenn ja, öffnen Sie das Dokument.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.doc finden.

Erfassen Sie zunächst die folgende Funktion:

```
Function DatExist(s As String) As Boolean
   DatExist = False
   If Len(s) > 0 Then DatExist = (Dir(s) <> "")
   Exit Function
End Function
```

Listing 4.3: Die Funktion zur Überprüfung der Existenz einer Datei

Die Funktion DatExist erwartet einen String, der sowohl den Namen der Datei als auch den Pfad enthält. Wird dieser String nicht bzw. leer übergeben, meldet die Funktion Len den Wert 0, was zum Abbruch der Funktion führt. Wird der Dateiname samt dem Pfad richtig an die Funktion übergeben, wenden Sie die Funktion Dir an, die Sie wahrscheinlich noch aus alten DOS-Tagen her kennen. Meldet diese Funktionen einen Wert ungleich Leer zurück, wurde die Datei gefunden.

Erfassen Sie nun das Makro, welches den Pfad- und Dateinamen der Funktion DatExist übergibt.

```
Sub DokDA()
Const Datei = "D:\eigene Dateien\Dok1.doc"

s = DatExist(Datei)
If s = "Wahr" Then
Documents.Open Datei
Else
MsgBox "Die Datei ist nicht vorhanden!"
End If
End Sub
```

Listing 4.4: Datei- und Pfadnamen werden an die Funktion DatExist übergeben

Im Makro DokDa wird das gewünschte Dokument, dessen Existenz geprüft werden soll, in einer Konstanten gleich zu Beginn des Makros angegeben. Diese Konstante wird der Funktion DatExist übergeben. Die Funktion ihrerseits liefert einen Wahrheitswert an die aufrufende Prozedur zurück, die Sie noch auswerten müssen. War die Suche nach dem Dokument erfolglos, wird der Wert False zurückgeliefert. Im anderen Fall, wenn also das gesuchte Dokument gefunden wird, wird der Wert True von der Funktion an die aufrufende Prozedur zurückgegeben. In diesem Fall wenden Sie die Methode Open an, um das Dokument zu öffnen.

4.3 Daten bereinigen

Oft sind einfach einige Zeichen in Daten unerwünscht, die Sie gerne herausnehmen würden. Ob es nun die schrägen Striche zwischen einer Vorwahl und einer Rufnummer, Punkte, Bindestriche oder Kommas sind, ist je nach Anwendungsfall unterschiedlich. Im folgenden Makro sollen in einer Excel-Tabelle in einem bestimmten Bereich alle Daten bereinigt werden. Die Ausgangstabelle sieht dabei wie folgt aus:

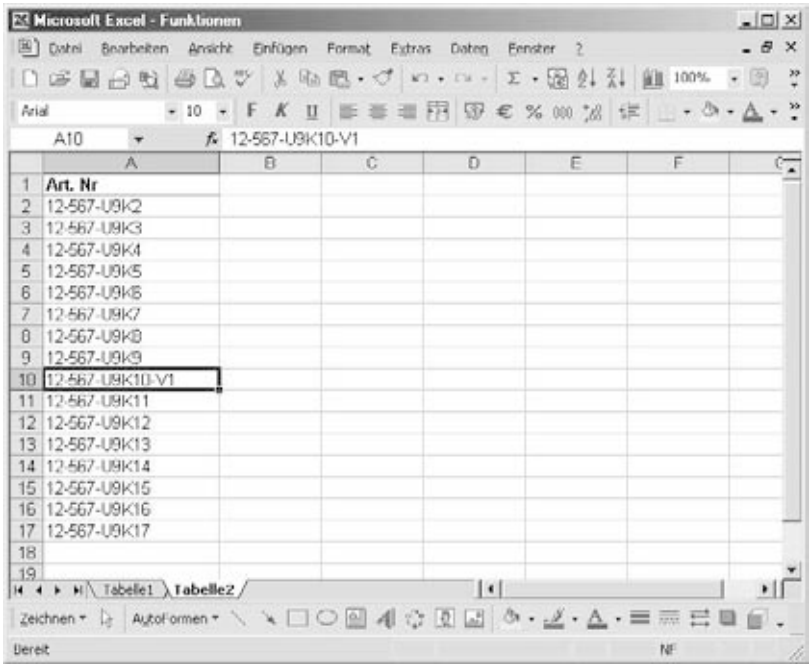


Abb. 4.3 Die Ausgangstabelle

In der Tabelle sollen nun alle Bindestriche entfernt werden.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.xls finden.

```
Function BindestrRaus(s As String) As String
Dim Puffer As String
Dim i As Integer

Puffer = ""
i = 1
While InStr(i, s, "-") > 0
    Puffer = Puffer & Mid(s, i, InStr(i, s, "-") - i)
    i = InStr(i, s, "-") + 1
Wend
Puffer = Puffer & Mid(s, i)
BindestrRaus = Puffer
End Function
```

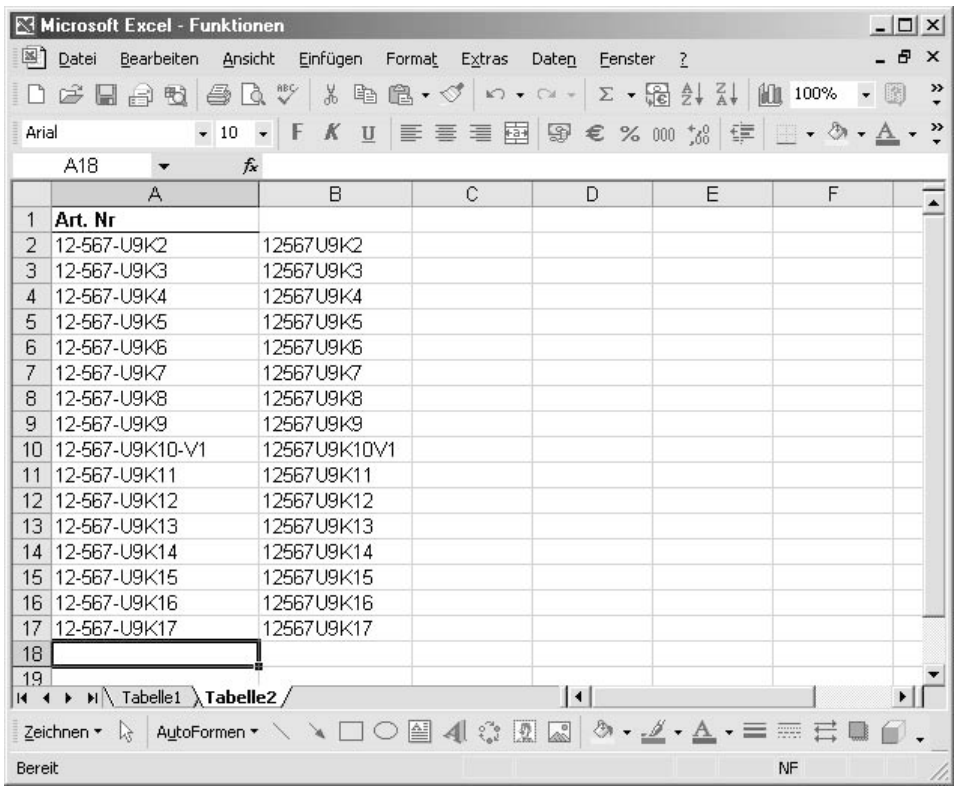
Listing 4.5: Eine Funktion, die alle Bindstriche entfernt

In der Funktion durchlaufen Sie eine Schleife, in der die Zeichen jeweils bis zum nächsten Bindestrich in den String `Puffer` übertragen werden. Dabei wird der Bindestrich aber nicht übertragen. Ermitteln Sie danach die Position des nächsten Bindestrichs über die Funktion `InStr`.

