EXCEL
FORMULARE UND VBA

Handbuch für Kurse und zum Selbststudium

Alexandra von Cube

September 2001

Inhaltsverzeichnis

Vorwort

Dieses Skript wurde für einen Kurs für den Fachbereich Maschinenbau - Lehrstuhl für Fertigungsvorbereitung - erstellt. Deshalb sind die Beispiele darauf zugeschnitten. Außerdem war nicht Vollständigkeit gefragt, sondern es sollte um die Formulare der Office-Anwendung Excel 2000 gehen. Grundlegende Programmierkenntnisse wie die Existenz von Variablen, ihre Datentypen wie z. B. String oder Integer, sowie Kontrollstrukturen wie IF/THEN/ELSE und ähnliches wurden vorausgesetzt.

Ein extra Dank geht an *StrgAltEntf* oder genauer *Schorsch Dabbeljuh* aus dem Spotlight-Forum, der mir lauter wertvolle Tipps gegeben hat.

Einleitung

Visual Basic for Application, so lautet der ausgeschriebene Name für die Makrosprache der Microsoft Anwendungen. Damit kann man häufig wiederkehrende Befehlsabfolgen automatisieren, Bedingungen einfügen, Schleifen durchlaufen lassen und vieles mehr. Der zugehörige Editor wird mit den Anwendungen mitgeliefert und hat inzwischen auch einen Debugger und andere Hilfen zur Erstellung von recht komplexen Programmen. Um den Editor und seine Funktionalität sowie grundlegende Strukturen eines Makros soll es in einem ersten Kapitel gehen. Das zweite Kapitel befasst sich mit der Logik der Objekte, die man in Excel manipuliert. Im Prinzip handelt es sich tatsächlich um eine Art objektorientierte Programmierung. Das dritte Kapitel widmet sich den Formularen und ihrer Beziehung zum Programmcode und zum Tabellenblatt, sowie der Modifikationen, die sich im Editor ergeben. Das vierte Kapitel schließlich will die Formulare mit ihren einzelnen Steuerelementen erklären. Und das fünfte Kapitel versucht, die gestellte Aufgabe umzusetzen

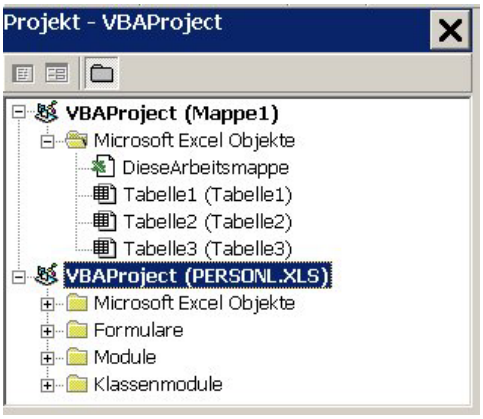
So, und nun viel Spaß ...

1 Der Editor und was man sonst noch ganz am Anfang wissen muss

1.1 Aufrufen und Einrichten

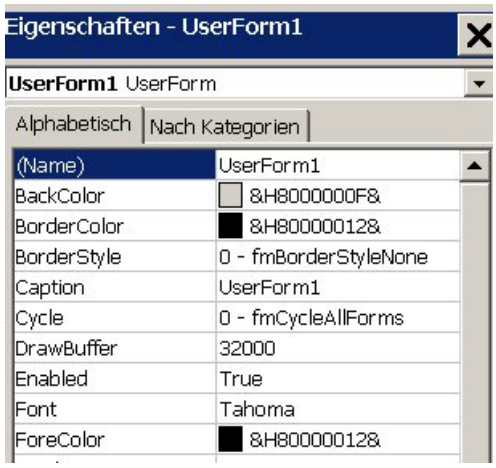
In allen Microsoft Anwendungen, also auch in Excel, kann man mit der Tastenkombination ALT + F11 den Editor aufrufen. Alternativ geht das auch mit der Befehlsfolge **Extras** ***Makro*** *Visual Basic Editor*. Es öffnet sich ein *eigenes Fenster*, das nicht Bestandteil des Excel-Bildschirms ist. Das Bild, das sich dann bietet, ist noch nicht besonders arbeitsfreundlich, deshalb sollte man etwas über die drei wichtigsten Teile der Fensters wissen.

Der ***Projekt-Explorer*** sitzt meistens oben links in der Ecke und zeigt wie in einem Verzeichnisbaum die geöffneten Dateien, wobei jede für ein Projekt steht. Unterhalb dieser Ebene sieht man die jeweils zugehörigen Objekte. Das können Module sein, Formulare, Klassenmodule oder auch Tabellenblätter.



In diesem Kurs interessieren nur die Formulare.

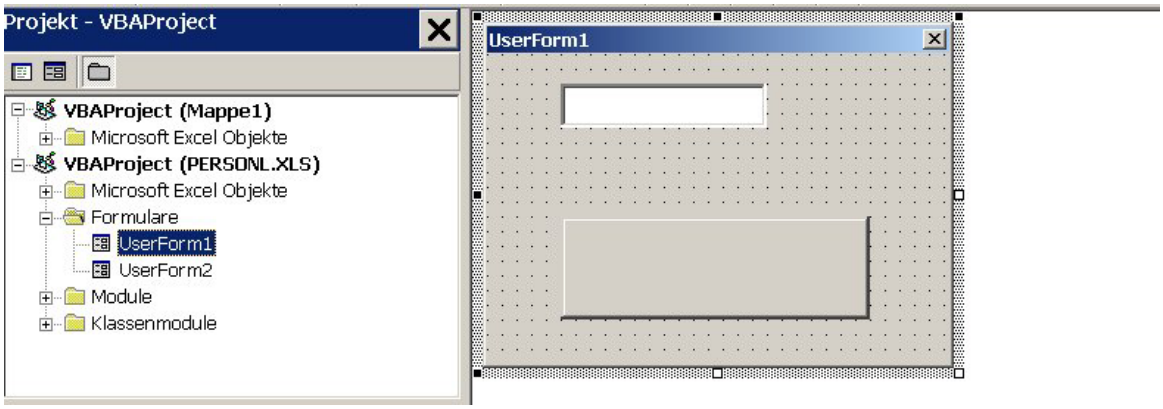
Das ***Eigenschaftsfenster*** sitzt direkt darunter und zeigt die verschiedenen einzustellenden Eigenschaften der jeweiligen Objekte an.



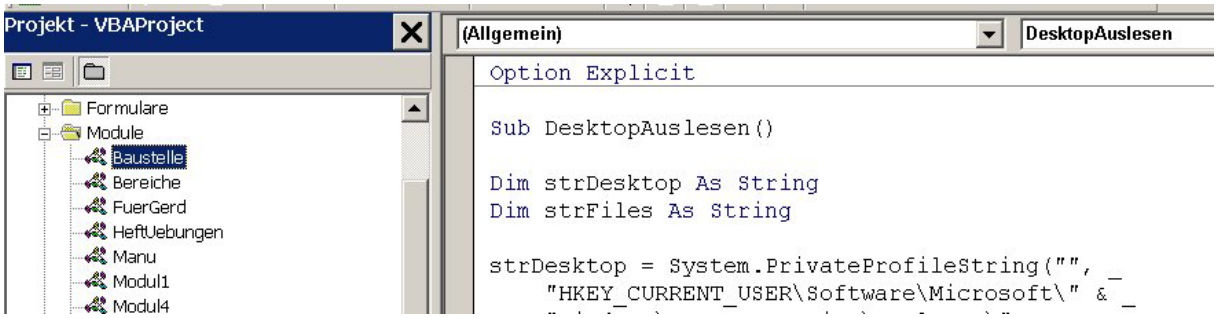
Die Graphik zeigt die Eigenschaften eines Formulars, wobei man hier mal das erste Fremdwort lernen sollte, nämlich USERFORM. So heißen diese Objekte in VBA.

Falls eines dieser beiden Fenster nicht zu sehen ist, kann man sie über **Ansicht** ***Projekt-Explorer*** bzw. ***Eigenschaftsfenster*** anschalten

Rechts davon, den größten Teil des Bildschirms einnehmend, findet das sogenannte ***Modulfenster*** seinen Platz. Je nach dem, was man als Objekt im Projektexplorer markiert hat, sieht man dort Programmcode oder ein Userform.

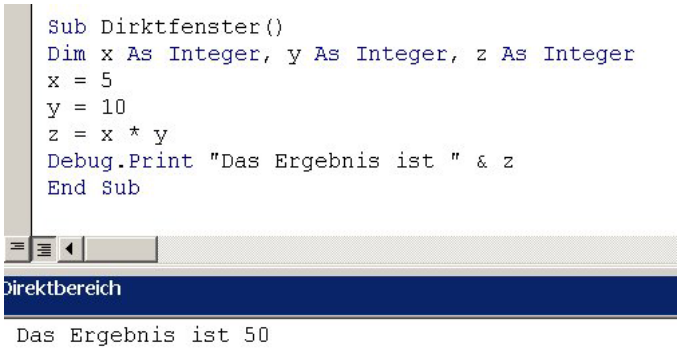


Bei diesem Beispiel ist links das UserForm1 markiert und rechts ist es als Graphik zu sehen.

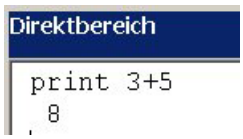


In dieser Ansicht ist links ein Modul markiert und rechts wird der dort abgelegte Code angezeigt und später dann natürlich auch eingegeben.

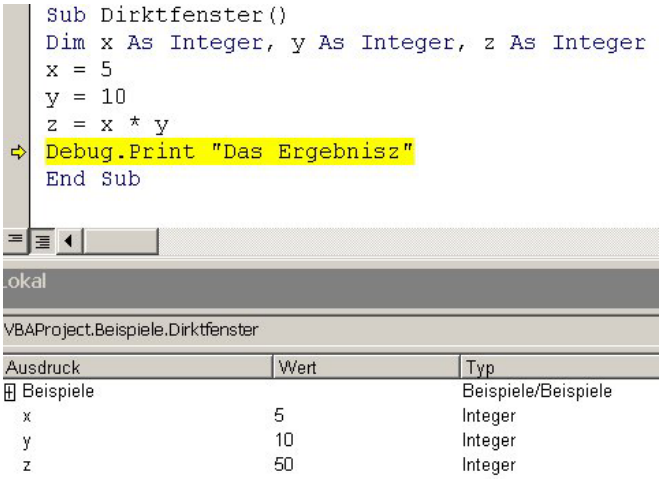
Zwei weitere Fenster sind noch interessant, allerdings hat man sie in den wenigsten Fällen immer eingeschaltet. Zum einen gibt es das ***Direktfenster***, in dem man mit `Debug.Print` Ergebnisse von Variablen ausdrucken...



