6 Beispiele für VBA-Excel

VBA - Visual Basic for Applications, die Programmiersprache für Microsoft Excel

Werner Holler

1. Was ist VBA

Visual Basic for Applications (im folgenden mit VBA bezeichnet) ist die Programmiersprache von Microsoft, mit der sich die tägliche Arbeit mit Excel automatisieren läßt, die es erlaubt, die Funktionen den eigenen Bedürfnissen anzupassen, oder mit der sich selbst vollständige Anwendungen erzeugen lassen.

In Microsoft Excel lassen sich bestimmte Aufgaben durch Verwendung von Makros automatisieren. Ein Makro ist eine Folge von Visual-Basic-Anweisungen, die Microsoft Excel mitteilen, was zu tun ist.

Microsoft Excel beinhaltet den sogenannten Makro-Recorder, der Makros aufzeichnet, indem er die bei der Arbeit mit Microsoft Excel durchgeführten Befehle und Schritte speichert. Später kann man dann das Makro wieder abspielen, um die aufgezeichneten Aktionen automatisch zu wiederholen und damit Zeit und Aufwand zu sparen. Für diese einfache Art der Makro-Aufzeichnung sind noch keinerlei Kenntnisse über VBA-Programmierung notwendig. Die Makros lassen sich allerdings noch wesentlich leistungsfähiger machen, wenn man den automatisch aufgezeichneten Code durch den eigenen Visual-Basic-Code verbessert oder erweitert.

Mit Visual Basic können angepaßte Befehle, Menüs, Dialogfelder, Nachrichten und Schaltflächen erstellt und angepaßte Hilfethemen für diese Elemente angezeigt werden. Von der Automatisierung wiederholter Aufgaben bis hin zur Entwicklung leistungsfähiger, voll funktionsfähiger Anwendungen ermöglichen die Werkzeuge von Visual Basic, Microsoft Excel so anzupassen, wie es genau den eigenen Bedürfnissen entspricht.

2. Wie lernt man VBA am besten?

Will man VBA für Microsoft Excel programmieren, so sollte man im Umgang mit Microsoft Excel doch schon einigermaßen geübt sein. Je mehr man von Microsoft Excel weiß, desto besser ist man auf VBA vorbereitet. Die meisten Makros führen irgendwelche Operationsfolgen in Microsoft Excel aus, und die meisten Anweisungen in einem Makro sind äquivalent zu Befehlen oder Operationen in Microsoft Excel. Die Arbeit mit VBA ähnelt eigentlich sehr der Arbeit mit Microsoft Excel, allerdings hat man keine so bequeme Benutzeroberfläche zur Verfügung. Statt der Befehle und Dialogfelder verwendet man die Visual-Basic-Anweisungen. Ausdrucke und Funktionen, aus denen sich die Anwendungen zusammensetzen, sind leichter verständlich, wenn man schon genau weiß, welche Aufgaben sie in Microsoft Excel ausführen. Ich selbst habe schon gelegentlich lange Makros für Aufgaben geschrieben, die mit einem einzigen Microsoft Excel-Befehl ausgeführt hätten werden können. Nachher ist man eben immer schlauer.

Hat man noch keinerlei Erfahrungen mit einer Makro-Programmiersprache, erscheint einem vielleicht VBA sehr unübersichtlich. Es hat sich aber in der Praxis sehr bewährt, nicht zuerst die gesamte Theorie auf einmal in sich aufzusaugen und dann erst in die Anwendung zu gehen, sondern stets nur das zu lernen, was man gerade für eine spezielle Aufgabe braucht. Hier bietet sich als Unterstützung die Online-Hilfe von VBA an:

In einem Visual-Basic-Modul kann man ein Visual-Basic-Schlüsselwort eintippen, den Cursor am Schlüsselwort stehen lassen und F1 drücken, um *die Visual-Basic-Hilfe* für diesen Begriff anzuzeigen. Die Hilfe enthält auch für die meisten Schlüsselwörter ein Beispiel, welches man kopieren und ins eigene Makro einfügen kann.

Eine andere Möglichkeit der Online-Hilfe wäre, im *Hilfe-Menü* die Excel-Themen anzuklicken. Man kann dann entweder „*Erste Schritte*“ im Inhalt-Fenster anklicken, ein bestimmtes Thema oder einen VB-Begriff in der *Index-Tabelle* nachsehen, oder über *Suchen* einen beliebigen Text auffinden.

