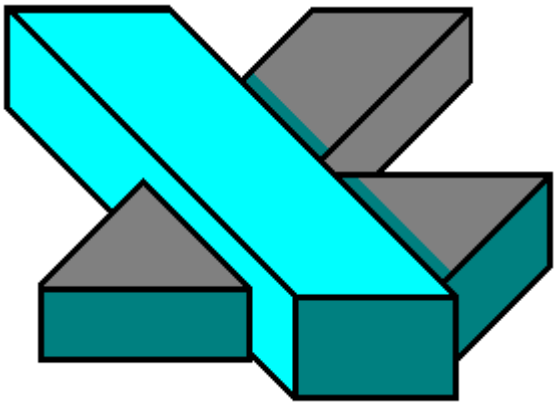
Einführung

in

MS-Excel

- Makroprogrammierung -



Kevelaer, im Juni 1999

Dipl.-Ing. Reinhard Peters
Ber. Ing für Informationsverarbeitung
Heideweg 60
47623 Kevelaer-Keylaer
Telefon 02832/6678
e-Mail: rpeters@pikt.de
Internet: www.pikt.de

Inhaltsverzeichnis

1 Einführung

Dieses Seminar richtet sich an Teilnehmer, die mit MS-Ecxel bereits gut vertraut sind. Das Seminar soll dem Teilnehmer anhand von Beispielen die Verwendung von Makros unter MS-Excel vermitteln. Grundlegende Programmiertechniken für Visual-Basic unter MS-Excel werden behandelt.

Folgende Themen werden behandelt:

- Aufzeichnen von Makros mit dem Makrorekorder
- Ausführen von Makros
- Umgang mit dem Visual-Basic-Editor
- Visual-Basic-Module
- Grundlagen von Visual-Basic

Überblick

Makros ermöglichen das Erstellen automatisierter Handlungsabläufe innerhalb eines Excel-Arbeitsblattes. In Excel ab der Version 5.0 werden die Makros in Form einer Programmiersprache realisiert. Diese Programmiersprache ist Visual-Basic für Applikationen (VBA). Basic ist eine weit verbreitete Programmiersprache im Bereich der Personalcomputer das Visual-Basic stellt eine Erweiterung dieser Programmiersprache dar. Zusätzlich sind in der Programmiersprache auch alle Excel-Funktionen und Prozeduren verfügbar, die auch über die Menüpunkte ausgewählt werden können.

Sprache

Bis zur Version 7.0 von Excel konnte die Sprache für die Makro-Module gewählt. So konnte man z. B. einen deutschen BASIC-Dialekt, zum Editieren der Makros verwenden. Ab der Version 97 steht nur noch ein Dialekt und zwar mit englischsprachigen Befehlen zur Verfügung.

Aufbau

Im einfachsten Fall sind die Makros nur eine Aneinanderreihung von mehreren Excel-Funktionen, Operationen und Befehlen. Diese werden unter einem Namen zusammengefaßt und können mit diesem Namen jederzeit wieder aufgerufen werden.

Ausblick

Die Möglichkeiten, die sich Ihnen mit der Makro-Programmierung bieten, sind mindestens so vielseitig und komplex wie die Bedienung von Excel selbst. Deshalb können Ihnen im Rahmen dieses Kurses nur Anregungen gegeben werden. Zur Realisierung auch umfangreicher Makros sollten Sie viel probieren und die Beispiele in den Handbüchern und den Beispiel-Dateien studieren.

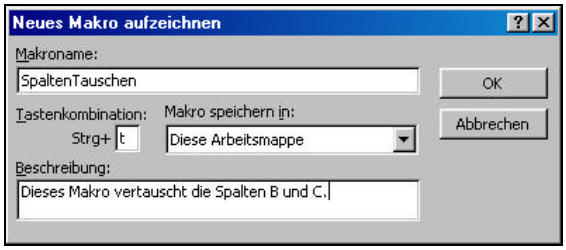
2 Der Makrorecorder

Die einfachste Art zur Erstellung eines Makros ist die Benutzung des Makrorecorders. Sie müssen sich den Makrorecorder vorstellen wie einen Kassettenrecorder. Zu einem bestimmten Zeitpunkt wird auf Aufnahme geschaltet. Die folgenden Handlungsabläufe werden aufgezeichnet. Wenn alles aufgezeichnet ist, wird die Stopptaste gedrückt. Anschließend können die Handlungsabläufe beliebig oft abgespielt werden.

2.1 Makros aufzeichnen

Beispiel

Sie möchten ein Makro erstellen, das in einer Tabelle zwei nebeneinander liegende Spalten vertauscht. Wählen Sie den Menüpunkt [Extras | Makro aufzeichnen | Aufzeichnen]. Die Dialogbox „Neues Makro aufzeichnen“ wird angezeigt.



In dieser Dialogbox können Sie zunächst den Namen des Makros festlegen. Hierfür gelten die gleichen Regeln wie für die Festlegung von Namen in Excel, keine Leerzeichen, nur Buchstaben und Ziffern, muß mit einem Buchstaben anfangen und darf nicht länger als 255 Zeichen sein. Eingebürgert hat sich eine Schreibweise bei der alles klein geschrieben wird und jedes neue Wort durch einen Großbuchstaben am Anfang kenntlich gemacht wird.

Tastenkombination

Mit der Tastenkombination, die Sie an dieser Stelle festlegen können aber nicht müssen, können Sie das Makro später aufrufen. Die Tastenkombination können Sie durch Drücken der entsprechenden Taste in dem nachfolgenden Editierfeld festlegen. Sie können auch Tastenkombinationen in Verbindung mit der <Umschalt>-Taste verwenden.

Makro speichern in

Mit dem Kombinationsfeld „Makro speichern in“ können Sie festlegen, wo das Makro gespeichert werden soll. Folgende Auswahlmöglichkeiten stehen Ihnen zur Verfügung:

- Persönliche Makro-Arbeitsmappe
  für Makros, die Ihnen immer zur Verfügung stehen sollen
- Neue Arbeitsmappe
  für Makros, die Sie bei Bedarf nachladen möchten
- Diese Arbeitsmappe
  für Makros, die an diese Arbeitsmappe gebunden sind

Speichern

Mit diesen Auswahlmöglichkeiten können Sie festlegen, wo das aufgezeichnete Makro gespeichert werden soll. Wenn Sie die Option „Diese Arbeitsmappe“ wählen, wird das Makro mit dieser Arbeitsmappe gespeichert. Das Makro steht Ihnen somit nur zur Verfügung, wenn diese Arbeitsmappe auch geöffnet ist.

Alternativ dazu können Sie die Makros auch in einer anderen Arbeitsmappe speichern („Neue Arbeitsmappe“). Auch diese Arbeitsmappe muß für die Ausführung des Makros zuvor geöffnet worden sein.

Personl.xls

Die „persönliche Makro-Arbeitsmappe“ ist eine Datei mit einer besonderen Bedeutung. Makros, die in dieser Arbeitsmappe gespeichert werden, stehen Ihnen in Zukunft jederzeit zur Verfügung. Falls noch keine persönliche Makro-Arbeitsmappe existiert, wird Sie automatisch angelegt. Die zugehörige Datei finden Sie im Excel-Startverzeichnis (..\Microsoft Office\Office\XLStart) unter dem Namen personl.xls. Sie wird automatisch bei jedem Start von Excel als ausgeblendetes Fenster mit geöffnet. Mit dem Menüpunkt Fenster|Einblenden können Sie die Datei jederzeit sichtbar machen.

Beschreibung

Darüber hinaus haben Sie die Möglichkeit, das Makro kurz zu beschreiben. Geben Sie einen beliebigen Text ein. Dieser wird mit dem Makro abgespeichert.

Aufzeichnung starten

Klicken Sie den Aktionsschalter OK, um Ihre Eingaben zu bestätigen. Ab diesem Zeitpunkt werden alle Ihre Aktionen in dem Makro aufgezeichnet.

Beginnen Sie nun die aufzuzeichnende Befehlssequenz auszuführen:

1. Spalte C markieren
2. [Bearbeiten|Ausschneiden]
3. Spalte B markieren
4. [Einfügen|Zellen ausschneiden]

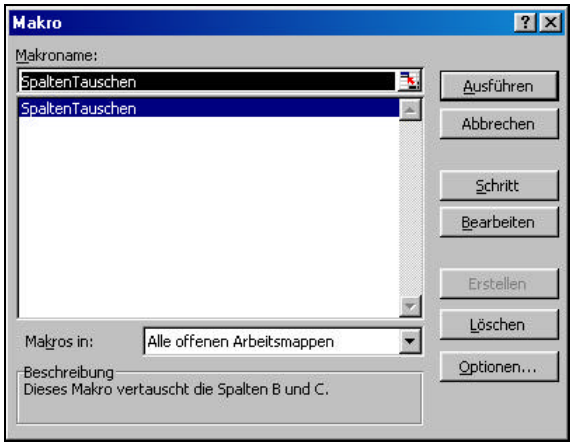
Damit haben Sie alle Aktionen für den aufzuzeichnenden Vorgang durchgeführt. Klicken Sie auf das Stoppsymbol oder wählen Sie den Menüpunkt [Extras|Makro aufzeichnen|Aufzeichnung beenden], um die Aufzeichnung zu beenden.

Hinweis

Speichern Sie nur solche Makros in der persönlichen Makro-Arbeitsmappe, die auch wirklich immer benötigt werden. Viele globale Makros verlangsamen den Start von Excel. Spezielle Makros speichern Sie immer in der jeweiligen Arbeitsmappe. Den Namen eines Makros und die zugeordnete Tastenkombination können Sie jederzeit ändern bzw. neu definieren.

2.2 Makros ausführen

Sie können das Makro nun mit Hilfe des Menüpunktes [Extras | Makro] jederzeit wieder ausführen lassen. Die folgende Dialogbox wird angezeigt:



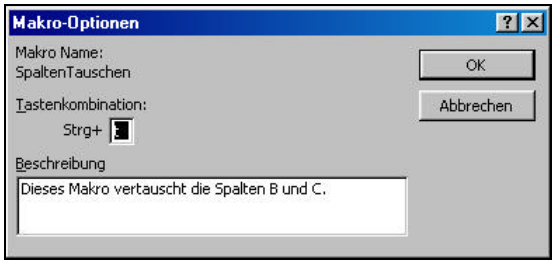
In der Auswahlliste werden die Namen aller verfügbaren Makros angezeigt. Wenn Sie ein Makro auswählen, wird unterhalb der Auswahlliste zusätzlich die Beschreibung des Makros angezeigt. Klicken Sie den Aktionsschalter ausführen, um das ausgewählte Makro auszuführen. Alle Aktionen, die Sie zuvor aufgezeichnet haben, werden nun automatisch abgespielt. In der Tabelle werden die Spalten B und C vertauscht.

Sie haben aber noch weitere Möglichkeiten Makros aufzurufen. Dies sind im einzelnen:

- Ausführen über eine Tastenkombination
- Ausführen über einen zusätzlichen Menüpunkt
- Ausführen über ein Symbol in der Symbolleiste
- Ausführen über grafische Symbole in einem Tabellenblatt

Tastenkombination

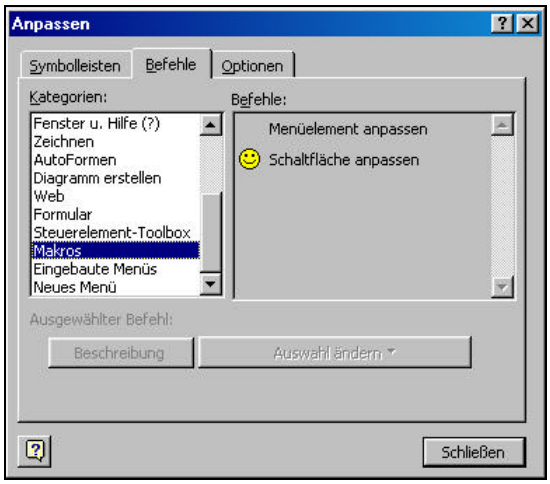
Die Tastenkombination gehört zu den Eigenschaften des Makros. Wir haben diese schon bei der Aufzeichnung des Makros festlegen können, um die Tastenkombination nachträglich zu ändern, wählen Sie den Menüpunkt [Extras | Makro] und wählen Sie das gewünschte Makro aus. Klicken Sie anschließend den Aktionsschalter Optionen. Die folgende Dialogbox wird angezeigt:



Mit dieser Dialogbox können Sie sowohl die Tastenkombination als auch die Beschreibung des Makros nachträglich verändern.

Menü/Symbolleiste

Zum Einbinden eines Makro in ein Menü oder eine Symbolleiste wählen Sie den Menüpunkt Extras|Anpassen. Die folgende Dialogbox wird angezeigt:

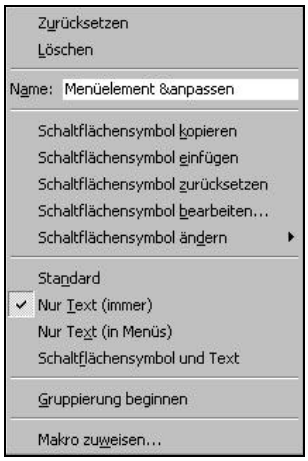


Klicken Sie auf die Registerkarte „Befehle“ und wählen Sie in der linken Auswahlliste den Punkt „Makros“. In der rechten Liste erscheinen zwei Auswahlmöglichkeiten.

Menüelement

Einen neuen Menüpunkt fügen Sie ein, in dem Sie mit der Maus auf den Eintrag „Menüelement anpassen“ in der rechten Liste gehen und diesen an die gewünschte Position in einem bereits existierenden Menü. Wenn Sie ein neues Menü anlegen möchten, dann müssen Sie zunächst mit dem Auswahlpunkt „Neues Menü“ (linke Liste) ein neues Menü erzeugen.

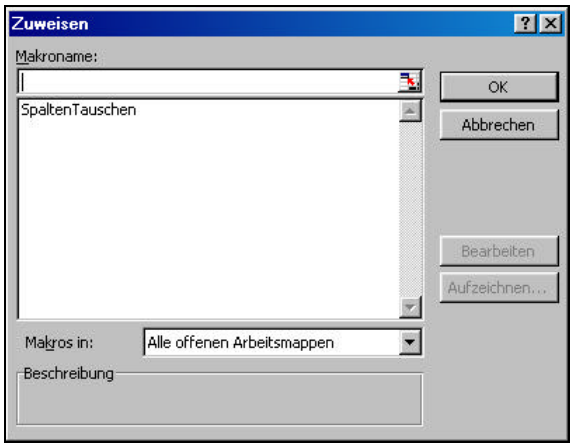
Klicken Sie anschließend auf den neu eingefügten Menüpunkt die rechte Maustaste. Ein Kontextmenü mit Auswahlpunkten zum Bearbeiten des Menüpunktes wird angezeigt:



Mit diesem Kontextmenü können Sie z. B. den Text für den Menüpunkt ändern. Mit dem „&“ im Text für den Menüpunkt bestimmen Sie die Zugriffstaste. Der entsprechende Buchstabe wird im fertigen Menü dann unterstrichen dargestellt. Die Menüpunkte für die Schaltflächensymbole betreffen die Symbolleisten, können aber auch in Verbindung mit einem Menüpunkt angewendet werden. Mit den nächsten vier Menüpunkten wird festgelegt, ob nur Text oder auch ein Symbol im Menü angezeigt werden soll. Mit dem Menüpunkt „Gruppierung beginnen“ fügen Sie einen Trennstrich in das Menü vor dem entsprechenden Menüpunkt ein.

Makro zuweisen

Wählen Sie den Menüpunkt „Makro zuweisen“, um dem neuen Menüpunkt das gewünschte Makro zuzuweisen. Die folgende Dialogbox wird angezeigt:



Wählen Sie das gewünschte Makro aus und klicken Sie auf den Aktionsschalter OK. Der Menüpunkt ist nun mit dem ausgewählten Makro verbunden.