...oder aber direkt Berechnungen ausführen kann:



Das ***Lokalfenster*** dient zur direkten Überprüfung von Variablen. Es liefert zu allen globalen und lokalen Variablen des aktuellen Moduls den Wert und den Datentyp, wenn Sie mit ***F8*** durch den Code hoppeln.



1.2 Öffnen, Speichern, Exportieren und Importieren

Programmcode und Formulare gehören immer zu bestimmten Dateien. Das kann eine Vorlage sein (*.xlt) oder aber auch eine normale Arbeitsmappe (*.xls) oder ein Excel-Addin (*.xla). Um an die Information heranzukommen und diese zu bearbeiten, muss die Datei geladen sein. Das geht allerdings nicht über den Editor, sondern man muss zu Excel wechseln (z. B. ganz schnell mit dem Symbol in der Buttonleiste) und dort mit **Datei *Öffnen*** arbeiten.
Eine besondere Datei in diesem Zusammenhang ist die Datei PERSONL.XLS. Sie liegt standardmäßig im Autostartverzeichnis von Excel und wird automatisch beim Starten mitgeladen, so dass Makros, die man dort programmiert, in jeder Lebenslage zur Verfügung stehen.

Speichern von Programmcode und Formularen geht im Editor selber mit dem Befehl **Datei *‚Dateiname' Speichern*** bzw. mit der Diskette in der Symbolleiste.

Muss man tatsächlich einmal Programmcode oder ein Formular unabhängig von der Datei transportieren, dann kann man mit dem Befehl **Datei *Exportieren*** das Objekt in eine *.bas (Basic), bzw. *.frm (Forms) Datei speichern. Diese Dateien kann man dann an anderer Stelle problemlos mit dem Befehl **Datei *Importieren*** wieder einlesen.

1.3 Formulare einfügen und löschen

Um mit einem Userform loszulegen braucht man ein neues Form. Mit dem Befehl **Einfügen *Userform*** wird ein jungfräuliches Objekt ohne alles erstellt. Aufpassen muss man dabei, welches Projekt (Datei) dabei im Projekt-Explorer markiert ist. Denn zu dieser Datei gehört das Form nachher.

Um ein Form wieder loszuwerden, muss man es markieren und dann leider mal nicht mit der ***Entf-Taste*** arbeiten, sondern die rechte Maustaste auf das Objekt anwenden und dort dann den Befehl ***Entfernen von UserForm1*** wählen. Sicherheitshalber fragt Excel nach, ob das

Objekt vor dem Löschen exportiert werden soll. Wenn man diese Frage verneint, wird das Form direkt gelöscht; ansonsten bekommt man die Möglichkeit zum Erstellen einer Basic-Datei.

1.4 Option Explicit und ein paar Einstellungen und Schreibweisen

Da es sich einfach gehört, sollte man Variablen immer deklarieren. Dies geschieht in VBA mit `DIM` und damit man es nicht vergisst, kann man unter **Extras *Optionen*** in der Registerkarte *Editor* ankreuzen, dass die Variablendeklaration erforderlich ist. In jedes Modul und jedes Formular wird dann ganz oben der Eintrag „Option Explicit“ geschrieben, der einen zu der Deklaration zwingt.

Zum ordentlichen Aufführen der Variablen gehört auch, dass man sich an ein paar Konventionen hält, was die Schreibweise der selben betrifft. Entsprechend der Datentypen sollte man Präfixe verwenden, an denen man sie sofort erkennen kann. Microsoft empfiehlt folgende Buchstabenkombinationen:

Variablentyp	*Präfix*
String	str
Integer	int
Objekt	obj
Boolean	bln
Byte	byt
Date/Time	dtm
Double	dbl
Error	err
Long	lng
Single	sng

Die verschiedenen Datentypen und ihre Bedeutung sollen hier kein Thema sein und werden unterstellt. Aber damit da nichts schief geht... Nur mit `DIM` werden die Variablen wirklich deklariert. Das Präfix dient der besseren Lesbar- und Erkennbarkeit.

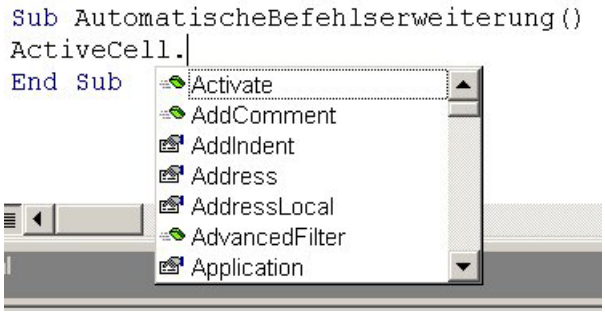
Die anderen Einträge des Befehls **Extras *Optionen***, die schon standardmäßig angekreuzt sind, beinhalten alle Hilfen für die Eingabe. Die kann man einfach so lassen. Auf der letzten Registerkarte sollte man darauf achten, dass die benutzten Fenster verankert werden.

Die **Benennung** der Prozeduren und Funktionen gehorcht im Prinzip immer der gleichen Logik: Sie beginnen mit `Sub` oder `Function` und einem Namen, der mit einem Klammerpaar abgeschlossen wird. Ob dort dann wiederum Werte stehen, hängt von der Art des Makros ab. Enden müssen die Programme mit `End Sub`, bzw. `Function`. Vorgestellte Bestimmung wie `Private` oder `Public` sagen etwas über die Reichweite der Variablen der Makros aus. Das führt aber hier zu weit. Außerdem sind die Prozedurnamen bei Formularen im Prinzip festgelegt.

Kommentare werden mit einem Apostroph eingeleitet und erscheinen in der Ansicht grün.

1.5 Die Automatische Befehlsergänzung

VBA ist nicht doof. Wenn man den Namen eines Objektes tippt und danach den Punkt setzt, zeigt es einem die möglichen weiteren „Befehle“. Das ist nützlich, da es so viele Objekte, Eigenschaften und Methoden gibt, dass man diese eh nicht im Kopf haben kann.

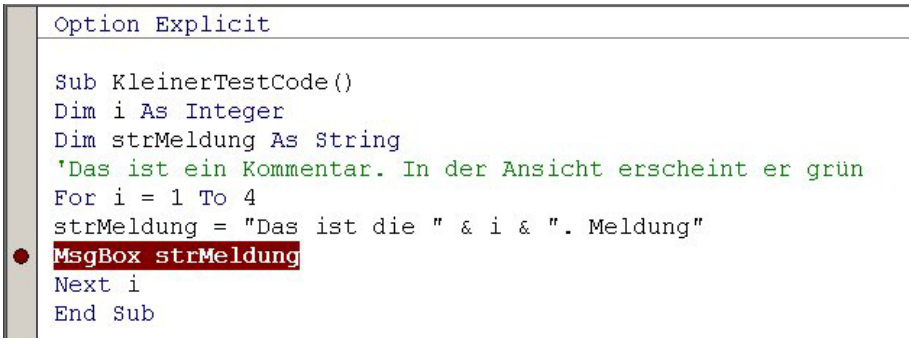


Aber anders als bei Word funktioniert das noch nicht perfekt. Sobald man z. B. ein Element einer Auflistung hat, versagt diese Hilfe. Manche Objekte werden auch stiefmütterlich behandelt: Selection z. B.

1.6 Starten von Code u. a. in der Editorumgebung

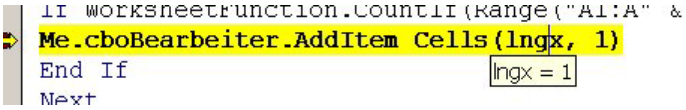
Steht man mit dem Cursor in den Befehlen, bzw. ist ein Userform markiert, kann man diese entweder mit ***F5*** oder dem Icon aus der Symbolleiste ▸ starten.

Um den Code zu verfolgen, kann man auch einen sogenannten Einzelschrittmodus benutzen. Mit ***F8*** hangelt man sich Schritt für Schritt durch das Programm. Ist das Makro länger und man weiß, dass man erst ab einer bestimmten Stelle in den Einzelschrittmodus wechseln möchte, dann setzt man in diese Codezeile mit der Maus einen Haltepunkt, indem man auf den Rand neben der Zeile klickt



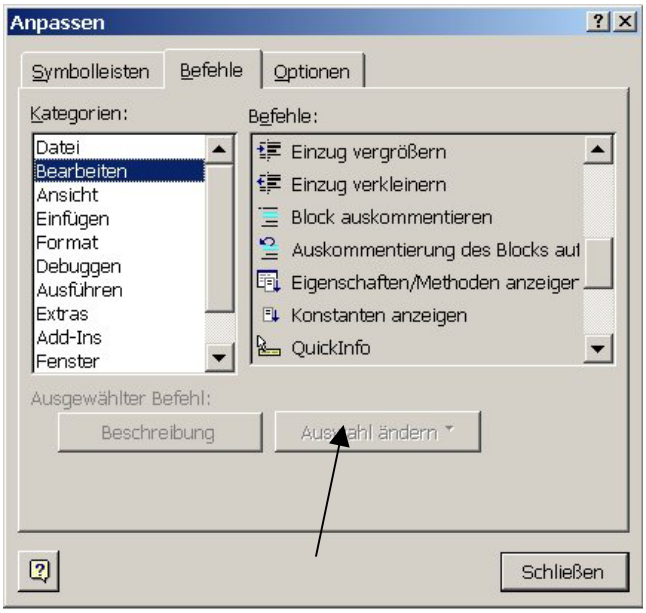
Die Zeile wird braun unterlegt und am Rand sieht man den Punkt. Der Code läuft bis zu dieser Stelle, stoppt dort und kann entweder mit ***F5*** bis zum Ende oder zum nächsten Haltepunkt oder aber mit ***F8*** im Einzelschrittmodus weitergeführt werden. Um einen solchen Stopp zu entfernen, klickt man einfach noch einmal dorthin.

Der Einzelschrittmodus hat noch einen großen Vorteil: fährt man mit der Maus über eine dargestellte Variable, so zeigt der TippText den Inhalt der selben an.

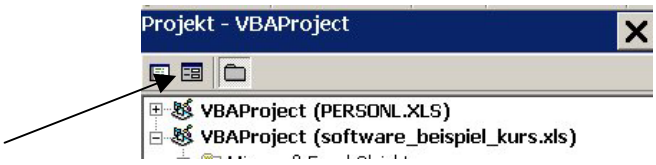


1.7 Anpassen der Symbolleiste

Um sich ein paar nützliche Buttons in die Symbolleiste zu integrieren, benutzt man den Befehl **Ansicht *Symbolleisten ->*** *Anpassen.*



Sehr praktisch sind die zwei Befehle, die sich in der Kategorie **Bearbeiten** befinden, nämlich ***Block auskommentieren*** und ***Auskommentierung des Blocks aufheben***. Sie dienen dazu, Zeilen als Kommentare zu kennzeichnen. Mit gedrückter linker Maustaste schiebt man sie nacheinander einfach in die Symbolleiste. Mit dem dann aktiven Button ***Auswahl ändern*** kann man ihr Aussehen bestimmen. In der Kategorie **Ansicht** findet man die beiden Befehle ***Code*** und ***Objekt***. Da man diese bei Formularen braucht, sollte man sie auch in die Symbolleiste ziehen. Allerdings sind sie auch Bestandteil des Projekt-Explorers. Es ist also mehr eine Geschmackssache.