Klickt man im Hilfe-Menü auf die *Microsoft Excel-Hilfethemen*, kann man anschließend mit dem *Hilfe-Assistenten* eine Frage stellen und die im Abschnitt *Programmier- und Sprachverzeichnis* des Dialogfelds dargestellten Informationen lesen.

Befindet man sich in einem aktiven VBA-Modul, klickt man auf den *Objekt-Katalog* im *Anzeige-Menü* und dann auf die Schaltfläche *Element-Hilfe* (das Fragezeichen unterhalb des Felds *Objekte/Module*), um Informationen über ein Objekt, eine Methode, eine Eigenschaft oder Funktionen zu erhalten.

Achtung: Hat man nur die Standardinstallation von Microsoft Excel verwendet, ist die Hilfe für VBA nicht installiert. In diesem Fall muß man das Microsoft Excel-Setup erneut ausführen.

3. Verwenden des Makro-Recorders

Der Makro-Recorder kann VB-Anwendungen für fast jede Operation aufzeichnen, die man unter Microsoft Excel ausführt. Mit Hilfe des Makro-Recorders kann man beobachten, wie die Operationen unter Microsoft Excel in VB-Anweisungen übersetzt werden und umgekehrt. Es ist allgemein schneller und einfacher, einen Teil eines Makros aufzuzeichnen, als die entsprechenden Anweisungen „händisch“ zu schreiben (siehe Beispiele unter 6.)

Wie ich im Umgang mit Schülern selbst feststellen konnte, eignet sich die Methode, VBA-Programme mit dem Makro-Recorder zu erstellen, sehr gut, um bei Anfängern ein gutes Basisverständnis für objektorientiertes Programmieren zu erzeugen.

4. Code-Beispiele

Das Microsoft Excel-Paket enthält Codebeispiele, die man öffnen und ausführen kann. Beliebige Teile dieser Beispiele kann man in eigene Anwendungen einkopieren und entsprechend ändern. Die Beispiele befinden sich im Beispiel-Ordner im Microsoft Excel-Ordner.

5. Englisch oder Deutsch?

Eine wichtige Entscheidung betrifft die Verwendung der zu verwendenden Programmiersprache; VBA in Deutsch etwa ist sicherlich für einen Anwender, der gewohnt ist, in Englisch zu programmieren, sehr gewöhnungsbedürftig.

Im Dialogfeld des Befehles *Extras - Optionen* kann man im Register *Modul Allgemein* wählen, ob man Deutsch oder Englisch programmieren will.

Ist allerdings momentan ein Modulblatt aktiv, ist die Option *Aktuelle Einstellungen* selektiert, und im Listenfeld *Sprache/Land* unterhalb wird die für dieses Modulblatt gültige Spracheinstellung angezeigt. Diese Einstellung ist nachträglich nicht mehr veränderbar. Wählt man hingegen *Standardeinstel-*

lungen aus, werden im Listenfeld die Sprachen angezeigt, die man für neu anzulegende Modulblätter benutzen kann.

Wenn die gewünschte Einstellung nicht im Feld *Sprache/Land* aufgelistet ist, muß man zuerst die gewünschten Objektbibliotheken *.olb installieren.

Als „Übersetzungshilfe“ zwischen Codes in englisch und deutsch sei die im Lieferumfang enthaltene Excel-Datei *VBAListe.xls* empfohlen.

Um Vergleiche über beide Code-Varianten anstellen zu können, habe ich im folgenden stets beide Versionen angeführt. Ebenfalls habe ich in allen Erläuterungen zusätzlich zur deutschen Bezeichnung auch noch den englischen Ausdruck *[kursiv und in eckigen Klammern]* geschrieben.

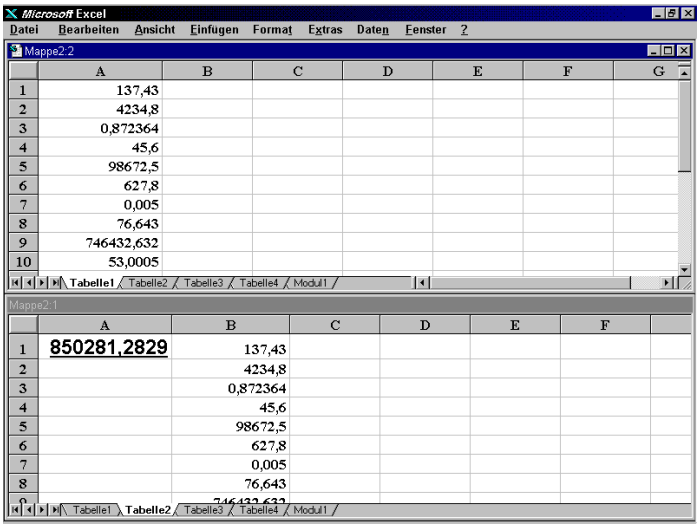
6. Einfaches Makro

Die folgenden Makros wurden vom System selbst mit dem Makro-Recorder erzeugt.