Symbolleiste

Das Hinzufügen von Symbolen zu Symbolleiste erfolgt analog zum Hinzufügen von Menüpunkten. Wählen Sie den Menüpunkt „Schaltfläche anpassen“ in der rechten Auswahlliste und ziehen Sie das Symbol an die gewünschte Position.

Bei den Schaltfläche besteht zusätzlich die Möglichkeit das Symbol zu verändern. So können Sie z. B. aus einer Liste von vorhandenen Symbolen auswählen. Klicken Sie dazu die rechte Maustaste und wählen Sie in dem Kontextmenü den Menüpunkt „Schaltflächensymbol ändern“. Eine entsprechende Auswahl wird angezeigt. Darüber hinaus haben Sie die Möglichkeit auch eigene Symbole zu erstellen oder bestehende Symbole zu verändern. Wählen Sie dazu den Menüpunkt „Schaltflächensymbol bearbeiten“. Die folgende Dialogbox wird angezeigt:

Durch Klicken mit der Maus auf die einzelnen Bildpunkte können Sie deren Farben und somit das Symbol verändern.

Hinweis

Grundsätzlich ist es möglich, auch die Bedeutung bestehender Symbole und Menüpunkte in der zuvor beschriebenen Art und Weise zu verändern. Dies sollte aber nur in begründeten Ausnahmefällen getan werden. Falls Sie vorhandene Menüs oder Symbolleisten verändert haben und die Änderungen wieder rückgängig machen möchten, können Sie dies über die Registerkarten „Symbolleisten“ in der Dialogbox „Anpassen“ machen. Wählen Sie in der Liste die entsprechende Menü- oder Symbolleiste aus und klicken Sie auf den Aktionsschalter zurücksetzen.

Hinweis

Von Ihnen erstellte oder angepaßte Symbolleisten stehen in allen Arbeitsmappen des Systems zur Verfügung. Wenn Makros mit den Symbolen verbunden sind, werden die zugehörigen Arbeitsmappen beim Anklicken des Menüpunktes oder Symbols automatisch geladen, um das Makro auszuführen. Um sicherzustellen, daß eine benutzerdefinierte Symbolleiste immer in einer bestimmten Arbeitsmappe verfügbar ist, können Sie die Symbolleiste an eine Arbeitsmappe anbinden. Dann kann der Benutzer die Symbolleiste zwar löschen aber mit dem nächsten Öffnen der Arbeitsmappe steht die Symbolleiste wieder zur Verfügung.

Falls Sie die Sichtbarkeit von Menü- oder Symbolleisten beim Öffnen einer Arbeitsmappe steuern möchten, müssen Sie entsprechende Befehle in dem sogenannten Autostart-Makro der Datei integrieren. Beachten Sie dazu die Ausführungen im Kapitel Ereignismakros.

Falls Sie Konfiguration der Menüleiste oder Symbolleiste so sehr verändert haben, daß es Ihnen schwerfällt die ursprünglichen Einstellungen wieder herzustellen, können Sie auch im Windows-Verzeichnis die entsprechende Datei löschen, die diese Einstellungen speichert. Wenn Sie das System ohne Anmeldung betreiben heißt die Datei excel8.xlb. Ansonsten setzt der Dateiname sich nach folgendem Muster zusammen *8.xlb, wobei * für den Namen des jeweiligen Benutzers besteht. Über den Menüpunkt Datei öffnen können Sie auch andere *.xlb-Dateien laden und damit z. B. die Einstellungen für die Symbolleisten anderer Benutzer übernehmen.

Grafische Elemente

Eine weitere Möglichkeit ein Makro ausführen zu lassen, ist die Verknüpfung des Makros mit einem grafischen Objekt innerhalb eines Tabellenblattes. Sie können aber jedes der angebotenen grafischen Objekte verwenden.

Zeichnen

Mit Hilfe der Symbolleiste „Zeichnen“ können Sie ein grafisches Objekt in Ihre Tabelle einfügen. Klicken Sie das Symbol an, und ziehen Sie an der gewünschten Position den Rahmen für das Objekt auf. Klicken Si auf dem Objekt die rechte Maustaste. In dem angezeigten Kontextmenü gibt es wie bei den Symbole in der Symbolleiste einen Menüpunkt „Makro zuweisen“. Wählen Sie in der dann angezeigten Dialogbox das entsprechende Makro aus, und klicken Sie den Aktionsschalter OK. Klicken Sie einmal an eine andere Stelle der Tabelle, damit das Objekt nicht mehr markiert ist.

Wenn Sie die Maus nun über das grafische Objekt bewegen, wird der Mauszeiger in Form einer Hand angezeigt. Sie können nun auf das Objekt klicken, um das Makro ausführen zu lassen.

Wenn Sie die Befehlsschaltfläche zu einem späteren Zeitpunkt noch einmal bearbeiten möchten, müssen Sie mit dem entsprechenden Symbol (Pfeil) der Symbolleiste „Zeichnen“ den Markierungsmodus aktivieren. Danach können Sie auf die Befehlsschaltfläche klicken, um den Text zu ändern oder ein anderes Makro zuzuweisen. Wählen Sie dazu den Menüpunkt „Makro zuweisen“ im Kontextmenü.

Steuerelemente

Eine weitere Möglichkeit besteht darin, Steuerelemente innerhalb des Tabellenblattes zu verwenden und diese mit dem gewünschten Makro zu verbinden. Diese Möglichkeit wird später noch einmal in Verbindung mit benutzerdefinierten Dialoge aufgegriffen.

2.3 Relative Aufzeichnung

Das im vorhergehenden Beispiel aufgezeichnete Makro wird immer, unabhängig von der Position der Markierung, die beiden Spalten B und C vertauschen. Bei einer so allgemeinen Aufgabe für ein Makro, wie dem Vertauschen zweier Spalten, wäre es jedoch wünschenswert, wenn das Makro die Spalten in Abhängigkeit von der Position der Markierung vertauschen würde. Dazu gibt es neben dem bereits praktizierten absoluten auch noch den relativen Aufzeichnungsmodus. Beim relativen Aufzeichnungsmodus werden alle Befehle in Abhängigkeit von der aktuellen Position der Markierung aufgezeichnet.

Bevor Sie nun die Makroaufzeichnung starten, müssen Sie sich darüber Gedanken machen, wo die Markierung beim Beginn der Aufzeichnung stehen muß. Beim späteren Abspielen des Makros werden die Aktionen immer relativ zur aktuellen Position der Markierung ausgeführt. Wenn Sie z. B. die Markierung zu Beginn der Aufzeichnung in Spalte A stehen haben und bei der Aufzeichnung die Spalten B und C vertauschen, dann werden bei einem späteren Abspielen unter der Voraussetzung, daß die Markierung in der Spalte B steht, die Spalten C und D vertauscht. Es ist also sinnvoll, die Markierung vor dem Start der Aufzeichnung in die Spalte B zu setzen, wenn während der Aufzeichnung die Spalten B und C vertauscht werden.

Starten Sie nun die Aufzeichnung mit dem Menüpunkt „Extras|Makro aufzeichnen|Aufzeichnen“. Nennen Sie das Makro SpaltenTauschen2. Wenn Die Makroaufzeichnung gestartet wurde, wird folgende zusätzliche Symbolleiste angezeigt.

Die Funktion des ersten Symbols haben wir bereits kennengelernt. Es dient zum Stoppen der Makroaufzeichnung. Das zweite Symbol dient zum Umschalten zwischen dem relativen und dem absoluten Aufzeichnungsmodus. Klicken Sie einmal auf das zweite Symbol, um den relativen Aufzeichnungsmodus einzuschalten.

Alle Handlungen werden jetzt relativ zur Position der Markierung aufgezeichnet. Zeichnen Sie das Makro genauso auf wie das erste Makro SpaltenTauschen. Beenden Sie die Aufzeichnung.

Sie können nun auch diesem Makro wieder einen Menüpunkt, eine Tastenkombination oder ein Symbol in der Symbolleiste zuordnen. Wenn Sie das Makro dann abspielen, wird das Makro alle Aktionen in Abhängigkeit von der Position beim Start des Makros ausführen. Wenn Sie die Markierung auf die Spalte D setzen, dann werden die Spalten D und E vertauscht.

Wichtig ist, daß Sie schon beim Aufzeichnen bedenken, wo der Benutzer bei der Ausführung die Markierung plazieren wird. Bei der Ausführung von Makros kann es auch zu sogenannten Laufzeitfehlern kommen. Die dadurch entstehen, daß einzelne Anweisungen des Makros nicht ausgeführt werden können. Ein Laufzeitfehler tritt z. B. auf, wenn Sie vor dem Start des Makros SpaltenTauschen2 die Markierung auf die Spalte IV setzen. Dies ist die letzte mögliche Spalte einer Tabelle. Rechts daneben existieren keine weiteren Spalten, also ist das Markieren der rechts daneben liegenden Spalte auch nicht möglich.

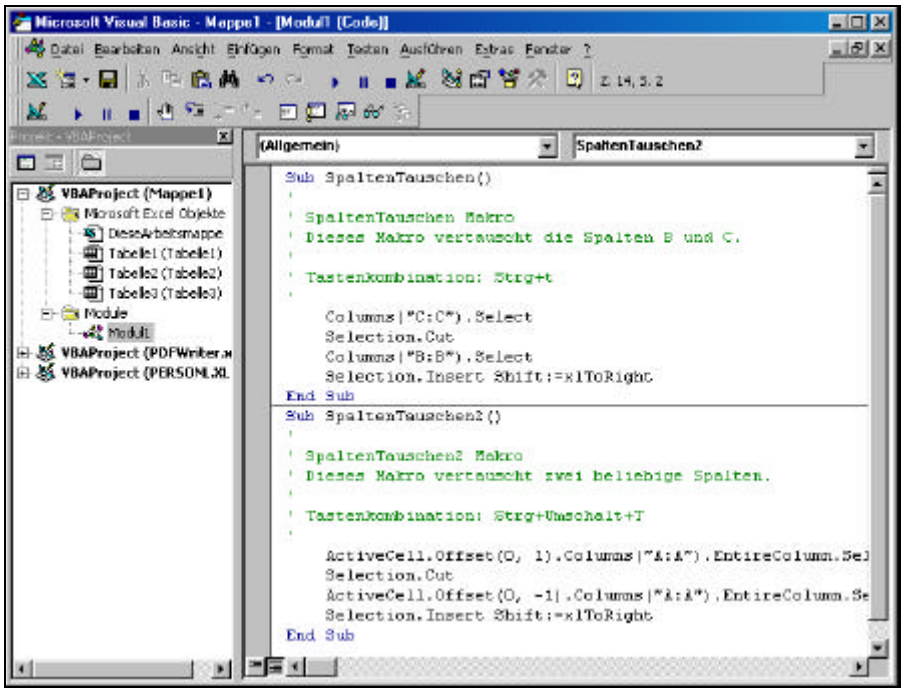
3 Der Visual-Basic-Editor

Visual-Basic-Modul

Die Makros werden durch den Makrorecorder automatisch in einem sogenannten Visual-Basic-Modul gespeichert. Dieses Modul wird zusammen mit der bei der Aufzeichnung ausgewählten Arbeitsmappe gespeichert.

Sie verwenden den Visual-Basic-Editor, um die Module sichtbar zu machen oder die Makros nachträglich noch zu editieren oder auch um neue Makros zu erstellen. Die Makros, die wir bei den ersten Übungen aufgezeichnet haben, finden wir im Modul 1.

Wählen Sie im Menü Extras den Menüpunkt Makro|Visual-Basic-Editor oder drücken Sie die Tastenkombination <Alt>+<F11>, um den Visual-Basic-Editor zu aktivieren. Ein neues Programmfenster für den Editor wird geöffnet. Die folgende Grafik zeigt das Aussehen dieses Fensters.



Im rechten Teil des Fensters werden die aufgezeichneten Makros angezeigt. Im linken Teil des Fensters sehen Sie den sogenannten Projekt-Explorer. Im Projekt-Explorer werden die verschiedenen geöffneten Mappen und die darin enthaltenen Objekte angezeigt. Ähnlich wie beim Explorer können Sie durch Klicken auf die Plus und Minuszeichen untergeordnete Strukturen sichtbar machen oder ausblenden.

Mit dem Visual-Basic-Editor steht Ihnen ein modernes und leistungsfähiges Programmentwicklungssystem zur Verfügung. Bevor wir uns näher die Handhabung des Editors anschauen. Betrachten wir zunächst die beiden zuvor aufgezeichneten Makros.

3.1 Analyse eines aufgezeichneten Makros

Das Visual-Basic-Modul enthält in den Zeilen die einzelnen Aufrufe der Funktionen. Diese werden in der vorgegebenen Reihenfolge ausgeführt. Die Funktionen, die zu einem Makro gehören, sind eingeklammert von den Schlüsselwörtern Sub und End Sub, die den Anfang, den Namen und das Ende des Makros festlegen. Durch unsere Übungen sind folgende Makros bereits entstanden:

```
Sub SpaltenTauschen()
' SpaltenTauschen Makro
' Dieses Makro vertauscht die Spalten B und C.
' Tastenkombination: Strg+t
'
    Columns("C:C").Select
    Selection.Cut
    Columns("B:B").Select
    Selection.Insert Shift:=xlToRight
End Sub

Sub SpaltenTauschen2()
' SpaltenTauschen2 Makro
' Dieses Makro vertauscht zwei beliebige Spalten.
' Tastenkombination: Strg+Umschalt+T
'
    ActiveCell.Offset(0,
1).Columns("A:A").EntireColumn.Select
    Selection.Cut
    ActiveCell.Offset(0,                                    -
1).Columns("A:A").EntireColumn.Select
    Selection.Insert Shift:=xlToRight
End Sub
```

Kommentare

Die grünen Textstellen sind Kommentare. Sie dienen lediglich zur Erläuterung. Hier wurden von Excel automatisch die von Ihnen eingegebenen Beschreibungen eingesetzt. Eine Kommentarzeile ist gekennzeichnet durch einen einfachen Anführungsstrich am Beginn der jeweiligen Zeile.

Schlüsselworte

Die blauen Textstellen sind die Excel-Schlüsselwörter. In diesen beiden Makros sind das nur die Worte Sub und End Sub. Es existieren aber noch weitere Schlüsselworte, von denen die meisten im Kapitel Kontrollstrukturen vorgestellt werden.

Anweisungen

Die Anweisungen und Excel-Funktionen werden in schwarzer Schrift angezeigt. Sie erkennen in beiden Makros genau die vier Anweisungen wieder, die Sie während der Aufzeichnung aufgenommen haben. Diese Anweisungen sollen im folgenden analysiert werden:

Visual-Basic ist eine objektorientierte Programmiersprache. Die meisten Befehle beziehen sich auf Objekte in einer Tabelle, einer Arbeitsmappe oder der Anwendung. Die Objekte haben Eigenschaften. Diese Eigenschaften können verändert werden. Zum Verändern der Eigenschaften eines Objektes können die Methoden benutzt werden. ActiveCell ist eine Eigenschaft (Property) der Anwendung. Sie liefert das Bereichsobjekt, das die aktuell aktive Zelle beschreibt. Mit der Methode Columns("A:A") wird die erste Spalte des Bereichs ermittelt. Mit der Eigenschaft EntireColumn kann das Bereichsobjekt ermittelt werden, das die ganze Spalte beschreibt. Mit der Methode Select wird der Bereich dieses Bereichsobjektes markiert. Im Gegensatz dazu sehen Sie im ersten Makro, daß direkt die Spalte C der Tabelle mit der Methode Columns("C:C") ausgewählt wird.