Erfassen Sie nun noch die aufrufende Prozedur und übergeben der Funktion `BindestrRaus` jeweils die gewünschte Zelle, aus der die Bindestriche entfernt werden sollen.

```
Sub TabelleBereinigen()
Sheets("Tabelle2").Activate
Range("A2").Select
Do Until ActiveCell.Value = ""
 ActiveCell.Offset(0, 1).Value = _ BindestrRaus(ActiveCell.Value)
 ActiveCell.Offset(1, 0).Select
Loop
End Sub
```

Listing 4.6: Die Tabelle2 wird bereinigt



	A	B
1	**Art. Nr**	
2	12-567-U9K2	12567U9K2
3	12-567-U9K3	12567U9K3
4	12-567-U9K4	12567U9K4
5	12-567-U9K5	12567U9K5
6	12-567-U9K6	12567U9K6
7	12-567-U9K7	12567U9K7
8	12-567-U9K8	12567U9K8
9	12-567-U9K9	12567U9K9
10	12-567-U9K10-V1	12567U9K10V1
11	12-567-U9K11	12567U9K11
12	12-567-U9K12	12567U9K12
13	12-567-U9K13	12567U9K13
14	12-567-U9K14	12567U9K14
15	12-567-U9K15	12567U9K15
16	12-567-U9K16	12567U9K16
17	12-567-U9K17	12567U9K17
18		
19		

Abb. 4.4 In der Nebenspalte werden die bereinigten Werte eingefügt

4.4 Aktive Zelle im Zielbereich?

Im folgenden Beispiel definieren Sie einen festen Bereich z.B. A1:D10. Ihre Aufgabe besteht nun darin zu überprüfen, ob die momentan aktive Zelle in diesem Bereich liegt.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.xls finden.

```
Function BereichsP(Bereich1 As Range, Bereich2 As Range) _
                  As Boolean
Dim SM As Range

Set SM = Application.Intersect(Bereich1, Bereich2)
BereichsP = Not SM Is Nothing
Set SM = Nothing
End Function
```

Listing 4.7: Liegt die aktive Zelle im Zielbereich?

Die Funktion `BereichsP` erwartet zwei Argumente. Im ersten Argument übergeben Sie die Adresse der aktiven Zelle, im zweiten Argument übergeben Sie der Funktion den definierten Zielbereich.

Über die Methode `Intersect` prüfen Sie, ob zwei Bereiche sich überschneiden, also die aktive Zelle im definierten Bereich liegt. Wenn ja, dann steht in der Variablen SM ein gültiger Wert, den Sie auswerten können. Wenn nicht, dann trifft die Bedingung `Nothing` zu. Was jetzt noch fehlt, ist die aufrufende Prozedur.

```
Sub AktiveZelleWo()
If BereichsP(ActiveCell, Range("A1:D10")) = True Then
    MsgBox "Die aktive Zelle " & ActiveCell.Address & _
    " liegt im Zielbereich!"
Else
    MsgBox "Die aktive Zelle " & ActiveCell.Address & _
    " liegt nicht im Zielbereich!"
End If
End Sub
```

Listing 4.8: Zielbereich und zu überprüfende Zelle werden übergeben

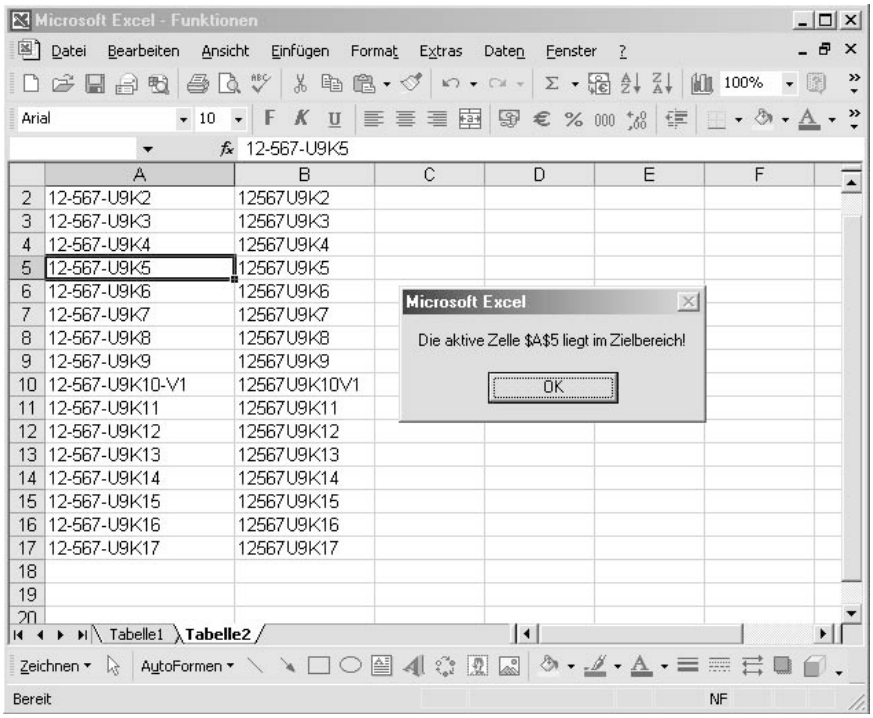


Abb. 4.5 Die aktive Zelle liegt im definierten Bereich

4.5 Dokumentschutz aufheben und neu setzen

Haben Sie Ihre Word-Dokumente mit einem Kennwort geschützt und möchten Sie dieses durch ein anderes Kennwort ändern, müssen Sie normalerweise alle Dokumente einzeln öffnen, den Dokumentschutz im Menü EXTRAS aufheben und diesen dann neu setzen. Dabei müssen Sie zuerst das alte Kennwort eingeben und danach das neue Kennwort festlegen und bestätigen. Diesen Aufwand können Sie dann minimieren, wenn Sie gewöhnlich dieselben Kennwörter benutzen. Schreiben Sie jetzt eine Funktion, das alle Dokumente, die unter den Namen Dok*.doc fallen und sich im Verzeichnis D:\EIGENE DATEIEN befinden, öffnet, das alte Kennwort übergibt und ein neues Kennwort einstellt.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.doc finden.

Die Funktion für diese Aufgabe entnehmen Sie dem folgenden Listing:

```
Function PassWortTauschen(AltesPW As String, _
NeuesPW As String) As Boolean
Dim DOK As Word.Document
Dim i As Integer

With Application.FileSearch
    .NewSearch
    .FileType = msoFileTypeWordDocuments
    .FileName = "Dok*.doc"
    .LookIn = "D:\Eigene Dateien"
    .SearchSubFolders = False
    .Execute
 For i = 1 To .FoundFiles.Count
   Set DOK = Documents.Open(.FoundFiles(i))
    With DOK
     If .ProtectionType <> wdNoProtection Then
      .Unprotect AltesPW
      .Protect Type:=wdAllowOnlyRevisions, Password:=NeuesPW
      Else
      .Protect Type:=wdAllowOnlyRevisions, Password:=NeuesPW
      End If
      .Close wdSaveChanges
       End With
    Next
End With
End Function
```

Listing 4.9: Dokumentschutz aufheben und neu setzen

Wenden Sie das Objekt `FileSearch` an, um die Dokumente zu suchen. Dieses Objekt bietet Ihnen einige Eigenschaften und Methoden an, die Sie einsetzen müssen, um Ihre Suche näher zu spezifizieren.

Über die Methode `NewSearch` setzen Sie die Einstellungen aller Suchkriterien sicherheitshalber auf die Standardeinstellungen zurück.

Über die Eigenschaft `FileType` legen Sie den Typ der Dateien fest, nach denen Sie suchen möchten. Da Sie in diesem Beispiel nach Word-Dokumenten suchen, versorgen Sie diese Eigenschaft mit der Konstanten `msoFileTypeWordDocuments`.

Mithilfe der Eigenschaft `FileName` legen Sie den Namen der gesuchten Dokumente fest. Dabei können Sie auch so genannte »WildCards« verwenden, um die Suche breiter auszulegen. So werden über den String "Dok*.doc" alle Dokumente gesucht, die mit den Buchstaben »Dok« beginnen und mit ».doc« enden.