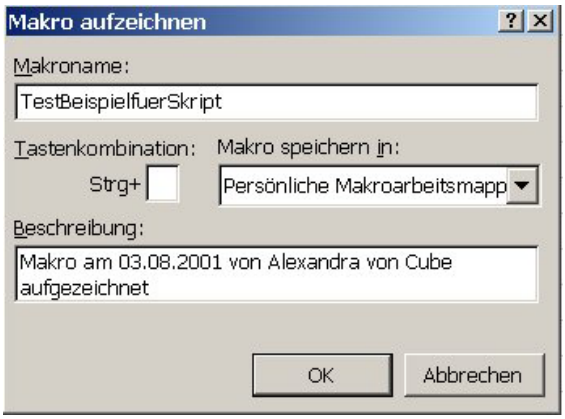


Außerdem kann man mit diesem Befehl Makros in der Schaltfläche der Anwendung integrieren. Dazu mehr auf Seite 16 in dem Kapitel **„Ein neues Formular..... starten“.**

1.8 Makros aufnehmen

Nicht immer aber manchmal ist es praktisch, den Befehl **Extras *Makro*** *Aufzeichnen* zu benutzen. Wie bei einem Videorekorder schreibt Excel dann alle Befehle mit, die man in dieser Zeit ausführt. Will man z. B. wissen, wie in VBA die aktive Zelle mit der Zahl 1234 gefüllt wird, wenn diese im Datumsformat formatiert ist, kann man das mit dem Rekorder herausbekommen:

Zuerst wird man gefragt, wie das Makro heißen soll, welche Beschreibung es bekommt und wo es gespeichert werden soll. Auch eine Tastenkombination kann man an diese Stelle vergeben.



Wird es in der „Persönlichen Markoarbeitsmappe“ gespeichert, dann hat man das Makro immer zur Hand.

Wenn man fertig ist, wird die Aufnahme beendet:

Und so sieht dann der fertig Code, den Excel in ein neues Modul einfügt aus:

```
Sub TestBeispielfuerSkript()
'
' TestBeispielfuerSkript Makro
' Makro am 03.08.2001 von Alexandra von Cube aufgezeichnet
'

'
    ActiveCell.FormulaR1C1 = "5/18/1903  12:00:00 AM"
    Range("A2").Select
End Sub
```

Nützlich ist dieses Verfahren, wenn man den Namen bestimmter Objekte, Eigenschaften oder Methoden nicht weiß. So bekommt man sie ohne viel Suchen heraus.

2 Objekte, Eigenschaften und Methoden

2.1 Allgemeines

Ganz kurz etwas sehr Prinzipielles zu der Art und Weise, wie in VBA programmiert und gedacht wird. Jedes Teil des Programms, des Tabellenblattes eines Formulars und so weiter wird als ***Objekt*** definiert, das bestimmte ***Eigenschaften*** hat und dem bestimmte ***Methoden*** zugeordnet sind. Das ist am Anfang etwas verwirrend und deshalb vielleicht mit einem Beispiel leichter zu verstehen:

Ein Auto ist unser ***Objekt***. Das ist noch einfach ;-). Dieses Auto kann nun weiß oder blau, grau oder grün... sein. Das ist eine **Eigenschaft**. Diese Eigenschaft ist natürlich nicht nur für dieses Auto vorhanden, sondern auch die Sitzbezüge können wieder die gleichen Farben haben. Es gibt also die Eigenschaft unabhängig von dem bestimmten Objekt. Um diese zusammenzubringen, werden die beiden Teile der Information mit einem Punkt verbunden, etwas so `Auto(„FiatPunto).Farbe` **`Farbart:`**`= „metallic“,` **`Farbton:`**`=rot` . Die Parameter, die mit Doppelpunkt und Gleichheitszeichen angeschlossen werden, geben dann den Inhalt der Eigenschaften wieder.

Ein Sitz ist auch ein Objekt. Da es aber ein Teil des Autos ist, wird dem auch Rechnung getragen, indem die Schreibweise so lautet:

```
Auto(„FiatPunto“).Sitz(„Fahrersitz“).Farbe Farbart:=“glänzend“, _
Farbton:=“grün.“
```

In der Umgangssprache für Excel könnte das so lauten:

```
Datei(„Beispiel“).Arbeitsblatt(„Datensammlung“).Zelle(„A1“).Schrift _
Schriftart= „Tahoma“
```

Oder wirklich in Excel:

```
Workbooks("Beispiel.xls").Worksheets("Datensammlung").Range("A1"). _
Font.Name = "Tahoma"
```

Leider stimmt das Beispiel nicht so ganz, weil sich Microsoft da nicht ganz konsequent verhält. Die Unterabteilung von Font sind zwar lauter Eigenschaften. Sie können aber nicht mit Kommas und dem „:=“ aneinander gereiht werden, sondern müssen tatsächlich einzeln gesetzt werden.

Ähnlich funktioniert es mit den Methoden, die ein Objekt oder Unterobjekt beherrscht. Ein Auto kann z. B. `fahren` oder `hupen` oder `stehen` oder `bremsen`.... Ein Sitz kann `quietschen` oder `hochklappen`... Auch hier verbindet VBA die Einzelnen Objekte mit ihren Methoden durch einen Punkt.

```
Auto(„FiatPunto“).Sitz(„Fahrersitz“).hochklappen
```

Umgangssprachlich:

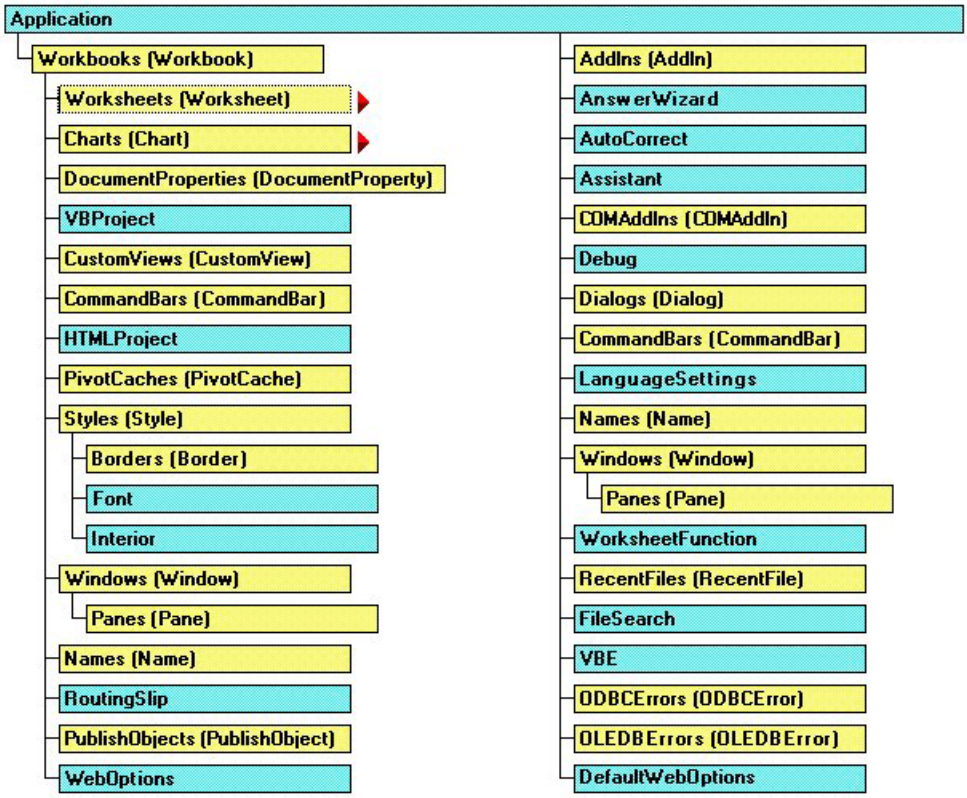
```
Datei(„Beispiel“).Speichern
```

Und wirklich in Excel:

```
Workbooks("Beispiel.xls").Save
```

2.2 Das Objektmodell in der Hilfe

Die Hilfe von Excel in VBA ist so lala. Wenn man schon ein bisschen was weiß, kann man damit durchaus zurechtkommen. Aufgerufen wird sie mit **? *Microsoft VisualBasic Hilfe*** oder mit ***F1***. Unter dem Stichwort Excel Objekt-Modell findet man folgende Übersicht:



Damit kann man sich zu diversen Objekten durchhangeln. Wenn man das Objekt schon kennt und nur wissen will, welche Methoden und Eigenschaften dazugehören, kann man auch in das schon getippte Word den Cursor stellen und ***F1*** drücken. Die Hilfe funktioniert dann kontext-sensitiv.

2.3 Auflistungen

Manche Objekte treten vornehmlich im Rudel auf. So können z. B. mehrere Dateien in Excel offen sein, oder aber eine Datei hat mehrere Tabellenblätter. Je nach dem, ob man alle Objekte der gleichen Art auf einmal ansprechen möchte oder nur eines, muss man Excel das mitteilen.

```
ActiveWorkbook.Worksheets.Delete
```

löscht auf jeden Fall mehr als

```
ActiveWorkbook.Worksheets(„Tabelle1“).Delete
```

Also, wenn man ein bestimmtes Objekt unter gleichartigen meint, dann braucht man einen Namen für dieses Objekt, der in Klammern und Anführungszeichen gesetzt wird. Oder aber man schreibt eine Zahl (Index) in die Klammer, da den Objekten intern eine Nummer zugeordnet ist.

```
ActiveWorkbook.Worksheets.Count
```

Gibt z. B. die Anzahl aller Arbeitsblätter der aktiven Datei zurück. Diesen Befehl gibt es nicht bezogen auf ein bestimmtes Tabellenblatt. Das wäre auch unsinnig.

2.4 Ein paar wichtige Objekte und Eigenschaften für den Hausgebrauch

2.4.1 Application

Application ist die Anwendung selber. Diese gibt es immer nur einmal. Deshalb wird sie normalerweise nicht mit aufgeschrieben.

`Application.ActiveWorkbook.ReadOnly` und

`ActiveWorkbook.ReadOnly`

ist also das gleiche. Nicht desto trotz sollte man wegen der Logik wissen, wie das höchste hierarchische Objekt heisst.