Selection ist eine Eigenschaft der Anwendung, die das Bereichsobjekt für den aktuell markierten Bereich beschreibt. Mit der Methode Cut wird auf diesen Bereich die Funktion des Menüpunktes „Bearbeiten|Ausschneiden“ angewandt. Durch die Methode Offset(0;-1) wird die Anwendungseigenschaft ActiveCell verändert. Sie wird um 0 Zeilen nach oben/unten und um eine Spalte nach links versetzt. Dadurch wird die zweite Spaltenmarkierung realisiert. Auch hier ist wie bei dem ersten Makro wieder direkt die Spalte B ausgewählt worden.

Mit der Methode Insert wird die Funktion des Menüpunktes „Einfügen|Zellen ausschneiden“ realisiert. Shift:=xlToRight ist ein Parameter für diese Funktion, der anzeigt, daß die vorhandenen Zellen nach rechts verschoben werden sollen.

3.2 Makros editieren

Sie können Makros auch von Hand erstellen. Um ein neues Modul anzulegen, wählen Sie den Menüpunt „Einfügen|Modul“. Die manuelle Erstellung eines Makros soll anhand eines kleinen Beispiels geübt werden. Geben Sie in ein leeres Makro-Modul folgende Zeilen ein:

```
Sub HelloWorld()
    MsgBox ("Hallo Welt")
End Sub
```

Setzen Sie die Schreibmarke in das Makro und wählen Sie den Menüpunkt [Ausführen|Sub/Userform ausführen] oder drücken Sie die Funktionstaste <F5>, um das Makro ausführen zu lassen. Mit diesem Makro wird lediglich in einem Hinweisfenster der Text „Hallo Welt“ angezeigt.

Durch die Zeile mit dem Schlüsselwort Sub wird der Name des Makros festgelegt. MsgBox ist eine Visual-Basic-Funktion, der als Parameter angegebene Text wird in dem Hinweisfenster angezeigt. Mit dem Schlüsselwort End Sub wird das Ende des Makros angezeigt. Innerhalb des Makro-Editors können Sie die von Texteditoren her bekannten Editier-Funktionen verwenden.

Mit einer einfachen Anweisung haben Sie auch die Möglichkeit, Eingaben vom Benutzer entgegen zu nehmen. Kopieren Sie das oben angegebene Makro in dem Sie es markieren und mit den Menüpunkt „Bearbeiten|Kopieren“ und „Bearbeiten|Einfügen“ kopieren. Erweitern Sie das Makro wie folgt:

```
Sub HalloName()
    Name = InputBox("Geben Sie Ihren Namen ein:")
    MsgBox ("Hallo " & Name)
End Sub
```

In der zweiten Zeile erkennen Sie eine sogenannte Zuweisung. Mit der Visual-Basic-Funktion InputBox wird eine Dialogbox für die Eingabe eines Wertes geöffnet. Der als Parameter angegebene Text wird in der Dialogbox angezeigt. Darunter kann der Benutzer in einem Editierfeld einen Wert eingeben. Nach dem Klicken des Aktionsschalters OK durch den Benutzer wird der Wert als Funktionswert der Funktion InputBox zurückgegeben.

Variablen

Mit Hilfe des Gleichheitszeichens wird dieser Wert der Variablen mit dem Namen Name zugewiesen. Eine Variable ist ein Name bzw. Platzhalter für eine Speicherzelle. Der Wert einer Variablen kann durch eine Zuweisung verändert werden. Die Variablen können Sie innerhalb von Formeln oder Funktionsaufrufen anstelle von Konstanten (Zahlen oder Zeichenketten) verwenden. Bei der Berechnung wird für die Variable ihr Wert verwendet. Dies sehen Sie am Beispiel der dritten Zeile. Für die Funktion MsgBox wird die Formel "Hallo " & Name als Parameter verwendet. Mit dieser Formel wird die Zeichenkette "Hallo " mit dem Wert der Variablen Name verknüpft. Das &-Zeichen ist der Operator zur Verknüpfung zweier Zeichenketten. Die resultierende Zeichenkette wird dann als Parameter für die Funktion verwendet.

Hinweis

Für die Variablennamen gelten die gleichen Bedingungen wie für die Namen in Excel.

Hilfe

Wenn Sie Makros editieren oder aufgezeichnete Makros analysieren wollen, kommt es häufig vor, daß Sie eine Beschreibung der benutzten Funktionen benötigen. Setzen Sie im Makro-Editor die Schreibmarke auf die entsprechende Funktion und drücken Sie die Taste <F1>. Die Hilfestellung zur markierten Funktion wird am Bildschirm angezeigt.

3.3 Die Makro-Symbolleiste

Mit Hilfe der Makro-Symbolleiste können Sie viele nützlichen Funktionen, die im Zusammenhang mit der Erstellung von Makros benötigt werden direkt aufrufen. Klicken Sie dazu auf eines der nachfolgend beschriebenen Symbole:

Mit diesem Symbol wechseln Sie wieder zurück zum Excel-Tabellenblatt. Da der Visual-Basic-Editor ein eigenes Programmfenster hat, können Sie aber auch die Tastenkombination <Alt>+<Tab> verwenden, um zwischen den Fenstern von Excel und Visual-Basic zu wechseln.

Mit diesem Symbol können Sie weitere Module und andere Visual-Basic-Elemente wie z. B. die später noch behandelte Userform einfügen.

Die folgenden sechs Symbole entsprechen in der Funktion den entsprechenden Symbolen, die Sie bereits vom Excel-Fenster her kennen. Das Fernglas dient zum Suchen von Texten in den Makro-Modulen.

Mit diesem Symbol können Sie die Ausführung eines Makros starten, dies entspricht dem Drücken der Taste <F5> oder der Auswahl des entsprechenden Menüpunktes. Es wird immer das Makro gestartet in dem sich gerade die Schreibmarke befindet.

Mit diesem Symbol können Sie einen laufendes Makro unterbrechen. Alternativ können Sie auch die Tastenkombination <Strg>+<Pause> verwenden. Dies ist insbesondere dann sinnvoll, wenn Sie das Symbol nicht klicken können, weil z. B. die Sanduhr anstelle des Mauszeigers angezeigt wird.

Mit diesem Symbol können Sie ein laufendes Makro beenden. Diese Funktion wird besonders in Verbindung mit dem Testmodus benötigt.

Mit diesem Symbol können Sie bei der Bearbeitung von sogenannten Userforms zwischen dem Entwurfs- und dem Testmodus umschalten. Die Userforms werden später noch ausführlicher behandelt.

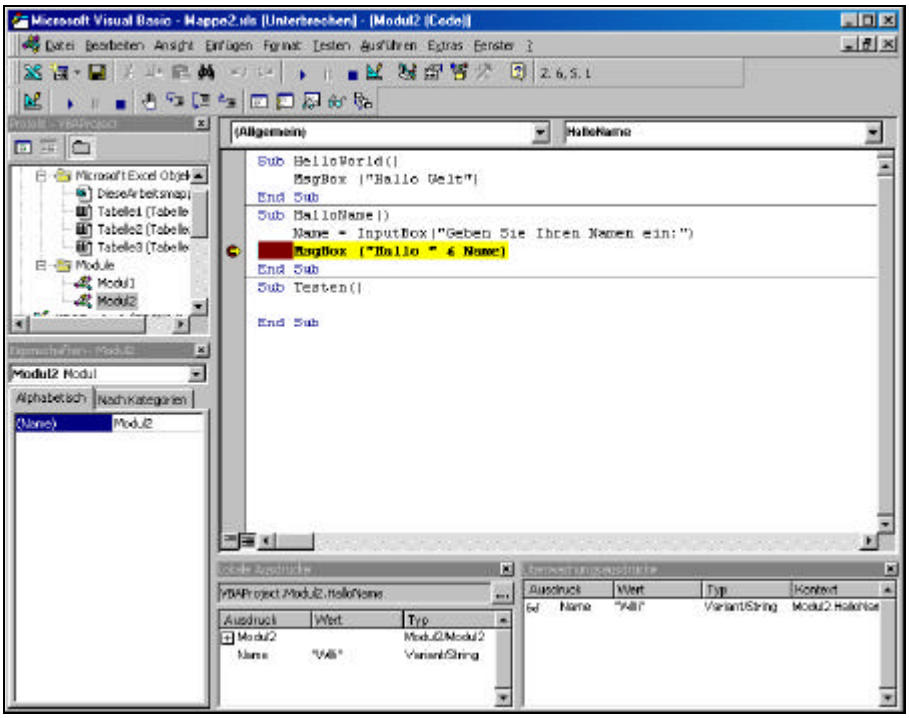
Mit diesem Symbol können Sie das Fenster des Projekt-Explorers sichtbar machen oder aktivieren. Das Ausblenden des Fensters erfolgt über das Kreuz in der Kopfzeile.

Mit diesem Symbol können Sie auf der linken Seite zusätzlich das Eigenschaftenfenster einblenden. Dieses Fenster zeigt Ihnen die Eigenschaften ausgewählter Objekte. Der Einsatz dieses Fensters ist insbesondere im Zusammenhang mit den Userforms sinnvoll.

Mit diesem Symbol blenden Sie das Fenster des Objektkataloges auf der rechten Seite ein. Der Objektkatalog gibt Ihnen eine Orientierung bei der Auswahl von Objekten, Eigenschaften und Funktionen. Das Ausblenden erfolgt durch das Einblenden eines Moduls mit einem Doppelklick im Fenster des Projekt-Explorers.

Mit diesem Symbol können Sie eine zusätzliche Symbolleiste anzeigen lassen, die zum Einfügen und Editieren der Dialogelemente in einer Userform verwendet werden.

Wenn Sie ein Makro geschrieben haben und das Makro funktioniert nicht in der gewünschten Weise, dann kann es sinnvoll sein, das Makro schrittweise ausführen zu lassen. Dabei beobachten Sie dann, ob alle Befehle korrekt ausgeführt werden und die verwendeten Variablen sich in der gewünschten Weise verändern. Im Programmierer-Latein spricht man von debuggen. Die nachfolgende Grafik zeigt, wie das Fenster des Visual-Basic-Editors in einer solchen Situation aussehen kann.



Mit den folgenden Symbolen haben Sie die Möglichkeit ein Makro schrittweise ausführen zu lassen und die Veränderung von benutzten Variablen zu beobachten.

Mit diesem Symbol können Sie auf die Zeile, die Sie zuvor im Editor ausgewählt haben, einen Haltepunkt setzen. Bei der Ausführung des Makros wird dann an dieser Stelle angehalten und Sie können die Funktion des Makros bis zu diesem Punkt überprüfen.

Mit diesem Symbol können Sie nach Erreichen eines Haltepunktes die Ausführung des Makros schrittweise fortsetzen. Mit jedem Klick auf dieses Symbol wird ein Befehl ausgeführt. Wenn Sie innerhalb eines Makros ein anderes Makro aufrufen. Man spricht dann von Unterprogrammen. Durch Klicken auf dieses Symbol werden auch die Unterprogramme schrittweise ausgeführt. Sie können das Symbol auch von Beginn an benutzen, um ein Makro komplett schrittweise ausführen zu lassen.

Mit diesem Symbol können Sie wie mit dem vorigen Symbol ein Makro schrittweise ausführen. Im Unterschied zum vorhergehenden Symbol werden Unterprogramme aber in einem Schritt ausgeführt.

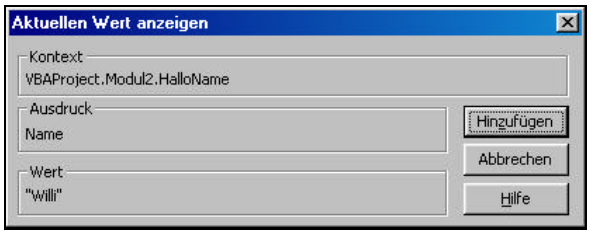
Mit diesem Symbol können Sie ein schrittweise aufgerufenes Unterprogramm in einem Schritt verlassen und zum nächsten Befehl in dem aufrufenden Programm gelangen.

Mit diesem Symbol können Sie das Fenster zur Anzeige aller in einem Makro benutzten sogenannten lokalen Variablen anzeigen lassen. Dies ist beim Debuggen sehr hilfreich, um die Veränderung der Werte sehen zu können.

Mit diesem Symbol können Sie das sogenannte Direktfenster anzeigen lassen. Dieses Fenster ermöglicht während eine Debugsitzung Befehle direkt einzugeben und damit das Verändern von benutzten Werten oder das Ausprobieren zusätzlicher Befehle.

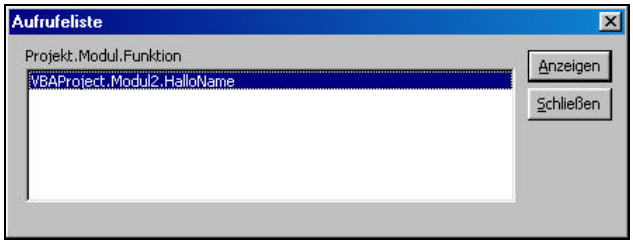
Mit diesem Symbol können Sie das Fenster zur Überwachung von beliebigen Variablen einblenden. Während das Fenster für die lokalen Symbole immer nur die Variablen anzeigt, die von dem im Moment debuggten Unterprogramm genutzt werden, können Sie hier auch Variablen sichtbar machen, die in einem anderen Kontext benutzt werden.

Mit diesem Symbol öffnen Sie eine Dialogbox, die Ihnen den aktuellen Zustand einer Variablen anzeigt. Die gewünschte Variable ist zuvor im Quelltext des Moduls mit der Maus anzuklicken und dann klicken Sie auf dieses Symbol. Die folgende Dialogbox wird angezeigt:



In diesem Fenster werden Name, Wert und Kontext der ausgewählten Variable angezeigt. Klicken Sie auf den Aktionsschalter Hinzufügen und die Anzeige erfolgt dauerhaft im Überwachungsfenster.

Bei Verwendung der Unterprogramm-Technik ist es oftmals sinnvoll herauszufinden. Wie es zu dem Aufruf eines bestimmten Unterprogramms kam. Die folgende Dialogbox zeigt Ihnen eine Liste aller aufgerufene Makros in der Reihenfolge ihres Aufrufs.



Unterprogramme

In Visual-Basic ist es möglich, in einem Makro andere Makros aufzurufen. Dadurch können Sie sich bei der Makro-Programmierung die Wiederholung von Befehlssequenzen ersparen oder komplexe Makros modularisieren, d. h. in mehrere kleine Teilaufgaben (Unterprogramme) zerlegen. Dies verbessert die Übersicht und erleichtert das Testen.

4 Grundlagen von Visual-Basic

Visual-Basic für Applikationen ist quasi der kleine Bruder des von Microsoft für Windows vertriebenen Entwicklungssystems Visual-Basic. Es ist an Excel angepaßt und enthält alle wichtigen Elemente einer Programmiersprache für Windows. Mit Visual-Basic ist die Entwicklung richtiger Applikationen für Excel möglich. Sie können sowohl Programme als auch eigene Excel-Funktions-Bibliotheken erstellen. In den folgenden Kapiteln sollen die wichtigsten Elemente dieser Programmiersprache vorgestellt und an kleinen Beispielen demonstriert werden.

4.1 Objekte, Eigenschaften und Methoden

Objekte

Visual-Basic ist eine objektorientierte Programmiersprache. Objekte sind in Excel wie Dinge im täglichen Leben. Diese Dinge haben bestimmte Eigenschaften (Properties). Nehmen wir z. B. ein Auto. Das Auto hat u. a. die Eigenschaften Farbe, Hubraum oder Leistung. Diese Eigenschaften bestimmen das Verhalten und das Aussehen des Objektes.

Eigenschaften

In Excel gibt es eine Reihe vordefinierter Objekte, die durch ihre Eigenschaften beschrieben sind, z. B. die Anwendung, die Arbeitsmappe (Workbook), das Tabellenblatt (Worksheet), der Bereich (Range) oder ein Diagramm. So hat z. B. das Tabellenblatt neben anderen Eigenschaften die Eigenschaft „Sichtbar“ (Visible). Diese beschreibt, ob das Tabellenblatt gerade sichtbar ist oder nicht.