Über die Eigenschaft `LookIn` legen Sie den Startordner fest, in dem mit der Suche begonnen werden soll.

Setzen Sie die Eigenschaft `SearchSubFolders` auf den Wert `True`, wenn auch in untergeordneten Verzeichnissen, die in unserem Beispiel also unterhalb des Verzeichnisses D:\Eigene Dateien liegen, gesucht werden soll.

Starten Sie letztendlich die Suche über die Methode `Execute`. Damit wird die Suche nach den angegebenen Dateien begonnen. Die Methode liefert einen Long-Wert zurück (null (0), wenn keine Dateien gefunden werden, eine positive Zahl, wenn eine oder mehrere Dateien gefunden werden).

Nach der Suche finden Sie die Ergebnisse im Objekt `FoundFiles`. Über die Eigenschaft `Count` ermitteln Sie die genaue Anzahl der gefundenen Dokumente. Setzen Sie im Anschluss daran eine Schleife auf, die ein Dokument nach dem anderen über die Methode `Open` öffnet, den alten Dokumentschutz bei Bedarf entfernt und durch den neuen Kennwortschutz ersetzt. Schließen Sie nach diesem Vorgang das Dokument, indem Sie die Methode `Close` einsetzen. Damit die Änderungen wirksam werden, geben Sie die Konstante `wdSaveChanges` an.

Was jetzt noch fehlt, ist die aufrufende Prozedur, welche Sie nun erfassen.

```
Sub DoksSchützen()
 b = PassWortTauschen("Test", "Test2")
End Sub
```

Listing 4.10: Dokumente werden mit neuem Kennwort versehen

4.6 Römische Zahlen wandeln

Da es nicht einfach ist, römische Zahlen in arabische umzuwandeln, schreiben Sie eine Funktion, die Sie im nächsten Listing sehen können.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.xls finden.

```
Function Arabisch(s As String) _
As Integer
Dim i    As Integer
Dim TeilW   As Integer
```

Listing 4.11: Römische Zahlen umwandeln

```
Dim TeilW2  As Integer
Dim GesamtW As Integer

GesamtW = 0
TeilW = 0
TeilW2 = 0

  For i = 1 To Len(s)
    Select Case Mid(s, i, 1)
      Case Is = "M"
        TeilW = 1000
      Case Is = "D"
        TeilW = 500
      Case Is = "C"
        TeilW = 100
      Case Is = "L"
        TeilW = 50
      Case Is = "X"
        TeilW = 10
      Case Is = "V"
        TeilW = 5
      Case Is = "I"
        TeilW = 1
      Case Else
        TeilW = 0
    End Select
    If TeilW2 < TeilW Then
    GesamtW = _
     GesamtW - TeilW2 * 2 + TeilW
    Else
     GesamtW = GesamtW + TeilW
    End If
    TeilW2 = TeilW
  Next i

  Arabisch = GesamtW
End Function
```

Listing 4.11: Römische Zahlen umwandeln

Testen Sie die Funktion anhand einer neuen Tabelle, in der Sie einmal ein paar römische Zahlen erfassen.

Schreiben Sie danach das Makro, welches die Funktion `Arabisch` aufruft.

```
Sub RomNAchArab ()
Sheets("Tabelle3").Activate
Range("A1").Select
Do Until ActiveCell.Value = ""
 ActiveCell.Offset(0, 1).Value = Arabisch(ActiveCell.Value)
 ActiveCell.Offset(1, 0).Select
Loop
End Sub
```

Listing 4.12: In einer Tabelle werden römische Zahlen umgesetzt

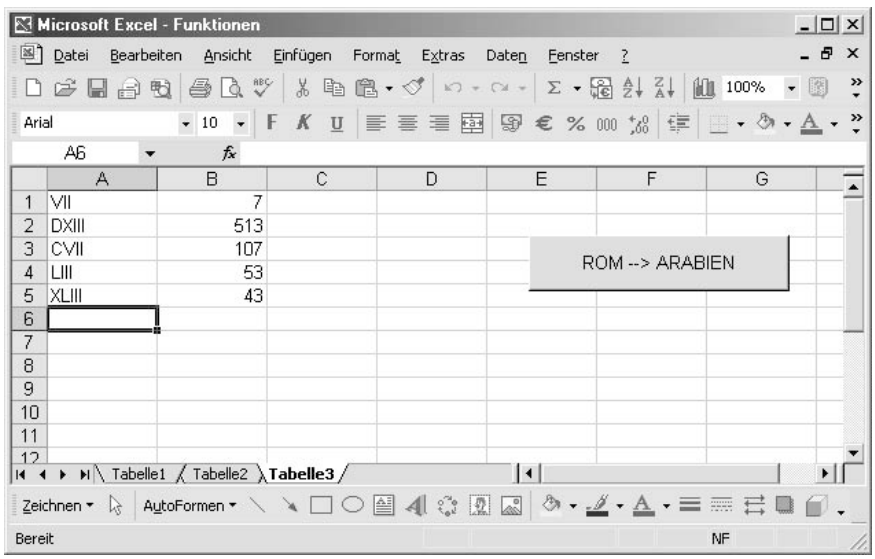


Abb. 4.6 Die römischen Zahlen wurden in Spalte B umgewandelt

4.7 Eingefügte Objekte in PowerPoint-Folien ermitteln

Beim Einfügen von Objekten in eine PowerPoint-Präsentation haben Sie die Möglichkeit, die Verknüpfung zur Quelldatei des Objekts beizubehalten. So werden Änderungen der Quelldatei auch in der PowerPoint-Präsentation durchgeführt.

Hinweis:

Den folgenden Quellcode können Sie auf *www.buch.cd* im Verzeichnis Kap04 unter dem Namen Funktionen.ppt finden.

4.7.1 Einzelne Verknüpfung gezielt aufspüren

Im folgenden Beispiel soll die Verknüpfungsadresse eines eingefügten Excel-Objekts in einer PowerPoint-Präsentation ermittelt werden. Kopieren Sie dazu einmal aus einer Excel-Arbeitsmappe einen Zellenbereich, starten PowerPoint und wählen aus dem Menü BEARBEITEN den Befehl INHALTE EINFÜGEN.

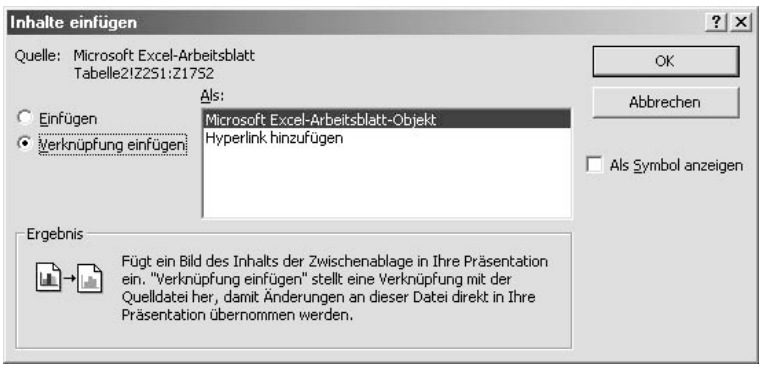


Abb. 4.7 Ein Excel-Objekt verknüpft einfügen

Aktivieren Sie die Optionsschaltfläche VERKNÜPFUNG EINFÜGEN und bestätigen Sie mit OK.

Erfassen Sie in der Entwicklungsumgebung von PowerPoint jetzt die Funktion, die die Verknüpfungsadresse des Objekts ermitteln soll.

```
Function Verknüpfung(obj As PowerPoint.Shape) As String
On Error GoTo Fehler
Verknüpfung = obj.LinkFormat.SourceFullName
Exit Function

Fehler:
Verknüpfung = ""
End Function
```

Listing 4.13: Adresse des verknüpften Objekts ermitteln

Die Eigenschaft `LinkFormat` liefert Ihnen alle verknüpften OLE-Objekte. Mithilfe der Eigenschaft `SourceFullName` können Sie den kompletten Namen samt Pfad des verknüpften Objekts ermitteln.

