Einführung

in

MS-Word für Windows

- Makroprogrammierung -

Kevelaer, im September 1999

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

Inhaltsverzeichnis

1 Einführung

Dieses Seminar richtet sich an Teilnehmer, die mit MS-Word bereits gut vertraut sind. Das Seminar soll dem Teilnehmer anhand von Beispielen die Verwendung von Makros unter MS-Word vermitteln. Grundlegende Programmiertechniken für Visual-Basic unter MS-Word 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
- Erstellen von eigenen Dialogen und Visual-Basic-Formularen

Überblick

Makros ermöglichen das Erstellen automatisierter Handlungsabläufe innerhalb eines Word-Dokumentes. In Word 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 Word-Funktionen und Prozeduren verfügbar, die auch über die Menüpunkte ausgewählt werden können.

Sprache

Bis zur Version 7.0 von Word konnte die Sprache für die Makros gewählt werden. So konnte man z. B. einen deutschen BASIC-Dialekt (zumindest teilweise), 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 Word-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 Möglichkeiten von Word 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

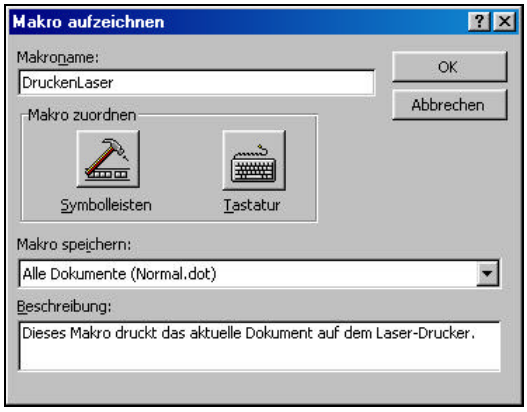
Die einfachste Methode zur Erstellung eines Makros ist die Benutzung des Makrorecorders. Sie müssen sich den Makrorecorder vorstellen wie einen Kassettenrecorder. Zu einem bestimmten Zeitpunkt wird auf Aufnahme geschaltet. Die folgenden Handlungsabläufe werden aufgezeichnet. Wenn alles aufgezeichnet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

2.1 Makros aufzeichnen

Beispiel

Sie haben Zugriff auf zwei verschiedene Drucker, z. B. einen Laser-Drucker und einen Tintenstrahl-Drucker. Sie möchten zwei Makros erstellen, die beim Drucken automatisch die Auswahl des jeweiligen Druckers beinhalten. Wählen Sie den Menüpunkt [Extras | Makro aufzeichnen | Aufzeichnen] oder aktivieren Sie die Visual-Basic-Symbolleiste:

Klicken Sie auf das Symbol mit dem Punkt. Die Dialogbox „Makro aufzeichnen“ wird angezeigt.



In dieser Dialogbox können Sie zunächst den Namen des Makros festlegen. Hierfür gelten folgende Regeln: Die Namen dürfen nur Buchstaben und Ziffern beinhalten (Leerzeichen sind nicht erlaubt). Sie müssen mit einem Buchstaben anfangen und dürfen 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.

Makro zuordnen

Mit der Zuordnung, die Sie an dieser Stelle festlegen können aber nicht müssen, können Sie das Makro später aufrufen. Sie sollten die Zuordnung des Makros zu einem Symbol, Menüpunkt oder einer Tastenkombination zu einem späteren Zeitpunkt vornehmen. Anmerkung: An dieser Stelle ist Word etwas ungeschickt programmiert, nach der Zuordnung wird sofort die Aufzeichnung gestartet. Sie kommen nicht zu der Dialogbox „Makro aufzeichnen“ zurück, auch wenn Sie noch nicht die anderen Felder ausgefüllt hatten.

Makro speichern

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

- Alle Dokumente (Normal.dot)
  für Makros, die Ihnen immer zur Verfügung stehen sollen
- Dieses oder ein anderes offenes Dokument
  für Makros, die Sie bei Bedarf nachladen möchten, bzw. die an ein bestimmtes Dokument gebunden sind.

Mit diesen Auswahlmöglichkeiten können Sie festlegen, wo das aufgezeichnete Makro gespeichert werden soll. Wenn Sie die eine andere als die Option „Alle Dokumente“ wählen, wird das Makro mit dem entsprechenden Dokument gespeichert. Das Makro steht Ihnen somit nur zur Verfügung, wenn dieses Dokument auch geöffnet ist.

Wenn Sie jedoch die Option „Alle Dokumente“ wählen, dann wird das Makro in der Normal-Vorlage (Normal.dot) gespeichert und steht Ihnen somit jederzeit zur Verfügung. Speichern Sie jedoch nur die Makros in der Normal-Vorlage, die Sie auch wirklich immer benötigen. Zu viele Makros in der Normal-Vorlage verlängern unnötig die Ladezeiten von Word.

Normal.dot

Die Datei Normal.dot ist (dot bedeutet Document Template, Dokument-vorlage) wird automatisch mit jeder neuen Datei geladen, somit stehen hier gespeicherte Makros auch in jedem Dokument zur Verfügung. Sie können die Datei auch direkt öffnen. Sie finden die im Vorlagen-Verzeichnis von Microsoft-Office (..\Microsoft Office\Vorlagen).

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:

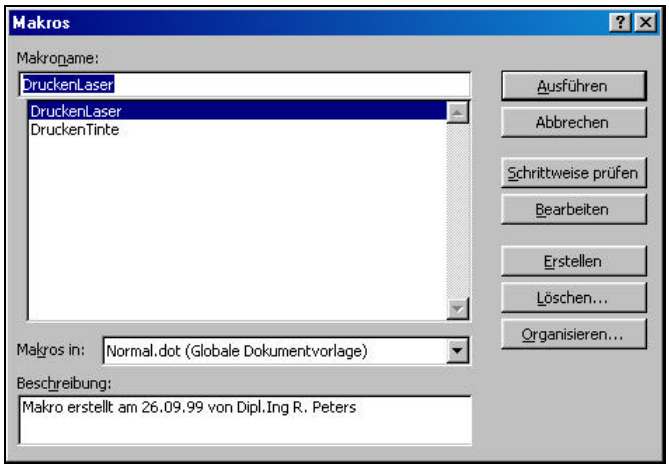
1. Menü Datei Drucken
2. Im Komninationsfeld „Drucker Name“ Auswahl des gewünschten Druckers (Tintenstrahl-Drucker)
3. Klicken Sie auf den Aktionsschalter Optionen in der Dialogbox Drucken
4. Markieren Sie das Feld „Umgekehrte Druckreihenfolge“
5. Klicken Sie auf „OK“ in der Dialogbox Optionen
6. Klicken Sie auf „OK“ in der Dialogbox Drucken

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.

Sie können die laufende Aufzeichnung auch anhalten. Aktionen die Sie danach machen werden nicht mit aufgezeichnet. Klicken Sie erneut auf das Symbol, um die Aufzeichnung fortzusetzen.

2.2 Makros ausführen

Sie können das Makro nun mit Hilfe des Menüpunktes [Extras | Makro] jederzeit wieder ausführen lassen. Alternativ dazu können Sie auch auf das Symbol mit dem Pfeil in der Visual-Basic-Symbolleiste klicken. Die folgende Dialogbox wird angezeigt:



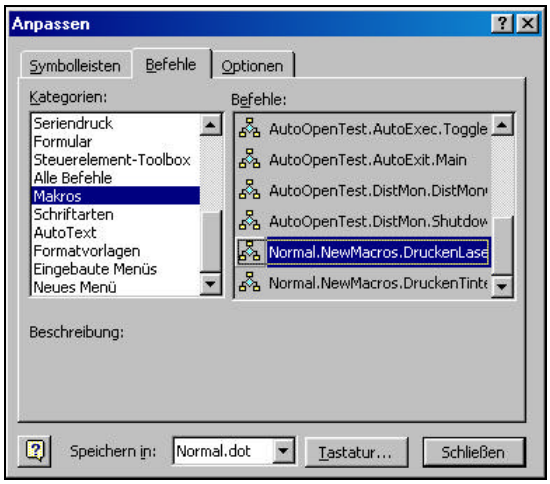
In der Auswahlliste werden die Namen der verfügbaren Makros in dem ausgewählten Dokument bzw. der ausgewählten Vorlage angezeigt. Wenn Sie ein Makro auswählen, wird unterhalb der Auswahlliste zusätzlich angezeigt, wer das Makro wann erstellt hat. Klicken Sie den Aktionsschalter ausführen, um das ausgewählte Makro auszuführen. Alle Aktionen, die Sie zuvor aufgezeichnet haben, werden nun automatisch abgespielt. Das Dokument wird mit den gewählten Einstellungen gedruckt.

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

- Ausführen über einen zusätzlichen Menüpunkt
- Ausführen über ein Symbol in der Symbolleiste
- Ausführen über eine Tastenkombination
- Ausführen über Steuerelement in einem Dokument

Menü/Symbolleiste

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

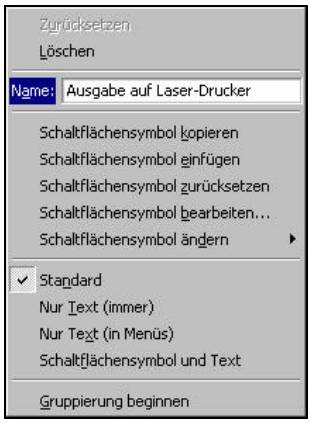


Klicken Sie auf die Registerkarte „Befehle“ und wählen Sie in der linken Auswahlliste den Punkt „Makros“. In der rechten Liste werden dann alle verfügbaren Makro angezeigt.

Menüelement

Einen neuen Menüpunkt fügen Sie ein, in dem Sie mit der Maus auf den Eintrag des gewünschten Makros 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:

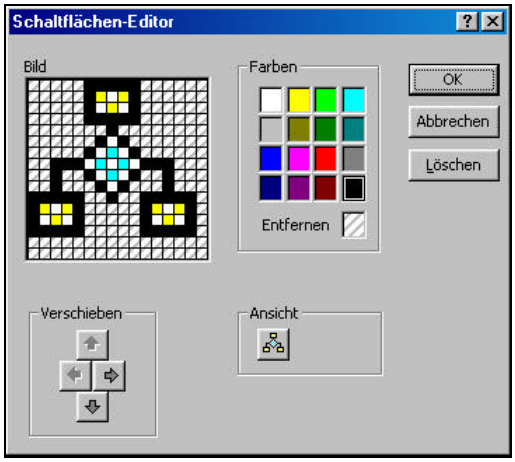


Mit diesem Kontextmenü können Sie z. B. den Text für den Menüpunkt ändern. Mit dem „&“ im Text für den Menüpunkt bestimmen Sie die Zugriffstaste. Der entsprechende Buchstabe wird im fertigen Menü dann unterstrichen dargestellt. Die Menüpunkte für die Schaltflächensymbole betreffen die Symbolleisten, 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.

Symbolleiste

Das Hinzufügen von Symbolen zur 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. Um anstelle von Text (standardmäßig der Makroname) ein Symbol angezeigt zu bekommen, wählen Sie im Kontextmenü den Menüpunkt Standard.

Bei den Schaltflächen 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“.