Methoden

Darüber hinaus gibt es für die Objekte bestimmte Methoden mit denen ein Objekt in Aktion versetzt werden kann. Beim Auto könnten das z. B. die Methoden Gang einlegen oder kuppeln sein. Für ein Tabellenblatt gibt es z. B. die Methode „Berechnen“ (Calculate). Mit dieser können Sie ein Tabellenblatt neu berechnen lassen.

Sie können die Objekte auf folgende Arten behandeln:

- Eigenschaften abfragen
  Sie können den Zustand eines Objektes ermitteln, in dem Sie die Eigenschaften des Objektes abfragen.
- Eigenschaften verändern
  Sie können den Zustand eines Objektes verändern, in dem Sie die Eigenschaften eines Objektes durch eine Zuweisung verändern. Nicht immer sind alle Eigenschaften veränderbar, manche Eigenschaften können auch nur abgefragt werden.
- Aktionen veranlassen
  Sie können ein Objekt zu einer bestimmten Aktion veranlassen, in dem Sie eine Methode aufrufen.

Wir wollen diese drei Möglichkeiten an einfachen Beispielen kennenlernen. Als Beispiel-Objekt nehmen wir den Bereich. Ein Bereich ist ein beliebiger Ausschnitt aus einer Tabelle. Er kann eine oder mehrere Zellen umfassen.

4.2 Anweisung, Zuweisung und Variablen

Das Makro zum Ermitteln bzw. Verändern der Eigenschaften eines Bereichsobjektes könnte dann wie folgt aussehen:

```
Sub ObjekteManipulieren()
    Worksheets("Tabelle1").Activate
    ZellenWert = Range("A2").Value
    Range("A3").Value = ZellenWert
End Sub
```

Beachten Sie bitte, daß bei der Verwendung der Objekte zunächst ein Objekt und dann abgetrennt durch einen Punkt eine Methode oder Eigenschaft angegeben wird. Wobei Worksheets und Range selbst wieder Methoden der Objekte Workbook bzw. Worksheet sind, sie beziehen sich in diesem Fall auf die jeweils aktive Arbeitsmappe bzw. Tabelle und liefern als Ergebnis eben ein Bereichsobjekt oder ein Tabellenblattobjekt zurück. In der dritten Zeile wird der Wert eines Objektes, hier speziell der einzelnen Zelle A2 gelesen und durch eine Zuweisung (mit Hilfe des Gleichheitszeichens) in die Variable Zellenwert übertragen. Eine Variable ist eine Speicherzelle die (zumindest in diesem Fall) beliebige Werte aufnehmen. Der Variablenname wird als Platzhalter in Ausdrücken oder Formeln verwendet. An dessen Stelle tritt dann bei der Ausführung des Makros der Wert der Variablen. Sie kennen sicherlich alle noch die Funktion von Platzhaltern aus dem Mathematikunterricht. Einer Variablen können Sie mit einer Zuweisung einen Wert zuweisen. Dann steht der Variablenname auf der rechten Seite des Gleichheitszeichens. Eine Variable, der noch kein Wert zugewiesen wurde, hat den Wert „Leer“. Den Namen einer Variablen können Sie beliebig auswählen, es gelten die normalen Regeln für Namen in Excel. Der Name darf nicht mit einem bereits existierenden Namen übereinstimmen.

In der vierten Zeile wird dann umgekehrt der Wert dieser Variablen einem Bereichsobjekt zugewiesen. Das Aktivieren des Tabellenblattes ist nicht unbedingt notwendig, wenn Sie kein Tabellenblatt auswählen, dann wird das aktuell ausgewählte Tabellenblatt verwendet.

Alternativ hätten Sie den gleichen Effekt, auch mit dem folgenden Makro erreichen können:

```
Sub ObjekteManipulieren2()
    ZellenWert = Worksheets("Tabelle1").Range("A2").Value
    Worksheets("Tabelle1").Range("A3").Value = ZellenWert
End Sub
```

Sie können sich sicherlich schon denken wie mit diesem oder einem ähnlichen Makro ein Wert von einem Tabellenblatt zum anderen kopiert werden kann.

Oder Sie hätten das ganze auch ohne Zuhilfenahme einer Hilfsvariablen durch eine direkte Zuweisung von einem Bereichsobjekt zum anderen machen können. Das folgende Beispiel zeigt Ihnen auch diese Möglichkeit:

```
Sub ObjekteManipulieren3()
   Worksheets("Tabelle1").Range("A3").Value = _
      Worksheets("Tabelle1").Range("A2").Value
End Sub
```

Hier fällt besonders auf, daß eine Visual-Basic-Anweisung über mehrere Zeilen geht. Visual-Basic arbeitet Ihre Makros zeilenweise ab. Wenn Sie eine Makroanweisung machen müssen, die über mehrere Zeilen geht, dann müssen Sie durch einen Unterstrich am Zeilenende anzeigen, daß auch die nächste Zeile noch zu dieser Anweisung gehört.

4.3 Eigenschaften und die With-Anweisung

Der Wert ist nicht die einzige veränderbare Eigenschaft eines Bereiches. Sie können z. B. auch die Formel für eine Zelle oder auch die Formatierung verändern. Weitere Möglichkeiten entnehmen Sie bitte der Excel-Hilfe. Die nachfolgenden Beispiele zeigen entsprechende Möglichkeiten:

```
Sub ObjekteManipulieren4()
   Range("A4").Formula = "=A2*A3"
   Range("A2:A4").NumberFormat = "#,##0.00"
End Sub
```

Objekte können komplexere Eigenschaften haben, die selbst wieder Objekte sind, und durch mehrere Eigenschaften beschrieben sind. Die Eigenschaft Schriftart z. B. ist selbst wieder ein Objekt, das quasi in dem Objekt Bereich enthalten ist, wie z. B. das Bereichsobjekt selbst in dem Objekt Tabellenblatt enthalten ist und dieses Objekt wiederum in dem Objekt Arbeitsmappe enthalten ist.

Sie können die Eigenschaften solcher Objekte entweder so:

```
Sub ObjekteManipulieren5()
   Range("A2:A4").Font.Name = "Times New Roman"
   Range("A2:A4").Font.Size = 24
   Range("A2:A4").Font.Bold = True
   Range("A2:A4").Font.Italic = True
End Sub
```

oder so verändern:

```
Sub ObjekteManipulieren6()
   With Range("A2:A4").Font
      .Name = "Times New Roman"
      .Size = 24
      .Bold = True
      .Italic = True
   End With
End Sub
```

With-Anweisung

Die With .. End With-Struktur dient zum Verkürzen der Codierung. Die zweite Möglichkeit bietet Ihnen geringeren Schreibaufwand und größere Übersichtlichkeit. Sie ist deshalb vorzuziehen. Grundsätzlich sind aber beide Möglichkeiten gleichwertig. Innerhalb der With-Anweisung erkennen Sie die zugehörige Eigenschaften an dem vorangestellten Punkt.

Zwar wurde schon der Aufruf einiger Methoden gezeigt, Worksheets, Activate oder Range. Es soll aber auch noch ein Beispiel für ein Bereichsobjekt gezeigt werden, das die Eigenschaften des Objektes sichtbar verändert.

Mit der Methode Autofit können Sie die Spaltenbreite optimal anpassen lassen. Diese Methode kann aber nur auf einen Bereich angewandt werden, der eine oder mehrere ganze Spalten repräsentiert. Die Eigenschaft EntireColumn ist für ein Bereichs-Objekt (hier der Bereich A2:A4) das entsprechende die ganze Spalte umfassende Bereichsobjekt zu dem der Bereich gehört hier die Spalte A.

```
Sub ObjekteManipulieren7()
   Range("A2:A4").EntireColumn.AutoFit
End Sub
```

An diesem Beispiel wird nochmals deutlich, daß eine Methode zur Veränderung der Eigenschaften und des Aussehens eines Objektes beitragen kann.

4.4 Adressierung von Objekten

Bisher haben Sie für die Adressierung von Bereichsobjekten oder Tabellenblättern immer die von den Bezügen her bekannte Notation mit Blattnamen und A1-Bezug benutzt. Grundsätzlich sind aber noch andere Möglichkeiten denkbar und in bestimmten Situationen vielleicht auch besser einzusetzen.

4.4.1 Arbeitsmappen

Workbooks

Workbooks ist ein Methode der Anwendung. Sie liefert aus der Liste der geöffneten Arbeitsmappen ein Arbeitsmappen-Objekt. Grundsätzlich haben Sie zwei Möglichkeit die gewünschte Arbeitsmappe zu selektieren. Entweder mit dem Namen oder mit dem Index.

```
Workbooks(2).Activate
```

Oder

```
Workbooks("Mappe2.xls").Activate
```

Der Index entspricht der Reihenfolge der letzten Aktivierung der Arbeitsmappen. Daher ist die Verwendung ist bei Verwendung des Index das Ergebnis nicht immer eindeutig. Die zweite Möglichkeit ist also zu bevorzugen.

Wenn Sie eine neue oder eine vorhandene Arbeitsmappe öffnen möchten können Sie auf die Workbooks-Liste die Methoden Add oder Open zurückgreifen.

```
Workbooks.Add
Workbooks.Open ("Mappe3.xls")
```

Mit dem ersten Befehl wird eine neue leere Arbeitsmappe angelegt mit dem zweiten Befehl wird die vorhandene Arbeitsmappe mit dem Namen mappe3.xls geöffnet. Bei der Methode Add können Sie einen Dateinamen als Parameter angeben, wenn Sie eine Vorlage verwenden möchten.

Zum Schließen von Arbeitsmappen verwenden Sie die Methode Close. Diese Methode steht sowohl für die ganze Arbeitsmappen-Liste als auch für eine einzelne Arbeitsmappe zur Verfügung.

```
Workbooks.Close
Workbooks("Mappe3.xls").Close(True)
```

Mit dem ersten Befehl werden alle derzeit offenen Arbeitsmappen geschlossen. Mit dem zweiten Befehl nur die mappe3.xls. Durch Angabe des Parameters True erreichen Sie, daß die Änderungen ohne Rückfrage gespeichert werden.

Sie haben auch die Möglichkeit Arbeitsmappen zu speichern, dazu stehen zwei Methoden zur Verfügung, die Sie Arbeitsmappen-Objekte anwenden können.

```
Workbooks("Mappe3.xls").Save
Workbooks("Mappe3.xls").SaveAs("Mappe4.xls")
```

Mit der ersten Methode wird Arbeitsmappe unter dem gleichen Namen gespeichert, z. B. zur Zwischenspeicherung. Mit der zweiten Anweisung können Sie die Arbeitsmappe unter einem anderen Namen speichern.

Die aktive Arbeitsmappe sprechen Sie mit der Eigenschaft ActiveWorkbook an.

4.4.2 Arbeitsblätter und Bereichsobjekte

Eine zweite wichtige Liste ist innerhalb der Arbeitsmappe die Liste der Tabellenblätter, Worksheets. Mit dieser Liste können Sie die einzelnen Blätter einer Arbeitsmappe auswählen. Auch bei der Tabellenblattliste können Sie sowohl mit dem Namen des Blattes als auch mit einem Index ein Tabellenblattobjekt auswählen.

```
Worksheets(2).Activate
Worksheets("Tabelle1").Activate
```

Mit dem ersten Methodenaufruf wählen Sie das zweite Tabellenblatt der Arbeitsmappe aus. Die Reihenfolge der Indizes entspricht der der Anordnung im Register. Die zweite Methode verwendet anstelle des Index wieder einen Namen, was in vielen Fällen besser ist.

Spalten

Innerhalb eines Tabellenblattobjektes oder eines Bereichsobjektes gibt es dann wiederum Möglichkeiten ganze Spalten auszuwählen.

```
Columns(2).Select
Columns("A:A").Select
Range("C1:F5").Columns("A:A").Select
```

Im ersten Beispiel wird die Spalte B des momentan aktiven Tabellenblattobjektes markiert. Im zweiten Beispiel wird die Spalte A. Im dritten Beispiel bezieht sich die Methode Columns nicht auf eine Tabelle sondern auf einen Bereich. In diesem Fall beziehen sich die Indizes, auch wenn Sie die Nomenklatur mit den Buchstaben verwenden, nur auf diesen Bereich. D. h. mit dem letzten Befehl werden die Zellen C1:C5 markiert.

Zeilen

Die Behandlung von Zeilen ist ähnlich. Die entsprechende Methode heißt Rows. Das folgende Beispiel zeigt wie Sie innerhalb eines Tabellenblattes die dritte Zeile markieren.

```
Rows(3).Select
```

Einzelne Zellen

Eine weitere Methode ermöglicht das Auswählen von einzelnen Zellen innerhalb einer Tabelle oder innerhalb eines Bereichsobjektes.

```
Cells(5,1).Select
Range("C1:F5").Cells(1,1).Select
```

Mit der ersten Anweisung wird die Zelle A5 ausgewählt. Beachten Sie, daß entgegen der sonst üblichen Notation zuerst die Zeilen- und dann die Spaltennummer anzugeben ist. Mit der zweiten Anweisung wird die Zelle C1 markiert. Die Cells-Methode bezieht sich wieder nur auf den angegebenen Bereich.

Bereiche

Ganze Bereiche von Zellen sprechen Sie am besten mit der Range-Methode an, die Sie bereits kennengelernt haben:

```
Range("A1:E5").Select
Range(Cells(1, 1), Cells(5, 5)).Select
```

Beide Anweisungen markieren den Bereich A1 bis E5. Im ersten Fall werden die von Formeln her vertrauten Bezüge zur Adressierung des Bereichs verwendet. Im zweiten Fall werden Zellobjekte zur Abgrenzung des Bereichs verwendet. Diese Möglichkeit erscheint zunächst umständlich, kann aber in Verbindung mit Makros durchaus sinnvoll eingesetzt.

Namen

Wenn Sie Bereichsobjekte anwenden möchten und in Ihren Tabellen für bestimmte Bereiche oder Zellen Namen festgelegt haben, so können Sie auch diese Namen zum Ansprechen eines Bereiches verwenden, z. B.

```
Range("Jahre").Select
```

Dieses Beispiel setzt voraus, daß zuvor in der Tabelle ein Bereich mit dem Namen Jahre definiert wurde.

4.5 Kontrollstrukturen

Mit Hilfe der Kontrollstrukturen können Sie den Programmfluß steuern. Als Beispiele sind hier zu nennen, die Fallunterscheidung oder auch die Wiederholung von bestimmten Anweisungen. Wenn Sie ein Makro aufzeichnen, werden Sie immer eine lineare Programmstruktur haben. Einer Anweisung folgt die nächste. Wenn Sie aber die Makros von Hand erstellen oder erweitern, können Sie Anweisungen wiederholen oder auch bestimmte Anweisungen auslassen oder nur unter bestimmten Bedingungen ausführen lassen. Diese Möglichkeit macht auch das Besondere bei der Erstellung eigener Makros aus.