2.4.2 Workbooks

Workbook heißen die in Excel geöffneten Dateien. Da es davon wieder mehrere geben kann, handelt es sich um eine Auflistung. Die nähere Bestimmung eines einzelnen Objekts wird über den Dateinamen gehandelt. Da man häufig mit der aktiven Datei arbeiten möchte gibt es das eigene Objekt *ActiveWorkbook*. Das ist logischer Weise keine Auflistung, da es davon immer nur eines geben kann. *ActiveWorkbook* wird auch häufig weggelassen, da es das unterstellte Objekt ist, wenn nicht anderes gesagt wird.

```
ActiveWorkbook.Worksheets("Tabelle1").Range("A1").Font.Size = 14
```

```
Worksheets("Tabelle1").Range("A1").Font.Size = 14
```

sind also identisch.

2.4.3 Worksheets

Das Objekt *Worksheets* spricht die einzelnen Arbeitsblätter einer Datei an. Die Benennung bei einzelnen Elementen der Auflistung besteht aus dem Namen des Tabellenblatts, oder aus einem Index. Beim Index ist die Reihenfolge der Tabellenblätter in der Datei ausschlaggebend. Auch hier gibt es wieder ein abgeleitetes Objekt mit Name *ActiveSheet*, das das aktuelle Tabellenblatt bezeichnet. Und auch hier wieder die Logik: Wenn kein Tabellenblatt angegeben wird, dann gilt das Objekt *ActiveSheet* als unterstellt.

`ActiveSheet.Range("A1").Font.Size = 14` und

`Range("A1").Font.Size = 14`

ist also mal wieder das Gleiche.

2.4.4 Range oder wie spreche ich Zellen an

Range ist das englische Wort für Bereich und entspricht bei Tabellenblättern den Zellen (bei Diagrammen natürlich etwas anderem). Damit klar ist, welche Zelle gemeint ist, muss die Adresse in Klammern dahinter gesetzt werden. Dabei kann man auch mit dem Doppelpunkt arbeiten (von... bis) oder statt des Semikolon mit dem Komma (Einzelzellen).

```
Sub DasRangeObjekt()
Range("A1:B5").ClearFormats
Range("A1,B2,C3").ClearContents
End Sub
```

Die A1-Schreibweise ist bei dem Objekt zwingend. Möchte man die Z1S1-Schreibweise benutzen, muss man dem Range-Objekt die Eigenschaft Cells hinzufügen.

```
Cells(1, 1).Font.Bold = True
```

formatiert die Zelle A1 fett.

```
Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```

formatiert den Bereich A1 bis C5 kursiv.

`Cells(Reihe, Spalte)` gibt immer eine einzelne Zelle zurück. `Range(Cells(Reihe, Spalte), Cells(Reihe, Spalte))` gibt einen Bereich zurück. Diese Schreibweise ist nützlich für Erstellen von Schleifen, weil man mit einem Zähler ganze Bereiche durchpflügen kann.

Zum Schluss sei auch noch auf die Variante:

```
[a1:b5].Select
```

hingewiesen. Diese Kurzschreibweise, bei der das Wort RANGE und die Anführungszeichen wegfallen ist aber nicht unbedingt empfehlenswert, da sie wohl nicht in allen Lebenslagen von Excel richtig interpretiert wird.

Auch in diesem Bereich gibt es wieder ein eigenes Objekt, das sich auf die momentan aktive Zelle bezieht, welchen `ActiveCell` heisst.

2.4.5 Das Markieren und Aktivieren von Zellen

Mit `BEREICH.SELECT` kann man da nicht viel falsch machen, wobei es egal ist, wie man den Bereich beschreibt.

```
Sub Zellen2()
'A1 bis B2 markieren
Range("a1:b2").Select
End Sub
```

Oder:

```
Sub Zellen3()
'A1 bis B2 markieren
Range(Cells(1, 1), Cells(2, 2)).Select
End Sub
```

Hier wird ein Bereich markiert und darin eine bestimmte Zelle aktiviert.

```
Sub Zellen4()
'A1 bis B2 markieren und Zelle A2 aktivieren
Range("a1:b2").Select
Cells(2, 1).Activate
End Sub
```

Alles markieren:

```
Sub Markieren1()
'Alles markieren im aktiven Arbeitsbatt
Cells.Select
End Sub
```

Zeilen oder Spalten markieren:

```
End Sub
```

```
Sub SpaltenUndZeilen1()
'Auch Zeilen und Spalten sind Eigenschaften ☹
Rows(1).Select
Columns(3).Select
End Sub
```

Außerdem gibt es noch die Selection-Eigenschaft, die einen markierten Bereich umfasst:

```
Sub Markierung1()
'Die Markierung selber ist eine Eigentschaft
'Das Objekt kann man weglassen...
Selection.Copy
MsgBox "Das markierte Objekt ist ein " & TypeName(Selection)
End Sub
```

Oder so:

```
Sub Markierung2()
'Formate löschen
Selection.ClearFormats
End Sub

Sub Markierung3()
'Alles Löschen
Selection.Clear
End Sub
```

Leider ist Selection eines der Schlüsselworte, die die automatische Befehlsergänzung nicht unterstützen. Man kann das ein bisschen austricksen, wenn man kurzfristig stattdessen `ActiveCell` in seinen Code schreibt, weil sehr viele Befehle gleich sind und hier die Unterstützung funktioniert.

3 Formulare, die Oberfläche für die Datenverwaltung

3.1 Ein neues Formular...

Formulare werden in der VBA-Umgebung erstellt. Nur dort hat man das ganze Instrumentarium zur Verfügung. Wie das geht, nämlich mit **Einfügen *Userform,*** wurde schon kurz in Kapitel 1.3 erläutert. Mehr muss man da auch nicht wissen.

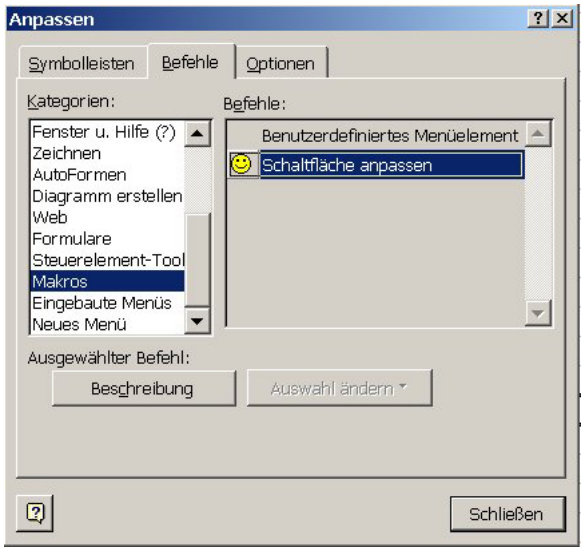
3.1.1 ... starten

Kurze Erinnerung an Kapitel *1.6 Starten von Code u. a. in der Editorumgebung*: Wenn das leere Formular im VBA-Editor markiert ist und man ***F5*** drückt, dann wird ein leeres Formular auf den Bildschirm geschickt. Logo. Oder aber man drückt auf . Diese beiden Methoden gelten allerdings eben nur für das Starten aus dem VBA-Editor. Wenn man sein Formular aus dem Tabellenblatt benutzen möchte, dann kann man damit nichts anfangen. In diesem Fall fügt man ein Modul ein, nennt es am besten FormsStarten und erstellt ein Makro, dass so aussieht:

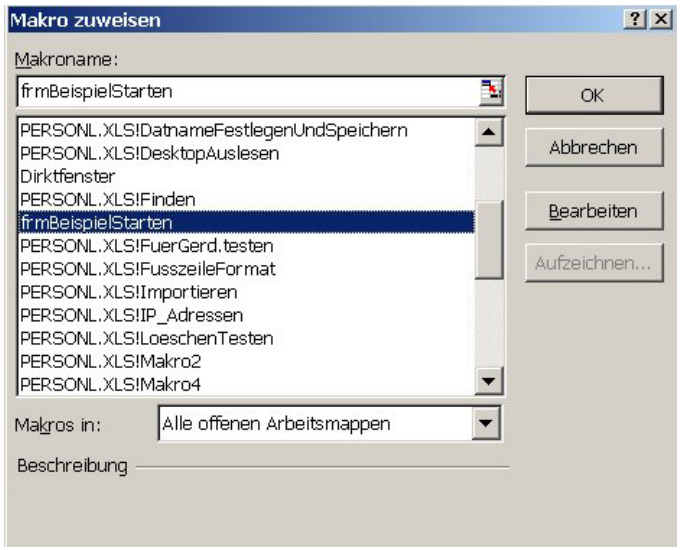
```
Sub usrBeispielStarten()
usrBeispiel.Show
End Sub
```

Dieses Makro kann man sich dann mit dem Befehl **Extras *Anpassen*** *Makros* auf einen Button legen. Allerdings geht das nicht ganz so komfortabel wie bei internen Befehlen von Excel.

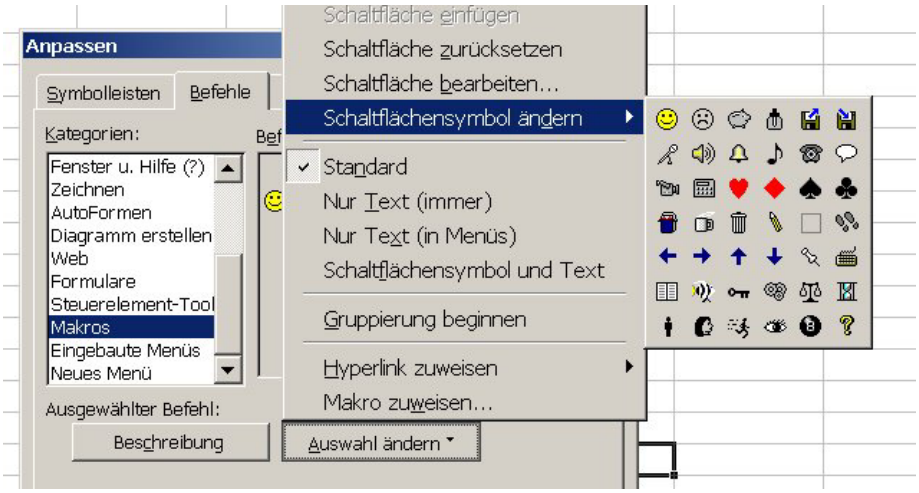
Wenn man in der linken Spalte den Ausdruck „Makros“ anklickt, dann kann man mit gedrückter linker Maustaste den Smiley, der rechts erscheint, packen und ihn oben in die Symbolfläche schieben.



Beim ersten Klick auf das neue Symbol, wird man von Excel gefragt, welches Makro man denn eigentlich starten will:



Da ist man dann nicht kleinlich und sagt es Excel ;-). In einem dritten Schritt kann man dann das Symbol ändern. Dazu braucht man wieder den Befehl **Extras** ***Anpassen***. Ist das entsprechende Fenster geöffnet, dann markiert man sein Symbol und kann nun über den Button ***„Auswahl ändern“*** einiges anpassen.



3.1.2 .... schließen

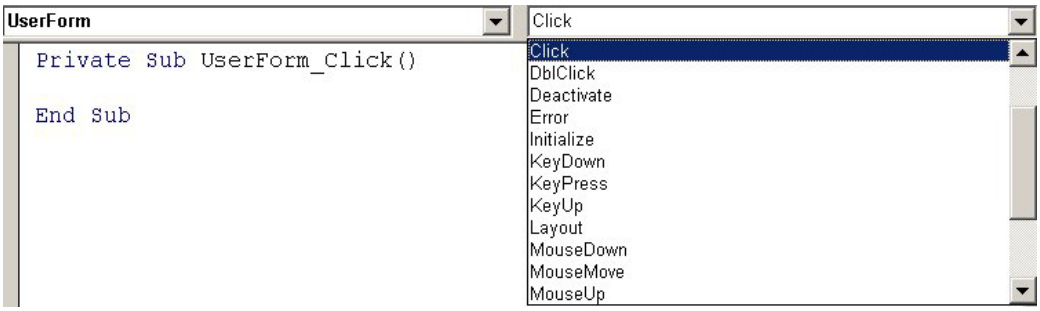
Man kann das neue Formular ohne einen einzigen eigenen Button schon schließen, nämlich mit dem Kreuz rechts oben, das Excel jedem leeren Formular mitgibt. Oder aber man benutzt die Tastenkombination ***ALT*** + ***F4***, die hier, wie bei vielen anderen Windows-Fenstern auch funktioniert.

3.2 Wo verstecken sich die Makros?

Bei einem Formular ist der Witz natürlich, dass man es nicht nur startet und beendet, sondern dass man dort Aktionen auslösen kann, z. B. indem man auf einen Button drückt. Das geht, indem man den einzelnen Objekten des Formulars und dem Formular selber auch bestimmte Makros zuordnet. Diese gehören tatsächlich zu diesem bestimmten Userform und sind in einem Modul zu finden. Und hier wird jetzt unsere Anpassungsorgie vom Beginn verständlich.

Die beiden Buttons ***„Code“*** und ***„Objekt“*** wechseln zwischen der Oberfläche des Formulars und den dahinter liegenden Makros hin und her.

In der Objektansicht sieht man all die Teile, aus denen man sich sein Formular zusammenstellt, in der Code-Ansicht kann man sich dann die verschiedenen Ereignisse raussuchen, die so einem Objekt zustoßen können und die dann bestimmte Reaktionen auslösen. Dazu sucht man in der linken Liste das Objekt heraus und bekommt dann in der rechten Liste die verschiedenen Ereignisse angezeigt.



Bei dem obigen Beispiel ist es das UserForm-Objekt selber, also das aktuelle Formular. Und wie man rechts sieht, sind einige Ereignisse daran geknüpft. Damit sind auch die Makronamen festgelegt, da VBA daran erkennt, welches Ereignis den Code auslösen soll. In unserem Falle heißt es UserForm_Click. Das kann man sich eigentlich ganz gut vorstellen ;-).

4 Steuerelemente der Formulare

Es gibt eine Menge Steuerelemente, auch Controls genannt, die auf Formularen Platz finden. Damit gibt man Text und Zahlen ein, schickt Befehle los oder lässt sich Ergebnisse ausgeben. Um sie auf den Formularen einzufügen, ist es am einfachsten, wenn man die Steuerelemente-Toolbox einblendet. Entweder wird sie schon aktiviert, wenn man ein Formular in der Bearbeitungsansicht (Objektansicht) markiert, oder aber man muss auf klicken, damit sie angeht.

Die allgemeine Logik ist immer die gleiche. Man klickt mit der Maus auf das Objekt, das man braucht und mit gedrückter linker Maustaste, zieht man für es einen Rahmen auf dem Userform. Danach gibt man dem Teil einen ordentlichen Namen im Eigenschaftsfenster.

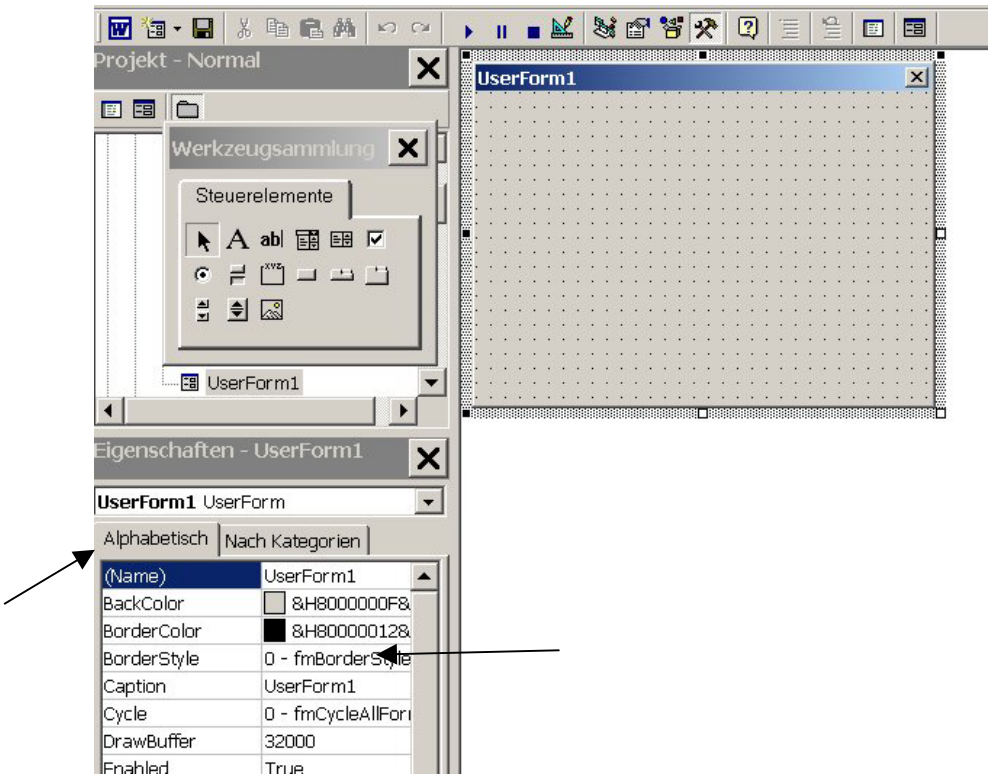
Auch hier bietet Microsoft wieder eine Konvention an, wie man diese benennen sollte:

Control-Typ	*Präfix*	*Symbol*	*Funktion*
CheckBox	chk		Kontrollkästchen
ComboBox	cbo		Auswahlliste
Command Button	cmd		Befehlsknopf
Frame	fra		Rahmen (für Optionsfelder)
Image	img		Graphikrahmen
Label	lbl		Beschriftungsfeld
ListBox	lst		Listenfeld
Multipage Page	mlp pg		Dialogunterteilung
Optionsfeld	opt		Optionsfeld
Registerkarte	tab		Dialogunterteilung
ScrollBar	(h/v)sb		Rollbalken
Spin	spn		Drehknopf zum Hörerstellen von Zahlen
TextBox	txt		Textfeld

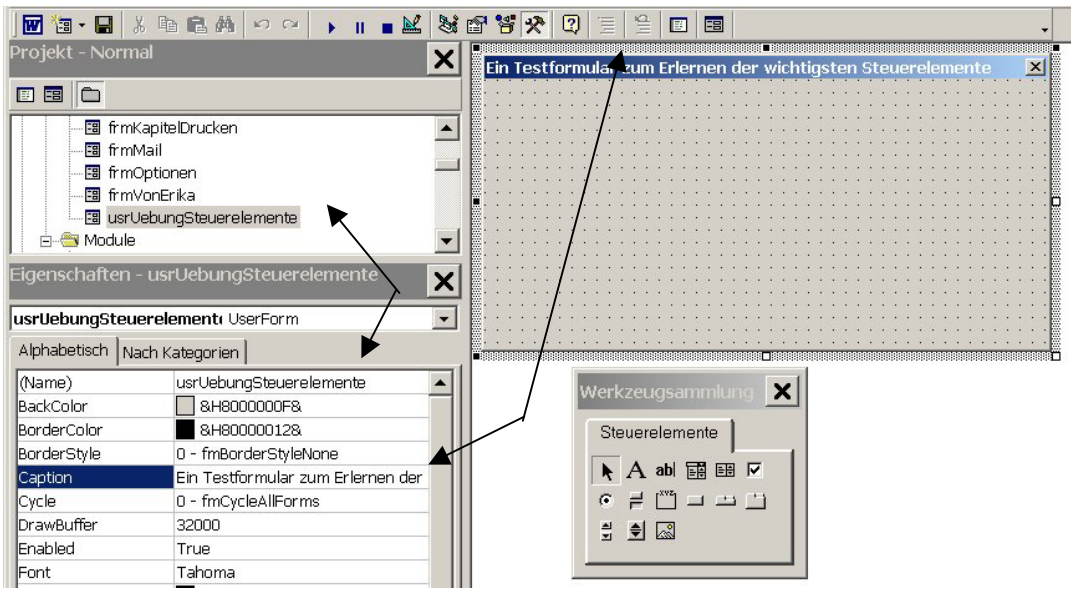