Schreiben Sie nun noch die Prozedur, die die Funktion Verknüpfung aufruft. Übergeben Sie dabei die gewünschte Folie, auf der sich das verknüpfte Objekt befindet und die Nummer des Objekts.

```
Sub VerknüpfungAufspüren()
Dim obj As PowerPoint.Shape

Set obj = ActivePresentation.Slides(1).Shapes(1)
MsgBox Verknüpfung(obj)
End Sub
```

Listing 4.14: Foliennummer und Objektnummer werden der Funktion übergeben

Über die Anweisung `ActivePresentation.Slides(1).Shapes(1)` greifen Sie auf das erste eingefügte Ole-Objekt in der ersten Folie der aktiven Präsentation zu. Übergeben Sie diese Information an die Funktion `Verknüpfung`.

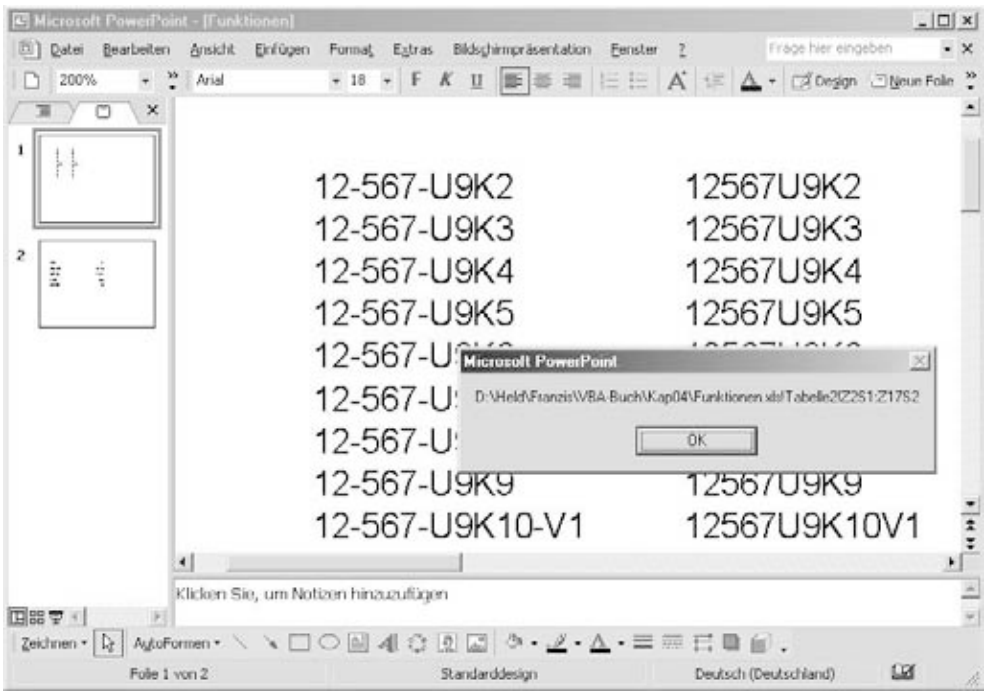


Abb. 4.8 Die Verknüpfungsadresse des OLE-Objekts wurde ermittelt

4.7.2 Alle eingefügten Objekte aufspüren

Die gerade vorgestellte Funktion können Sie noch weiter ausbauen, indem Sie die Verknüpfungen aller in der Präsentation enthaltenen Ole-Objekte ermitteln und anzeigen. Die Funktion für diese Aufgabe lautet:

```
Function AlleVerkn(PräsName As String) As String
Dim s As String
Dim sh As Shape
Dim i As Integer

For i = 1 To ActivePresentation.Slides.Count
For Each sh In ActivePresentation.Slides(i).Shapes
    If sh.Type = msoLinkedOLEObject Then
        With sh.LinkFormat
            s = s & Chr(13) & .SourceFullName
        End With
    End If
Next
Next i
AlleVerkn = s
End Function
```

Listing 4.15: Alle Verknüpfungsadressen der Präsentation werden angezeigt

Die Funktion `AlleVerkn` erwartet den Namen der Präsentation, von der Sie die Adressen aller verknüpften Ole-Objekte erhalten möchten. Als Ergebnis bekommen Sie diese in einer String-Variablen gemeldet. Innerhalb der Funktion selbst ermitteln Sie im ersten Schritt die Anzahl der eingesetzten Folien der Präsentation mithilfe der Eigenschaft `Count`. Diese Anzahl bildet auch das Endekriterium für die erste Schleife in der Funktion. Es sollen somit alle Folien der Präsentation abgearbeitet werden. In der zweiten Schleife werden alle Objekte in der jeweiligen Folie durchforstet. Ergibt die Prüfung über die Eigenschaft `Type`, dass es sich um ein eingefügtes Ole-Objekt handelt, geben Sie über die Eigenschaft `LinkFormat` die Adresse der Verknüpfung an eine String-Variable weiter. Setzen Sie die Anweisung `Chr(13)` ein, um die einzelnen Adressen ordendlich untereinander zu schreiben. Geben Sie am Ende der Funktion den so gebildeten String, der jetzt die Verknüpfungsadressen aller eingefügten Ole-Objekte beinhaltet, an die Prozedur zurück.

Erfassen Sie jetzt die aufrufende Prozedur.

```
Sub AlleOLEermitteln()
 MsgBox AlleVerkn(ActivePresentation.Name)
End Sub
```

Listing 4.16: Der Funktion AlleVerkn wird der Namen der aktiven Präsentation übergeben

8 VBE-Programmierung in Office

Unter der VBE-Programmierung versteht man den Zugriff auf die Entwicklungsumgebung in VBA. Damit können Sie beispielsweise ermitteln, welche Module in Ihren VBA-Projekten enthalten sind. Diese Module, UserForms oder sonstige VBE-Bestandteile können Sie des weiteren exportieren bzw. importieren. Ferner können Sie benötigte VBA-Bibliotheken elegant per Makro einbinden und vieles mehr. Dieses klassische Thema wird anhand einiger Beispiele in Word beschrieben.

Hinweis:

Die folgenden Makros können Sie auf *www.buch.cd* im Verzeichnis Kap08 unter dem Namen VBE.doc finden.

8.1 Die Voraussetzung

Möchten Sie VBE-Programmieren, dann müssen Sie ab der XP-Office-Version eine sehr wichtige Einstellung vornehmen. Gehen Sie dazu wie folgt vor:

- Wählen Sie aus dem Menü EXTRAS den Befehl MAKRO/SICHERHEIT.
- Wechseln Sie auf die Registerkarte VERTRAUENSWÜRDIGE QUELLEN.

Abb. 8.1 Vertrauenswürdige Quellen aktivieren

- Aktivieren Sie das Kontrollkästchen ZUGRIFF AUF VISUAL BASIC-PROJEKT VERTRAUEN.
- Bestätigen Sie mit OK.

Erst nach dieser Einstellung können Sie die Programmierung mit VBE beginnen.

8.2 Bibliotheken einbinden

Die VBE-Programmierung ist seit der Office-Version 97 möglich. Dazu benötigen Sie die Objektbibliothek MICROSOFT VISUAL BASIC FOR APPLICATION EXTENSIBILITY 5.3. Diese binden Sie ein, indem Sie in der Entwicklungsumgebung aus dem Menü EXTRAS den Befehl VERWEISE wählen.