Vorgang: Im Menü *Extras Makro aufzeichnen* anwählen und *Aufzeichnen* bestätigen. Makro-Namen angeben bzw. vorgeschlagenen übernehmen und bestätigen. Danach die gewünschten Microsoft Excel-Befehlsschritte eingeben und zuletzt den Schalter *Makro-Aufzeichnung beenden* anklicken. Das so entstandene Makro kann in der Tabelle Modulx (x=1, 2, ..) angesehen bzw. erweitert werden.

Beispiel 1:

Zahlen im Bereich von A1 bis A10 einer Tabelle1 sollen markiert, in eine Tabelle2 von B1 bis B10 kopiert, in A1 aufsummiert und das Ergebnis dort fett und unterstrichen und in Arial Schriftgrad 14 ausgegeben werden



Code in deutsch:

```
Sub Makro1_deutsch()
    Bereich("A1:A10").Auswählen
    Auswahl.Kopieren
    BlattListe("Tabelle2").Auswählen
    Bereich("B1:B10").Auswählen
    AktivesBlatt.Einfügen
   Bereich("A1").Auswählen
  Anwendung.AusschneidenKopierenModus = Falsch
    AktiveZelle.Z1S1Formel = "=SUMME(ZS(1):Z(9)S(1))"
    Auswahl.Schriftart.Fett = Wahr
    Auswahl.Schriftart.Unterstreichen = xlEinfach
    Mit Auswahl.Schriftart
        .Name = "Arial"
        .Schriftstil = "Fett"
        .Grösse = 14
        .Durchstreichen = Falsch
        .Hochgestellt = Falsch
        .Tiefgestellt = Falsch
        .Kontur = Falsch
        .Schatten = Falsch
        .Unterstreichen = xlEinfach
        .FarbIndex = xlAutomatisch
    Ende Mit
Ende Sub
```

Code in englisch:

```
Sub Makro1_englisch()
    Range("A1:A10").Select
    Selection.Copy
    Sheets("Tabelle2").Select
   Range("B1:B10").Select
    ActiveSheet.Paste
    Range("A1").Select
    Application.CutCopyMode = False

ActiveCell.FormulaR1C1 = "=SUM(RC[1]:R[9]C[1])"
    Selection.Font.Bold = True
    Selection.Font.Underline = xlSingle
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Fett"
        .Size = 14
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlSingle
        .ColorIndex = xlAutomatic
    End With
End Sub
```

7. Steuerelemente

Steuerelemente wie etwa Schaltflächen, Kontrollkästchen oder Listenfelder können auf Arbeitsblättern, Tabellenblättern und Dialogblättern plaziert werden.

Durch die Verwendung von Steuerelementen kann die Benutzeroberfläche ganz auf den Anwender zugeschnitten und so der Umgang mit dem Programm vereinfacht werden. Plaziert man Steuerelemente in einem Dialogfeld, so wird ein angepaßtes Dialogfeld erzeugt.

Mit der *Werkzeugleiste* (einblenden über *Ansicht - Symbolleiste - Dialog*) kann man ein Steuerelement anwählen und auf dem Blatt plazieren. Nachdem man ein Steuerfeld plaziert hat, legt man die Eigenschaften fest, etwa ob ein Optionsfeld ausgewählt dargestellt wird, oder ob die Größe einer Schaltfläche angepaßt werden soll, wenn sich die Größe einer zugrundeliegenden Zelle in einem Arbeitsblatt ändert.

Danach wird dem Steuerelement eine VB-Prozedur zugeordnet. Wenn der Anwender auf eine Schaltfläche, auf ein Kontrollkästchen oder ein Optionsfeld klickt, führt VB die entsprechende Prozedur aus.