Speichern

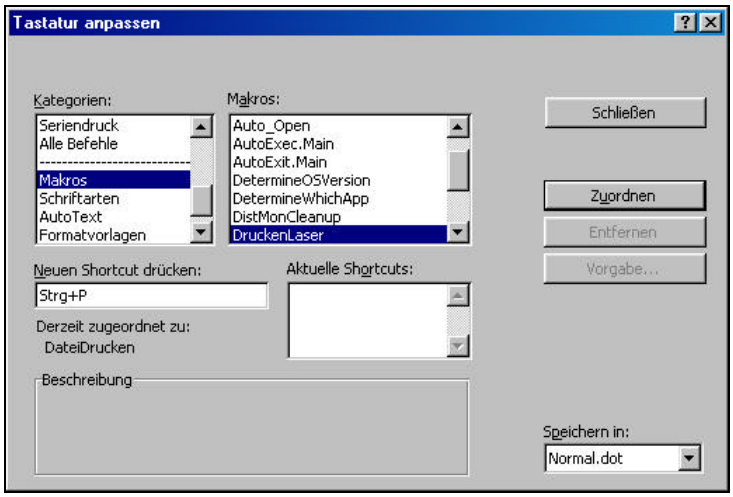
Von Ihnen erstellte oder angepaßte Symbolleisten stehen in allen Dokumenten zur Verfügung wenn Sie diese Änderungen in der Normal-Vorlage (normal.dot) speichern. Sie können aber auch durch Auswahl des jeweiligen Dateinamens erreichen, daß die Änderungen zusammen mit dem aktiven Dokument gespeichert werden. Dann werden die geänderte Symbol- oder Menüleiste mit Ihren Änderungen immer dann angezeigt, wenn das entsprechende Dokument geöffnet bzw. aktiviert wird.

Hinweis

Achten Sie darauf, die gewünschten Änderungen werden nur dann wirksam, wenn Sie die Einstellung für den Speicherort geändert haben, bevor Sie die Änderungen vornehmen.

Tastenkombination

Auch die Tastenkombination werden bei Word über die Dialogbox Anpassen festgelegt. Klicken Sie auf den Aktionsschalter „Tastatur“ im unteren Teil der Dialogbox „Anpassen“. Die folgende Dialogbox wird angezeigt:



In der Dialogbox können Sie sowohl den eingebauten Word-Funktionen als auch Ihren Makros, aber auch anderen selbstdefinierten Elementen wie z. B. Autotexten oder Formatvorlagen Tastenkombinationen zuweisen. Um Makros mit einer Tastenkombination zu verbinden, wählen Sie in der Liste “Kategorien” den Eintrag Makros, dann suchen Sie in der rechten Liste das gewünschte Makro und markieren den Eintrag.

Klicken Sie dann auf das Feld “Neuen Shortcut drücken” und geben Sie die gewünschte Tastenkombination ein. Falls diese Tastenkombination bereits belegt ist, wird unter “derzeit zugeordnet zu” angezeigt, welche Funktion im Moment mit dieser Tastenkombination verbunden ist. Falls Sie die Tastenkombination trotzdem zuordnen ist die alte Verbindung gelöscht. Falls Ihr Makro bereits einer anderen Tastenkombination zugeordnet, wird diese in der Liste “Aktuelle Shortcuts” angezeigt. Sie können einem Makro mehrere Tastenkombinationen zuordnen. Falls Sie eine Tastenkombination löschen möchten, markieren Sie diese zunächst in der Liste “Aktuelle Shortcuts” und klicken Sie anschließend auf den Aktionsschalter Entfernen. Zum Zurücksetzen aller Tastenkombinationen auf die Voreinstellungen. Klicken Sie auf den Aktionsschalter “Vorgabe”, dadurch werden alle Tastenkombinationen auf die Voreinstellungen zurückgesetzt. Auch die Tastenkombinationen können Sie für alle Dokumente (normal.dot) oder für ein bestimmtes Dokument definieren. Wählen Sie vorher in dem Kombinationsfeld “Speichern in” den gewünschten Speicherort aus.

Klicken Sie auf den Aktionsschalter „Schließen“ die Tastenkombinationen sind nun wirksam.

Steuerelemente

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

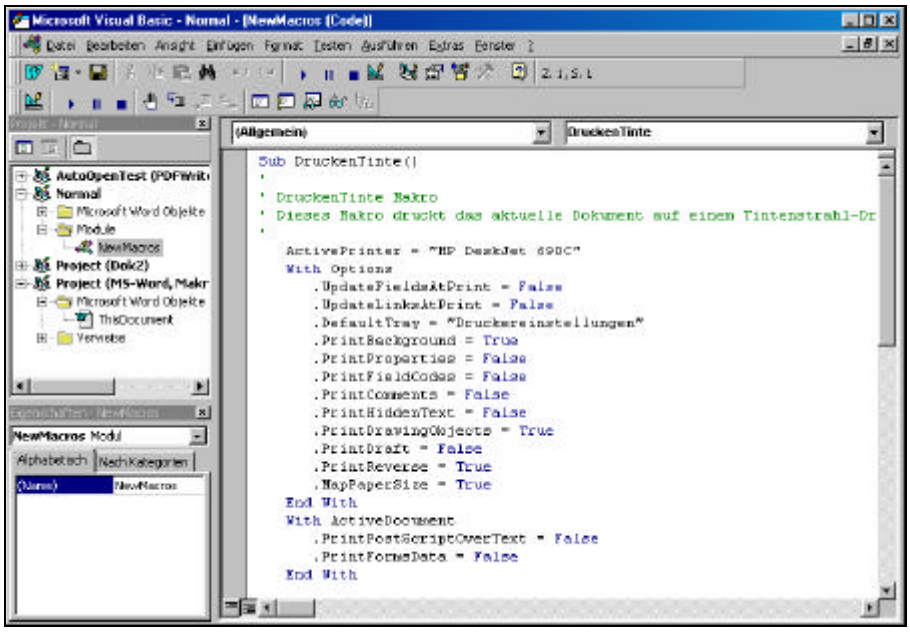
3 Der Visual-Basic-Editor

Visual-Basic-Modul

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

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 Dokumente (Visual-Basic-Projekte) 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 bzw. Zuweisungen der Eigenschaften. 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 zwei Makros entstanden, die im wesentlichen gleich aufgebaut sind. Das erste Makro hat folgenden Inhalt:

```
Sub DruckenTinte()
'
' DruckenTinte Makro
' Dieses Makro druckt das aktuelle Dokument auf einem
' Tintenstrahl-Drucker.
'
   ActivePrinter = "HP DeskJet 690C"
   With Options
      .UpdateFieldsAtPrint = False
      .UpdateLinksAtPrint = False
      .DefaultTray = "Druckereinstellungen"
      .PrintBackground = True
      .PrintProperties = False
      .PrintFieldCodes = False
      .PrintComments = False
      .PrintHiddenText = False
      .PrintDrawingObjects = True
      .PrintDraft = False
      .PrintReverse = True
      .MapPaperSize = True
   End With
   With ActiveDocument
      .PrintPostScriptOverText = False
      .PrintFormsData = False
   End With
   Application.PrintOut FileName:="", _
      Range:=wdPrintAllDocument, Item:= _
      wdPrintDocumentContent, Copies:=1, _
      Pages:="", PageType:=wdPrintAllPages, _
      Collate:=True, Background:=True, PrintToFile:=False
End Sub
```

Kommentare

Die grünen Textstellen sind Kommentare. Sie dienen lediglich zur Erläuterung. Hier wurden von Word 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 Visual-Basic-Schlüsselwörter. In diesen beiden Makros sind das die Worte Sub und End Sub sowie With und End With. Es gibt aber noch weitere Schlüsselworte, von denen die meisten im Kapitel Kontrollstrukturen vorgestellt werden.

Anweisungen

Die Anweisungen und Word-Funktionen werden in schwarzer Schrift angezeigt. Sie erkennen in beiden Makros mehrere Anweisungen, die den Aufrufen der einzelnen Dialogboxen entsprechen. Diese wollen wir im folgenden analysieren:

Visual-Basic ist eine objektorientierte Programmiersprache. Die meisten Befehle beziehen sich auf Objekte in Word selbst oder in einem Dokument. Die Objekte haben Eigenschaften. Diese Eigenschaften können verändert werden. Zum Verändern der Eigenschaften eines Objektes können die Methoden benutzt werden oder die Eigenschaften werden direkt durch eine Zuweisung verändert.

Schauen Sie sich die erste Anweisungszeile an. „ActivePrinter“ ist eine Eigenschaft der Anwendung durch die Zuweisung eines Druckernamens kann der aktuell ausgewählte Drucker geändert werden. Dies war unsere erste Aktion nach dem Öffnen der Dialogbox „Drucken“.

Im zweite Schritt haben wir die Druckoptionen geändert. Die Druck-Optionen werden repräsentiert durch ein Anwendungsobjekt und zwei Einstellungen für das aktive Dokument. Die Anwendungseigenschaft „Options“ repräsentiert diese Einstellungen. Dabei ist diese selbst wieder ein Objekt, daß mehrere Eigenschaften enthält. Damit Visual-Basic bei der Ausführung die einzelnen Eigenschaften dem Objekt „Options“ zuordnen kann werden, diese Eigenschaften durch eine With-Anweisung geklammert aufgeführt. Die Anweisungszeile „With Options“ bewirkt, daß im folgenden direkt auf die Eigenschaften des Objektes „Options“ zugegriffen werden kann. Die folgenden 12 Zeilen repräsentieren entsprechende Zuweisungen (Einstellungen) der Eigenschaften des Objektes. Die verschiedenen Eigenschaften korrespondieren mit den Einstellmöglichkeiten in der Dialogbox Optionen. Vergleichen Sie selbst: „UpdateFieldsAtPrint“ und „Felder aktualisieren“, „PrintBackground“ und „Drucken im Hintergrund“ oder „PrintReverse“ und „Umgekehrte Druckreihenfolge“. Überall dort wo Sie das Feld markiert haben, werden Sie im Makro sehen, daß der Eigenschaft der Wert „True“ (Wahr) zugewiesen wird und sonst wird „False“ (Falsch) zugewiesen. Das Schlüsselwort „End With“ kennzeichnet das Ende der Zuweisungen zu den Eigenschaften des Objektes „Options“.

ActiveDocument

Zwei weitere Eigenschaften gehören nicht zu den Anwendungsoptionen (Options), sie sind vielmehr Eigenschaften des Objektes, das das aktive Dokument repräsentiert. Dieses Objekt wird mit der Eigenschaft ActiveDocument angesprochen, da auch hier wieder mehrere Eigenschaften geändert werden, werden die Eigenschaften dieses Objekt wieder mit einer With-Anweisung geklammert angesprochen. Die beiden Eigenschaften „PrintPostScriptOverText“ und „PrintFormsData“ entsprechen den Einstellungen „Postscript über Text drucken“ und „in Formularen nur Daten drucken“ aus der Dialogbox Optionen.

Sie sehen, daß durch die Makro-Aufzeichnung alle aktuell eingestellten Werte gespeichert wurden, obwohl wir nur eine Einstellung (Umgekehrte Druckreihenfolge) geändert haben. Dies wäre eigentlich nicht nötig, ist aber bei den aufgezeichneten Makros so üblich. Um nur die Druckreihefolge zu ändern hätte allein die folgende Anweisung ausgereicht:

```
Options.PrintReverse = True
```

In dieser Anweisung wird zunächst das Objekt „Options“ angesprochen durch den Selektor (.) wird dann eine Eigenschaft ausgewählt, die Eigenschaft „PrintReverse“ und dieser durch den Zuweisungsoperator (=) der Wert „True“ zugewiesen.

Die letzte Anweisung dient dann zum Ausdrucken des Dokumentes. Dies haben wir bei der Aufzeichnung durch das Klicken auf den Aktionsschalter „OK“ in der Dialogbox „Drucken“ ausgelöst.

Bei dieser Anweisung fällt auf, daß sie über mehrere Zeilen verteilt ist. Auch dies sieht man häufig in aufgezeichneten Makros, da auch hier wieder alle Parameter an die jeweilige Methode übergeben, obwohl dies nicht nötig wäre. Das die restlichen Zeilen des Makros nur eine Anweisung darstellen erkennen Sie an den Unterstrichen (_) am Zeilenende.