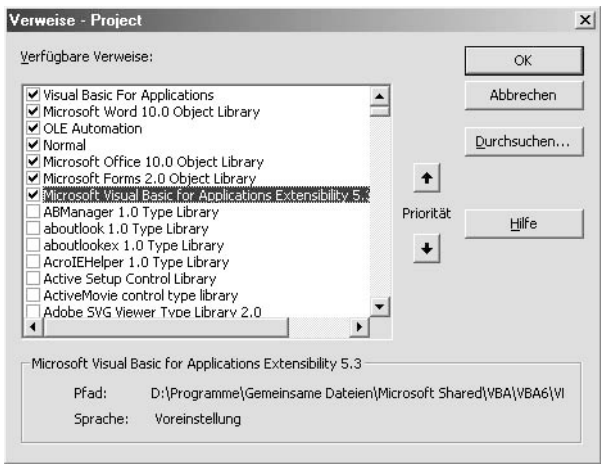


Abb. 8.2 Zusätzliche Bibliotheken einbinden

Im Listenfeld VERWEISE – PROJECT aktivieren Sie im Listenfeld VERFÜGBARE VERWEISE die Bibliothek MICROSOFT VISUAL BASIC FOR APPLICATION EXTENSIBILITY 5.3 und bestätigen mit OK.

Verweise können Sie selbstverständlich auch über den Einsatz eines VBA-Makros setzen bzw. entfernen. Dazu brauchen Sie aber noch einige Informationen darüber.

8.2.1 Gesetzte Verweise ermitteln

Das folgende Makro schreibt diese Informationen in eine Tabelle eines neuen Word-Dokuments.

```
Sub GesetzteVerweiseAbfragen()
Dim ref As Reference
Dim Tabelle As Table
Dim NeuDoku As Document
Dim i As Integer

On Error Resume Next
Set NeuDoku = Documents.Add
Set Tabelle = NeuDoku.Tables.Add(Selection.Range, 20, 4)
i = 1
With Tabelle
  .Cell(i, 1).Range.InsertAfter "Verweis"
  .Cell(i, 2).Range.InsertAfter "GUID"
  .Cell(i, 3).Range.InsertAfter "Major"
  .Cell(i, 4).Range.InsertAfter "Minor"
End With
For Each ref In Application.VBE.ActiveVBProject.References
  With Tabelle
   .Cell(i + 1, 1).Range.InsertAfter ref.Name
   .Cell(i + 1, 2).Range.InsertAfter ref.GUID
   .Cell(i + 1, 3).Range.InsertAfter ref.Major
   .Cell(i + 1, 4).Range.InsertAfter ref.Minor
   i = i + 1
  End With
Next ref
End Sub
```

Listing 8.1: Die gesetzten Verweise in eine Tabelle schreiben

Über die Eigenschaft `Name` können Sie den Namen des Verweises erfahren. Die Eigenschaft `Guid` gibt einen Wert vom Typ `String` zurück, der die Klassen-ID eines Objekts enthält. Bei der `Guid` handelt es sich um eine eindeutige Nummer, welche die Bibliothek identifiziert. Das Argument `Major` gibt einen Wert vom Typ `Long` zurück, der die Hauptversionsnummer der Klassenbibliothek, auf die verwiesen wird, enthält. Das Argument `Minor` gibt einen Wert vom Typ `Long` zurück, der die Nebenversionsnummer der Klassenbibliothek, auf die verwiesen wird, anzeigt. Beide Nummern sind notwendig, um die Bibliothek richtig zu adressieren. Anhand dieser beiden Nummern kann in der Registrierung der hinzuzufügende Verweis ermittelt werden.

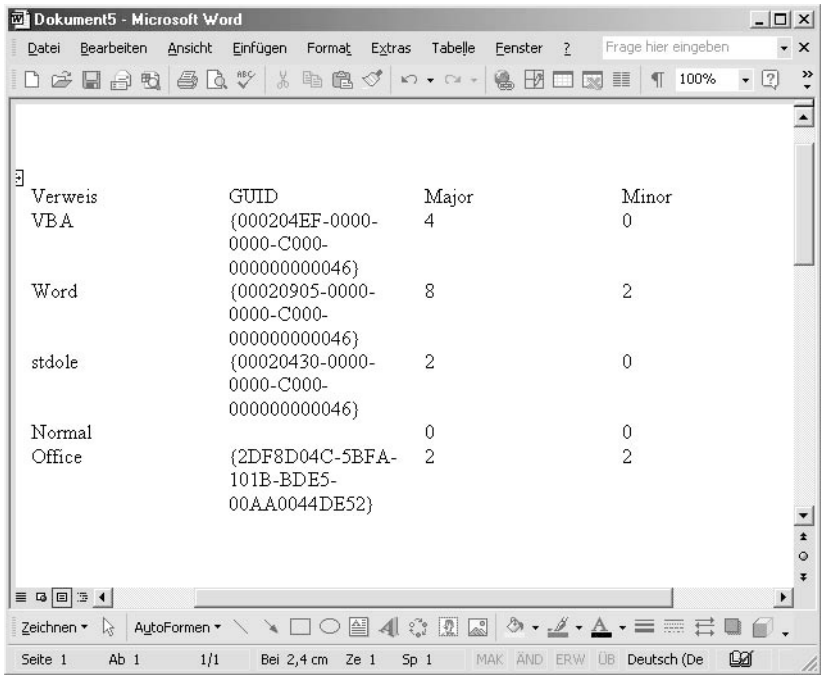


Abb. 8.3 Alle gesetzten Verweise ermitteln

8.2.2 Die VBE-Bibliothek einbinden

Möchten Sie die Bibliothek MICROSOFT VISUAL BASIC FOR APPLICATION EXTENSIBILITY 5.3 per Makro einbinden, starten Sie foglendes Makro:

```
Sub AktivierenVBE()
Dim VBEobj As Object

On Error Resume Next
VBEobj = Application.VBE.ActiveVBProject.References. _
AddFromGuid("{0002E157-0000-0000-C000-000000000046}", 5, 3)
End Sub
```

Listing 8.2: Die VBE-Bibliothek einbinden

Die Methode `AddFromGuid` fügt der `References`-Auflistung einen Verweis hinzu, wobei der global eindeutige Bezeichner (`GUID`) des Verweises verwendet wird. Die komplette Syntax lautet:

```
AddFromGuid(GUID, Major, Minor) As Reference
```

8.2.3 Die VBE-Bibliothek entladen

Selbstverständlich können Sie einmal gesetzte Verweise auf Bibliotheken auch wieder zurücknehmen. Im folgenden Makro wird die Bibliothek MICROSOFT VISUAL BASIC FOR APPLICATION EXTENSIBILITY 5.3 entladen:

```
Sub DeaktivierenVBE()
Dim VBEObj As Object

 On Error Resume Next
 Set VBEObj = Application.VBE.ActiveVBProject.References
 VBEObj.Remove VBEObj("VBIDE")
End Sub
```

Listing 8.3: Die VBE-Bibliothek entladen

Mithilfe der Methode `Remove` entfernen Sie den Verweis auf die eingebundene Bibliothek aus dem aktiven Dokument.

8.2.4 Bibliotheksinfos schreiben

Um zu sehen, welche Verweise auf Bibliotheken in Ihrem Dokument gesetzt sind, wie die Bibliotheken heißen und wo diese gespeichert sind, wenden Sie das folgende Makro an:

```
Sub BibliothekenInfosSchreiben()
Dim Verweis As Reference

  On Error Resume Next
  For Each Verweis In _
  Application.VBE.ActiveVBProject.References
   Debug.Print "Bezeichnung: " & Verweis.Description & _
   Chr(13) & "Speicherort: " & Verweis.FullPath _
```

Listing 8.4: Bibliotheksinformationen im Direktfenster ausgeben

```
  & Chr(13) & "Name: " & _
    Verweis.Name & Chr(13) & Chr(13)
  Next Verweis
End Sub
```

Listing 8.4: Bibliotheksinformationen im Direktfenster ausgeben

Neben dem Objektnamen jeder eingebundenen Bibliothek geben Sie zusätzlich auch noch die genaue Bezeichnung sowie den Speicherort der Bibliothek im Direktfenster aus.