Um ein Steuerelement auf einem Blatt zu plazieren, geht man folgendermaßen vor:

In der Werkzeugleiste auf die Schaltfläche für das gewünschte Steuerelement klicken

An gewünschter Stelle auf das Blatt klicken und ziehen, bis das Steuerelement die gewünschte Größe hat

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Werkzeugleiste

Bedeutung der Symbole:

1 Bezeichnungsfeld: Text, den man für den Benutzer angibt und der etwa Namen, Anweisungen und Warnungen enthalten kann

2 Bearbeitungsfeld: Feld, in das der Benutzer Text, Zahlen oder Zellenverweise eingeben kann

3 Gruppenfeld: Rahmen, der eine Gruppe von Optionsfeldern oder anderer Steuerelemente umschließt

4 Schaltfläche: Befehlsschaltfläche wie etwa *OK* oder *ABBRECHEN*

5 Kontrollkästchen: Zeigt an, ob eine Option gesetzt ist, unabhängig vom Status anderer Optionen im Dialogfeld

6 Optionsfeld: Auswahl einer Option aus einer Gruppe von sich gegenseitig ausschließenden Optionen

7 Listenfeld: Liste von Text-Zeichenfolgen, von denen eine oder mehrere ausgewählt werden können

8 Dropdown-Feld: Nicht editierbares Textfeld und ein Pfeil, kombiniert mit einer Dropdown-Liste, die erscheint, wenn der Benutzer auf den Pfeil klickt

9 Liste-Feld-Kombinationsfeld: Editierbares Textfeld kombiniert mit einem Listenfeld

10 Dropdown-Kombinationsfeld: Leeres Bearbeitungsfeld, kombiniert mit einem Dropdown-Listenfeld, das erscheint, wenn der Benutzer auf den Pfeil klickt

11 Bildlaufleiste: Zum Verändern numerischer Werte

12 Drehfeld: Schaltflächenpaar zum Inkrementieren oder Dekrementieren eines angezeigten Wertes

13 Steuerelement-Eigenschaften: Wird verwendet, um die Eigenschaften des ausgewählten Objektes anzusehen oder zu bearbeiten

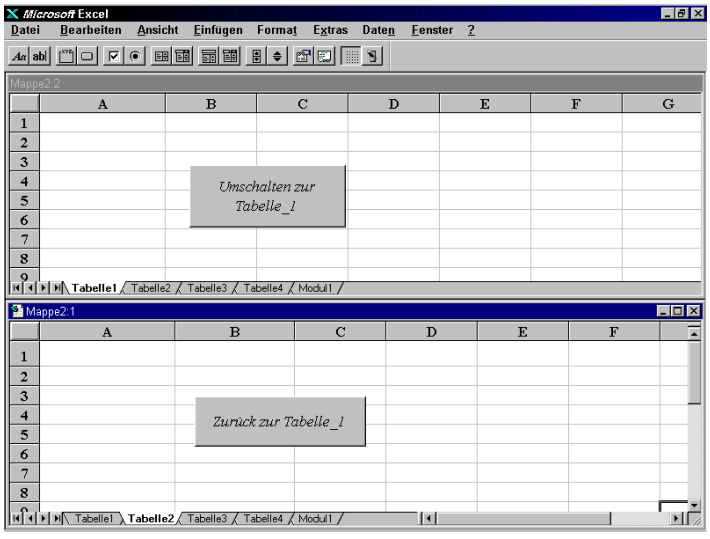
14 Code_Bearbeiten: Zum Erzeugen oder Bearbeiten von Codes für das ausgewählte Objekt

15 Raster: Zum vereinfachten Ausrichten von Steuerelementen

16 Dialog_ausführen: Zeigt das Dialogfeld so an, wie es bei der Ausführung erscheint

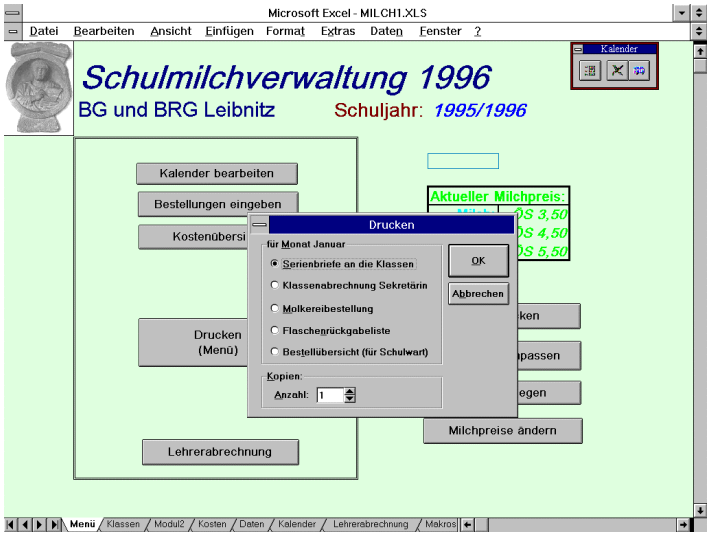
Beispiel 2:

Auf einem Tabellenblatt soll das Steuerelement *Schaltfläche* plaziert werden, welches nach Anklicken ein Makro aufruft, welches wiederum ein anderes Tabellenblatt öffnet.



Vorgang:

Tabelle öffnen, Steuerelement *Schaltfläche* auf Tabelle ziehen und wie oben beschrieben plazieren. *Aufzeichnen* anwählen (bzw. falls schon ein Makro existiert, zum Zuweisen einfach diesen Makro-Namen eingeben) und *Makro-Aufzeichnen* wie unter 6. beschrieben. Am Ende *Makro-Aufzeichnen beenden* bestätigen (wird manchmal ganz vergessen ...).



Beispiel für eine Excel-Tabelle mit Dialogfeldern

8. Verwenden des Makro-Rekorders zum Erzeugen eines Ausdrucks

Wie schon oben mehrmals erwähnt, stellt der Makro-Rekorder wahrscheinlich die beste Möglichkeit dar, einen ersten Ausdruck zu erzeugen, der die benötigten Eigenschaften und Methoden enthält, und der für seine Endfassung danach nur mehr entsprechend modifiziert zu werden braucht.

Beispiel 3:

Angenommen, wir wollen einen Ausdruck erzeugen, der den Schriftstil und die Schriftgröße für ein Diagramm ändert. Mit dem Makro-Rekorder hätten wir dann etwa folgenden Code aufgezeichnet:

```
Sub Makro2_deutsch()
AktivesDiagramm.Diagrammtitel.Auswählen
      Mit Auswahl.Schriftart
        .Name = "Times New Roman"
        .Schriftstil = "Fett"
        .Grösse = 24
        .Durchstreichen = Falsch
        .Hochgestellt = Falsch
        .Tiefgestellt = Falsch
        .Kontur = Falsch
        .Schatten = Falsch
        .Unterstreichen = xlKein
        .FarbIndex = xlAutomatisch
        .Hintergrund = xlAutomatisch
    Ende Mit
Ende Sub
Sub Makro2_englisch()
   ActiveChart.ChartTitle.Select
    With Selection.Font
        .Name = "Times New Roman"
        .FontStyle = "Bold"
        .Size = 24
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlNone
        .ColorIndex = xlAutomatic
        .Background = xlAutomatic
    End With
End Sub
```

Der Recorder hat das Objektmodell durchsucht und die Zugriffsfunktionen für die Objekte Chart *[Diagramm],* ChartTitle *[Diagrammtitel]* und Font *[Schriftart]* aufgezeichnet. Dieser Code soll jetzt modifiziert werden, um folgende Dinge auszuführen:

Veränderung der AktivesDiagramm *[ActiveChart]*-Eigenschaft in die Diagramm *[Charts]* - Methode und Angabe eines Diagrammnamens als Argument. Damit wird es möglich, daß das Makro

von jedem beliebigen Blatt in der Arbeitsmappe ausgeführt werden kann, nicht nur vom Diagrammblatt aus

Entfernen des nicht benötigten Codes für die Selektion. Nach der Veränderung muß auch das Schlüsselwort Mit *[With]* verschoben werden

Entfernen der Eigenschaft des Schriftarten *[Font]* - Objekts, das das Makro nicht verändern sollte (es darf nur die Eigenschaften Schriftstil *[FontStyle]* und Grösse *[Size]* ändern, aber keine anderen)

Der Prozedurname soll verändert werden

Das folgende Beispiel zeigt den Code nach der Veränderung:

```
Sub FormatChartTitle()
       Mit DiagrammListe("Chart1").Diagrammtitel.Schriftart
           .Schriftstil = "Fett"
           .Grösse = 24
       Ende Mit
Ende Sub

Sub FormatChartTitle()
       With Charts("Chart1").ChartTitle.Font
            .FontStyle = "Bold"
            .Size = 24
       End With
End Sub
```

Beispiel 4:

Über eine bequeme Eingabebox sollen in eine Tabelle1 in die Zellen D3 und D4 zwei Werte eingegeben werden. Um das Gesamtbild der Tabelle1 allerdings während der Eingabe nicht zu verändern, soll die Eingabe selbst zuerst über eine Eingabebox in eine Tabelle2 in die (vor der Eingabe zu löschenden) Zellen B3 und B4 erfolgen, die anschließend in die Zellen D3 bzw. D4 in der Tabelle1 kopiert werden.