4.5.1 If..Then..Else

Für die Fallunterscheidung gibt es mehrere Anweisungen, die Sie einsetzen können. Im einfachsten Fall müssen Sie nur feststellen, ob eine bestimmte Bedingung erfüllt ist, nur dann soll eine Aktion ausgeführt werden. Im ersten Beispiel soll für den Wert in der aktuell markierten Zelle ein Rabatt von 20% abgezogen und der neue Wert in diese Zelle zurückgeschrieben werden. Wenn die Zelle leer ist, soll vom Benutzer ein Wert für die Zelle angefordert werden. Wenn der Benutzer in dem Eingabedialog auf Abbrechen klickt, soll nichts weiter passieren, die Zelle bleibt leer. Die Lösung für dieses Problem könnte wie folgt aussehen:

```
Sub Rabatt1()
   Zellenwert = ActiveCell.Value
   If IsEmpty(ActiveCell) Then
      Zellenwert = InputBox("Geben Sie den Betrag ein:")
   End If
   If Zellenwert <> "" And IsNumeric(Zellenwert) Then
      ActiveCell.Value = 0.8 * Zellenwert
   End If
End Sub
```

Ein Fall

Hinter dem Schlüsselwort If wird eine Bedingung formuliert. Diese Bedingung kann eine beliebige Formel sein, die als Ergebnis einen logischen Wert liefert. Bei der ersten Wenn-Anweisung wird die Informations-Funktion IsEmpty aufgerufen, deren Funktionswert ist ein logischer Wert. Bei der zweiten If-Anweisung handelt es sich um einen Vergleich und den Aufruf einer logischen Funktion, die untereinander mit logischen Operator And verknüpft sind.

Oftmals werden zwei Fälle unterschieden und in beiden Fällen ist unterschiedlich zu reagieren. Für diesen Zweck werden Sie die If-Then-Else-Anweisung verwenden. Das vorhergehende Beispiel soll dahingehend erweitert werden, daß nur für den Fall, daß der Betrag größer ist als 1000 der Rabatt 20% betragen soll. Für kleinere Beträge soll nur ein Rabatt von 10% gelten.

Diese Aufgabe könnte wie folgt gelöst werden:

```
Sub Rabatt2()
   ZellenWert = ActiveCell.Value
   If IsEmpty(ActiveCell) Then
      ZellenWert = InputBox("Geben Sie den Betrag ein:")
   End If
   If ZellenWert <> "" And IsNumeric(ZellenWert) Then
      If ZellenWert >= 1000 Then
         ActiveCell.Value = 0.8 * ZellenWert
      Else
         ActiveCell.Value = 0.9 * ZellenWert
      End If
   End If
End Sub
```

Zwei Fälle

In diesem Beispiel sehen Sie gleich zwei Dinge. Zum einen, daß Sie die If-Anweisungen auch geschachtelt einsetzen können, und zum anderen die Möglichkeit bei der If-Anweisung neben dem Then-Teil einen Else-Teil einzusetzen. Dieser wird ausgeführt, wenn die Bedingung nicht erfüllt ist.

Oftmals müssen aber auch mehr als zwei Fälle unterschieden werden. Vom Prinzip her kann eine solche Aufgabenstellung mit mehreren in einander geschachtelten Wenn-Anweisungen gelöst werden. Eleganter ist aber oftmals die Verwendung des Schlüsselwortes ElseIf. Dies könnte dann wie folgt aussehen:

```
Sub Rabatt3()
   ZellenWert = ActiveCell.Value
   If IsEmpty(ActiveCell) Then
      ZellenWert = InputBox("Geben Sie den Betrag ein:")
   End If
   If ZellenWert <> "" And IsNumeric(ZellenWert) Then
      If ZellenWert >= 10000 Then
         ActiveCell.Value = 0.7 * ZellenWert
      ElseIf ZellenWert >= 1000 Then
         ActiveCell.Value = 0.8 * ZellenWert
      ElseIf ZellenWert >= 100 Then
         ActiveCell.Value = 0.9 * ZellenWert
      Else
         ActiveCell = ZellenWert
      End If
   End If
End Sub
```

In diesem Fall werden gleich vier Fälle unterschieden, wenn eine weitere Auswertung des Wertes notwendig ist, kann dies im Else-Teil mit dem Schlüsselwort ElseIf erfolgen. Beim letzten Fall kann immer noch das Schlüsselwort Else verwendet werden.

4.5.2 Select .. Case

Bei der If-Anweisung können Sie eine Fallunterscheidung durch mehrere hintereinander geschaltete Vergleiche vornehmen. Oftmals steht aber fest, daß nur ganz bestimmte Werte auftreten können. Dann können Sie auch die Select-Anweisung zur Fallunterscheidung verwenden. Die Aufgabenstellung wird dazu wie folgt abgeändert, wenn die aktive Zelle einen Wert enthält, dann wird in einer Dialogbox die Rabatt-Klasse abgefragt. Dies können zwölf Klassen 1 (5%), Klassen 2 und 3 (10%), Klassen 4 bis 6 und Klasse 8 (20%) und die übrigen Klassen (30%) sein. Entsprechend wird der Wert berechnet und in die aktive Zelle eingetragen. Die Lösung könnte dann wie folgt aussehen:

```
Sub Rabatt4()
   ZellenWert = ActiveCell.Value
   If IsNumeric(ZellenWert) Then
      Rabattklasse = InputBox("Rabattklasse?")
      If IsNumeric(Rabattklasse) Then
         Select Case Rabattklasse
            Case 1
               ActiveCell.Value = ZellenWert * 0.95
            Case 2, 3
               ActiveCell.Value = ZellenWert * 0.9
            Case 4 To 6, 8
               ActiveCell.Value = ZellenWert * 0.8
            Case 7, Is < 12
               ActiveCell.Value = ZellenWert * 0.7
            Case Else
               MsgBox ("Ungültige Rabattklasse")
         End Select
      Else
         MsgBox ("Ungültige Eingabe für Rabattklasse")
      End If
   End If
End Sub
```

Bei der Select-Anweisung muß der Wert der zu prüfenden Variablen genau mit einem der angegebenen Werte hinter dem Schlüsselwort Case übereinstimmen, trifft dies nicht zu, dann wird der nächste Fall (Case) überprüft. Wie Sie bei dem zweiten Case sehen, ist es möglich, durch Komma getrennt mehrere Werte in einem Fall (Case) zusammen zu fassen. Bei dem dritten Case wird von der Möglichkeit Gebrauch gemacht mit dem Schlüsselwort To auch Bereiche anzugeben, diese können dann wiederum Bestandteil einer Aufzählung sein. Der vierte Fall zeigt die Verwendung des Schlüsselwortes Is. Damit ist es möglich, Vergleichsoperatoren bei der Fallunterscheidung zu verwenden. Mit dem letzten Case werden dann alle anderen Fälle behandelt. Die Angabe eines Case Else ist optional. Dies kann weggelassen werden, dann würde durch die Select-Anweisung unter Umständen eben nichts ausgeführt.

Hinweis

Wichtig für Funktion ist meist die Reihenfolge der Abfragen. Dies gilt sowohl für die Select- als auch für ElseIf-Anweisung. Wenn z. B. im oben gezeigten Beispiel der vierte Fall zuerst angegeben würde, dann würde für alle Rabattklassen ein Rabatt von 70% berechnet. In jedem Fall läßt sich eine Select-Anweisung durch eine entsprechende Konstruktion von If-Anweisungen ersetzen.

4.5.3 For .. Next

Häufig ist es auch nötig eine Anweisung mehrfach ausführen zu lassen. Für solche Fälle gibt es die Wiederholungs-Anweisungen. Diese ermöglichen es, eine oder mehrere Anweisungen wiederholen zu lassen. Die einfachste Form der Wiederholungs-Anweisungen, die For-Schleife, kann eingesetzt werden, wenn die Anzahl der Wiederholungen bekannt ist. Wir wollen z. B. in einer Tabelle einen Bereich mit berechneten Werte füllen und zwar in der folgenden Art und Weise:

2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19
11	12	13	14	15	16	17	18	19	20

Die Inhalte der einzelnen Zellen ergeben sich aus der Summe der Spaltennummer und der Zeilennummer innerhalb des Bereichs. Die Lösung für diese Aufgabe könnte wie folgt aussehen:

```
Sub BereichFüllen()
   For SpaltenNr = 1 To 10
      For ZeilenNr = 1 To 10
         Cells(ZeilenNr, SpaltenNr).Value = _
           SpaltenNr + ZeilenNr
        Next ZeilenNr
    Next SpaltenNr
End Sub
```

In dem Beispiel erkennen Sie zwei ineinander geschachtelte For-Anweisungen. Die Anweisungen zwischen den beiden Schlüsselworten For und Next werden jeweils 10mal wiederholt, da die Zähler für die beiden For-Anweisungen (SpaltenNr und ZeilenNr) automatisch jeweils von 1 bis 10 gezählt werden und bei jedem Mal die Anweisungen in der For-Anweisung wiederholt werden. Da in diesem Beispiel zwei For-Anweisungen ineinander verschachtelt sind, wird die innere Anweisung (Cells...) insgesamt 10 mal 10mal also 100mal ausgeführt. Der Bereich wird in der angebenen Weise gefüllt.

In der nächsten Aufgabe soll das Beispiel so abgeändert werden, daß nur jede zweite Zelle gefüllt werden soll, also eine Art Schachbrettmuster aus gefüllten und nicht gefüllten Zellen entsteht.

Die Lösung für die Aufgabe könnte wie folgt aussehen:

```
Sub BereichFüllen2()
   For SpaltenNr = 1 To 10
      ' Prüfen, ob Spaltennummer gerade ist
      If SpaltenNr Mod 2 = 0 Then
         For ZeilenNr = 2 To 10 Step 2
            Cells(ZeilenNr, SpaltenNr).Value = _
              SpaltenNr + ZeilenNr
         Next ZeilenNr
      Else
         For ZeilenNr = 1 To 10 Step 2
            Cells(ZeilenNr, SpaltenNr).Value = _
              SpaltenNr + ZeilenNr
         Next ZeilenNr
      End If
   Next SpaltenNr
End Sub
```

In dem Beispiel sehen Sie, daß Sie zum Zählen nicht immer mit 1 beginnen müssen. Vielmehr können Sie beliebige Start- und Endwerte verwenden. Ferner erkennen Sie, daß die letzte Ausführung mit der größten Zahl die kleiner oder gleich der bei Bis angegebenen Zahl ist, stattfindet. Neu an diesem Beispiel ist, die Angabe einer Schrittweite für die inneren Schleifen. Damit erreichen Sie, daß der Schleifenzähler, hier die Variable ZeilenNr, bei jedem Schleifendurchlauf nicht um 1 sondern wie im Beispiel um 2 erhöht wird. Sie können als Schrittweite aber auch Dezimalbrüche verwenden, wie das nächste Beispiel zeigt:

```
Sub BereichFüllen3()
   For NeuerWert = 1 To 5 Step 0.5
      ActiveCell.Offset(1, 0).Select
      ActiveCell.Value = NeuerWert
   Next NeuerWert
End Sub
```

Dieses Makro füllt ab der aktuellen Position der Markierung eine Spalte mit Werten zwischen 1 und 5 im Abstand von 0,5.

Nicht immer muß die Anzahl der Durchläufe bei der Programmierung schon bekannt sein. Dies zeigt z. B. das nachfolgende Beispiel:

```
Sub VieleBereicheLöschen()
   For nIndex = 1 To Selection.Areas.Count
      Selection.Areas(nIndex).ClearContents
   Next nIndex
End Sub
```

Dieses Makro löscht alle augenblicklich markierten Bereiche auch bei einer Mehrfachauswahl. Dazu ermittelt es durch die Anweisung Selection.Areas.Count die Anzahl der markierten Bereiche. Durch die For-Anweisung wird dann die Anweisung in der dritten Zeile für alle markierten Bereiche wiederholt.

Eine weitere Variante der For-Anweisung ermöglicht es, z. B. alle markierten Zellen eines Bereiches anzusprechen. Nehmen wir als Beispiel die folgende Aufgabenstellung: In dem markierten Bereich sollen die Werte aller Zellen überprüft werden, wenn die Zahlen in dem Bereich größer sind als ein einzugebender Grenzwert, sollen sie durch rote Schriftfarbe hervorgehoben werden und sonst nicht.

Die Lösung für diese Aufgabe könnte wie folgt aussehen:

```
Sub Zahlenmarkieren()
   Grenzwert = InputBox("Geben Sie den Grenzwert ein:")
   If IsNumeric(Grenzwert) Then
      For Each Zellen In Selection
         If IsNumeric(Zellen.Value) Then
            If Zellen.Value > CDbl(Grenzwert) Then
               Zellen.Font.ColorIndex = 3
            Else
               Zellen.Font.ColorIndex = xlAutomatisch
            End If
         End If
      Next Zellen
   Else
      MsgBox ("Falsche Eingabe")
   End If
End Sub
```

Mit der For Each-Anweisung können Sie eine für alle Objekte (hier Zellen) einer Objektmenge (hier Selection, dies ist der markierte Bereich) die Anweisungen zwischen den Schlüsselworten For und Next wiederholen lassen. Dies ist oftmals ganz nützlich, wenn es darum geht wie hier mit einer zum Zeitpunkt der Programmierung nicht bekannten Auswahl zu arbeiten.

Eine weitere Besonderheit erkennen Sie in diesem Beispiel. Die Variable Grenzwert hat den Datentyp String (Zeichenfolge). Zellen.Value jedoch hat den Datentyp Double. Daher wird ein Vergleich mit folgender Anweisung Zellen.Wert > Grenzwert nicht das gewünschte Ergebnis bringen. Für den Vergleich müssen die Datentypen der beiden Variablen zunächst angepaßt werden. Dies geschieht mit Umwandlungsfunktion CDbl. Diese Funktion liefert den der Zeichenfolge entsprechenden Zahlenwert. Die ebenfalls existierende Funktion Val sollten Sie nicht verwenden, da diese nicht die länderspezifischen Einstellungen bezüglich des Dezimaltrennzeichens berücksichtigt. Beachten Sie hierzu auch das Kapitel zu den Datentypen.

Hinweis

Grundsätzlich haben Sie die Möglichkeit eine For-Anweisung vorzeitig durch Verwendung der Anweisung Exit For an beliebiger Stelle innerhalb des For-Anweisungsblockes zu verlassen. Im Sinne der strukturierten Programmierung wird von der Verwendung dieser Möglichkeit jedoch abgeraten.

4.5.4 Do .. Loop

Die For-Anweisung können Sie immer dann einsetzen, wenn die Anzahl der Schleifendurchläufe zum Zeitpunkt der Programmierung bekannt ist oder sich berechnen läßt bzw. sich z. B. aus der aktuellen Auswahl ergibt. Was aber, wenn die Anzahl der Durchläufe allein von dem Verhalten des Benutzers während des Durchlaufs einer Schleife abhängt? Als Beispiel soll die folgende Aufgabenstellung dienen: Schreiben Sie ein Makro, daß ab der aktuellen Position der Markierung Werte in eine Spalte einfüllt. Die Werte sollen vom Benutzer mit einem Eingabe-Dialog abgefragt werden. Wenn der Benutzer in dem Dialog auf den Aktionsschalter Abbrechen klickt, dann soll die Abfrage beendet werden. Die Lösung für die Aufgabe könnte wie folgt aussehen:

```
Sub BereichFüllen4()
   Do
      WertfürZelle = InputBox("Geben Sie den Wert ein:")
      If WertfürZelle <> "" Then
         ActiveCell.Value = WertfürZelle
         ActiveCell.Offset(1, 0).Select
      End If
   Loop While WertfürZelle <> ""
End Sub
```

Bei dieser Schleifenkonstruktion wird die Schleife solange ausgeführt, wie die Bedingung am Ende der Schleife erfüllt ist. Der erste Schleifendurchlauf findet in jedem Fall statt, da die Abbruchbedingung am Ende der Schleife angegeben ist. Damit haben Sie die Gelegenheit auf das zum Zeitpunkt der Programmierung nicht bekannte Verhalten des Benutzers zu reagieren.

Die Do-Schleife gibt es in unterschiedlichen Varianten. Die je nach Aufgabenstellung verwendet werden können. Neben der vorgestellten Variante gibt es noch die folgenden:

```
Do While Bedingung
    Anweisungen
Loop
```

Bei dieser Konstruktion können Sie auch schon den ersten Schleifendurchlauf vermeiden. Die Schleife wird solange wiederholt, wie die Bedingung wahr ist.

```
Do
    Anweisungen
Loop Until Bedingung
```

Diese Schleife läuft bis die Abbruchbedingung erfüllt ist. Der erste Schleifendurchlauf wird auf jeden Fall ausgeführt.

```
Do Until Bedingung
    Anweisungen
Loop
```

Wie bei der vorhergehenden Konstruktion, wobei Sie auch wieder den ersten Schleifendurchlauf vermeiden können. Man spricht deshalb auch von abweisenden Schleifen.