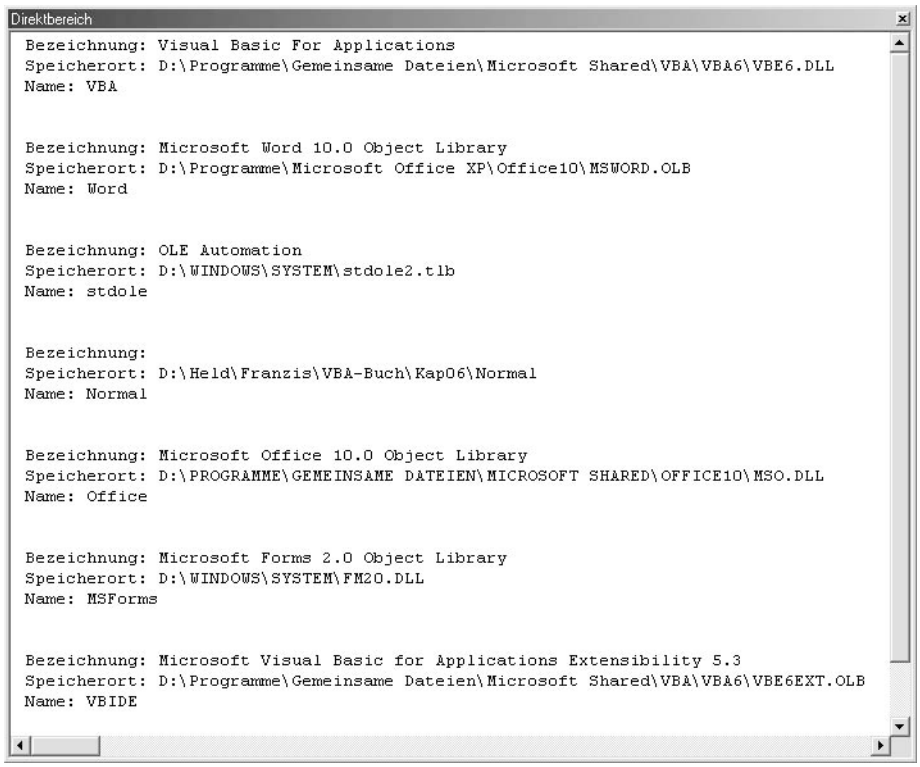


Abb. 8.4 Informationen über eingebundene Bibliotheken ausgeben

8.3 Die VBE-Komponenten

Um mehr über die einzelnen VBE-Komponenten zu erfahren, können Sie nach dem Einbinden der Bibliothek MICROSOFT VISUAL BASIC FOR APPLICATION EXTENSIBILITY 5.3 mehr erfahren. Dazu rufen Sie in der Entwicklungsumgebung den Objektkatalog auf, indem Sie die Taste F2 drücken. Danach stellen Sie im ersten Dropdown den Eintrag VBIDE ein.

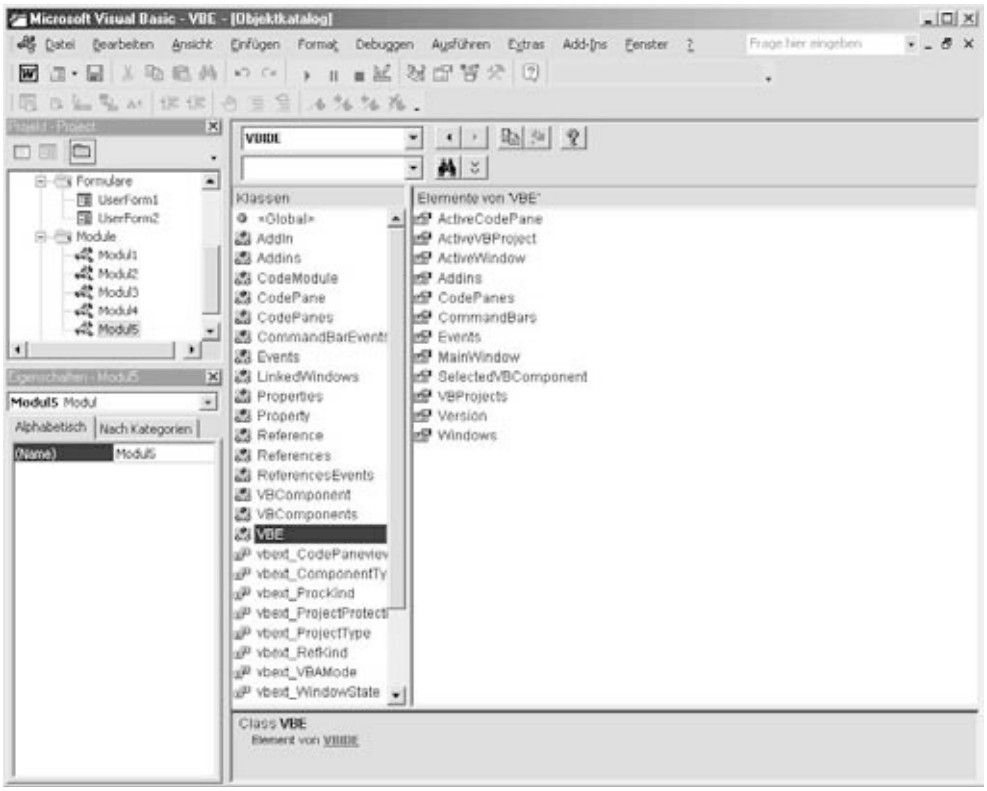


Abb. 8.5 Die VBE-Bibliothek ansehen

Das VBE-Objekt hat mehrere Auflistungsobjekte und Eigenschaften für den Zugriff auf die einzelnen Elemente. Unter anderem sind dies folgende:

- `VBProjects`: In diesem Auflistungsobjekt sind alle geöffneten Projekte in der Entwicklungsumgebung verzeichnet.
- `AddIns`: Sie regelt den Zugriff auf die Auflistung der Add-Ins.
- `Windows`: Stellt Methoden und Eigenschaften für den Zugriff auf die Fenster, wie z.B. Projekt- und Eigenschaftenfenster, bereit.
- `CodePanes`: Ist für den Zugriff auf die geöffneten Code-Bereiche eines Projekts verantwortlich.
- `CommandBars`: Kümmert sich um den Zugriff auf die Auflistung der Befehlsleisten.
- `Events`: Liefert alle Eigenschaften, die Add-Ins eine Verbindung zu allen Ereignissen in Visual Basic für Applikationen ermöglichen.
- `Version`: Gibt einen Wert vom Typ String zurück, der die Version von Visual Basic für Applikationen enthält, die von der Anwendung verwendet wird.

8.4 VBE-Komponenten auflisten

Möchten Sie wissen, welche VBE-Komponenten in Ihrem Dokument enthalten sind, dann können Sie dies in Erfahrung bringen, indem Sie die Eigenschaft `VBCompontents` einsetzen. Diese Eigenschaft liefert Ihnen eine Auflistung der in einem Projekt enthaltenen Komponenten zurück. Diese Komponenten sollen jetzt in einem Listenfeld einer UserForm angezeigt werden. Zeichnen Sie nun eine UserForm nach folgendem Vorbild:

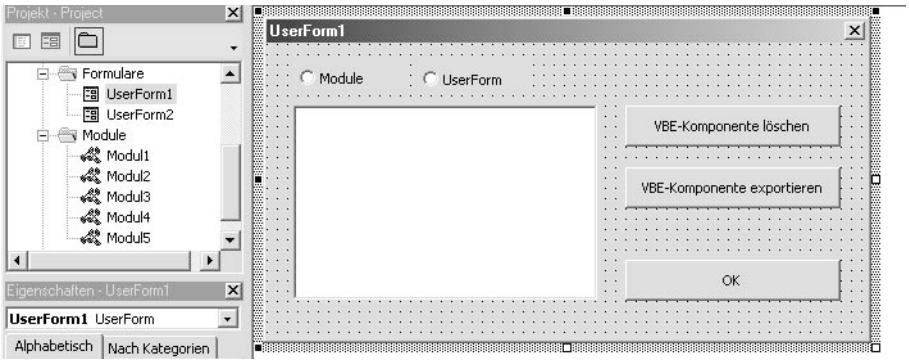


Abb. 8.6 Die UserForm soll die Komponenten anzeigen

Führen Sie jetzt einen Doppelklick auf die Optionsschaltfläche MODULE durch und erfassen das nächste Makro:

```
Private Sub OptionButton1_Click()
Dim VBkomp As VBComponent

UserForm1.ListBox1.Clear
 For Each VBkomp In ThisDocument.VBProject.VBComponents
  If VBkomp.Type = 1 Then _
    UserForm1.ListBox1.AddItem VBkomp.Name
 Next VBkomp
End Sub
```