Zeilenorientiert

Visual-Basic ist, im Gegensatz zu vielen anderen Programmiersprachen, eine zeilenorientierte Programmiersprache. Das bedeutet: Eine Zeile ist eine Anweisung. Oftmals reicht aber, wie auch im vorliegenden Fall, eine Zeile für die Darstellung einer Anweisung nicht aus. Deshalb gibt es die Möglichkeit durch einen Unterstrich am Ende einer Zeile dem Visual-Basic mitzuteilen, daß auch die nächste Zeile noch zu dieser Anweisung gehört und keine eigene Anweisung ist.

Diese Anweisung repräsentiert die Dialogbox „Drucken“. Durch das Schließen mit dem Aktionsschalter wird die Methode „PrintOut“ der Anwendung („Application“) mit den entsprechenden Parametern aufgerufen.

Vergleichen Sie selbst die Parameter mit den einzelnen Einstellmöglichkeiten in der Dialogbox: Der Parameter „Pagetype:=wdPrintAllPages“ entspricht der Einstellung „Alles“ im Gruppenfenster „Seitenbereich“, der Parameter „Item:= wdPrintDocumentContent“ entspricht der Einstellung „Dokument“ im Kombinationsfeld „Drucken“ oder der Parameter „Copies:=1“ entspricht der Einstellung „1“ im Editierfeld „Exemplare“.

Auch hier sehen Sie wieder, daß auch Einstellungen, die bei der Makro-Aufzeichnung nicht verändert wurden, mit aufgezeichnet wurden. Dies üblich bei Makro-Aufzeichnung, um das aktuelle Dokument mit den Voreinstellungen auszudrucken hätte auch die folgende Anweisungszeile ausgereicht:

```
Application.PrintOut
```

Durch diese Anweisung wird die Methode „Printout“ der Applikation („Application“) mit den Voreinstellungen aufgerufen. Das aktuelle Dokument wird ausgedruckt.

Auch am letzten Beispiel sehen wir, daß durch die Makroaufzeichnung unter Umständen sehr lange Makros entstehen, mit überflüssigen Anweisungen. Außerdem lassen sich bestimmte Anweisungen nicht mit Hilfe der Aufzeichnung realisieren. Deshalb gibt es auch immer wieder Fälle in denen Makros nicht aufgezeichnet werden (können) sondern von Hand geschrieben werden (müssen).

3.2 Makros editieren

Sie können, manchmal müssen Sie, Makros auch von Hand erstellen. Um ein neues Modul anzulegen, wählen Sie den Menüpunkt „Einfügen|Modul“ im Visual-Basic-Editor. 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 Makro-Namen.

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 Word-Dokument. Da der Visual-Basic-Editor ein eigenes Programmfenster hat, können Sie aber auch die Tastenkombination <Alt>+<Tab> verwenden, um zwischen den Fenstern von Word 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 Word-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.

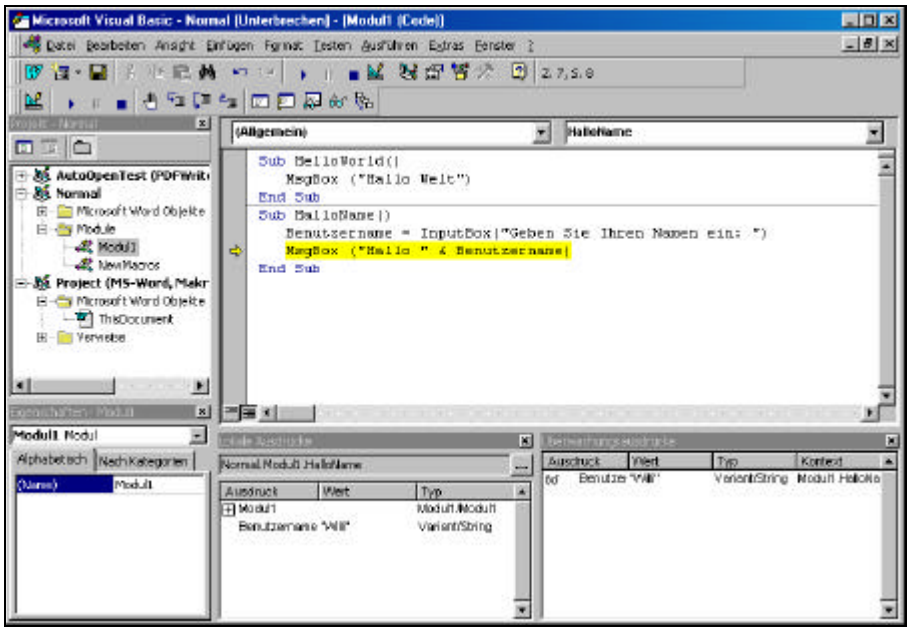
Mit diesem Symbol können Sie das Fenster des Projekt-Explorers sichtbar machen oder aktivieren. Das Ausblenden des Fensters erfolgt über das Kreuz in der Kopfzeile.

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 der 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 bei der schrittweisen Ausführung aufgerufenes Unterprogramm in einem Schritt verlassen und zum nächsten Befehl in dem aufrufenden Programm gelangen.

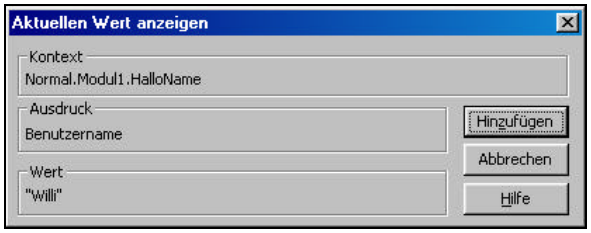
Mit diesem Symbol können Sie das Fenster zur Anzeige aller in einem Makro benutzten sogenannten lokalen Variablen anzeigen lassen. Dies ist beim Debuggen sehr hilfreich, um die Veränderung der Werte sehen zu können.

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.

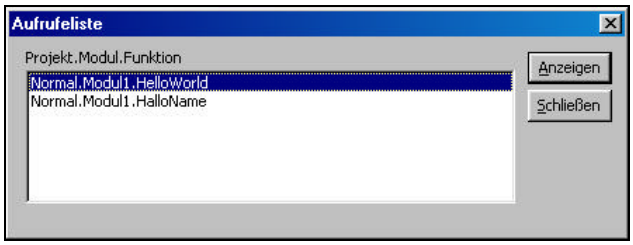
Mit diesem Symbol können Sie das Fenster zur Überwachung von beliebigen Variablen einblenden. Während das Fenster für die lokalen Symbole immer nur die Variablen anzeigt, die von dem im Moment debuggten Unterprogramm genutzt werden, können Sie hier auch Variablen sichtbar machen, die in einem anderen 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 Word angepaßt und enthält alle wichtigen Elemente einer Programmiersprache für Windows. Mit Visual-Basic ist die Entwicklung richtiger Applikationen für Word möglich. Sie können quasi eigene Programme als Zusatz für Word 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 Word 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 Word gibt es eine Reihe vordefinierter Objekte, die durch ihre Eigenschaften beschrieben sind, z. B. die Anwendung, das Dokument (Document), der Abschnitt (Section), der Absatz (Paragraph) oder die Wörter (Words). So hat z. B. das Dokument neben anderen Eigenschaften die Eigenschaft „Voller Name“ („Fullname“). Diese beinhaltet den vollen Dateinamen mit Pfadangabe.

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 Dokument gibt es z. B. die Methode „Drucken“ („PrintOut“). Mit dieser können Sie ein Dokument ausdrucken 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 die Auswahl (Selection). Die Selection ist die aktuelle Position der Schreibmarke oder ein markierter Bereich in einem Word-Dokument.

4.2 Anweisung, Zuweisung und Variablen

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

```
Sub Duplizieren()
   Documents("dok2.doc").Activate
   TextTeil = Selection.Text
   Selection.MoveDown Unit:=wdParagraph
   Selection.Text = TextTeil
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 Documents und Selection selbst wieder Methoden des Objektes Application sind. Sie beziehen sich auf die Anwendung und liefern als Ergebnis eben ein Bereichsobjekt oder ein Dokument-Objekt zurück. In der dritten Zeile wird der Wert eines Objektes, hier speziell des markierten Textes, durch eine Zuweisung (mit Hilfe des Gleichheitszeichens) in die Variable TextTeil 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 bereits angesprochenen Regeln wie für die Makro-Namen. Der Name darf nicht mit einem bereits existierenden Namen übereinstimmen.

In der vierten Zeile wird die Markierung um einen Absatz nach unten versetzt. Dies geschieht mit der Methode MoveDown der Parameter Unit:=wdParagraph bestimmt dabei das die Markierung um einen Absatz versetzt wird, ohne Parameter oder mit Unit:=wdLine könnten Sie das Versetzen um eine Zeile erreichen.

In der fünften Zeile wird dann umgekehrt der Wert dieser Variablen der Markierung zugewiesen. Das Aktivieren des Dokumentes ist nicht unbedingt notwendig, wenn Sie kein Dokument auswählen, dann wird das aktuell ausgewählte Dokument verwendet. Da die Markierung leer ist, wird der Text in das Dokument eingefügt. Wäre die Markierung nicht leer, würde der markierte Text durch diese Anweisung ersetzt.

Was macht das Makro? Vorausgesetzt es wurde zuvor ein Absatz markiert, dann dupliziert das Makro diesen Absatz. Allerdings hat das Makro einen kleinen Schönheitsfehler es setzt nämlich voraus, daß zuvor ein kompletter Absatz markiert wurde. Ist dies nicht der Fall, dann kopiert das Makro nur den markierten Text an das Ende des Absatzes.

In einem solchen Fall sollten Sie in dem Makro sicherstellen, daß die Markierung die gewünschte Ausdehnung hat. Dazu ergänzen Sie die folgenden Zeilen vor der ersten Zuweisung:

```
   Selection.MoveUp Unit:=wdParagraph
   Selection.MoveDown Unit:=wdParagraph, Extend:=wdExtend
```

Durch diese Anweisungen wird Markierung zunächst auf den Anfang des aktuellen Absatzes gesetzt und anschließend bis zum Ende des Absatzes erweitert.

Ein weiterer Schönheitsfehler unseres Makros ist, daß es die Formatierung nicht mitkopiert. Das folgende Makro zeigt, wie Sie auch die Formatierung mitkopieren können.