Im Folgenden sollen die wichtigsten Steuerelemente an Hand von Beispielen erläutert werden.

4.1 Das Userform selber

Wenn man ein neues braucht, dann benutzt man den Befehl **Einfügen *Userform***. Das ist ja ziemlich einfach.

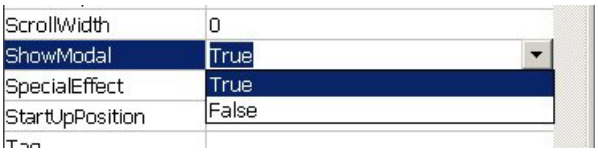


Nach dem Einfügen sollte man es direkt umbenennen, weil „UserForm1“ als Name sicher nicht der Renner ist. Bei der Benennung von Formularen unterscheidet man zwei Begriffe, nämlich *Caption*, was für die Beschriftung in der blauen Leiste steht und *(Name),* womit der programminterne Name des Objekts gemeint ist. Und hier hat sich das Präfix ***usr*** eingebürgert. Wenn man also in unserem Fall als Name ***usrUebungSteuerelemente*** eingibt, dann ändert sich die Beschriftung dadurch noch nicht. Dazu muss man bei *Caption* den Text ändern und dort dann hinschreiben „Ein Testformular zum Erlernen der wichtigsten Steuerelemente“.



Da am Anfang das Formular zu schmal ist, um den ganzen Titel aufzunehmen, wird er mit drei Pünktchen abgekürzt, aber mit der Maus kann man das Formular einfach verbreitern. Um das Formular zu starten, erinnere man sich an die Lektion „Ein neues Formular... ... starten“ auf Seite 16.

Seit Excel 2000 kann man in den Eigenschaften des Formulars angeben, ob man es so starten möchte, dass man vor Schließen des selben nichts im Arbeitsblatt machen kann, oder aber gleichzeitig in der Tabelle und im Formular Einträge erlaubt sind. Die Eigenschaft nennt sich „Showmodal“ und kann auf **True** oder **False** stehen. Bei **True** kann man nicht ins Arbeitsblatt wechseln, bei **False** ist es erlaubt.

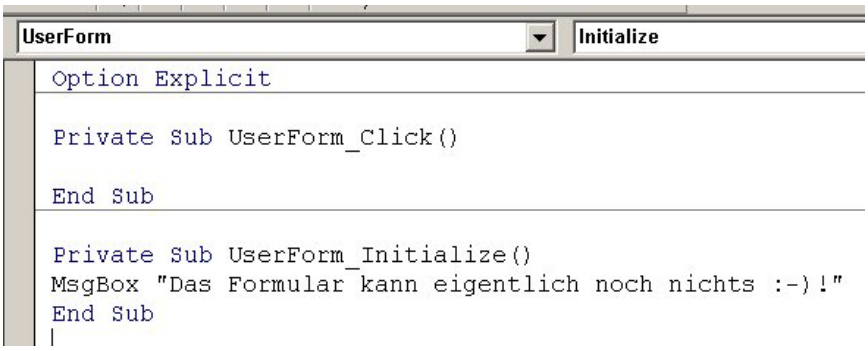


4.1.1 Das Ereignis „Initialize“...

Mit dem Starten des Formulars kann man ein Ereignis verbinden. Im Normalfall werden damit die Combo- und Listboxen gefüllt. Aber man kann auch z. B. eine Nachrichtenmeldung damit losschicken.

Um überhaupt an das Ereignis heranzukommen und den Code eingeben zu können, müssen wir in das Codefenster wechseln (Seite 17)

Eines der wichtigsten Ereignisse ist sicherlich das **„Initialize“-Ereigniss,** das eintritt, wenn ein Formular geladen wird. Zu diesem Zeitpunkt kann dann mit diesem Befehl die Auswahllisten und ähnliches des Formulars füllen. Zum Testen kann man jetzt einfach mal ein Meldungsfenster an die Menschheit schicken.

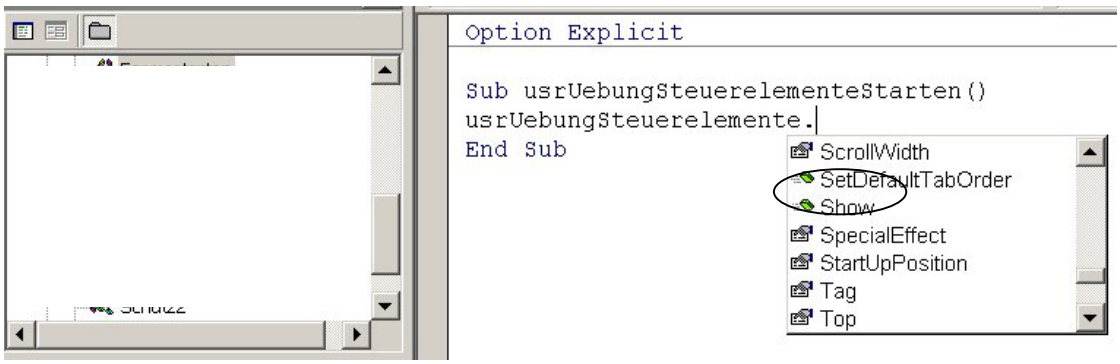


Wenn man das nun losschickt mit ***F5*** z. B., dann kommt erst das Meldungsfenster und dann das leere Formular. Schließen kann man das oben rechts in der Ecke mit dem Kreuz (siehe Seite 17).

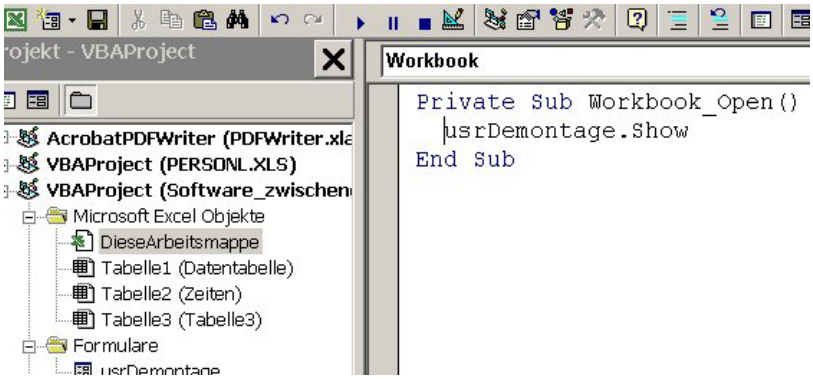
4.1.2 ...und wie man es überhaupt auslöst

Wie schon besprochen, wird ein Formular wird aus einem normalen Modul heraus gestartet. Dazu braucht man den Befehl **Einfügen *Modul***. Als Übung sollte man das jetzt ruhig noch einmal machen. Rechts im Projekt-Explorer erscheint das Teil mit dem schicken Namen *Modul1*. Auch hier sollte man sofort Hand anlegen und das Modul umbenennen. Eines könnte man z. B. *FormsStarten* nennen. Der Code ist denkbar einfach. Man gibt der Sub einen Na-

men – am Besten den des Formulars, das man starten möchte -, und `Objektname.Show` lässt die Sache auf dem Bildschirm erscheinen:



Dieses Modul kann man sich nun auf eine Button legen (siehe Seite 9) oder aber mit einem speziellen Namen belegen, so dass es automatisch beim Starten der Arbeitsmappe ausgeführt wird. Dazu muss man ein Modul in dem Objekt „Diese Arbeitsmappe“ mit dem Namen **`Private Sub Workbook_Open()`** erstellen.



4.1.3 Das Ereignis Click

Ist das Userform auf dem Bildschirm sichtbar, dann gibt es auch hier wieder verschiedene Ereignisse. Das Click-Ereignis tritt beim Daraufklicken ein, also eigentlich dauernd.

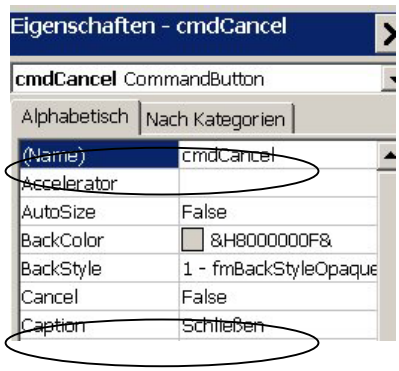
```
Private Sub UserForm_Click()
 MsgBox "Nicht!!! Ich bin kitzelig!!!"
End Sub
```

Damit hat man garantiert alle Lacher auf seiner Seite, aber es nicht unbedingt nützlich.

4.2 Die Befehlsschaltfläche

Ein solcher Button kann für Vieles benutzt werden, z. B. um das Formular elegant zu schließen, um eine Liste neu einlesen zu lassen, um Bedingungen ausführen zu lassen, um Eingaben in die Tabelle zu übernehmen und und und.... Das Hauptereignis ist logischer Weise der Click.

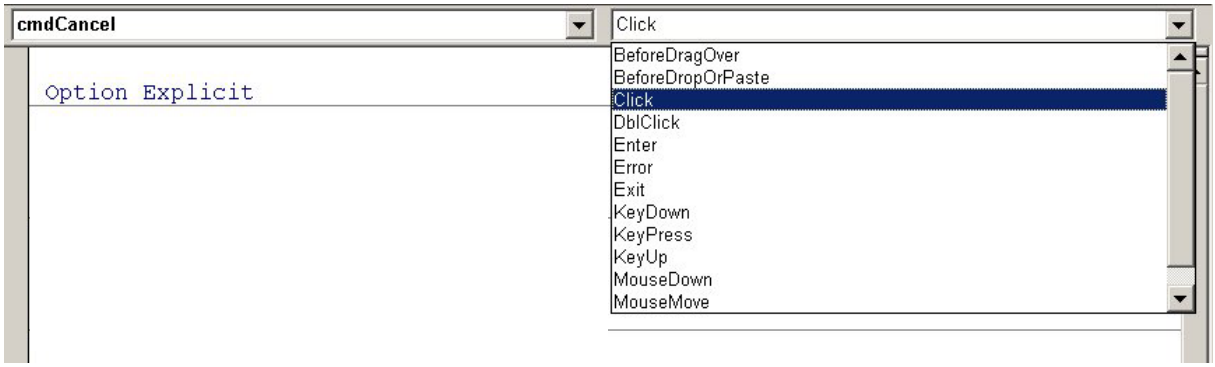
Um einen Knopf zu basteln, brauchen wir die Toolbox (siehe Seite 19). Nach Anklicken des Schalters „Befehlsschaltfläche“ kann man auf dem Userform mit gedrückter linker Maustaste einen Kasten ziehen. Man sollte sich angewöhnen, direkt die Benennung der Schaltfläche zu ändern, da die Standardbezeichnung „CommandButton1“ nicht sehr viel weiter hilft. Die Graphik zeigt ein Beispiel:



Der Button heißt jetzt `cmdCancel` und die Beschriftung lautet ***Schließen***. Man kann hier jetzt noch einiges an Formaten einstellen, aber das ist mehr Spielerei und kann selber ausprobiert werden.

4.2.1 Das Ereignis Click...

... ist eigentlich das einzig wichtige Ereignis des Befehlsbuttons. Schon der Doppelklick ist ungewöhnlich und ungebräuchlich. Aber auch hier hat man wieder sehr viel mehr Auswahl, als man wirklich braucht. Wechselt man mit Doppelklick oder den Buttons oder aber ***F7*** in die Code-Ansicht, wählt sich links das Objekt *„cmdCancel“* sieht man rechts die ganze Latte von Möglichkeiten:



Um nun ein Formular zu schließen, muss man auf das Ereignis Click folgende Befehle legen:

```
Private Sub cmdCancel_Click()
Unload usrUebungSteuerelemente
End Sub
```

An dieser Stelle ein paar Worte zum Objekt `Me`. Um unsereins Tipperei zu ersparen, hat Microsoft das Objekt `Me` erfunden. Gemeint ist damit das gerade aktuelle Formular. Mit dem obigen Code äquivalent ist also folgende Formulierung:

```
Private Sub cmdCancel_Click()
Unload Me
End Sub
```

Und noch ein paar Worte zum Starten des Codes in Userforms: Auch wenn man nur einen bestimmten Teil des Codes ausprobieren möchte, also z. B. den eigenständigen Code einer Sub-Prozedur, wird immer erst mal das Form gestartet. Nur so kommt man an die einzelnen Objekte heran.

4.3 Das Beschriftungsfeld

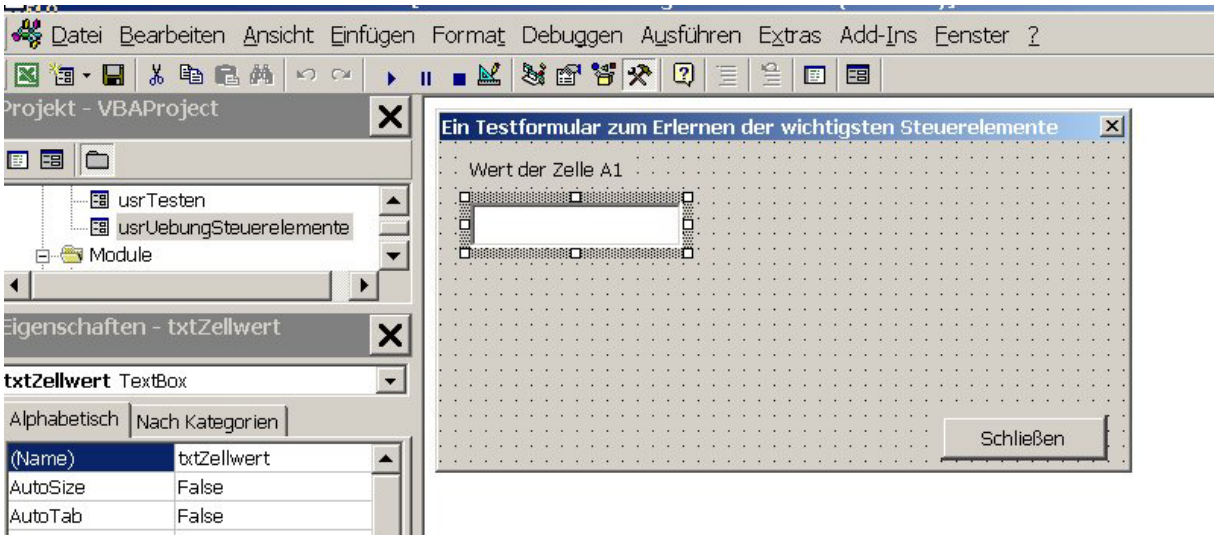
Das Beschriftungsfeld ist ein reines Textfeld, mit dem man Formularteile beschriften kann. Normalerweise fügt man es einfach ein, ändert die Beschriftung und Schluss ist. Es kann allerdings auch als Ausgabefenster benutzt werden, was manchmal ganz praktisch ist, da man den Wert dort nicht ändern kann. Z. B. könnte man so den Namen des aktuellen Arbeitsblatts ausgeben. Dazu muss man entweder einen Button erstellen, der das antriggert, oder aber beim Laden des Userforms dem Ereignis Initialize diese Information mitgeben. Ein Beispiel, wenn das Beschriftungsfeld `lblDynamisch` heisst.

```
Private Sub UserForm_Initialize()
MsgBox "Das Formular kann eigentlich noch nichts :-)!"
Me.lblDynamisch.Caption = ActiveSheet.Name
End Sub
```

4.4 Die Textbox

Ein vielbenutztes Control-Element ist die Textbox. Mit ihr kann man Werte in eine Tabelle schreiben oder auch aus ihr herauslesen.

Wenn man ein solches Element auf das Formular bringt, sollte man es auch gleich wieder umbenennen:

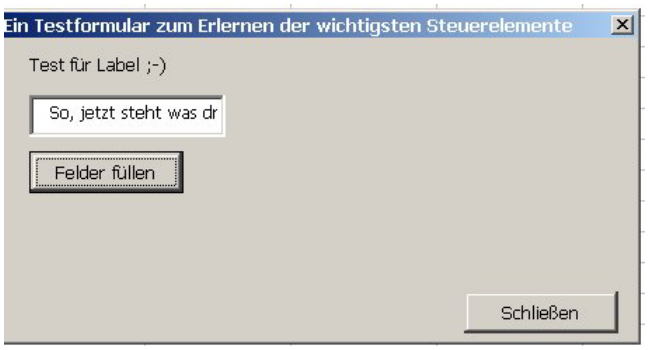


Um nun einen Wert auszulesen oder einen eingegebenen Wert irgendwo hinzu schreiben, braucht man das Objekt `.Value` und eine Aktion, die das dann auch ausführt. Bei diesem Beispiel und bei den folgenden wird das mit einem Button erledigt, der „*cmdAktionStarten*“ heißen soll.

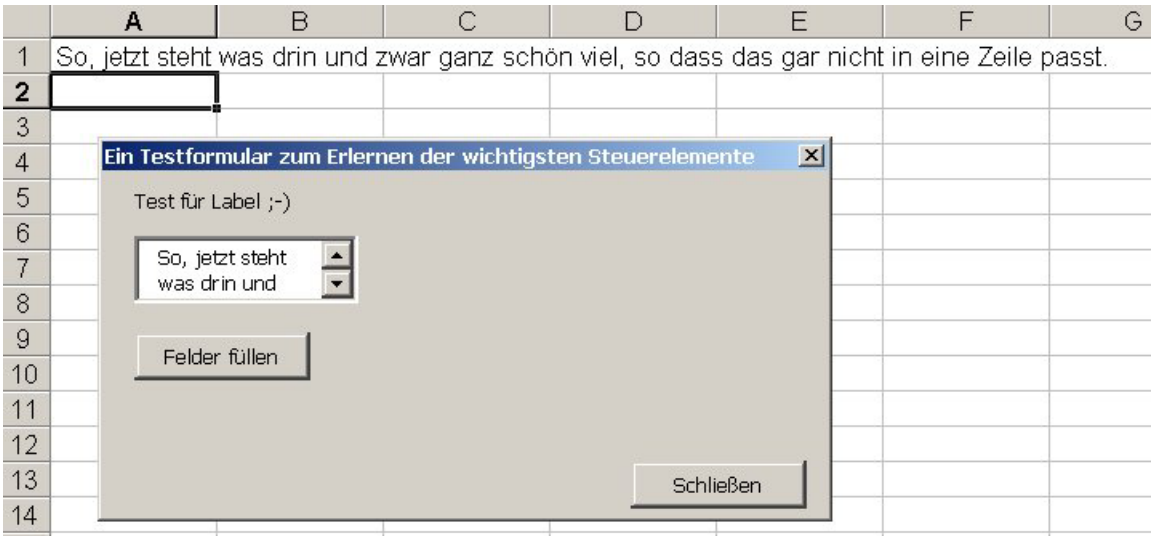
Der Code lautet:

```
Private Sub cmdStarten_Click()
Me.txtZellwert.Value = [a1]
End Sub
```

Und das Ergebnis nach Anklicken des Knopfes



zeigt einem, dass in der Zelle A1 „So, jetzt steht was dr....“ steht. Da wird wohl noch etwas kommen. Um sicher zu sein, dass man auch alles sieht, sollte bei den Eigenschaften der Textbox *MultiLine* auf *True* setzen. Das heisst, das der Text in der Textbox umgebrochen werden kann. Wenn man die Box dann noch etwas länger zieht, oder sogar noch die Eigenschaften *Scrollbar* auf *2-fmScrollbarsVertical* dann kann man sicher sein, dass nichts verloren geht:



Die Laufleisten sieht man allerdings erst, wenn man in das Textfeld klickt.

Mit der Formel `[a1] = Me.txtZellwert.Value` bekommt man den Wert, den man dort eingetragen hat in die Zelle A1 zurückgeschrieben.

4.5 Das Dropdownfeld und das Listenfeld

Ein häufiges Element ist das Dropdownfeld, im Englischen: die Combobox, weshalb ihr Präfix auch *cbo* heisst. Genauso funktioniert auch das Listenfeld, dessen Präfix *lst* lautet. Aus einer Liste kann man auswählen, welcher Wert erscheinen soll. Das Hauptereignis ist auch hier das Click-Ereignis. Hier wird das Beispiel an einer Combobox nachvollzogen.

Um die einzelnen Punkte in das Feld zu bekommen, gibt es zwei Möglichkeiten:

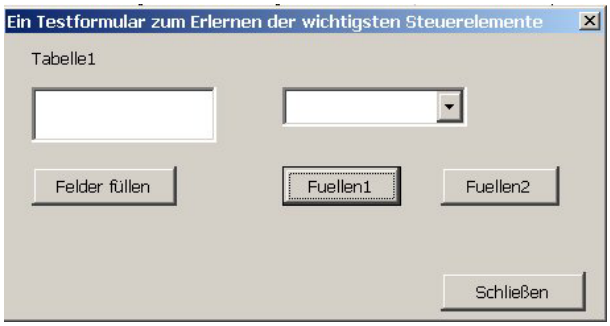
1. „Von Hand“

Meistens werden die Werte beim Initialisieren des Userforms eingelesen. In diesem Beispiel soll es allerdings wieder mit einem Commandbutton gemacht werden, damit nicht alles durcheinander steht:

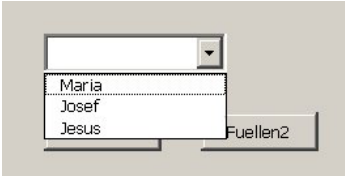
```
Private Sub cmdcboFuellen1_Click()
Me.cboNamensliste.AddItem "Maria"
Me.cboNamensliste.AddItem "Josef"
Me.cboNamensliste.AddItem "Jesus"
```

```
End Sub
```

Mit AddItem also bekommt man die gewünschten Einträge. Das Ergebnis sieht dann so aus:



Das ist irgendwie noch nichts. Erst wenn man auf den Pfeil nach unten klickt und das Dropdownfeld benutzt, sehen wir etwas:



Aber das geht auch noch besser. Mit der zusätzlichen Zeile:

```
Me.cboNamensliste.ListIndex = "0"
```

Wird der *erste* Eintrag der Liste gleich in das Feld geschrieben. Was lernt uns das? Die Listenelemente beginnen beim Zählen mit 0. Na gut...

Noch eine Verbesserung ist möglich. VBA kennt den `With`-Befehl. Dieser wird benutzt, um sich wiederholende Schreibarbeit zu sparen.