Listing 8.5: Module ermitteln

Löschen Sie im ersten Schritt das Listenfeld mithilfe der Methode `Clear`. Durchlaufen Sie danach eine Schleife, in der Sie alle Komponenten der Entwicklungsumgebung abarbeiten. Prüfen Sie, ob es sich um ein Modul handelt. In diesem Fall meldet die Eigenschaft `Type` den Wert `1`. Fügen Sie dann den Namen der Komponenten im Listenfeld ein, indem Sie die Methode `AddItem` einsetzen. Übergeben Sie dieser Methode die Eigenschaft `Name`, um den Namen der Komponente zu ermitteln.

Sorgen Sie jetzt dafür, dass auch die UserForms im Listenfeld angezeigt werden. Führen Sie daher einen Doppelklick auf die Optionsschaltfläche USERFORM durch und erfassen folgendes Makro:

```
Private Sub OptionButton2_Click()
Dim VBkomp As VBComponent

UserForm1.ListBox1.Clear
 For Each VBkomp In ThisDocument.VBProject.VBComponents
  If VBkomp.Type = 3 Then _
     UserForm1.ListBox1.AddItem VBkomp.Name
 Next VBkomp
End Sub
```

Listing 8.6: UserForms ermitteln

Löschen Sie im ersten Schritt das Listenfeld mithilfe der Methode `Clear`. Durchlaufen Sie danach eine Schleife, in der Sie alle Komponenten der Entwicklungsumgebung abarbeiten. Prüfen Sie, ob es sich um eine UserForm handelt. In diesem Fall meldet die Eigenschaft `Type` den Wert `3`. Fügen Sie dann den Namen der Komponenten im Listenfeld ein, indem Sie die Methode `AddItem` einsetzen.

Öffnen Sie nun die UserForm, indem Sie das folgende Makro starten.

```
Sub UserFormStarten()
 UserForm1.Show
End Sub
```

Listing 8.7: UserForm starten

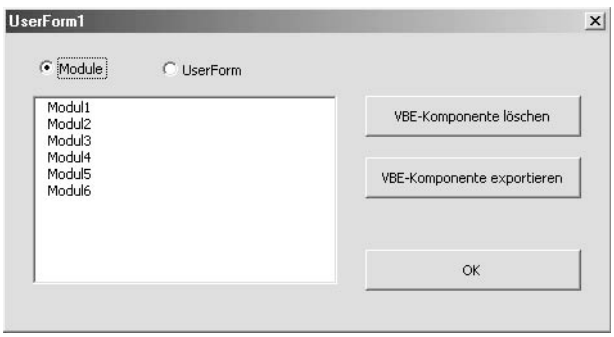


Abb. 8.7 Alle Module werden im Listenfeld angezeigt

8.5 VBE-Komponenten entfernen

Die gerade erstellte UserForm können Sie einsetzen, um einzelne VBE-Komponenten zu löschen, ohne in die Entwicklungsumgebung wechseln zu müssen. Führen Sie einen Doppelklick auf die Schaltfläche VBE-KOMPONENTE LÖSCHEN durch und erfassen folgendes Makro:

```
Private Sub CommandButton1_Click()
 On Error Resume Next
 With ThisWorkbook.VBProject
    .VBComponents.Remove .VBComponents(ListBox1.Value)
 End With
 MsgBox "Die VBE-Komponente wurde entfernt!"
 ListBox1.RemoveItem (ListBox1.ListIndex)
End Sub
```

Listing 8.8: VBE-Komponente löschen

Mit der Methode `Remove` können Sie eine VBE-Komponente löschen. Die `On Error`-Anweisung verhindert einen Makroabsturz, wenn die VBE-Komponente nicht gefunden werden kann, weil es eventuell bereits gelöscht wurde. Nach dem Entfernen der VBE-Komponenten aus der Entwicklungsumgebung müssen Sie den entsprechenden Eintrag auch aus dem Listenfeld entfernen. Dazu setzen Sie die Methode `RemoveItem` ein und übergeben dieser Methode den gerade markierten Eintrag im Listenfeld.

8.6 VBE-Komponenten exportieren

Möchten Sie Ihren Quellcode sichern, indem Sie diesen in einer Textdatei speichern, dann können Sie ebenfalls die vorher eingefügte UserForm einsetzen. Führen Sie einen Doppelklick auf die Schaltfläche VBE-KOMPONENTE EXPORTIEREN durch und erfassen Sie das folgende Makro:

```
Private Sub CommandButton2_Click()
Dim VBkomp As Object

On Error Resume Next
ChDir "C:\"
Set VBkomp = ThisWorkbook.VBProject.VBComponents(ListBox1.Value)
With VBkomp
    .Export ListBox1.Value & "_" & Date & ".txt"
```

Listing 8.9: VBE-Komponente exportieren

```
End With

MsgBox "Die VBE-Komponente wurde exportiert!"
End Sub
```

Listing 8.9: VBE-Komponente exportieren

Mit der Methode `Export` sichern Sie eine VBE-Komponente als Textdatei. Im Falle, dass Sie eine UserForm sichern, wird zusätzlich zur Textdatei eine Datei mit der Endung FRX gespeichert, die Informationen über den Aufbau der UserForm enthält. Als Dateinamen für die Sicherung verwenden Sie den Namen der VBE-Komponenten und hängen das aktuelle Datum über die Funktion `Date` dran.

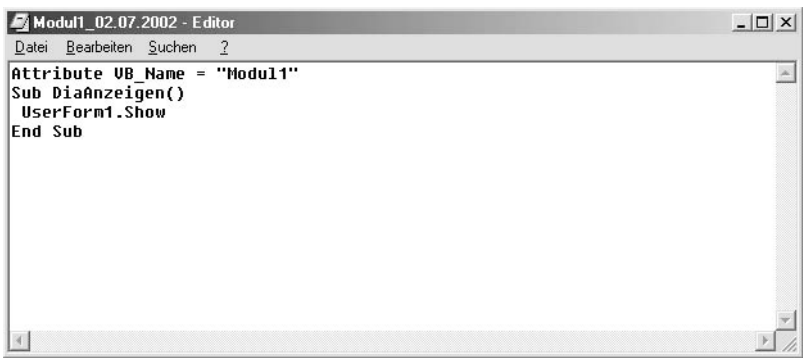


Abb. 8.8 Ein exportiertes Modul in einer Textdatei

8.7 VBE-Komponenten importieren

Nach dem Export von VBE-Komponenten haben Sie natürlich auch die Möglichkeit, einen Quellcode als neues Modul in Ihre Entwicklungsumgebung einzufügen. Sehen Sie sich jetzt einmal das folgende Makro an:

```
Sub ModulAusTextdateiImportieren()
Dim VBKomp As VBComponent
Dim CodeModul As CodeModule
Dim i  As Integer
Const ImportDat = "D:\Modul1_02.07.2002.txt"

Set VBKomp = _
```

Listing 8.10: VBE-Komponente importieren

```
 ThisDocument.VBProject.VBComponents.Add _
 (vbext_ct_StdModule)
 VBKomp.Name = "NeuesModul"
 Application.Visible = True

Set CodeModul = _
 ThisDocument.VBProject.VBComponents _
 ("NeuesModul").CodeModule
With CodeModul
     .AddFromFile ImportDat
 End With
End Sub
```

Listing 8.10: VBE-Komponente importieren

Die Methode `Add` verwendet die Konstante `vbext_ct_StdModule`, welche ein normales Modul repräsentiert. Selbstverständlich können Sie ebenso Klassenmodule und UserForms über diese Methode in Ihr Projekt einfügen. Entnehmen Sie dazu die notwendigen Konstanten der folgenden Tabelle.