```
Sub Duplizieren3()
   Selection.MoveUp Unit:=wdParagraph
   Selection.MoveDown Unit:=wdParagraph, Extend:=wdExtend
   Set TextTeil = Selection.FormattedText
   Selection.MoveDown Unit:=wdParagraph
   Selection.FormattedText = TextTeil
End Sub
```

Die Eigenschaft Text des Objektes Selection enthält nur den unformatierten Text des Objektes. Der Text mit der Formatierung ist in der Eigenschaft FormattedText hinterlegt. Diese Eigenschaft ist selber wieder ein Objekt. Durch die Zuweisung in der dritten Anweisungszeile wird der Variablen TextTeil ein Verweis auf dieses Objekt zugewiesen. Weil hier kein Wert sondern eine Referenz zugewiesen wird ist hier auch das Schlüsselwort Set erforderlich. Die übrigen Anweisungen entsprechen dem vorhergehenden Beispiel.

Ein weiterer Schönheitsfehler des Makros ist, daß nach Ausführung die ursprüngliche Markierung verändert ist. Mit Hilfe der Eigenschaft Range können Sie die ursprüngliche Position der Markierung aber retten und nachher wieder herstellen. Die Range-Objekte sind der Selection sehr ähnlich, Sie können viele Eigenschaften und Methoden der Selection auch auf Range-Objekte anwenden. Range ist wie Selection ein Bereichsobjekt. Ergänzen Sie als erste und letzte Zeile in dem Makro die folgenden beiden Zeilen:

```
   Set AltePos = Selection.Range
...
   AltePos.Select
```

Durch die erste Anweisung wird der anfangs markierte Bereich in einer Objekt-Variablen (AltePos) gesichert. Nach Abschluß der Aktionen wird mit Hilfe der Methode Select der ursprüngliche Bereich wieder ausgewählt.

Das nächste Beispiel zeigt ein Makro zum Vertauschen zweier Worte. Es zeigt nochmal eine neue Eigenschaft, die Wortliste Words und die Anwendung der Methoden Cut und Paste (Ausschneiden und Einfügen)

```
Sub WorteTauschen()
   Selection.Words(1).Cut
   Selection.MoveRight Unit:=wdWord
   Selection.Paste
End Sub
```

Mit der ersten Anweisung wird aus dem markierten Text zunächst das erste Wort ausgewählt und dieses dann ausgeschnitten (Zwischenablage). Mit der zweiten Anweisung wird die Markierung um ein Wort nach rechts versetzt. Die dritte Anweisung fügt schließlich das zuvor ausgeschnittene Wort wieder in den Text ein.

4.3 Eigenschaften und die With-Anweisung

Der Text ist nicht die einzige veränderbare Eigenschaft eines Bereiches. So können Sie z. B. auch die Formatierung verändern. Weitere Möglichkeiten entnehmen Sie bitte der Word-Hilfe. Die nachfolgenden Beispiele zeigen entsprechende Möglichkeiten.

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 einem Bereichsobjekt enthalten ist, wie z. B. das Bereichsobjekt selbst in dem Objekt Auswahl enthalten ist und dieses Objekt wiederum in dem Objekt Anwendung enthalten ist.

Sie können die Eigenschaften solcher Objekte entweder so:

```
Sub Hervorheben()
   Selection.Font.Name = "Arial"
   Selection.Font.Size = 24
   Selection.Font.Bold = True
   Selection.Font.Italic = True
End Sub
```

oder so verändern:

```
Sub Hervorheben2()
   With Selection.Font
      .Name = "Arial"
      .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.

Das folgende Beispiel zeigt noch einmal die Veränderung eines ganz anderen Objektes.

```
Sub Dokumenteinrichten()
   With ActiveDocument.Sections(1).PageSetup
      .BottomMargin = CentimetersToPoints(2.5)
      .TopMargin = CentimetersToPoints(2.5)
      .LeftMargin = CentimetersToPoints(2.5)
      .RightMargin = CentimetersToPoints(2.5)
      .PaperSize = wdPaperA4
      .Orientation = wdOrientPortrait
   End With
End Sub
```

Dieses Makro verändert für den ersten Abschnitt des aktiven Dokumentes die aus der Dialogbox Seite einrichten (PageSetup) bekannten Einstellungen. Interessant ist hier auch die Funktion zur Umrechnung von cm-Angaben in Punkt (1 Punkt = $^{1}/_{72}$ Zoll (Inch)).

4.4 Adressierung von Objekten

Bisher haben wir hauptsächlich mit dem Objekt Selection gearbeitet, um einen bestimmten Bereich im Dokument anzusprechen. Grundsätzlich sind aber noch andere Möglichkeiten denkbar und in bestimmten Situationen vielleicht auch besser einzusetzen.

4.4.1 Dokumente

Documents

Documents ist ein Methode der Anwendung. Sie liefert aus der Liste der geöffneten Dokumente ein Dokument-Objekt. Grundsätzlich haben Sie zwei Möglichkeit das gewünschte Dokument zu selektieren. Entweder mit dem Namen oder mit dem Index.

```
Documents("dok2.doc").Activate
```

Oder

```
Documents(2).Activate
```

Der Index entspricht der Reihenfolge der letzten Aktivierung der Dokumente (ausgeblendete Dokumente zählen mit). Daher ist die Verwendung ist bei Verwendung des Index das Ergebnis nicht immer eindeutig. Die erste Möglichkeit ist also zu bevorzugen.

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

```
Documents.Add
Documents.Open ("dok2.doc")
```

Mit dem ersten Befehl wird ein neues leeres Dokument angelegt mit dem zweiten Befehl wird das vorhandene Dokument mit dem Namen dok2.doc geöffnet. Bei der Methode Add können Sie einen Dateinamen als Parameter (Template) angeben, wenn Sie eine Vorlage verwenden möchten.

```
Documents.Add("Brief.dot")
```

Alternativ zu der zuerst vorgestellten Aufrufform mit den Klammern können Sie auch die folgende Form verwenden.

```
Documents.Add Template:="Brief.dot"
```

Außerdem können Sie mit einem weiteren Parameter (NewTemplate) festlegen, ob Sie ein neues Dokument oder eine neue Vorlage erzeugen möchten. Der folgende Aufruf zeigt das Erzeugen einer neuen Vorlage auf Basis der Vorlage Brief.dot:

```
Documents.Add Template:="Brief.dot", NewTemplate:=True
```

Die zweite Form des Aufrufes ist immer dann sinnvoller, wenn Sie nur einen Parameter aus einer langen Liste von Parametern benötigen.

```
Documents.Add NewTemplate:=True
```

Dieser Aufruf erzeugt z. B. eine neue Vorlage auf Basis der Normalvorlage (Normal.dot).

Zum Schließen von Dokumenten verwenden Sie die Methode Close. Diese Methode steht sowohl für die ganze Dokumenten-Liste als auch für ein einzelnes Dokument zur Verfügung.

```
Documents.Close
Document("dok2.doc").Close(True)
```

oder

```
Documents("dok2.doc").Close SaveChanges:=True
```

Mit dem ersten Befehl werden alle derzeit offenen Dokumente geschlossen. Mit dem zweiten Befehl nur das Dokument dok2.doc. Durch Angabe des Parameters True erreichen Sie, daß die Änderungen ohne Rückfrage gespeichert werden.

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

```
Documents("dok2.doc").Save
Documents("dok2.doc").SaveAs FileName:="dok3.doc"
```

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

Das aktive Dokument sprechen Sie mit der Eigenschaft ActiveDocument an.

Mit der Methode PrintOut können Sie z. B. das aktive oder ein bestimmtes Dokument ausdrucken lassen

```
ActiveDocument.PrintOut
Documents("dok2.doc").PrintOut
```

4.4.2 Weitere Bereichsobjekte

Innerhalb der Dokumente gibt es wiederum Listen, die Elemente eines Word-Dokuments wiederspiegeln. Sie erinnern sich noch: Ein Word-Dokument besteht aus folgende Elementen:

- Abschnitte
- Absätze
- Sätze
- Wörter
- Zeichen

Diese Objekte finden Sie auch im Objekt-Modell von Word, sie können auf das Dokument oder auf Teilobjekte eines Dokuments bezogen sein:

```
MsgBox( ActiveDocument.Words(1))
MsgBox( Selection.Words(1))
```

Durch die erste Anweisung wird in einer Messagebox das erste Wort eines Dokumentes durch die zweite Anweisung das erste Wort der Auswahl ausgegeben.

Abschnitte

Innerhalb eines Dokumentes gibt als nächst kleinere Einheit die Abschnitte. Sie können durch die Sections-Auflistung angesprochen werden.

```
ActiveDocument.Sections.Add
ActiveDocument.Sections(1).PageSetup
```

Weitere interessante Eigenschaften der Sections sind:

- Footers
  Dieses Objekt repräsentiert die Fußzeilen eines Abschnittes. In Abhängigkeit von den Einstellungen für die Eigenschaften DifferentFirstPageHeaderFooter und OddAndEvenPagesHeaderFooter des PageSetup-Objektes kann es bis zu drei (Kopf- und) Fußzeilen für einen Abschnitt geben.
- Headers
  Dieses Objekt repräsentiert die Kopfzeilen eines Abschnittes.
- ProtectedForForms
  Mit dieser Eigenschaft können Abschnitte für die Formulareingabe geschützt werden.

Absätze

Auflistungen von Absätzen finden Sie in den Range-Objekten beliebiger Objekte, z. B.

```
Documents(1).Range.Paragraphs(1).Range.Select
Documents(1).Sections(2).Range.Paragraphs(2).Range.Select
```

Die erste Anweisung markiert den ersten Absatz im Dokument. Das zweite Beispiel zeigt die Markierung des zweiten Absatzes im zweiten Abschnitt des Dokumentes. Mit Hilfe der Eigenschaften des Objektes Paragraphs können Sie die Formatierung eines oder mehrerer ausgewählter Absätze vornehmen.

```
Selection.Paragraphs(1).Alignment = wdAlignParagraphRight
```

Diese Anweisung legt fest, daß der erste Absatz der Markierung rechtsbündig formatiert wird. Weitere Eigenschaften ermöglichen die Einstellung aller Absatz-Eigenschaften. Mit der Eigenschaft Format können Sie das Absatz-Format als ganzes ansprechen, um so alle Absatz-Einstellungen in einmal anzusprechen.

```
Selection.Paragraphs(1).Previous.Format = _
   Selection.Paragraphs(1).Format
```

Dieses Beispiel überträgt alle Absatz-Einstellungen auf den vorhergehenden Absatz. Ein weitere nützliche Eigenschaft ist die Eigenschaft Style, mit dieser können Sie die dem Absatz zugewiesene Formatvorlage ermitteln oder bestimmen.

```
Selection.Paragraphs(1).Style = "Standard"
```

Dieses Beispiel weist dem ersten Absatz in der Markierung das Absatzformat „Standard“ zu.

Sätze

Mit der Auflistung Sentences können Sie auf einzelne Sätze in einem Range-Objekt zugreifen.

```
Selection.Sentences(1).FormattedText.Bold = True
```

Das Beispiel formatiert den ersten Satz innerhalb der Markierung in Fettschrift.

Wörter

Mit der Auflistung Words können die einzelnen Worte innerhalb eines Range-Objektes angesprochen werden.

```
Selection.Words(1).LanguageID =wdEnglishUS
```

Das Beispiel zeigt, wie Sie für das markierte Wort die Sprache ändern können.

Zeichen

Mit der Auflistung Characters können die einzelnen Zeichen innerhalb eines Range-Objektes angesprochen werden. Sie erhalten als Ergebnis ein Range-Objekt, das nur ein einzelnes Zeichen beinhaltet.

Weitere interessante Objekte bzw. Objektlisten sind:

- AutotextEntries
  Liste mit den Autotext-Einträgen einer Vorlage. Der Zugriff auf Vorlagen kann mit den Methoden NormalTemplate (Normal-Vorlage) oder AttachedTemplate (aktuell dem Dokument zugeordnete Vorlage) erfolgen.
- Bookmarks
  Liste der Textmarken innerhalb eines Dokumentes, die Eigenschaft kann für die Navigation innerhalb von Dokumenten nützlich sein.
- Styles
  Liste der Formatvorlagen innerhalb eines Dokumentes.
- Templates
  Liste der verfügbaren Dokumentvorlagen.
- FormFields
  Liste der Formularfelder innerhalb eines Dokumentes.
- Windows
  Liste der innerhalb von Word geöffneten Fenster.