Grundsätzlich könnten Sie die For-Schleife immer auch mit einer Do-Schleife realisieren. Doch in vielen Fällen ist die Verwendung der For-Schleife einfach bequemer.

Hinweis Ein vorzeitiges Verlassen ist mit der Anweisung Exit Do. Siehe hierzu auch Hinweis im vorhergehenden Abschnitt.

4.6 Variablen und Datentypen

In Visual-Basic haben alle implizit deklarierten Variablen den Datentyp Variant. Dieser Datentyp ist in der Lage beliebigen Werte anzunehmen. Dies gilt für alle bisher von uns verwendeten Variablen.

Wenn Sie größere Projekte mit Visual-Basic realisieren, empfiehlt es sich die Variablen explizit zu deklarieren. Dies hat in Visual-Basic mehrere Vorteile.

- Besseres Laufzeitverhalten
- Bessere Speicherplatzausnutzung
- Typ-Überprüfung während der Laufzeit, dies hilft logische Programmier-Fehler zu finden

In Visual-Basic sind folgende Datentypen integriert:

Boolean Für logische Größen, Wahrheitswerte, Speicherplatzbedarf 2 Byte, mögliche Wert sind True (Wahr) oder False (Falsch).

Byte Für ganze Zahlen ohne Vorzeichen, Speicherplatzbedarf 1 Byte, mögliche Werte zwischen 0 und 255.

Integer Für ganze Zahlen, Speicherplatzbedarf 2 Byte, mögliche Werte zwischen -32768 und 32767. Das Typkennzeichen für die implizite Deklaration ist das %.

Long Für ganze Zahlen, Speicherplatzbedarf 4 Byte, mögliche Werte zwischen –2147483648 und 2147483647. Das Typkennzeichnen für die implizite Deklaration ist das &.

Currency Für das Rechnen mit Geldbeträgen, Speicherplatzbedarf 8 Byte, mögliche Werte zwischen -922.337.203.685.477,5808 und 922.337.203.685.477,5807. Bei den Currency-Variablen, handelt es sich um einen Festkommadatentyp. Das Typkennzeichnen für die implizite Deklaration ist das @.

Single Für Gleitkommanzahlen, Speicherplatzbedarf 4 Byte (IEEE-Standard), mögliche Werte liegen im Bereich zwischen 1,401298E-45 bis 3,402823E38 sowohl für positive als auch für negative Zahlen. Das Typkennzeichen für die implizite Deklaration ist !. Genauigkeit ca. 7 signifikante Dezimalstellen.

Double Für Gleitkommazahlen, Speicherplatzbedarf 8 Byte (IEEE-Standard), mögliche Werte liegen im Bereich zwischen 4,94065645841247E-324 bis 1,79769313486232E308 sowohl für positive als auch für negative Zahlen. Das Typkennzeichen für die implizite Deklaration ist #. Genauigkeit ca. 15 signifikante Dezimalstellen.

Date Für Datumsangaben, Speicherplatzbedarf 8 Byte, intern behandelt wie eine Double. Der Vorkammateil repräsentiert den Tag, 0 = 1.1.1900 und der Nachkommateil repräsentiert einen Bruchteil von 24 Stunden, z. B. 0,5 entspricht 12 Uhr.

String Für Zeichenketten, der Speicherplatzbedarf hängt von der Länge der Zeichenkette ab, bei der Deklaration mit variabler Länge können dies bis zu ca. 2 Milliarden Zeichen sein, bei fester Länge wird der Speicherplatz durch die entsprechende Angabe bei der Deklaration bestimmt und kann maximal ca. 64000 Zeichen betragen. Jedes Zeichen belegt ein Byte. Das Typekennzeichen für Zeichenketten ist $.

Object Für Objektreferenzen, der Speicherplatzbedarf beträgt 4 Byte. Die Variable speichert eine Referenz (Speicheradresse) eines beliebigen Objektes.

Variant Für beliebige Daten, Speicherplatzbedarf mindestens 16 Byte, darüber hinaus abhängig vom tatsächlich zugewiesenen Wert, wird ggf. dynamisch erweitert.

Implizite Deklaration

Die implizite Deklaration ist das was bisher in den Beispielen gezeigt wurde. Die Variable wird einfach verwendet. Wenn kein besonderes Typkennzeichen angegeben wird, haben die Variablen automatisch den Typ Variant und können somit jeden beliebigen Wert annehmen.

Eine Ausnahme kann mit Verwendung der Def*typ*-Anweisung erreicht werden. Dies ist jedoch veralteter Programmierstil und sollte nicht verwendet werden. Durch nachgestellte Typkennzeichnen können Sie aber auch bestimmte Datentypen für Ihre Variablen festlegen, siehe dazu auch nachfolgende Beispiele:

```
Zellenwert = 12
```

Deklariert implizit eine Variable vom Typ Variant.

```
Zellenwert$ = "Test"
```

Deklariert implizit eine Variable vom Typ String. Sie erkennen dies an dem nachgestellten $-Zeichen.

```
Zellenwert$ = 12
```

Diese Zuweisung weist der Variablen nicht den Wert der Zahl 12 zu sondern deren Darstellung als Zeichenkette.

```
Zahlwert = CInt( ZellenWert$ )
```

Falls eine Typumwandlung nicht automatisch in der gewünschten Weise erfolgt (Typkonflikt), ist es besser diese mit Hilfe einer Typumwandlungsfunktion zu programmieren. Entsprechende Typumwandlungsfunktionen stehen auch für die anderen Datentypen zur Verfügung.

Implizite Variablen gelten nur in dem Anweisungsblock in dem Sie verwendet wurden. Verwenden Sie in einem anderen Makro (Sub) eine Variable mit dem gleichen Namen so ist dies eine andere Variable.

Explizite Deklaration

Die explizite Deklaration erfolgt gewöhnlich am Anfang eines Blockes unter Verwendung des Schlüsselwortes DIM und unter Angabe des Datentyps:

```
Dim Beliebig
Dim Zähler As Integer
Dim Name As String * 20
```

Das erste Beispiel zeigt die Deklaration einer Variablen mit dem Namen Beliebig ohne Angabe eines Datentyps. Damit hat diese Variable automatisch den Datentyp Variant. Das zweite Beispiel zeigt die Deklaration einer Variablen mit dem Namen Zähler durch das Schlüsselwort As und der nachfolgenden Typangabe wird als Typ für diese Variable Integer bestimmt. Das dritte Beispiel zeigt die Deklaration einer Variable mit dem Namen Name als String-Variable mit fester Länge. Die Länge wird durch den Nachspann * 20 festgelegt.

Hinweis

Gewöhnlich werden Strings variabler Länge verwendet, dann entfällt die Längenangabe, die Variable wird einfach mit As String deklariert.

Geltungsbereich

Entscheidend für den Geltungsbereich einer Variablen ist, wo sie deklariert wird. Wird die Deklaration innerhalb eines Makros vorgenommen, so gilt diese Variable nur in diesem Makro (lokal). Wird die Deklaration hingegen außerhalb des Makro vorgenommen gilt sie im gesamten Modul in allen Makros, wenn nicht innerhalb eines Makros eine Variable mit dem gleichen Namen deklariert wird. Denn dann hat die lokale Definition Vorrang vor der globalen Definition. Soll eine Variable für das ganze Projekt deklariert werden, dann ist ihr das Schlüsselwort Public voranzustellen.

```
Public Wichtig As Double
```

Mit diesem Beispiel wird eine projektglobale Variable mit dem Namen Wichtig vom Typ Double deklariert. Die Verwendung des Schlüsselwortes Public ist nur außerhalb von Makros zulässig. Globale Variablen sollten im Sinne einer guten Programmierpraxis vermieden werden.

Option Explicit

Wenn Sie diese Anweisung als erste in einem Modul angeben, ist die implizite Deklaration in diesem Modul nicht mehr zulässig. Für größere Projekte wird dies empfohlen.

4.7 Unterprogramm-Technik

Wenn Sie umfangreiche Makro-Projekte zu erstellen haben, führt dies leicht zu großen und unübersichtlichen Programmstrukturen. Im Sinne einer guten Programmierpraxis sollten Sie ihr Projekt modularisieren, d. h. in Teilaufgaben zerlegen, diese einzeln und in einem überschaubaren Rahmen lösen und später wieder zusammenführen. Dieses Vorgehen hat verschiedene Vorteile:

- Die einzelnen Makros werden übersichtlicher
- Die Makros können einzeln ausgetestet werden
- Einzelne Makros können unter Umständen mehrfach verwendet werden

Dies soll an einem einfachen Beispiel verdeutlicht werden. Mit folgender Aufgabe sollen zwei Blöcke von Daten in einem Tabellenblatt gefunden werden, deren Länge und Breite nicht bekannt sind. Die Blöcke sind nebeneinander angeordnet. Der erste Block beginnt in A1. Die Blöcke sollen durch das Makro markiert und dann in ein anderes Tabellenblatt kopiert werden.

```
Sub UPDemo()
   nSpalte = 1                        ' letzte Spalte suchen--
   Do While Not IsEmpty(Cells(1, nSpalte))
      nSpalte = nSpalte + 1
   Loop
   nSpalte = nSpalte - 1
   nZeile = 1                         ' letzte Zeile suchen---
   Do While Not IsEmpty(Cells(nZeile, 1))
      nZeile = nZeile + 1
   Loop
   nZeile = nZeile - 1
   ' Tabellenbereich markieren und in Tabelle2 übertragen ---
   Range(Cells(1, 1), Cells(nZeile, nSpalte)).Select
   Selection.Copy
   Worksheets("Tabelle2").Activate
   Cells(1, 1).Select
   Selection.Insert
   Worksheets("Tabelle1").Activate
   nAnfang2 = nSpalte + 1   ' Anfang des zweiten Bereichs
suchen
   Do While IsEmpty(Cells(1, nAnfang2))
      nAnfang2 = nAnfang2 + 1
   Loop
   nSpalte = nAnfang2                 ' letzte Spalte suchen--
   Do While Not IsEmpty(Cells(1, nSpalte))
      nSpalte = nSpalte + 1
   Loop
   nSpalte = nSpalte - 1
   nZeile = 1                         ' letzte Zeile suchen---
   Do While Not IsEmpty(Cells(nZeile, nAnfang2))
      nZeile = nZeile + 1
   Loop
   nZeile = nZeile - 1
   ' Bereich markieren und in Tabelle2 übertragen -----------
   Range(Cells(1, nAnfang2), Cells(nZeile, nSpalte)).Select
   Selection.Copy
   Worksheets("Tabelle2").Activate
   Selection.End(xlDown).Select
   ActiveCell.Offset(1, 0).Select
   Selection.Insert
   Worksheets("Tabelle1").Activate
End Sub
```

Wenn Sie das Makro betrachten, stellen Sie zum einen fest, daß sich bestimmte Befehlssequenzen wiederholen. Zum anderen sehen Sie, daß dieses Makro aus verschiedenen Teilaufgaben besteht. Die wie folgt aufzulisten sind.

1. Ende des ersten Blockes suchen
2. Block zu Tabelle2 übertragen
3. Anfang des zweiten Blockes suchen
4. Ende des zweiten Blockes suchen
5. Block zu Tabelle2 übertragen

Wir haben also drei Teilaufgaben die sich zum Teil wiederholen und vielleicht sogar in anderen Makros noch genutzt werden können. Hier bietet es sich an diese als Unterprogramme zu realisieren. Die erste Teilaufgabe ist das Suchen eines Blockes und die zweite das übertragen eines Blockes in eine andere Tabelle.

Informationen können zwischen einem Unterprogramm und dem aufrufenden Programm über die Parameter-Liste des Unterprogramms ausgetauscht werden. Die Parameterliste muß entsprechend der Aufgabenstellung für die Teilaufgabe immer neu gestaltet werden. Für das Suchen eines Blockendes könnte das wie folgt aussehen:

```
Sub BlockEndeSuchen(nSZeile, nSSpalte, nEZeile, nESpalte)
   nSpalte = nSSpalte                ' letzte Spalte suchen--
   Do While Not IsEmpty(Cells(nSZeile, nSpalte))
      nSpalte = nSpalte + 1
   Loop
   nESpalte = nSpalte - 1
   nZeile = nSZeile                  ' letzte Zeile suchen---
   Do While Not IsEmpty(Cells(nZeile, nSSpalte))
      nZeile = nZeile + 1
   Loop
   nEZeile = nZeile - 1
End Sub
```

Als Parameter werden bei diesem Unterprogramm die Zeilen- und Spaltennummer der oberen linken und der unteren rechten Ecke des Bereichs ausgetauscht. Die ersten beiden Parameter sind Eingabeparameter. Sie werden durch das Unterprogramm verwendet aber nicht verändert. Die letzten beiden Parameter sind Ausgabeparameter Sie werden durch das Unterprogramm verändert. Der Algorithmus (Lösungsweg) ist der gleiche wie in dem großen Makro.

Das nächste Unterprogramm dient zur Realisierung des Kopierens.

```
Sub BlockKopieren(Bereich As Object, Quelle As Object, _
   Ziel As Object, Optional nZeile = 0)
   Bereich.Select
   Selection.Copy
   Ziel.Activate
   If nZeile = 0 Then
      Selection.End(xlDown).Select
      ActiveCell.Offset(1, 0).Select
   Else
      Cells(nZeile, 1).Select
   End If
   Selection.Insert
   Quelle.Activate
End Sub
```

In der Parameterliste dieses Unterprogramms sehen Sie zwei Besonderheiten. Erstens zur Übergabe von Objekte werden die Parameter für den Bereich und Tabellenblätter als Objekte definiert und zweitens der letzte Parameter ist als optional definiert, er kann, muß aber nicht bei einem späteren Aufruf angegeben werden. Wenn er nicht angegeben wird hat er automatisch den Wert Null. Dies hat zur Folge, daß sich das Kopierprogramm selbst die Zeile sucht in die kopiert wird.

Ein letztes Unterprogramm ist noch nötig, um für den zweiten Block die Anfangsspalte zu finden. Dies könnte wie folgt aussehen.

```
Sub BlockNächsterAnfang(nZeile, nSpalte)
   nSpalte = nSpalte + 1         ' Anfang des Bereichs suchen
   Do While IsEmpty(Cells(nZeile, nSpalte))
      nSpalte = nSpalte + 1
   Loop
End Sub
```

Die Realisierung entspricht der in dem ursprünglichen Programm. Das folgende Makro zeigt nun die Verwendung der verschiedenen Unterprogramme:

```
Sub UPDemo2()
   ' Ersten Block suchen und übertragen
   nSZeile = 1
   nSSpalte = 1
   Call BlockEndeSuchen(nSZeile, nSSpalte, nEZeile, nESpalte)
   Call BlockKopieren(Range(Cells(nSZeile, nSSpalte), _
      Cells(nEZeile, nESpalte)), ActiveSheet, _
      Worksheets("Tabelle2"), 1)
   ' Zweiten Block suchen und übertragen
   nSSpalte = nESpalte
   Call BlockNächsterAnfang(nSZeile, nSSpalte)
   Call BlockEndeSuchen(nSZeile, nSSpalte, nEZeile, nESpalte)
   Call BlockKopieren(Range(Cells(nSZeile, nSSpalte), _
      Cells(nEZeile, nESpalte)), ActiveSheet, _
      Worksheets("Tabelle2"))
End Sub
```

Die Unterprogramme werden über das Schlüsselwort Call aufgerufen. Dies ist immer dann zu verwenden, wenn die Unterprogramme eine Parameterliste verwenden, was in der Regel der Fall ist. Wenn keine Parameterliste vorhanden ist, kann das Schlüsselwort Call auch weggelassen werden. Beachten Sie die Aufrufe des Unterprogramms BlockKopieren, einmal wurde der optionale 4 Parameter angegeben, einmal wurde er weggelassen.