Der Befehl zum Öffnen einer Eingabebox lautet EingabeDlg("*text*") *[InputDlg("text")]*, der Inhalt des Bearbeitungsfeldes wird danach zurückgeliefert.

Die Funktion IstWahr(*Ausdruck*) *[IsNumeric(Ausdruck)]* liefert den Wert wahr *[true]*, wenn der gesamte Klammerinhalt *Ausdruck* als Zahl ausgewertet wird, andernfalls ist der Rückgabewert falsch *[false]*.

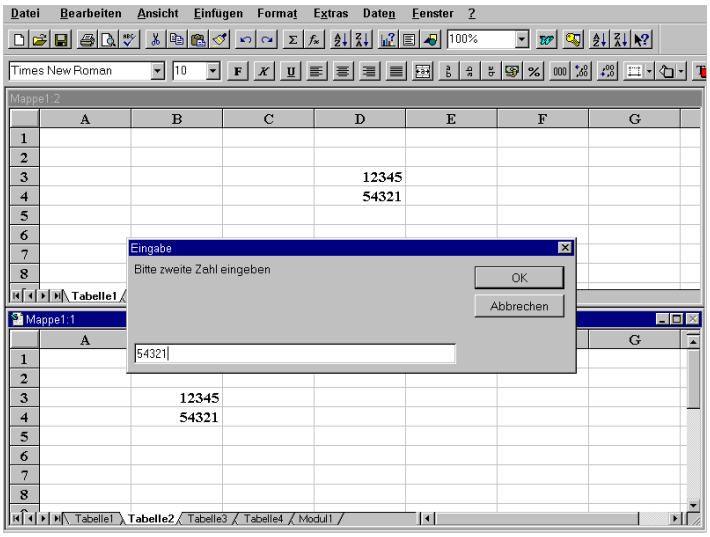
Den Eingabeteil des Makros mit Ausgabe in die entsprechenden Bereiche in Tabelle2 wird man zuerst quasi „händisch" erstellen müssen:

```
Durchlaufe
  Zahl1 = EingabeDlg("Bitte erste Zahl eingeben")
Schleife BisWahr IstZahl(Zahl1)
Durchlaufe
  Zahl2 = EingabeDlg("Bitte zweite Zahl eingeben")
Schleife BisWahr IstZahl(Zahl2)
```

oder

```
Do
  Zahl1 = InputBox("Bitte erste Zahl eingeben")
Loop Until IsNumeric(Zahl1)
Do
  Zahl2 = InputBox("Bitte zweite Zahl eingeben")
Loop Until IsNumeric(Zahl2)
```

Den weiteren Teil des Makros kann man wiederum mit dem Makro-Rekorder erstellen (lassen), ich habe etwa von Tabelle1 auf Tabelle2 umgeschaltet, dort die Zellen B3 und B4 gelöscht, (leer) markiert und wiederum nach Tabelle1 verschoben. Danach braucht man nur mehr das oben „händisch" erstellte Makro an geeigneter Stelle einfügen.




```
Sub Makro3_deutsch()
    BlattListe("Tabelle2").Auswählen
    Bereich("B3:B4").Auswählen
    Auswahl.InhalteLöschen
     Durchlaufe
              Zahl1 = EingabeDlg("Bitte erste Zahl eingeben")
    Schleife BisWahr IstZahl(Zahl1)
    Durchlaufe
       Zahl2 = EingabeDlg("Bitte zweite Zahl eingeben")
    Schleife BisWahr IstZahl(Zahl2)
    Bereich("B3") = Zahl1
    Bereich("B4") = Zahl2
    Bereich("B3:B4").Auswählen
    Auswahl.Kopieren
    BlattListe("Tabelle1").Auswählen
    Bereich("D3:D4").Auswählen
    AktivesBlatt.Einfügen
Ende Sub
Sub Makro3_englisch()
    Sheets("Tabelle2").Select
    Range("B3:B4").Select
    Selection.ClearContents
    Do
Zahl1 = InputBox("Bitte erste Zahl eingeben")
    Loop Until IsNumeric(Zahl1)
    Do
       Zahl2 = InputBox("Bitte zweite Zahl eingeben")
    Loop Until IsNumeric(Zahl2)    Range("B3") = Zahl1
    Range("B4") = Zahl2
    Range("B3:B4").Select
    Selection.Copy
    Sheets("Tabelle1").Select
    Range("D3:D4").Select
    ActiveSheet.Paste
End Sub
```