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-Anweisung

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 eine vom Benutzer einzugebene Zahl ein Rabatt von 20% bestimmt werden und der berechnete Wert soll an der aktuellen Position der Markierung in den Text eingefügt werden:

Im ersten Ansatz könnte die Lösung für die Aufgabe etwa so aussehen:

```
Sub Rabatt()
   Wert = InputBox("Wert? ")
   Wert = Wert * 0.8
   Selection.InsertAfter Wert
   Selection.MoveRight Unit:=wdWord
End Sub
```

Leider hat dieses Makro einige „Schönheitsfehler“. Was passiert z. B., wenn der Benutzer statt einer Zahl in dem Eingabefeld irgendeinen Text eingibt? In diesem Fall erhält der Anwender eine Laufzeitfehlermeldung. Dies sollte Sie vermeiden. Sie können es vermeiden indem Sie die Berechnung nur durchführen, wenn der eingegebene Wert eine Zahl ist. Das nachfolgende Beispiel:

```
Sub Rabatt1()
   Wert = InputBox("Wert? ")
   If IsNumeric(Wert) Then
      Wert = Wert * 0.8
      Selection.InsertAfter Wert
      Selection.MoveRight Unit:=wdWord
   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. Als Bedingung wird in diesem Fall die Informations-Funktion IsNumeric aufgerufen, deren Funktionswert ist ein logischer Wert. Die nachfolgend zwischen den Schlüsselworten Then und End If stehenden Anweisungen werden nur dann ausgeführt, wenn die hinter dem Schlüsselwort If stehende Bedingung erfüllt ist.

Oftmals werden zwei Fälle unterschieden und in beiden Fällen ist unterschiedlich zu reagieren. In unserem Beispiel hat das Makro nun den Nachteil, daß es den Anwender völlig im Unklaren darüber läßt, warum es nichts macht. Es wäre schöner, wenn für den Fall einer unzulässigen Eingabe eine entsprechende Hinweismeldung am Bildschirm angezeigt würde. Für diesen Zweck werden Sie die If-Then-Else-Anweisung verwenden. Das nachfolgende Makro zeigt eine entsprechende Erweiterung:

```
Sub Rabatt2()
   Wert = InputBox("Wert? ")
   If IsNumeric(Wert) Then
      Wert = Wert * 0.8
      Selection.InsertAfter Wert
      Selection.MoveRight Unit:=wdWord
   Else
      MsgBox ("Eingabe von Zahlen erforderlich!")
   End If
End Sub
```

Zwei Fälle

Das Makro entspricht im wesentlichen dem vorhergehenden Beispiel. Zusätzlich wird jedoch auch der Fall bearbeitet, daß die hinter dem Schlüsselwort If stehende Bedingung nicht erfüllt ist. Für den Fall, daß die Bedingung erfüllt ist, werden die zwischen den Schlüsselworten Then und Else stehenden Anweisung ausgeführt. Für den Fall das die Bedingung nicht erfüllt ist., werden die zwischen Else und End If stehenden Anweisungen ausgeführt.

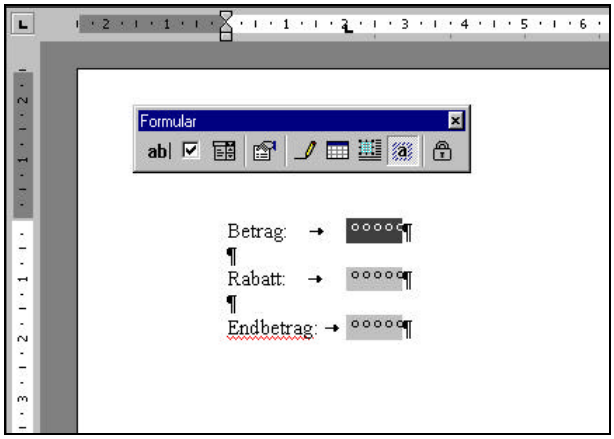
Oftmals müssen aber auch mehr als zwei Fälle unterschieden werden. Vom Prinzip her kann eine solche Aufgabenstellung mit mehreren in einander geschachtelten Wenn-Anweisungen gelöst werden. Eleganter ist aber oftmals die Verwendung des Schlüsselwortes ElseIf. Dies könnte dann wie folgt aussehen:

```
Sub Rabatt3()
   Wert = InputBox("Wert? ")
   If IsNumeric(Wert) Then
      If Wert >= 10000 Then
         Wert = Wert * 0.7
      ElseIf Wert >= 1000 Then
         Wert = Wert * 0.8
      ElseIf Wert >= 100 Then
         Wert = Wert * 0.9
      Else
         MsgBox ("Kein Rabatt, da kleiner 100.")
      End If
      Selection.InsertAfter Wert
      Selection.MoveRight Unit:=wdWord
   Else
      MsgBox ("Eingabe von Zahlen erforderlich!")
   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.

Beispiel

Als ein weiteres Beispiel für die Anwendung einer If-Anweisung soll noch die Anwendung einer entsprechenden Anweisung in Verbindung mit den Formularfeldern gezeigt werden. Geben Sie bitte folgende Formularfelder ein:



Über das Eigenschaftenfenster (Symbol mit der Hand) benennen Sie die drei Formularfelder so wie Sie beschriftet haben. Bei den Feldern Rabatt und Endbetrag sperren Sie die Eingabe.

Dann erfassen Sie das folgende Makro:

```
Sub RabattBestimmen()
   Eingabe = ActiveDocument.FormFields("Betrag").Result
   If IsNumeric(Eingabe) Then
      Betrag = CDbl(Eingabe)
      If Betrag >= 10000 Then
         Endbetrag = Betrag * 0.7
      ElseIf Betrag >= 1000 Then
         Endbetrag = Betrag * 0.8
      ElseIf Wert >= 100 Then
         Endbetrag = Betrag * 0.9
      Else
         Endbetrag = Betrag
      End If
      Rabatt = Betrag - Endbetrag
      ActiveDocument.FormFields("Rabatt").Result = Rabatt
      ActiveDocument.FormFields("Endbetrag").Result          =
Endbetrag
   Else
      ActiveDocument.FormFields("Betrag").Result = ""
      ActiveDocument.FormFields("Rabatt").Result = ""
      ActiveDocument.FormFields("Endbetrag").Result = ""
      MsgBox ("Eingabe von Zahlen erforderlich!")
   End If
End Sub
```

Anschließend wechseln Sie zu dem Dokument mit den Formular und öffnen das Eigenschaftenfenster für das erste Feld. In dem Feld „Makro starten beim Verlassen“ wählen Sie das Makro Rabatt bestimmen aus. Dadurch erreichen Sie, daß dieses Makro bei jedem Verlassen des Feldes automatisch den eingegebenen Wert überprüft. Falls dieser Wert gültig ist, wird die entsprechende Berechnung durchgeführt und die beiden Felder Rabatt und Endbetrag werden automatisch gefüllt, falls der eingegebene Wert ungültig ist, werden alle Felder automatisch gelöscht und eine entsprechende Fehlermeldung ausgegeben.

4.5.2 Select-Anweisung

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 Eingabe gültig war, 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 Rabattbestimmen()
   Betrag = ActiveDocument.FormFields("Betrag").Result
   Fehler = True
   If IsNumeric(Betrag) Then
      Rabattklasse = InputBox("Rabattklasse?")
      If IsNumeric(Rabattklasse) Then
         Select Case Rabattklasse
            Case 1
               Endbetrag = Betrag * 0.95
            Case 2, 3
               Endbetrag = Betrag * 0.9
            Case 4 To 6, 8
               Endbetrag = Betrag * 0.8
            Case 7, Is < 12
               Endbetrag = Betrag * 0.7
            Case Else
               MsgBox ("Ungültige Rabattklasse")
         End Select
         Rabatt = Betrag - Endbetrag
         ActiveDocument.FormFields("Rabatt").Result = Rabatt
         ActiveDocument.FormFields("Endbetrag").Result = _
            Endbetrag
         Fehler = False
      Else
         MsgBox ("Ungültige Eingabe für Rabattklasse")
      End If
   Else
      MsgBox ("Eingabe von Zahlen erforderlich!")
   End If
   If Fehler Then
      ActiveDocument.FormFields("Betrag").Result = ""
      ActiveDocument.FormFields("Rabatt").Result = ""
      ActiveDocument.FormFields("Endbetrag").Result = ""
   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. Diese 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.

Fehlermerker

Eine weitere Besonderheit bei diesem Beispiel ist der Fehlermerker, die Variable Fehler die am Anfang auf True gesetzt wird und im Falle einer erfolgreichen Bearbeitung auf False gesetzt wird. Am Ende wird diese Variable dann benutzt, um zu steuern, ob eine Fehlerbehandlung ausgeführt werden muß oder nicht. Diese Vorgehensweise ist immer dann sinnvoll, wenn Sie mehrere in einander geschachtelte Gültigkeitsprüfungen haben. Wenn Sie keinen Fehlermerker einführen würden, müßten Sie sonst bei jedem Fehlerfall eine Fehlerbehandlung programmieren. So können Sie dies einmal am Ende des Makros machen.

4.5.3 For-Anweisung

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 oder berechenbar ist. Geben Sie das folgende Makro ein und überlegen Sie, was als Ergebnis dieses Makros ausgegeben wird:

```
Sub Kalender()
   StartDatum = CDate(Selection.Text)
   AnzahlTage = InputBox("Wieviel Tage?")
   For nIndex = 1 To AnzahlTage
      Datum = StartDatum + nIndex
      Selection.InsertParagraphAfter
      Selection.MoveRight Unit:=wdCharacter
      Selection.InsertAfter Text:=Format(Datum, "DD.MM.YY")
   Next nIndex
End Sub
```

Das Makro ermittelt aus einem markierten Text ein Anfangsdatum. Über eine Eingabe-Funktion, wird vom Benutzer die Anzahl der Wiederholungen abgefragt. In der For-Anweisung werden dann die zwischen den Schlüsselworten For und Next stehenden Anweisungen entsprechend oft wiederholt. Der erste Ausdruck hinter dem Schlüsselwort For bestimmt den Namen des sogenannten Schleifenzählers und den Startwert, der Ausdruck hinter dem Schlüsselwort To bestimmt den Endwert. Beim ersten Durchlauf hat der Schleifenzähler nIndex in diesem Fall den Wert 1. Bei jedem Schleifendurchlauf wird der Schleifenzähler um 1 erhöht. Die Schleife wird solange durchlaufen, bis der Schleifenzähler den Endwert (AnzahlTage) erreicht hat. Danach werden die hinter Next stehenden Anweisungen weiter abgearbeitet.

Die Anweisungen innerhalb der For-Anweisung dienen dazu, die Markierung um jeweils einen Absatz (Zeile) tiefer zu setzen und das nächst Datum einzufügen. Beachten Sie bitte wie mit Hilfe der Format-Anweisung die Datumsausgabe aufbereitet wird.

Eine Variante des vorhergehenden Beispiels soll Ihnen weitere Möglichkeiten der For-Anweisung zeigen:

```
Sub Kalender2()
   StartDatum = CDate(Selection.Text)
   EndDatum = CDate(InputBox("Enddatum:"))
   For Datum = StartDatum To EndDatum Step 7
      Selection.InsertParagraphAfter
      Selection.MoveRight Unit:=wdCharacter
      Selection.InsertAfter Text:=Format(Datum, "DD.MM.YY")
   Next Datum
End Sub
```

Das Makro setzt das als Startwert markierte Datum bis zu dem als Endwert eingegebene Datum fort. Durch das Schlüsselwort Step wird allerdings erreicht, das der Schleifenzähler in diesem Fall die Variable Datum nicht um 1 sondern jeweils um 7 erhöht wird. Das Datum wird also nur für alle 7 Tage ausgegeben. Falls eine spezielle Anwendung es erfordert, können Sie den Schleifenzähler auch um Werte, die kleiner sind als 1 erhöhen, z. B. Step 0.5. Das folgende Makro zeigt noch einmal die Anwendung der For-Anweisung. In diesem Fall werden alle Formularfelder des aktiven Dokumentes gelöscht.

```
Sub AlleFelderLöschen()
   For nIndex = 1 To ActiveDocument.FormFields.Count
      ActiveDocument.FormFields(nIndex).Result = ""
   Next nIndex
End Sub
```

Mit diesem Makro werden alle Elemente einer Auflistung FormFields angesprochen. Die Anzahl der Elemente in der Auflistung läßt sich durch die Eigenschaft Count ermitteln. In der Schleife werden die einzelnen Elemente der Auflistung dann mit Hilfe des Schleifenzählers indiziert angesprochen. In diesem speziellen Fall geht es um die Behandlung aller Elemente einer Auflistung. Dies läßt sich eleganter lösen mit einer weiteren Variante der For-Anweisung. Das nachfolgende Beispiel zeigt ein Makro, das den gleichen Zweck erfüllt wie das vorhergehende Beispiel:

```
Sub AlleFelderLöschen2()
   For Each Feld In ActiveDocument.FormFields
      Feld.Result = ""
   Next Feld
End Sub
```

In diesem Makro wird durch das Schlüsselwort Each automatisch erreicht, daß alle Elemente der Auflistung aufgerufen werden. Ist die Auflistung leer, werden die Anweisungen in der Schleife nicht durchlaufen, ansonsten wird die Schleife sooft durchlaufen wie es Objekte in der Auflistung gibt. Innerhalb der Schleife können Sie die Objekte mit Hilfe des Namens der Schleifenvariablen (in diesem Fall Feld) ansprechen. Das nachfolgende Makro zeigt ein weiteres Beispiel:

```
Sub AlleSpeichern()
   For Each Dokument In Documents
      Dokument.Save
   Next Dokument
End Sub
```

Bei diesem Beispiel werden alle derzeit offenen Dokumente gespeichert.