Wenn Sie Unterprogramme aus anderen Modulen aufrufen möchten, dann müssen Sie beim Aufruf den Namen des Moduls vor dem Makronamen angeben, z. B.

```
Call Modul2.BlockNächsterAnfang(nSZeile, nSSpalte)
```

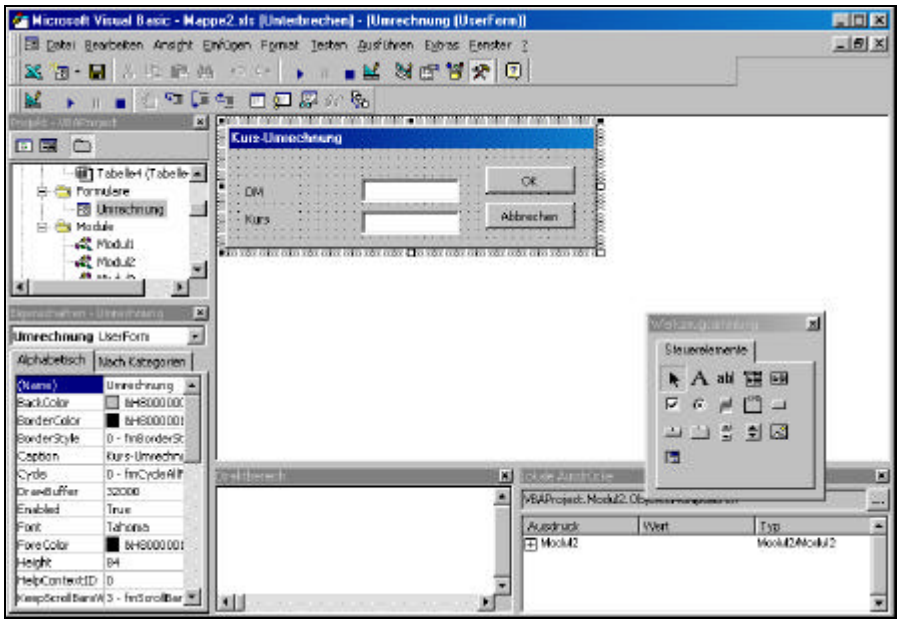
Die Namen der Module können Sie mit Hilfe des Eigenschaftenfensters auch ändern.

4.8 Einbinden eigener Dialogboxen

Mit Visual-Basic für Applikationen ist es Ihnen möglich, neben den bereits vorgestellten Dialogen Eingabe-Dialog (InputBox) und Meldungs-Dialog (MsgBox) auch eigene Dialogboxen zu erstellen und in Ihre Makros einzubinden.

4.8.1 Entwerfen eigener Dialoge

Zum Erstellen eines Dialoges fügen Sie zunächst eine Userform in Ihr Projekt ein. Wählen Sie dazu den Menüpunkt „Einfügen|Userform“. Eine leere Userform wird angezeigt:



In dieser Userform können Sie nun die Dialogbox entwerfen. Sie können die Userform auch als Rahmen für eine Excel-Anwendung verwenden. Dazu stehen Ihnen alle von Windows her bekannten Dialog-Elemente zur Verfügung. Mit Hilfe der Maus und der Symbolleiste Werkzeugsammlung können Sie neue Dialog-Elemente einfügen, verschieben und in der Größe verändern. Die Texte und Eigenschaften der Dialogelemente können Sie über das Eigenschaftenfenster verändern.

Die Elemente der Symbolleiste Werkzeugsammlung haben folgende Bedeutung:

Mit diesem Symbol können Sie nach Auswahl eines der anderen Symbole den Markierungsmodus wieder einschalten. Wenn Sie mehrere Dialogelemente des gleichen Typs zeichnen möchten, können Sie einen Doppelklick auf dem entsprechenden Symbol machen. Das Symbol rastet ein, das Ausschalten dieser Funktion kann dann ebenfalls mit diesem Symbol erfolgen.

Mit diesem Symbol fügen Sie ein Textfeld (Bezeichnungsfeld) in die Dialogbox ein (txt). Über das Eigenschaftenfenster können Sie den Namen und weitere Eigenschaften des Dialogelementes (Controls) festlegen. Um eine Identifizierung der Dialogelement zu erleichtern, sollte Sie direkt den Namen ändern. Empfehlenswert ist es, vor den eigentlichen Namen ein Kürzel zu setzen, um zu erkennen um welche Art von Dialogelemente es sich handelt. Bei einem Textfeld könnte dieses Kürzel z. B. txt sein. Statt einfach nur Betrag nennen Sie das Feld txtBetrag. Weitere Vorschläge für die Kürzel sind bei den entsprechenden Elemente in Klammern angegeben.

Mit diesem Symbol fügen Sie ein Editierfeld (Bearbeitungsfeld) in die Dialogbox ein (edt).

Mit diesem Symbol fügen Sie ein aufklappbares Listenfeld (Dropdown) in die Dialogbox ein (clst).

Mit diesem Symbol fügen Sie ein normales Listenfeld in die Dialogbox ein (lst).

Mit diesem Symbol fügen Sie ein Markierungsfeld (Kontrollkästchen) in die Dialogbox ein (chb).

Mit diesem Symbol fügen Sie ein Auswahlfeld (Optionsfeld) in die Dialogbox ein (rdb).

Mit diesem Symbol fügen Sie einen Toggle-Button in die Dialogbox ein. Der Toggle-Button sieht aus wie ein Aktionsschalter bleibt nach dem Klicken aber gedrückt (tbtn).

Mit diesem Symbol fügen Sie ein Gruppenfeld in die Dialogbox ein (grb).

Mit diesem Symbol fügen Sie einen Aktionsschalter (Button) in die Dialogbox ein (btn).

Mit diesem Symbol fügen Sie ein Tab-Control (Register-Karten) in eine Dialogbox ein (tbc).

Mit diesem Symbol fügen Sie ein Multi-Page-Control in eine Dialogbox ein (page).

Mit diesem Symbol fügen Sie eine Bildlaufleiste (Scroller) in die Dialogbox ein (scr).

Mit diesem Symbol fügen Sie einen Mini-Scroller (Drehfeld) in die Dialogbox ein (spin).

Mit diesem Symbol fügen Sie ein Bild in die Dialogbox ein (pic). Mit dem Eigenschaftenfenster können Sie eine beliebige Bilddatei diesem Element zuordnen (pic).

Mit diesem Symbol fügen Sie ein Referenz-Editierfeld in die Dialogbox ein. Ein solches Feld ermöglicht die Eingabe von Bezüge mit Hilfe der Maus. Sie kennen diese Felder auch aus entsprechenden Dialogboxen von Excel (ref).

Wenn Sie hier z. B. hier für die Gültigkeit die Option Zahl auswählen, dann läßt Excel als Eingabe für das entsprechende Feld nur Zahlen zu. Sie brauchen also bei der Bearbeitung der Ergebnisse die Gültigkeit der Eingaben des Benutzers in dieser Hinsicht nicht mehr zu überprüfen. Für die verschiedenen Dialogelemente stehen Ihnen unterschiedliche Einstell-Möglichkeiten zur Verfügung.

Testen Zum Testen der Dialogbox drücken Sie die Taste <F5>. Damit können Sie Aussehen und Verhalten zur Laufzeit prüfen.

Tab-Reihenfolge Zum Festlegen der Tab-Reihenfolge wählen Sie den Menü „Ansicht|Aktivier-reihenfolge“. Eine Dialogbox wird angezeigt, ordnen Sie die Elemente mit der Dailogbox in der gewünschten Reihenfolge an.

4.8.2 Verarbeitung der Eingaben

Die Verarbeitung der Eingabe in einer Dialogbox kann auf zwei Arten erfolgen:

1. Auswertung in dem aufrufenden Makro
   Bei dieser Möglichkeit lassen Sie die Dialogbox anzeigen und nachdem der Benutzer die Dialogbox geschlossen hat, werden in dem aufrufenden Makro die Eingaben ausgewertet.
2. Auswertung durch Ereignis-Makros
   Sie lassen die Dialogbox anzeigen und bei der Veränderung oder Anklicken eines Dialogelementes werden durch Excel automatisch Ereignis-Makros aufgerufen, die eine Auswertung der Eingaben vornehmen.

Beide Möglichkeiten können kombiniert werden. Das nachfolgende Beispiel zeigt Ihnen ein entsprechendes Beispiel:

Der Aufruf und die Anzeige der Dialogbox kann durch ein Makro wie folgt vorgenommen werden:

```
Sub DialogTesten()
   With KursUmrechnung
      .edtBetrag = "100"
      .edtKurs = "1,85"
      .Show
       If .btnOK.Tag = "OK" Then
         If IsNumeric(.edtBetrag.Text) And _
            IsNumeric(.edtKurs.Text) Then
            Kurs = CDbl(.edtKurs.Text)
            If Kurs <> 0 Then
               DMBetrag = CDbl(.edtBetrag.Text)
               Ergebnis = DMBetrag / Kurs
            Else
               Ergebnis = "Nicht definiert"
            End If
         Else
            Ergebnis = "Ungültige Eingabe"
         End If
         MsgBox ("Der Benutzer hat auf OK geklickt." & _
            Chr(10) & "Das Ergebnis lautet: " & Ergebnis)
      Else
         MsgBox ("Der Benutzer hat auf Abbrechen geklickt")
      End If
   End With
End Sub
```

Die Userform (Dialogbox) wird mit Ihrem Namen angesprochen ebenso können Sie deren Elemente mit dem jeweiligen Namen ansprechen. So heißt z. B. die Dialogbox KursUmrechnung und das Eingabefeld für den Betrag z. B. btnBetrag. Ein Eingabefeld hat als wichtigste Eigenschaft ein Feld Namens Text. Über diese Eigenschaft können Sie den enthaltenen Text vorbesetzen oder auslesen. Dies geschieht dann z. B. mit einer entsprechenden Zusweisung.

```
Kursumrechnung.btnBetrag.Text = "100"
```

Damit Sie nicht jedesmal den relativ langen Namen der Userform, Kursumrechnung, schreiben müssen, benutzen Sie wie im Beispiel gezeigt, die With-Anweisung.

Die Anzeige des Dialoges erfolgt mit der Methode Show. Sie zeigt die Userform am Bildschirm an. Der Benutzer kann nun in dem Dialog seine Eingaben vornehmen. Zum Schließen der Dialogbox klickt der Benutzer wahrscheinlich auf einen der Aktionsschalter OK oder Abbrechen. Diese sind jedoch noch ohne Funktion. Sie müssen für die Aktionsschalter erst entsprechende Funktionalität hinterlegen. Dazu wechseln Sie zur Userform und machen einen Doppelklick zunächst auf dem Schalter Abbrechen.

Ein leerer Unterprogrammrumpf für eine Ereignis-Prozedur mit dem btnCancel_Click wird angezeigt. Diese Prozedur wird automatisch aufgerufen, wenn der Benutzer zur Laufzeit auf den Aktionsschalter klickt. Zum Schließen des Dialoges rufen Sie die entsprechende Funktion auf.

```
Private Sub btnCancel_Click()
   Me.Hide
End Sub
```

Diese Ereignis-Prozedur ist eine Methode des Objektes KursUmrechnung vom Typ Userform. Das Objekt selbst können Sie innerhalb der Methode mit dem Schlüsselwort Me ansprechen.

Eine entsprechende Prozedur ist nun auch für den Aktionsschalter OK zu erstellen. Machen Sie wieder einen Doppelklick und füllen Sie die Prozedur entsprechend dem nachfolgenden Beispiel:

```
Private Sub btnOK_Click()
   btnOK.Tag = "OK"
   Me.Hide
End Sub
```

Mit der Eigenschaft Tag des Buttons haben Sie die Möglichkeit auch für einen Aktionsschalter einen Wert zu hinterlegen. Wir benötigen dies, um in dem rufenden Makro feststellen zu können, ob dieser Aktionsschalter geklickt wurde. Beachten Sie hierzu auch die If-Anweisung nach der Aufruf der Methode Show in dem aufrufenden Makro. Über dieses „Tag“ können Sie feststellen, welcher der beiden Aktionsschalter OK oder Abrechen geklickt wurde. Um allerdings sicherzustellen, daß dieses „Tag“ in allen anderen Fällen keinen Wert enthält ist es beim Öffnen der Dialogbox zu initialisieren. Sie könnten das ähnlich wie bei den Edit-Feldern vor dem Aufruf einer Methode machen. Eine Alternative bietet aber noch die Methode Initialize, die automatisch vor jeder Anzeige einer Userform aufgerufen wird. Sie können entsprechende Initialisierungen auch hier vornehmen. Machen Sie dazu einen Doppelklick auf der Userform selbst. Eine Rumpf einer Methode mit dem Namen Userform_Click wird generiert. Ändern Sie den Namen Click in den Namen Initialize und füllen Sie die Methode entsprechend dem nachfolgenden Beispiel:

```
Private Sub UserForm_Initialize()
   btnOK.Tag = ""
End Sub
```

Nach dem Schließen hat btnOK.Tag den Wert „OK“ oder nicht, je nachdem ob der Aktionsschalter geklickt oder nicht geklickt wurde. Die Funktion der nachfolgenden Anweisungen zur Auswertung der Dialogbox-Inhalte wurden schon erläutert.

Hinweis

Neben der Möglichkeit die Steuerelemente in den Userformen zu verwenden, haben Sie auch die Möglichkeit die Steuerelemente direkt in Tabellen einzusetzen und über die Ereignis-Prozeduren die Tabelle zu verändern oder andere Makros aufzurufen.

4.8.3 Verwendung von integrierten Dialogen

Neben der Möglichkeit eigene Dialogbox zu entwerfen und diese in Makros zu verwenden, können Sie auch alle Dialogboxen von Excel in Ihren Makros aufrufen.

Diese Dialogboxen sind in einer Liste mit dem Namen Dialogs hinterlegt, die wiederum dem Application-Objekt zugeordnet ist. Ausgewählt wird der gewünschte Dialog mit einem Index. Die Parameterliste für die Show-Methode unterscheidet sich bei den unterschiedlichen Dialogen. Das nachfolgende Beispiel zeigt den Aufruf des Dialoges zum Öffnen von Dateien:

```
Application.Dialogs(xlDialogOpen).Show
```

Die ausgewählte Datei wird automatisch auch geöffnet.

Die Liste der verfügbaren eingebauten Dialoge und der zugehörigen Parameter sind finden Sie im Hilfesystem, wenn Sie im Visual-Basic-Editor den Text „Dialogs“ eingeben und die Taste <F1> drücken. Klicken Sie im Hilfetext auf den Verweis Dialogs und anschließend auf den Verweis Argumentlisten integrierter Dialogfelder.

4.9 Definition von Makro-Funktionen

Funktionen

Bisher wurde Ihnen nur die Realisierung von Befehls-Makros (Sub) vorgestellt. Die Sub-Makros führen nur die enthaltenen Anweisungen aus, geben aber keine Werte zurück, es sei denn über die Parameterliste. Ein Funktions-Makro hingegen gibt immer mindestens einen Wert zurück, nämlich über den Funktionsnamen selbst, wie Sie es bereits von den eingebauten Excel-Funktionen her kennen gelernt haben.