9. Programmieren ohne Makro - Rekorder am Beispiel *AktuelleRegion* und *VerwendeterBereich*

Beide Eigenschaften sind sehr praktisch, wenn der Code mit Bereichen operiert, über deren Größe man keine Kontrolle hat. Der aktuelle Bereich ist ein Zellenbereich, der durch leere Zeilen und leere Spalten begrenzt ist, oder durch eine Kombination aus leeren Zeilen, leeren Spalten und den Rändern einer Tabelle.

Die Eigenschaft AktuelleRegion *[CurrentRegion]* wird auf ein Bereich *[Range]* - Objekt angewendet. In einer Tabelle kann es viele verschiedene aktuelle Bereiche geben, abhängig von dem Bereich *[Range]* - Objekt, auf das man die AktuelleRegion *[CurrentRegion]* - Eigenschaft anwendet.

Beispiel 5

Angenommen, Tabelle1 enthält eine Liste, auf die man ein Zahlenformat anwenden will. Das einzige, was man über die Liste weiß, ist, daß sie in Zelle A1 beginnt. Nicht weiß man, wie viele Zeilen und Spalten sie enthält. Der folgende Code zeigt, wie die

Liste unter Verwendung der AktuelleRegion *[CurrentRegion]* - Eigenschaft formatiert werden kann:

```
Sub Makro4_deutsch()
Setze MeinBereich =
TabellenblattListe("Tabelle1").Bereich("A1").AktuelleRegion
MeinBereich.ZahlenFormat = "0,0"
Ende Sub
Sub Makro4_englisch()
Set MeinBereich =
Worksheets("Tabelle1").Range("A1").CurrentRegion
MeinBereich.NumberFormat = "0.0"
End Sub
```

Der verwendete Bereich wird durch die oberen linken und unteren rechten nichtleeren Zellen in der Tabelle begrenzt. Es handelt sich um einen Bereich, der alle nichtleeren Zellen in der Tabelle enthält, ebenso wie alle leeren Zellen, die dazwischenliegen. Es kann nur einen benutzten Bereich in einer Tabelle geben.

Die VerwendeterBereich *[UsedRange]* - Eigenschaft wird auf ein Tabellenblatt *[Worksheet]* - Objekt angewendet, nicht auf ein Bereichs *[Range]* - Objekt .

Beispiel 6

Angenommen, die aktive Tabelle enthalte Daten aus einem Experiment, für das die jeweiligen Zeiten angegeben werden müssen. Der verwendete Bereich umfaßt die Daten in der zweiten Spalte, die Zeiten in der dritten Spalte sowie die Meßergebnisse in der vierten und fünften Spalte. Wir wollen nun einen Code schreiben, der Datum und Zeit zu einem einzigen Wert zusammenfaßt, diesen Wert von der Greenwich Mean Time (GMT) in die Pacific Standard Time (Minus 8 Stunden) konvertiert und darauf ein Datumsformat anwendet; Daten-Spalten D und E sollen in Spalten C und D übergehen.

Die Rohdaten mögen wie im folgenden Beispiel erscheinen:

	A	B	C	D	E
1					
2		09-Feb-96	12:23:03	5433	12362
3		09-Feb-96	12:31:04	5621	12735
4		09-Feb-96	12:35:26	5682	12930
5					
6		10-02-1996	18:32:52	5318	12274
7					
8		10-02-1996	20:21:35	5483	12327
9		10-02-1996	20:33:42	5561	12119

Man beachte, daß die Tabelle leere Zeilen und Spalten enthalten kann.

Mit Hilfe der VerwendeterBereich *[UsedRange]* - Eigenschaft kann man den gesamten benutzten Bereich zusammen mit den beiden Leerzeilen zurückgeben.