4.5.4 Do-Anweisung

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 wollen wir noch einmal das erste Rabatt-Makro aufgreifen. Dieses soll nun so erweitert werden, daß es die Abfrage wiederholt, bis der Benutzer eine gültige Eingabe gemacht hat oder den Vorgang mit einem entsprechenden Aktionsschalter abbricht.

```
Sub Rabatt5()
   Do
      Wert = InputBox("Wert? ")
      If IsNumeric(Wert) Then
         Wert = Wert * 0.8
         Selection.InsertAfter Wert
         Selection.MoveRight Unit:=wdWord
         EingabeOK = True
      Else
         nTest = MsgBox("Ungültige Eingabe", vbRetryCancel)
      End If
   Loop Until EingabeOK Or nTest = vbCancel
End Sub
```

Die Anweisung zwischen den Schlüsselworten Do und Loop werden sooft wiederholt, bis der Benutzer entweder eine gültige Eingabe gemacht hat oder in der Hinweismeldung auf den Aktionsschalter Abbrechen klickt. Das besondere an dieser Anweisung ist, daß zum Zeitpunkt der Programmierung nicht feststeht, wie oft die Anweisungen ausgeführt werden müssen. Die Zahl der Wiederholungen ist weder festgelegt noch berechenbar. Sie hängt vielmehr vom Verhalten des Benutzers während der Laufzeit ab.

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.

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 Typkennzeichen 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 Typkenn-zeichen 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:

```
Wert = 12
```

Deklariert implizit eine Variable vom Typ Variant.

```
Wert$ = "Test"
```

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

```
Wert$ = 12
```

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

```
Zahlwert = CInt( Wert$ )
```

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 Beispiel verdeutlicht werden. Für einen Vertrag sollen zwei Adressen aus einer Liste gesucht und in einen Vertragstext eingefügt werden. Der Vertragstext ist in der Vorlage „Vertrag.dot“. Eine neue Datei wird auf Basis dieser Vorlage angelegt. Anhand eines Kurznamens, der in einer Inputbox abgefragt wird, wird in der Datei „WMAdressen.doc“ eine Adresse für den Verkäufer gesucht, anschließend wird die Adresse an die Position einer entsprechenden Textmarke eingefügt. Im Anschluß daran wird das gleiche für die Adresse des Käufers vorgenommen. Die Adressenliste ist in einer Tabelle erfaßt. Für jede Adresse gibt es eine Zeile, in der ersten Spalte steht der als Suchbegriff verwendete Kurzname.

Modularisierung

Wenn Sie nun darüber nachdenken, wie sich diese Aufgabenstellung umsetzen läßt, werden Sie sehr schnell feststellen, daß dies schon eine recht komplexe Aufgabe wird. Sie werden feststellen, daß bestimmen Anweisungssequenzen häufiger gebraucht werden, und daß bestimmte Anweisungssequenzen eventuell auch noch für andere Aufgabenstellungen gebraucht werden könnten. Dies alles sind Ansatzpunkte, für eine Modularisierung. Sie sollten versuchen das Projekt in Teilaufgaben zu zerlegen, und diese dann relativ kleinen und abgeschlossenen Aufgaben jeweils für sich lösen (frei nach dem Motto: Divide et impera).

Gesamtablauf

Wir wollen zunächst den gesamten Ablauf betrachten, um die sich wiederholenden Teilaufgaben erkennen zu können:

1. Dokument mit der Adressenliste (ggf.) laden
2. Kurzname für den Verkäufer abfragen
3. Zeilennummer der Adresse suchen
4. Adresse anhand der Zeilennummer aus der Tabelle holen
5. Adresse für den Verkäufer in das Dokument einfügen
6. Kurzname für den Käufer abfragen
7. Zeilennummer der Adressen suchen
8. Adresse anhand der Zeilennummer aus der Tabelle holen
9. Adresse für den Käufer einfügen

Sie erkennen, daß sich einige Teilaufgaben wiederholen, und im Gegensatz zum Punkt 2 ist der Punkt 3 sicherlich auch nicht mit einer Anweisung erledigt. Solche Teilaufgaben bieten für Sie die Ansatzpunkte zur Erstellung von Unterprogrammen. Bei der Erstellung solcher modularisierten Programme gibt es nun zwei Strategien Top-Down oder Bottom-Up. Bei der ersten erstellen Sie zunächst das Gesamtprogramm und lösen dann die Teilaufgaben. Bei der zweiten Strategie gehen Sie genau umgekehrt vor. Sie lösen erst die Teilaufgaben und bauen Sie anschließend zum Gesamtprogramm zusammen.

Teilaufgabe

Schauen wir uns zunächst die erste Teilaufgabe an. Zur Lösung der Aufgabe müssen wir über alle Zeilen der ersten Spalte schauen und solange prüfen welcher Kurzname dort steht bis wir entweder den Kurznamen gefunden haben oder das Ende der Liste erreicht ist. Welche Angaben benötigen Sie, um diese Aufgabe zu erfüllen? Sie benötigen das Dokument oder zumindest die Tabelle, und Sie benötigen den Kurznamen nach dem gesucht werden soll. Als Ergebnis geben Sie zurück in welcher Zeile der Kurzname steht. Sie müssen sich außerdem Gedanken darüber machen, welche Ausnahmen möglich sind. So kann es in unserem Fall z. B. vorkommen, daß der gesuchte Begriff nicht in der Tabelle vorkommt, was soll in diesem Fall passieren? Wir einigen uns darauf, daß in diesem Fall die ungültige Zeilennummer 0 zurückgegeben wird.

Wenn wir nun darüber nachdenken, wie die Aufgabe gelöst werden kann, stellen wir fest, daß sich in dieser Aufgabe noch mal eine kleinere Teilaufgabe verbirgt, die wir auch bei der Lösung der Teilaufgabe mit dem Holen der kompletten Adresse später benötigen werden nämlich das Auslesen eines Textes aus einer Tabellen-Zelle. Für diese Aufgabe brauchen wir als Parameter einen Verweis auf die Tabelle und die Nummern der entsprechenden Zeile und Spalte. Als Ergebnis wird der Inhalt (Text) der Zelle zurückgegeben. Fangen wir zunächst mit der Lösung dieser Teilaufgabe an:

```
Sub TabellenInhalt(Inhalt, Tabelle, Zeile, Spalte)
   ' Dieses Makro ermittelt den Inhalt einer Tabellen-Zelle
   Set Zelle = Tabelle.Cell(Zeile, Spalte).Range
   Zelle.MoveEnd Unit:=wdCharacter, Count:=-1
   Inhalt = Zelle.Text
End Sub
```

Dieses Makro hat vier sogenannte formale Parameter, die zur Übergabe der benötigten Ausgangswerte und des Ergebnisses dienen, diese Parameter müssen später beim Aufruf entsprechend angegeben werden.

Das Makro selbst ermittelt das zu der Zelle gehörende Range-Objekt, da dieses aber auch die Zellenendemarke beinhaltet, muß anschließend die Selection um ein Zeichen nach links versetzt werden. Damit steht in der Text-Eigenschaft des Range-Objektes dann nur der Text zur Verfügung, der mit der letzten Anweisung dem Rückgabewert zugewiesen wird.

Parameter

Bei der Beschreibung der Parameterliste haben wir auf irgendwelche Typangaben verzichtet durch Hinzufügen des Schlüsselwortes As und dem entsprechenden Typbezeichner können Sie auch eine entsprechende Überprüfung der Werte zur Laufzeit erreichen. Der Kopf der Prozedur könnte dann etwa so aussehen:

```
Sub TabellenInhalt(Inhalt As String, Tabelle As Object, _
   Zeile As Integer, Spalte As Integer)
```

Referenzparameter

Sie werden feststellen, daß wenn Sie den Kopf Ihres (Sub-) Makros in dieser Art definieren, so wird es nur funktionieren, wenn die Aufruf-Parameter genau den erwarteten Typen entsprechen. Als Parameter werden in einem solchen Fall die Referenzen auf Variablen vom erwarteten Typ übergeben. Sie können jedoch auch nur Werte übergeben, daß wäre z. B. für die Zeilen- und Spaltennummern möglich. Der Kopf könnte dann so aussehen:

```
Sub TabellenInhalt( ByRef Inhalt As String, _
                    ByRef Tabelle As Object, _
                    ByVal Zeile As Integer, _
                    ByVal Spalte As Integer)
```

Werteparameter

Werteparameter (ByVal) können innerhalb eines (Sub-) Makros verändert werden, ohne daß diese Änderungen nach außen sichtbar werden. Bei Werte-Parametern wird innerhalb des Unterprogramms quasi mit einer Kopie der entsprechenden Variablen gearbeitet. Während bei der Übergabe als Referenz auch innerhalb des Unterprogramms mit der Variablen selbst gearbeitet wird.

Optionale Parameter

Eine weitere Variante ist das Auslassen von Parametern, sogenannten optionalen Parametern. Für das oben angegebene Unterprogramm könnte man sich z. B. die Spaltennummer als optionalen Parameter vorstellen. Wenn keine Spaltennummer angegeben wird, wird automatisch die Spalte 1 verwendet. Dies könnte in der Deklaration der formalen Parameterliste dann wie folgt aussehen:

```
Sub TabellenInhalt( ByRef Inhalt As String, _
                    ByRef Tabelle As Object, _
                    ByVal Zeile As Integer, _
                    Optional ByVal Spalte As Integer = 1)
```

In diesem Beispiel könnten Sie beim späteren Aufruf des Unterprogramms den letzten Parameter, die Spaltennummer, auch weglassen, es würde dann automatisch die Spalte 1 angenommen. Sie können mehrere optionale Parameter haben, diese müssen jedoch immer die letzten Parameter in der formalen Parameterliste sein.

Wir wollen bei dem Ausgangsbeispiel bleiben und alle Parameter wie dies standardmäßig erfolgt, „ByRef“ und als „Variant“ (ohne Typangabe) übergeben.

Den ersten Teil der ersten kleinen Teilaufgabe hätten Sie damit erledigt, jetzt müssen Sie das eigentliche Suchen der Adresse codieren. Die Lösung könnte etwa wie folgt aussehen:

```
Sub AdresseSuchen(Dokument, SuchText, Zeile)
   ' Dieses Makro ermittelt die Zeilen-Nummer einer Adresse
   With Dokument
      nIndex = 1
      bFound = False
      Do
         TabellenInhalt Inhalt:=ZellenText, _
            Tabelle:=.Tables(1), Zeile:=nIndex, Spalte:=1
         If ZellenText = SuchText Then
            bFound = True
         Else
            nIndex = nIndex + 1
         End If
      Loop Until bFound Or nIndex > .Tables(1).Rows.Count
      If Not bFound Then
         nIndex = 0
         MsgBox ("Ungültiger Kurzname")
      End If
   End With
   Zeile = nIndex
End Sub
```

Beachten Sie, wie in diesem Beispiel der Aufruf des zuvor erstellen Unterprogramms Tabelleninhalt erfolgt.