Wie Sie eigene Funktionen realisieren können, wurde noch nicht gezeigt. Als Beispiel soll Ihnen zunächst eine Funktion mit dem Namen MwSt gezeigt werden, die als Funktionswert für die als Parameter übergebene Zahl die Mehrwertsteuer zurückliefert. Das nachfolgende Beispiel zeigt die Realisierung dieses Funktions-Makros:

```
Function MWSteuer(Betrag)
   MWSteuer = Betrag * 0.15
End Function
```

In der ersten Zeile wird der Name der Funktion und die Anzahl und Namen der Parameter definiert. In der Zeile zwei wird der Funktionswert berechnet und der Funktion zugewiesen. Durch das Schlüsselwort in der Zeile wird das Ende der Funktion festgelegt. Beachten Sie das anstelle des Schlüsselwortes Sub, das bei den Befehls-Makros Anwendung findet, bei den Funktions-Makros das Schlüsselwort Funktion verwendet wird.

Sie können eine so definierte Funktion genauso verwenden, wie die eingebauten Excel-Funktionen. Einzige Voraussetzung ist, daß die Arbeitsmappe in der die Funktion definiert ist, geladen ist. Die Funktion wird im Funktions-Assistenten unter der Kategorie Benutzerdefiniert angezeigt.

Beispiel

Abschließend noch ein Beispiel für ein komplexeres Funktions-Makro zur Berechnung eines Rabattes:

```
Function Rabatt(Betrag)
   If Betrag >= 10000 Then
      Ergebnis = 0.7 * Betrag
   ElseIf Betrag >= 1000 Then
      Ergebnis = 0.8 * Betrag
   ElseIf Betrag >= 100 Then
      Ergebnis = 0.9 * Betrag
   Else
      Ergebnis = Betrag
   End If
   Rabatt = Ergebnis
End Function
```

An diesem Beispiel sehen Sie, daß innerhalb solcher Funktionsmakros nicht nur lineare Berechnungen sondern auch die Verwendung von Kontrollstrukturen möglich ist.

Mit Hilfe dieser Möglichkeit können Sie sich eigene Funktions-Bibliotheken für Excel aufbauen. Häufig verwendete Formeln können als Funktions-Makros realisiert werden.

5 Fortgeschritte Programmiertechniken

In den nachfolgenden Abschnitten sollen kurz fortgeschrittene Programmiertechniken vorgestellt werden. Zum einen soll Ihnen dies die Einarbeitung in solchen Techniken erleichtern und zum anderen soll es noch einmal einen Ausblick auf die vielfältigen Möglichkeiten der Makro-Programmierung mit Excel geben.

5.1 Ereignis-Makros

Im Zusammenhang mit den Dialogen haben wir bereits Makros kennengelernt, die nicht direkt aufgerufen werden müssen, sondern bei einem bestimmten Ereignis z. B. beim Klicken auf ein Dialogelement von Excel automatisch aufgerufen werden.

Auto-Makros

Eine besondere Kategorie der Ereignis-Makros sind die Auto-Makros. Sie werden ebenfalls bei bestimmten Ereignissen automatisch aufgerufen. Hier sind die Makros Auto_Open, Auto_Close, Auto_Activate und Auto_Deactivate zu nennen, die in den entsprechenden Situationen, also z. B. beim Öffnen oder Schließen einer Datei aufgeführt werden, wenn Sie entsprechend definiert sind.

Die Namen der Auto-Makros sind fest vorgeschrieben und müssen so verwendet werden. Wenn Sie ein Makro mit dem Namen Auto_Open in einem Modul definieren, wird dieses Makro dann automatisch beim Öffnen der Datei aufgerufen. Es darf in allen Modulen nur genau ein Makro mit diesem Namen ansonsten erhalten Sie beim Öffnen der Arbeitsmappe eine Fehlermeldung. Beachten Sie dazu das nachfolgende Beispiel:

```
Sub Auto_Open()
   MsgBox ("Hello World")
End Sub
```

Ähnliches können Sie auch mit einem Tabellenblatt machen. Allerdings müssen Sie hierzu dem Tabellenblatt eine entsprechende Methode hinzufügen. Machen Sie dazu im Projekt-Explorer einen Doppelklick auf dem Namen eines Tabellenblattes und geben Sie ein entsprechendes Makro ein, auch hier sind die Namen vorgeschrieben:

```
Private Sub Worksheet_Activate()
   MsgBox ("Tabelle aktiviert  ")
End Sub
```

Das Beispiel zeigt ein Makro, das automatisch ausgeführt, wenn das Tabellenblatt aktiviert wird. Weitere Ereignis-Makros können für Tabellenblätter definiert werden:

- WorkSheet_BeforeDoubleClick
- WorkSheet_RightDoubleClick
- WorkSheet_Calculate
- WorkSheet_Change
- WorkSheet_SelectionChange

5.2 Tastatur- und Zeitereignisse

Eine besondere Form von Ereignissen sind die Tastatur- und Zeitereignisse, Ihnen können mit den Methoden OnKey und OnTime Behandlungs-Routinen zugewiesen werden. Damit besteht die Möglichkeit ein Makro mit einer bestimmten Taste oder Tastenkombination oder einem Zeitereignis zu verbinden. Das nachfolgende Beispiel zeigt einige der Möglichkeiten:

```
Sub EreignisProzedurenInstallieren()
    ' Tastatur Ereignisse verarbeiten
    Application.OnKey "{F12}", "Modul2.HalloName"
    Application.OnKey "^{F12}", "Modul2.HelloWorld"
    ' Zeitereignisse verarbeiten
    Application.OnTime Now + TimeValue("00:00:15"), _
       "Modul2.HalloName"
End Sub
```

Mit dem erstem Beispiel wird der Funktionstaste <F12> das Makro HalloName aus dem Modul2 zugeordnet. Wenn Sie nach Ausführen der Prozedur EreignisProzedurInstallieren die Taste <F12> drücken, wird das entsprechende Makro automatisch aufgerufen. Sie können diese Methode wiederholt für unterschiedliche Tasten aufrufen. Das zweite Beispiel zeigt die Zuordnung des Makros HelloWorld zur Tastenkombination <Strg>+<F12>. Dies ist durch das vorangstellte „^“ kenntlich gemacht. Weitere Kennzeichen für Umschalt-Tasten sind das „+“ für die <Umsch>-Taste sowie das „%“-Zeichen für die <Alt>-Taste.

Wenn Sie als Makro-Namen “” angeben, wird die Definition eines Makros für die entsprechende Taste wieder rückgängig gemacht. Die Taste erhält Ihre ursprüngliche Bedeutung zurück.

Mit dem zweiten Beispiel wird nach 15 Sekunden ein Zeitereignis erzeugt, das automatisch das Makro HalloName aufruft. Sie könnten auch den Aufruf der Funktion Now weglassen und direkt eine konkrete Zeit angeben. Wenn Sie zyklische Zeitereignisse erzeugen wollen, müssen Sie dafür Sorge tragen, daß nach Bearbeitung des ersten Zeitereignisse die OnTime-Methode erneut aufgerufen wird. Dies geschieht am besten in dem automatisch aufgerufenen Makro als letzte Anweisung.

Zum Installieren solcher Ereignis-Makros bieten sich die Auto-Makros der Arbeitsmappen an.

5.3 Funktionen aus anderen Programmen

Mit Visual-Basic für Applikationen ist auch möglich Funktionen aus anderen Programmen und insbesondere auch aus der Windows-API (Application Programmer Interface) aufzurufen. Damit stehen Ihnen alle Möglichkeiten zur Verfügung wie Sie auch Programmierer anderer Entwicklungssysteme haben. Dazu ist lediglich eine Deklaration der entsprechenden Funktionen in einem Ihrer Module notwendig. Nachfolgend werden Ihnen einige Beispiele gezeigt:

```
Declare Sub MessageBeep Lib "User32.dll" (ByVal n As Integer)
```

Mit dieser Zeile wird die Windows-API-Funktion MessageBeep für Excel nutzbar gemacht. Die Funktion befindet sich in der Datei user32.dll (Lib-Anweisung) und einen Integer-Parameter.

```
Declare Function GetPrivateProfileString Lib "kernel32.dll" _
    Alias "GetPrivateProfileStringA" _
    (ByVal lpSection As String, _
    ByVal lpSetting As String, _
    ByVal lpDefault As String, _
    ByVal lpReturnedString As String, _
    ByVal nSize As Long, _
    ByVal lpFileName As String) As Long
```

Dieses Beispiel zeigt wie die Windows-API-Funktion GetPrivateProfileString für Excel nutzbar gemacht werden kann. Diese Funktion ermöglicht das Lesen von Einträge aus Windows-INI-Dateien. Die Funktion befindet sich in der kernel32.dll und hat dort intern den Namen GetPrivateProfileStringA. Die Parameter-Liste umfaßt mehrere Parameter und hat als Rückgabewert einen Wert vom Typ Long. Die Bedeutung der Parameter: Sektionsname, Schlüsselname, Default-Wert, Rückgabewert (Zeichenkette hinter dem Gleichheitszeichen, ist der Eintrag nicht vorhanden, dann wird de Default-Wert zurückgegeben), maximale Länge für die zurückzugebende Zeichenkette, Name der INI-Datei.

Das nachfolgende Beispiel zeigt die Nutzung der so eingebundenen Funktionen:

```
Sub Test()
   Dim Name As String * 20
   MessageBeep (1)
   Call GetPrivateProfileString("Peters", "Kind", "", Name,_
      20, "win.ini")
   MsgBox (Name)
End Sub
```

Mit der ersten Zeile wird eine Zeichenkette für den Rückgabewert in der Funktion GetPrivateProfileString definiert. Mit der zweiten Zeile wird die Funktion MessageBeep aufgerufen. Der Parameter gibt die Art des Tons an. Die dritte und vierte Zeile zeigen die Verwendung der Funktion GetPrivateProfileString. Der Aufruf liest aus der Datei win.ini aus der Sektion [Peters] den Eintrag Kind. Nehmen wir an folgendes wäre in der Datei win.ini enthalten:

```
[Peters]
Kind=Willi
```

Dann würde durch die letzte Anweisung eine Messagebox mit dem Namen Willi ausgegeben. Auf die gleiche Art lassen sich auch Funktionen aus selbst erstellten Dll oder Exe-Files zugänglich machen.

5.4 Fehlerbehandlung

Wenn zur Laufzeit des Programms unzulässige Berechnungen, Zugriffe auf nicht vorhandene Objekte oder Bereichsüberschreitungen auftreten, dann wird die Abarbeitung des Makros mit einer Laufzeit-Fehlermeldung abgebrochen.

Die Ausgabe dieser Laufzeit-Fehlermeldungen können Sie durch eine entsprechende eigene Fehlerbehandlung unterbinden.

Dazu gibt zwei Varianten. Die erste setzt voraus, daß Sie innerhalb eines jedes Unterprogramms für das Sie eine eigene Fehlerbehandlung benötigen, diese entsprechend deklarieren.

```
Sub Fehlerbehandlung()
   On Error GoTo Fehler
   ActiveCell.Offset(-1, -1).Select
   Exit Sub
Fehler:
   MsgBox ("Versetzen außerhalb der Tabelle nicht möglich")
End Sub
```

Das Beispiel zeigt die Anweisung ActiveCell.Offset(-1,-1).Select. Diese führt zu einer Laufzeit-Fehlermeldung immer dann, wenn die Markierung sich in der ersten Spalte oder Zeile befindet. In diesem Fall führt die Offset-Methode zu keinem gültigen Objekt. Durch die On Error-Konstruktion wird nun erreicht, das erstens die Laufzeit-Fehlermeldung nicht mehr ausgegeben wird und zum anderen falls ein Fehler auftritt, automatisch zu der Sprungmarke Fehler: verzweigt wird. Die dort stehende Anweisung(en) werden in diesem Fall ausgeführt. Um eine Ausführung dieser Anweisungen für den Nichtfehlerfall zu verhindern ist die Anweisung Exit Sub vor der Sprungmarke notwendig. Sie sorgt für ein Ende des Makros in der entsprechenden Zeile.

Im Sinne der strukturierten Programmierung ist diese Konstruktion nicht empfehlenswert verwenden Sie besser die folgende Anweisung.

```
Sub Fehlerbehandlung2()
   On Error Resume Next
   ActiveCell.Offset(-1, -1).Select
   If Err.Number <> 0 Then
      MsgBox ("Versetzen außerhalb der Tabelle nicht
möglich")
   End If
End Sub
```

Mit der Anweisung in der ersten Zeile erreichen sie jetzt nur, daß die Laufzeit-Fehlermeldung nicht mehr ausgegeben wird und mit der Ausführung der Zeile hinter der den Fehler verursachenden Zeile fortgefahren wird. Die Behandlung von möglichen Fehlern wird durch Überprüfung des Error-Status der Anwendung erreicht. Dazu fragen Sie die Eigenschaft Number des Err-Objektes ab. Für den Fall, daß eine Anweisung erfolgreich ausgeführt werden konnte, ist dieser Wert 0. Wenn der Wert ungleich 0 dann hat die letzte ausgeführte Anweisung einen Fehler verursacht.

Fehlernummer

Bei einigen Anweisungen ist sogar eine differenzierte Auswertung der Fehlerursache möglich, da die Werte der Fehlernummern Auskunft über die Ursache des Fehlers geben können. Eine Liste mit den mögliche Fehlernummern finden Sie im Index der Visual-Basic-Hilfe unter der Überschrift „Auffangbare Fehler“.

5.5 Ein/Ausblenden von Menüs/Symbolleisten

Wenn Sie eine besondere Arbeitsmappe für einen bestimmten Zweck präparieren, möchten Sie in diesem Moment vielleicht auch eine besondere Kombination von Menü- und Symbolleisten-Elementen haben. Sie können dies durch entsprechende Modifikationen der CommandBars-Liste des Applikations-Objektes erreichen. Nachfolgend einige Beispiele:

```
Application.CommandBars("Chart").Visible = True
Application.CommandBars("Symbolleiste 1").Controls.Add _
   Type:=msoControlButton, Id:=2949, Before:=2
Set NeuesElement = Application.CommandBars(_
   "Worksheet Menu Bar").Controls.Add(Type:= _
    msoControlButton, Id:=2949, Before:=2)
NeuesElement.OnAction = "Modul2.HelloWorld"
Application.CommandBars("Edit").Controls(17).Delete
```

Die erste Anweisung blendet die Symbolleiste „Diagramm“ ein. Mit der zweiten Anweisung wird der „Symbolleiste 1“ ein benutzerdefiniertes Menüelement hinzugefügt. Mit der dritten Anweisung wird ein solches in Arbeitsblatt-Menüzeile eingefügt. Durch die vierte Anweisung wird dem neuen Steuerelement das Makro „HelloWorld“ zugewiesen. Mit der letzten Anweisung wird der 17. Befehl aus dem Menü „Bearbeiten“ entfernt. Das Zusammenstellen besonderer Menüs für bestimmte Arbeitsmappen kann so schnell zu sehr umfangreichen Makros führen.

Neben der Möglichkeit die Symbolleisten per Makro-Anweisung(en) individuell zusammenzustellen, besteht auch die Möglichkeit eine besondere Datei zu laden, die die Einstellungen gespeichert hat. Dies dürfte in vielen Fällen einfacher sein. Voraussetzung ist die Symbolleisten wurden einmal über die Benutzeroberfläche entsprechend eingestellt, und anschließend wurde die xlb-Datei unter einem anderen Namen gesichert. Dann könnte man mit Hilfe des folgenden Makros die Datei jederzeit wieder laden.

```
Sub LadeSymbole()
   AlteSymbole = Application.UserName & "8.xlb"
   FileCopy AlteSymbole, "save8.xlb"
   Workbooks.Open ("EigeneSymbole.xlb")
End Sub
```

Diese Makro könnte dann z. B. im Auto_Open-Makro der entsprechenden Arbeitsmappe aufgerufen werden. Die ersten beiden Anweisungen dienen zum Sichern der aktuellen Einstellungen des Benutzers diese sollten dann durch ein entsprechendes Laden der gesicherten Datei „save8.xlb“ im Auto_Close-Makro wiederhergestellt werden.