Der folgende Code zeigt eine Möglichkeit, Datums- und Zeitwerte zu konvertieren und wie gewünscht zu formatieren:

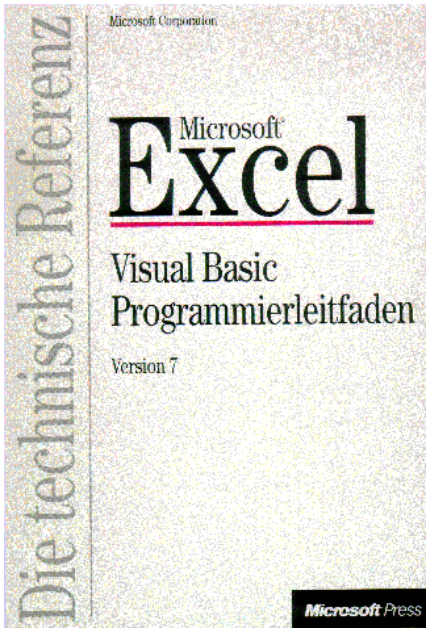
```
Sub Makro5_deutsch()
Setze MeinBereich = AktivesBlatt.VerwendeterBereich
MeinBereich.SpalteListe("D").Einsetzen
   Setze DatumsSpalte = MeinBereich.SpalteListe("D")
  Für Alle c In DatumsSpalte.ZelleListe
    Wenn c.Versetzen(0; -1).Wert <> "" Dann
       c.Z1S1Formel = "=ZS(-1)+ZS(-2)-8/24"
    Ende Wenn
  Nächste c
 DatumsSpalte.Zahlenformat = "MMM-TT-JJJJ hh:mm"
 DatumsSpalte.Kopieren
 DatumsSpalte.InhalteEinfügen Einfügen:=xlWerte
 MeinBereich.SpalteListe("B:C").Löschen
 DatumsSpalte.OptimalAnpassen
Ende Sub
```

```
Sub Makro5_englisch()
   Set MeinBereich = ActiveSheet.UsedRange
   MeinBereich.Columns("D").Insert
    Set DatumsSpalte = MeinBereich.Columns("D")
    For Each c In DatumsSpalte.Cells
        If c.Offset(0, -1).Value <> "" Then
             c.FormulaR1C1 = "=RC[-2]+RC[-1]-8/24"
        End If
   Next c
   DatumsSpalte.NumberFormat = "mmm-dd-yyyy hh:mm"
   DatumsSpalte.Copy
   DatumsSpalte.PasteSpecial Paste:=xlValues
   MeinBereich.Columns("B:C").Delete
   DatumsSpalte.AutoFit
End Sub
```

	A	B	C	D	E
1					
2		Feb-09-1995 04:23	5433	12362	
3		Feb-09-1996 04:31	5621	12735	
4		Feb-09-1996 04:35	5682	12930	
5					
6		Feb-10-1996 10:32	5318	12274	
7					
8		Feb-10-1996 12:21	5483	12327	
9		Feb-11-1996 12:33	5561	12119	

10. Weiterführende Literatur

Grundsätzliche Ideen aus diesem Bericht stammen aus dem Buch der Microsoft Corporation „Microsoft Excel - Visual Basic Programmierleitfaden - Version 7“. Das Buch soll ein Leitfaden sein, mit der Programmiersprache von EXCEL schlanke und effiziente Programme erstellen zu lernen. Es wurde von den Mitgliedern des Microsoft - Entwicklungsteams geschrieben und enthält Informationen, die nur Insider geben können. Vorausgesetzt werden allerdings bereits gewisse Erfahrungen im Umgang mit Microsoft Excel und VBA; als Ersteinstieg zum Programmieren von Excel eignet sich das Buch sicherlich nur beschränkt. Die Autoren selbst geben als Zielgruppe Entwickler, Berater und Excel - Profis an.



Inhalt:

● Grundlagen Excel/VBA ● Variablen, Datentypen, Konstanten ● Arbeiten mit Objekten ● Optimierung ● Debugging ● Dialogfelder, Menüs, Symbolleisten ● Kommunikation mit anderen Anwendungen ● Zugriff auf externe Daten ● Erstellung von Add-Ins ● Internationale Anwendungen ● Umstieg von der Makrosprache aus Excel 4 ● Objektmodell von Excel

Microsoft Corporation; MICROSOFT EXCEL - Visual Basic Programmierleitfaden; Microsoft Press; ISBN: 3-86063-231-0; 358 Seiten, öS 460.-

□