In einer Do-Loop-Anweisung wird solange der Inhalt der ersten Zelle einer jeden Zeile überprüft, bis der Wert gefunden wurde (bFound=True) oder das Ende der Tabelle erreicht wurde.

In der zweiten Teilaufgabe kommt dann ebenfalls das Unterprogramm zum Holen eines Tabellenwertes zum Einsatz. Mit Hilfe der zuvor gesuchten Zeilennummer kann nun die gesamte Adresse in eine Text-Variable übernommen werden.

```
Sub AdresseHolen(Dokument, Adresse, Zeile)
   If Zeile > 0 Then
      ' Adresse zusammensetzen
      With Dokument
         TabellenInhalt Tabelle:=.Tables(1), Zeile:=Zeile, _
            Spalte:=2, Inhalt:=Anrede
         TabellenInhalt Tabelle:=.Tables(1), Zeile:=Zeile, _
            Spalte:=3, Inhalt:=Name1
         TabellenInhalt Tabelle:=.Tables(1), Zeile:=Zeile, _
            Spalte:=4, Inhalt:=Name2
         TabellenInhalt Tabelle:=.Tables(1), Zeile:=Zeile, _
            Spalte:=5, Inhalt:=Straße
         TabellenInhalt Tabelle:=.Tables(1), Zeile:=Zeile, _
            Spalte:=6, Inhalt:=Ort
         Adresse = Anrede + vbCr
         Adresse = Adresse + Name1 + vbCr
         Adresse = Adresse + Name2 + vbCr
         Adresse = Adresse + Straße + vbCr
         Adresse = Adresse + Ort + vbCr
      End With
   Else
      Adresse = "Nicht gefunden"
   End If
End Sub
```

Die einzelnen Bestandteile der Adresse werden anschließend zu einer Zeichenkette zusammengesetzt. Beachten Sie, daß Sie durch die Konstante vbCr quasi den Druck auf die Taste <Eingabe> ersetzen können. Für den Fall, daß die Zeilennummer ungültig ist, wird als Rückgabewert nur der Text „Nicht gefunden“ übergeben.

Hauptprogramm

Damit sind die einzelnen Teilaufgaben gelöst. Das Haupt-Makro (Programm) kann erstellt werden. Eine mögliche Lösung könnte etwa so aussehen:

```
Sub VertragFüllen()
   ' Wenn nötig Dokument mit Adressen öffnen, sonst auf
   ' vorhandenes Dokument zurückgreifen
   Set Vertrag = ActiveDocument
   DateiName = "WMAdressen.doc"
   Documents.Open FileName:=DateiName, Revert:=False
   Set Adressenliste = ActiveDocument
   ' Verkäufer suchen und eintragen
   KurzName = InputBox("Verkäufer?")
   ' Zeilennummer der Adresse suchen
   AdresseSuchen Dokument:=Adressenliste, SuchText:=KurzName,
_
      Zeile:=nZeile
   ' Zugehörige Adresse lesen
   AdresseHolen Dokument:=Adressenliste, Adresse:=Adresse, _
      Zeile:=nZeile
   ' Adresse für den Verkäufer einfügen
   Vertrag.Bookmarks("Verkäufer").Select
   Selection.InsertAfter Adresse
   ' Käufer suchen und eintragen
   KurzName = InputBox("Käufer?")
   ' Zeilennummer der Adresse suchen
   AdresseSuchen Dokument:=Adressenliste, SuchText:=KurzName,
_
      Zeile:=nZeile
   ' Zugehörige Adresse lesen
   AdresseHolen Dokument:=Adressenliste, Adresse:=Adresse, _
      Zeile:=nZeile
   ' Adresse für den Verkäufer einfügen
   Vertrag.Bookmarks("Käufer").Select
   Selection.InsertAfter Adresse
   ' Vertrags-Text wieder aktivieren
End Sub
```

In diesem Makro erkennen Sie den zuvor diskutierten Gesamtablauf wieder. Zunächst wird die Datei mit den Adressen geöffnet. Durch den Parameter Revert:=False wird erreicht, daß falls die Datei bereits geöffnet ist, die sonst übliche Abfrage entfällt und auf die bereits vorhandene Datei zurückgegriffen wird. Dann erfolgt die Abfrage des Kurznamens für den Verkäufer und anschließend das Suchen und Holen der Adresse mit den zuvor beschriebenen Unterprogrammen. Eingefügt wird die Adresse an der Position einer Textmarke, die bereits in der Vorlage definiert ist. Diese wird ausgewählt und dahinter wird die Adresse eingefügt. Der Vorgang wird dann für den Käufer wiederholt.

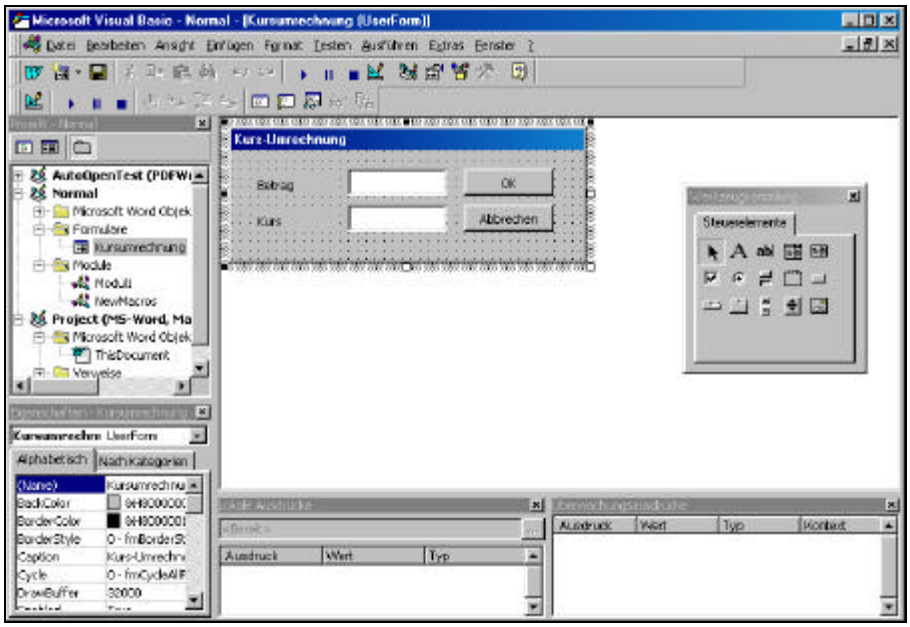
So haben Sie mit vier überschaubaren Makros die geforderte Aufgabenstellung gelöst. Hätten Sie alles in einem Makro unterbringen wollen, wäre dies zum einen erheblich länger geworden als die Summe der Einzellösungen ist und zum anderen wäre es wesentlich unübersichtlicher gewesen.

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 Word-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 Dialogelement 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 (cb).

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 (tb).

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 (pg).

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 (spn).

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

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|Aktivierreihenfolge“. Eine Dialogbox wird angezeigt, ordnen Sie die Elemente mit der Dialogbox 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) And _
            IsNumeric(.edtKurs) Then
            Kurs = .edtKurs
            If Kurs <> 0 Then
               DMBetrag = .edtBetrag
               Ergebnis = DMBetrag / Kurs
               Ergebnis = Format(Ergebnis, "#,##0.00")
            Else
               Ergebnis = "Nicht definiert"
            End If
         Else
            Ergebnis = "Ungültige Eingabe"
         End If
         MsgBox ("Der Benutzer hat auf OK geklickt." & _
            vbCr & "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 Dialogelement hat als wichtigste Eigenschaft ein Feld Namens Value. Über diese Eigenschaft können Sie den aktuellen Wert (in diesem Fall den enthaltenen Text) vorbesetzen oder auslesen. Dies geschieht dann z. B. mit einer entsprechenden Zuweisung.

```
Kursumrechnung.btnBetrag.Value = "100"
```

Das Feld Value ist die Standard-Eigenschaft eines Dialogelementes. Deshalb können Sie diese Anweisung auch in der folgenden Form verkürzen:

```
Kursumrechnung.btnBetrag = "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 Namen btnAbbrechen_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 btnAbbrechen_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 oder die Methode auch direkt aufrufen, dies zeigt das nächste Beispiel.

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:

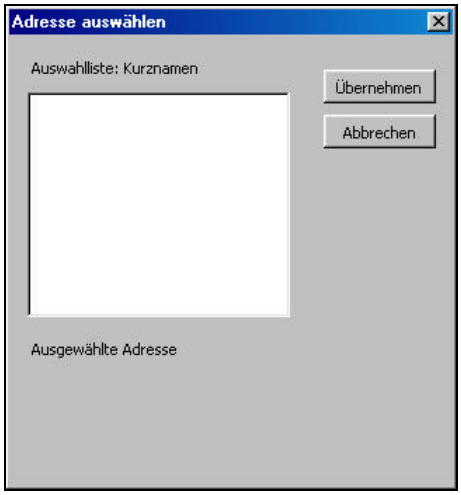
```
Private Sub btnOK_Click()
   btnOK.Tag = "OK"
   Hide
End Sub
```

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.

Ein weiteres Beispiel soll die Verwendung von Listenfeldern in einer Dialogbox zeigen. Entwerfen Sie eine Dialogbox in folgender Form:



In dem Bereich unter dem Text “Ausgewählte Adresse” ist ein leeres Textfeld mit dem Namen “txtAdresse”, die Liste bekommt den Namen “lstKurznamen”, die Aktionsschalter bekommen die Namen “btnÜbernehmen” und “btnAbbrechen”.

Aufgabe

In der Auswahlliste sollen die Kurznamen der verfügbaren Adressen angezeigt werden. Bei Auswahl einer Adresse in der Liste soll in dem freien Textfeld die vollständige ausgewählte Adresse angezeigt werden. Durch Klicken auf den Aktionsschalter Übernehmen soll die Auswahl bestätigt und die Adresse in den aktuellen Text an der Position der Markierung eingefügt werden. Wenn der Aktionsschalter Abbrechen geklickt wird, soll der Auswahl-Dialog geschlossen werden und keine weitere Aktion erfolgen.

Aufrufrahmen

Der Aufrufrahmen für diesen Dialog ist schnell erstellt. Sie brauchen den Dialog nur zur Anzeige zu bringen und nach erfolgreicher Auswahl aus dem Dialogelement “txtAdresse” den Text zu übernehmen und in das Dokument einzufügen.