Konstante	*Erklärung*
`vbext_ct_ClassModule`	fügt der Auflistung ein Klassenmodul hinzu
`Vbext_ct_MSForm`	fügt der Auflistung ein Formular hinzu
`Vbext_ct_StdModule`	fügt der Auflistung ein Standardmodul hinzu

Die Methode `AddFromFile` setzen Sie ein, um den Quellcode aus der Textdatei in das neu eingefügte Modul einzufügen. Dabei ist entscheidend, welcher Eintrag in der ersten Zeile der Textdatei steht. Dieser Eintrag wird bei der Umbenennung des Moduls NEUESMODUL herangezogen.

Eine weitere Möglichkeit. ein Makro in ein bestehendes Modul einzufügen ist, wenn Sie die einzelnen Zeilen des Makros eine nach der anderen einfügen. Bei der folgenden Lösung wird das Makro `WerBinIch` Zeile für Zeile in ein bestehendes Modul eingefügt.

```
Sub MakroZeilenweiseHinzufügen()
Dim CodeModul As CodeModule
Dim i As Integer

 Set CodeModul = _
 ThisDocument.VBProject.VBComponents _
```

Listing 8.11: Makro zeilenweise einfügen

```
 ("Modul8").CodeModule
 With CodeModul
     i = .CountOfLines + 1
     .InsertLines i, _
 "Sub WerBinIch()" & Chr(13) & _
 " Msgbox ""Ich bin der Anwender " & _
  Application.UserName & "!"" " & Chr(13) & _
 "End Sub"
 End With
End Sub
```

Listing 8.11: Makro zeilenweise einfügen

Im Makro wird Zeile um Zeile übertragen. Die Eigenschaft `CountOfLines` ermittelt, wie viele Codezeilen im Modul bereits enthalten sind, und addiert den Wert 1 darauf. Diese Maßnahme ist notwendig, um eventuell bereits bestehende Makros nicht zu überschreiben. Die Funktion `Chr(13)` sorgt jeweils für den Zeilenvorschub. Die Eigenschaft `UserName` gibt den aktuellen Benutzer des Dokuments zurück.

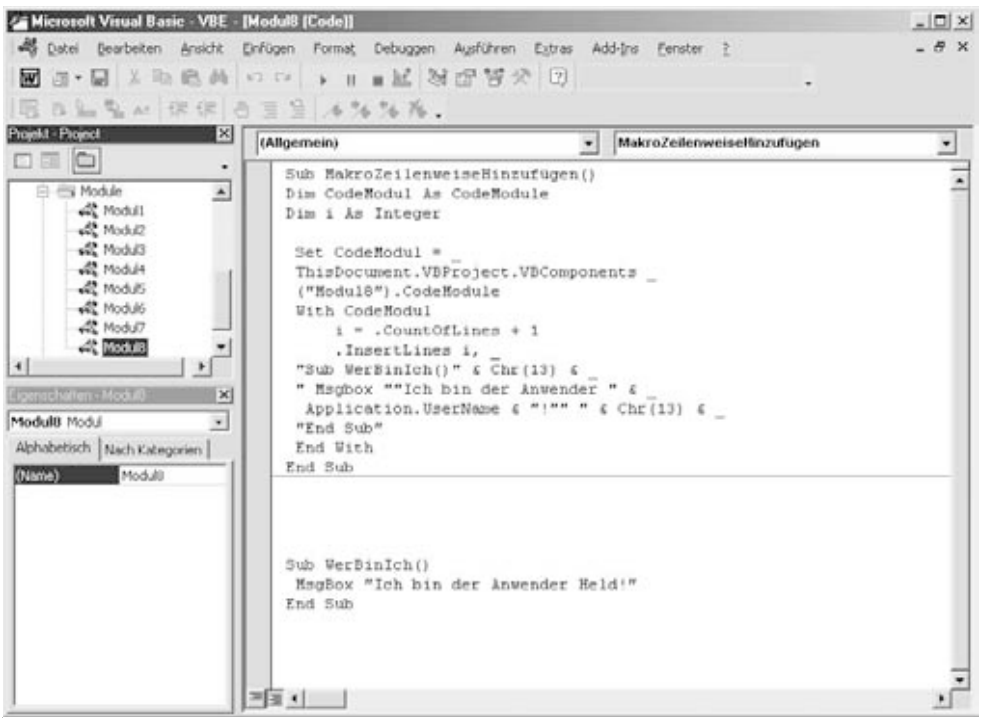


Abb. 8.9 Ein Makro Zeile für Zeile einfügen

Bernd Held

Studienausgabe

VBA-Programmierung

für Word, Excel und Access

Passen Sie Microsoft Office mit neuen Funktionen an Ihre Bedürfnisse an! Gewiss, Office bietet eine riesige Optionsvielfalt – aber ist auch wirklich alles dabei, was Sie brauchen? Wenn Ihnen immer wiederkehrende Arbeitsabläufe zu mühsam sind, brauchen Sie selbst programmierte Lösungen. Hier kommt Visual Basic für Applikationen ins Spiel. Denn mit VBA können Sie einfache, aber auch komplexere Lösungen für Ihre Office-Aufgaben selbst entwickeln.

Neue Office-Funktionen erstellen
In diesem Buch finden Sie eine detaillierte und praxisbezogene Anleitung zur VBA-Programmierung in Excel, Word, Access, Outlook, PowerPoint und anderen Office-Komponenten. Bernd Held zeigt Ihnen, worauf Sie bei der Programmierung achten müssen, wie Sie Fehler finden und wie Sie VBA-Programme tunen. Setzen Sie Steuerelemente geschickt ein, tauschen Sie Daten zwischen den verschiedenen Office-Applikationen problemlos aus und erzeugen Sie für Ihre Anwendungen die passenden Menü- und Symbolleisten. Egal, ob Sie ambitionierter Anwender oder Programmierer sind – der Autor vermittelt Ihnen das nötige Know-How, mit dem Sie Ihr persönliches Office gestalten.

Das Web in Office integrieren
Internetfunktionen lassen sich sehr einfach per VBA in Office-Dokumente einbinden. Verlinken Sie zum Beispiel die Tabellen einer Excel-Arbeitsmappe durch Hyperlinks, fragen Sie automatisch Aktienkurse ab, versenden Sie Excel-Bereiche per Mail oder richten Sie einen E-Mail-Direktversand ein.

Hilfen für Anwender einbauen
Machen Sie Office verständlicher! Mit Meldungen und Dialogen helfen Sie Anwendern, bei falschen Eingaben oder sonstigen Fehlern richtig zu reagieren. Durch einfache Eingabemasken fragen Sie Informationen von den Anwendern ab und werten sie dann aus. Oder Sie unterstützen die Benutzer, indem Sie ihnen bei Eingabedialogen Informationen zur Verfügung stellen, die sich aus den bereits eingegebenen Daten ergeben.

Aus dem Inhalt:

Über den Autor:

Bernd Held ist langjähriger Dozent, VBA-Entwickler und Autor von über 90 Fachbüchern und unzähligen Computer-Fachartikeln. Er entwickelt unter anderem Tools und Add-Ins für Excel und Access. Er ist einer der bekanntesten VBA-Spezialisten in Deutschland und von Microsoft mit dem Titel „Most Valuable Professional" ausgezeichnet worden. Bernd Held lebt in Vaihingen an der Enz.

Auf www.buch.cd:

- Sämtliche Programmbeispiele
- Testdateien
- Excel-Tools

9 783645 600705

25,00 EUR [D]
ISBN 978-3-645-60070-5

Besuchen Sie unsere Website
www.franzis.de