```
Sub AdressenAuswählen()
   Set Dokument = ActiveDocument
   With AdressAuswahl
      .Show
      Dokument.Activate
      If .BtnÜbernehmen.Tag = "OK" Then
         Selection.InsertAfter Text:=.txtAdresse
      End If
   End With
End Sub
```

Das Muster für den Aufruf und die Anzeige der Dialogbox ist wie bei dem ersten Beispiel. Die ausgewählte Adresse kann aus dem Textfeld übernommen werden.

Initialisierung

Die wichtigste und wohl auch kompliziertes Teilaufgabe ist die Initialisierung der Userform mit dem Füllen der Dialogelemente insbesondere des Listenfeldes mit den Kurznamen. Hier können wir uns nochmal die bereits zu der Aufgabe mit den Unterprogrammen erstellten Teilmakros zu Nutze machen. Eine mögliche Lösung für die Initialisierung könnte etwa so aussehen:

```
Private Sub UserForm_Initialize()
   Documents.Open FileName:="WMAdressen.doc", Revert:=False
   Set Dokument = ActiveDocument
   With Dokument
      For nIndex = 2 To .Tables(1).Rows.Count
         Modul1.TabellenInhalt Inhalt:=KurzName, _
            Tabelle:=.Tables(1), Zeile:=nIndex, Spalte:=1
         lstKurznamen.AddItem (KurzName)
      Next nIndex
   End With
   lstKurznamen.Selected(0) = True
   Modul1.AdresseHolen Dokument, Adresse, 2
   txtAdresse = Adresse
End Sub
```

Hier erfolgt die Initialisierung in einer speziellen Ereignis-Prozedur der Userform. Sie hätten die Initialisierung auch in dem aufrufenden Makro machen können, aber da gehört meines Erachtens von der Logik her nicht hin. Sie gehört vielmehr zum Dialog selbst.

Das Gerüst für die Initialisierungs-Routine können Sie von Hand eingeben ober besser Sie erzeugen durch einen Doppelklick auf die Userform selbst automatisch einen Rahmen und ändern in dem so entstandenen Prozedurkopf den Begriff „Click“ in „Initialize“.

Zunächst muß die Adressenliste geöffnet werden. Dann wird mit Hilfe des Makros TabellenInhalt aus dem Modul1 die Liste gefüllt. Dazu benutzen Sie die Methode AddItem des Listenfeld-Objektes, diese Methode fügt einen einzelnen Eintrag der Liste hinzu.

Mit den letzten drei Anweisungen wird zunächst die Auswahl auf das erste Element der Liste gesetzt. Anschließend wird die zugehörige Adresse eingelesen und in dem leeren Textfeld dargestellt.

Auswahl

Wenn der Benutzer in der Liste durch Klicken mit der Maus die Auswahl verändert, dann muß auch die Anzeige in dem Textfeld mit der Adresse geändert werden. Dazu ist eine Ereignis-Prozedur für das Listenfeld zu erstellen. Diese könnte wie folgt aussehen:

```
Private Sub lstKurznamen_Click()
   Auswahl = lstKurznamen.ListIndex + 2
   Modul1.AdresseHolen Dokument, Adresse, Auswahl
   txtAdresse = Adresse
End Sub
```

Mit der Methode ListIndex kann der Index des aktuell ausgewählten Listenelementes ermittelt werden. Zu dem ermittelten Index muß eine 2 addiert werden, zum einen weil die Zählung in dem Listenfeld mit 0 beginnt und zum anderen weil unsere Tabelle mit der zweiten Zeile beginnt.

Anschließend wird mit dem Makro AdresseHolen die Adresse aus der Datei geholt und in dem Textfeld dargestellt.

Die Makros für die Aktionsschalter Abbrechen und Übernehmen sind so zu erstellen, wie die für Aktionsschalter Abbrechen und OK aus dem vorhergehenden Beispiel.

4.8.3 Verwendung von integrierten Dialogen

Neben der Möglichkeit eigene Dialogboxen zu entwerfen und diese in Makros zu verwenden, können Sie auch alle Dialogboxen von Word 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(wdDialogFileOpen).Show
```

Die ausgewählte Datei wird automatisch auch geöffnet.

Die Liste der verfügbaren eingebauten Dialoge und der zugehörigen Parameter finden Sie im Hilfesystem, wenn Sie im Index den Eintrag „Argumentlisten integrierter Dialogfelder“ auswählen, wird eine entsprechende Liste angezeigt.

Parameter

Aus der Auflistung ersehen Sie auch die Möglichkeit Parameter beim Aufruf anzugeben, wenn Sie dies möchten sollten Sie die Dialogbox-Objekte in der folgenden Form verwenden:

```
With Application.Dialogs(wdDialogFileOpen)
   .Name = "*.*"
   If .Show = -1 Then
      DateiName = .Name
   End If
End With
```

Sie können die in der Auflistung angegeben Parameter vor der Anzeige vorbesetzen oder nach der Anzeige auswerten. Mit der oben angegebenen Sequenz wird der Datei-Öffnen-Dialog zur Auswahl beliebiger Datei .Name=„*.*“ geöffnet, wenn der Benutzer die Auswahl erfolgreich abgeschlossen hat, liefert die Show-Methode als Funktionswert den Wert –1 zurück (für Abbrechen 0). In diesem Fall enthält die Eigenschaft .Name den Namen der geöffneten Datei.

Anzeigen

Die Methode Show zeigt den Dialog an und führt auch die damit verbundene Aktion aus. Im Beispiel des Datei-Öffnen-Dialoges wird die Datei auch automatisch geöffnet. Nun könnte es sein, daß Sie die Datei gar nicht öffnen möchten, sondern nur einen Dateinamens auswählen möchten. Für diesen Fall können Sie dann anstelle der Methode Show die Methode .Display verwenden.

```
If .Display = -1 Then
   DateiName = .Name
End If
```

Würden Sie diese Zeilen in dem obigen Beispiel verwenden, würde der Auswahl-Dialog zwar angezeigt, die Datei aber anschließend nicht geöffnet.

Ausführen

Eine weitere Möglichkeit ist die Methode .Execute. Durch Verwendung dieser Methode würde die mit dem Dialog verbundene Aktion zwar ausgeführt, der Dialog aber nicht angezeigt.

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. Sie können solche Funktions-Makros zum Beispiel für Berechnungen verwenden, um sich so eigene Rechenfunktionen zu definieren oder aber auch Prozedur-Makros, die nur einen Werte zurückgegeben als Funktionen definieren, sie lassen sich dann in dem meisten Fällen eleganter aufrufen.

Als Beispiel soll Ihnen zunächst eine Funktion für eine Berechnung gezeigt werden. Die Funktion MWSteuer liefert als Funktionswert für die als Parameter übergebene Zahl die Mehrwertsteuer zurück. Das nachfolgende Beispiel zeigt die Realisierung dieses Funktions-Makros:

```
Function MWSteuer(Betrag)
   MWSteuer = Betrag * 0.16
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.

Das nächste Beispiel zeigt wie das Makro zum Lesen des Inhaltes einer Tabellenzelle auch als Funktion hätte realisiert werden können.

```
Function FTabellenInhalt(Tabelle, Zeile, Spalte)
   ' Dieses Makro ermittelt den Inhalt einer Tabellen-Zelle
   Set Zelle = Tabelle.Cell(Zeile, Spalte).Range
   Zelle.MoveEnd Unit:=wdCharacter, Count:=-1
   FTabellenInhalt = Zelle.Text
End Function
```

Die Rückgabe des Ergebnisses erfolgt über den Funktionsnamen, der Parameter Inhalt kann deshalb entfallen. Wichtig ist, daß an mindestens einer Stelle in der Funktion der Funktionswert zugewiesen wird. Das folgende Beispiel zeigt, wie diese Funktion dann aufgerufen werden kann.

```
Sub FTesten()
   MsgBox (FTabellenInhalt(ActiveDocument.Tables(1), 1, 1))
End Sub
```

Dieses Makro zeigt in einem Meldungsfenster den Inhalt der ersten Zelle aus der ersten Tabelle des aktiven Dokumentes.

5 Fortgeschrittene 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 Word 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 Word 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 AutoExec, AutoOpen, AutoNew, AutoClose und AutoExit 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 AutoOpen 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 geben, ansonsten erhalten Sie beim Öffnen der Datei eine Fehlermeldung. Beachten Sie dazu das nachfolgende Beispiel:

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

Dieses Makro wird beim Öffnen der entsprechenden Datei automatisch ausgeführt und zeigt in einem Meldungsfenster den Text „Hello World“.

Die übrigen Automakros werden in folgenden Situationen aufgerufen:

AutoExec	Beim Starten von Word oder Laden einer globalen Vorlage (normal.dot)
AutoNew	Beim Erstellen eines neuen Dokuments
AutoClose	Beim Schließen eines Dokuments
AutoExit	Beim Beenden von Word oder Entladen einer globalen Vorlage

5.2 Zeitereignisse

Eine besondere Form von Ereignissen sind die Zeitereignisse, Ihnen kann mit der Methode OnTime eine Behandlungs-Routine zugewiesen werden. Damit besteht die Möglichkeit ein Makro mit einem Zeitereignis zu verbinden. Dazu das nachfolgende Beispiel:

```
Sub EreignisProzedurenInstallieren()
' Zeitereignisse verarbeiten
    Application.OnTime Now + TimeValue("00:00:15"), _
       "Modul1.HalloName"
End Sub
```

Mit dem 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 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 Word nutzbar gemacht. Die Funktion befindet sich in der Datei user32.dll (Lib-Anweisung) und hat 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 Word 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 der 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 System-Objekte

An dieser Stelle sei für Word noch auf die System-Objekte hingewiesen. Mit diesen Objekten ist es möglich auf Funktionen des Betriebssystems bzw. des Task-Managers zuzugreifen. In diesem Zusammenhang sind zwei Objekte zu nennen.

System

Das Objekt mit dem Namen System hat zahlreiche Eigenschaften mit denen Sie Informationen über den verwendeten Computer und das Betriebssystem erhalten können. So gibt es z. B. Eigenschaften wie .ComputerType, .OperatingSystem, oder auch .FreeDiskSpace und .Country. Mit besonderen Eigenschaften wie z. B. .Cursor können Sie auch Systemeigenschaften setzen.

```
System.Cursor = wdCursorWait
```

Diese Anweisung zeigt z. B. die Sanduhr als Mauszeiger. Mit den Eigenschaften .ProfileString bzw. .PrivateProfileString können Sie auf Einträge in der Registrier-Datenbank von Windows bzw. in INI-Dateien zurückgreifen. Erwähnenswert ist auch noch die Methode .Connect mit der Sie eine Verbindung zu einem Server bzw. Netzlaufwerk herstellen können.

Tasks

Eine weiteres interessantes Objekt ist die Task-Liste, die über den Namen Tasks angesprochen werden kann. Die folgende Sequenz zeigt die Anwendung dieses Objektes:

```
If Tasks.Exists("Rechner") Then
    With Tasks("Rechner")
        .Activate
        .WindowState = wdWindowStateNormal
    End With
Else
    Shell "calc.exe"
    Tasks("Rechner").WindowState = wdWindowStateNormal
End If
```

Dieses Makro prüft, ob eine Anwendung mit dem Namen „Rechner“ bereits ausgeführt wird. Ist dies der Fall, dann wird die Anwendung aktiviert. Wenn nicht wird Sie mit Hilfe der Funktion Shell gestartet. Es bestünde auch die Möglichkeit eine einzelne Task mit der Methode .Close zu schließen.

Eine weitere interessante Methode der Task-Liste ist die Methode .ExitWindows. Sie ermöglicht das Herunterfahren des Systems aus einem Word-Makros heraus. Änderungen an Word-Dateien sollten zuvor gespeichert werden, da Word direkt verlassen wird. Alle weiteren Anwendungen verhalten sich wie beim normalen Abmelden oder Herunterfahren.

5.5 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 es zwei Varianten. Die erste setzt voraus, daß Sie innerhalb eines jeden Unterprogramms für das Sie eine eigene Fehlerbehandlung benötigen, diese entsprechend deklarieren.

```
Sub Fehlerbehandlung()
   On Error GoTo Fehler
     ...
   Exit Sub
Fehler:
   MsgBox ("Fehlermeldung")
End Sub
```

Das Beispiel zeigt eine On Error-Konstruktion mit eigener Fehlerbehandlung. Durch die On Error-Konstruktion wird nun erreicht, das erstens eine 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
     ... (Anweisung, die eventuell zum Fehler führt)
   If Err.Number <> 0 Then
      MsgBox ("Fehlermeldung")
   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“.