```
Me.cboNamensliste.AddItem "Maria"
Me.cboNamensliste.AddItem "Josef"
Me.cboNamensliste.AddItem "Jesus"
Me.cboNamensliste.ListIndex = "0"
```

Es ist klar, dass in jeder Zeile der Großteil des Befehls gleich lautet, weil es sich um das gleiche Objekt handelt. In einem solchen Fall kann man auch

```
Private Sub cmdcboFuellen1_Click()
With Me.cboNamensliste
  .AddItem "Maria"
  .AddItem "Josef"
  .AddItem "Jesus"
  .ListIndex = "0"
End With
End Sub
```

schreiben, also in die erste Zeile `With` und der Teil der sich dauernd wiederholt, darunter dann nur noch die Unterbefehle mit dem Punkt davor. Und wenn man fertig ist, wird die Geschichte mit `End With` abgeschlossen. Ziemlich elegant...

Um den markierten Wert auszulesen, braucht man den Befehl: `Me.cboNamensliste.Value`, also den gleichen Befehl wie bei der Textbox. Schreibt man diesen in einen Variable, kann man damit weiterrechnen.

2. Auslesen aus dem Tabellenblatt

Die zweite Methode in Excel eine Combobox zu füllen besteht in dem Auslesen einer Liste auf einem Tabellenblatt.

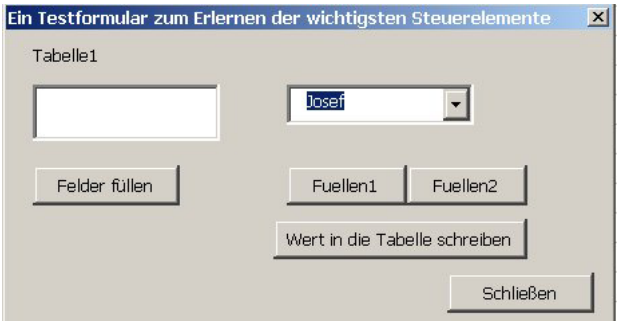
	A
1	Jesus
2	Maria
3	Josef
4	Esel
5	Ochs
6	

Nimmt man die obige Liste als Ausgangspunkt, dann lautet der Code

```
Private Sub cmdcboFuellen2_Click()
Me.cboNamensliste.RowSource = ("A1:A5")
Me.cboNamensliste.ListIndex = "0"
End Sub
```

Das ist sicher die elegantere Methode und auch die, die mehr Dynamik zulässt. Man kann sich durchaus ein Tabellenblatt einfügen, indem man nur diese Listen verwaltet, die irgendwelche Comboboxen füllen sollen.

Um einen markierten Wert in die Tabelle zurückzuschreiben, hat man auch wieder verschiedene Möglichkeiten.



Zum einen geht auf jeden Fall:

```
Private Sub cmdWertZurueckschrbeiben_Click()
Dim varWert As Variant
varWert = Me.cboNamensliste.Value
[b5] = varWert
End Sub
```

Bei Betätigen des neuen Buttons wird unserem Fall „Josef“ in die Zellen B5 geschrieben.

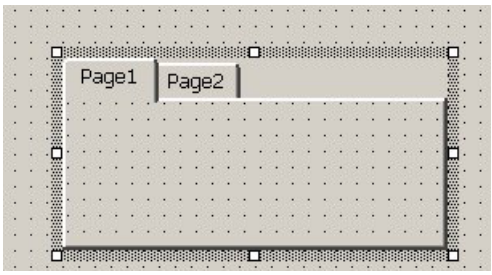
Die andere Möglichkeit, die allerdings nur für Comboboxen und nicht für Listfelder gilt, sieht so aus:

```
Private Sub cmdWertZurueckschrbeiben_Click()
Me.cboNamensliste.ControlSource = "c6"
End Sub
```

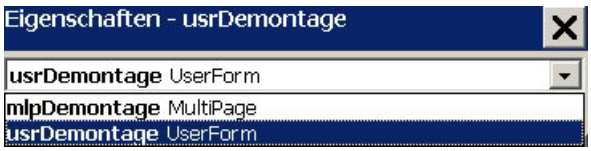
Hier würde „Josef“ in die Zelle C6 gepackt. Ebenso wie Rowsource ist es wieder die kürzere Möglichkeit.

4.6 Das Multipage-Steuerelement

Mit dem Multipage-Steuerelement kann man auf seinen Formularen Register einrichten, damit sich nicht alle Befehle auf einer Seite rumknubbeln. Das Präfix des Elements ist *mlp*. Wenn man ein solches einfügt, bekommt man eigentlich drei Objekte, nämlich das Multipage-Objekt selber und zwei Pages, die dazugehören und die eigentlichen Unterteilungen bilden.



Bei den Pages bietet sich *pg* als Präfix an. Bei der Beschriftung (Caption) sollte man darauf achten, dass der Benutzer damit etwas anfangen kann. Danach kann man auf den Registerkarten ganz normal wie auf einem Formular arbeiten. Um das Element neu zu beschriften, hat man allerdings etwas Probleme mit dem Markieren. Es wird immer das Page-Element „gefokust“, um mal ein schönes neues Wort zu erfinden. Man sollte in diesem Fall versuchen, nur das Formular zu markieren und dann auf der rechten Seite im Eigenschaftsfenster an das gewünschte Multipage-Element zu kommen.

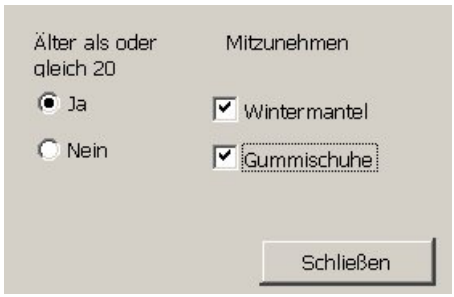


Bei diesem Beispiel ist es schon umbenannt worden.

4.7 Optionsfelder und Kontrollkästchen

Diese beiden Elemente dienen zum Ankreuzen von Möglichkeiten. Dabei kann man die Informationen der Optionsfelder ***ausschließend*** gestalten, also immer nur ein Element einer Gruppe ist angeschaltet. Kontrollkästchen können auch ***gemeinsam*** aktiviert werden.

Ein Beispiel:



Erstellt werden diese Elemente ganz normal mit gedrückter Maustaste auf dem Formular. Die Präfixe sollten *opt* bzw. *chk* lauten.

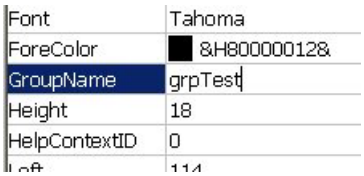
Um einen Wert eines solchen Feldes auszulesen, braucht man:

```
Dim blnWert As Boolean
blnWert = Me.optAelterAls20Ja.Value
```

```
MsgBox blnWert
```

Als Ergebnis bekommt man dann WAHR(TRUE, –1) oder FALSCH (FALSE, 0) und kann damit weiterrechnen.

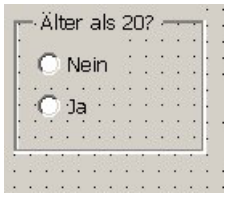
Um die Optionskästchen wissen zu machen, dass sie zusammengehören, kann man unter den Eigenschaften einen *Gruppennamen* vergeben.



Damit ist auch gewährleistet, dass man mehrere Gruppen auf einem Userformular benutzen kann.

4.8 Rahmen

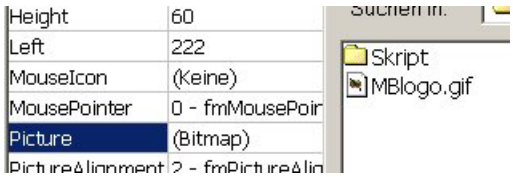
Frames (Rahmen) - Präfix ist *frm* - dienen in den meisten Zwecken nur der optischen Einteilung des Userforms. Allerdings können sie auch als Container für Optionsfelder dienen, so dass dann zusammengehört, was auch zusammengehören soll. Zu beachten dabei ist, dass man den Frame zuerst malt und dann die anderen Elemente dort hinschreibt. Ansonsten verdeckt der Rahmen die Elemente.



Die Eigenschaft *Caption* ist bei diesem Element die Beschriftung oben im Rahmen.

4.9 Graphikrahmen

Ein eigenes Element dient als Container für Graphiken, die man auf dem Formular platzieren möchte. Das Präfix lautet *img* und in den Eigenschaften kann man den Ursprung des Bildes eingeben.



5 Umsetzung der gestellten Aufgabe

5.1 Vorneweg zur Erinnerung: ein paar Programmierstrukturen in VBA

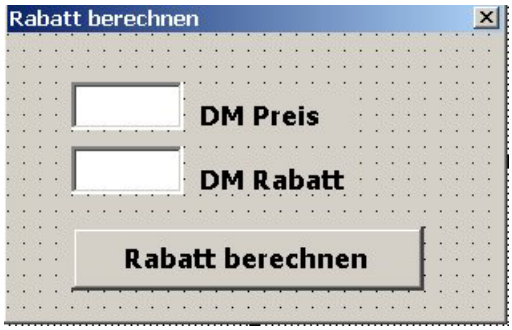
Eigentlich sind ja die normalen Kontrollstrukturen vorausgesetzt in diesem Kurs, aber da man sich nicht alles merken kann noch mal ein kurze Wiederholung der wichtigsten Werkzeuge.

5.1.1 IF Then Else

Das ist wohl die bekannteste Bedingungsabfrage in jeder Sprache.

```
If Bedingung Then
  Anweisung, wenn Bedingung wahr ist
Else
  Anweisung, wenn Bedingung falsch ist.
End If
```

Ein Beispiel, mit dem Sie den Rabatt berechnen können, den es aber erst gibt, wenn der Preis über 200 DM liegt. Der Preis steht in der Zelle A2 der Tabelle.




```
Private Sub cmdRabattberechnen_Click()

Me.txtPreis.value = [a2]

If [a2] > 200 Then
 Me.txtRabatt.Value = [a2] * 0.04 `Das Dezimalzeichen muss ein Punkt sein
Else
 Me.txtRabatt.Value = 0
End If

End Sub
```

Das gibt es dann auch noch eine Stufe verschachtelt mit dem zusätzlichen Keyword **ElseIf**, um mehrere Stufen durchlaufen zukönnen. In unserem Beispiel hieße das, dass man ab einer Rechnung von 1000 DM nicht nur 4 Prozent, sondern 6 Prozent Rabatt bekommt.

```
Private Sub cmdRabattberechnen_Click()

Me.txtPreis.value = [a2]

If [a2] > 1000 Then
 Me.txtRabatt.Value = [a2] * 0.06
ElseIf [a2] > 200 Then
  Me.txtRabatt.Value = [a2] * 0.04
  Else
   Me.txtRabatt.Value = 0
```

```
End If
End Sub
```

5.1.2 Select Case

Muss man mit mehreren Überprüfungen der Bedingungen arbeiten, dann bietet sich allerdings eher der Befehl **`Select Case`** an, der übersichtlicher ist, als das verschachtelte If-Konstrukt.

```
Select case Ausdruck, der überprüft wird
  case Wert1 des Ausdrucks
  Anweisung
  case Wert2 des Ausdrucks
  Anweisung
...
  case else
End select
```

Nimmt man das obige Beispiel, sähe das so aus.

```
Private Sub cmdSelectCaseRabattberechnen_Click()
Dim dblPreis As Double
Me.txtPreis.Value = [a2]
dblPreis = [a2]
 Select Case dblPreis
  Case Is > 1000
  Me.txtRabatt.Value = [a2] * 0.06
  Case Is > 200
  Me.txtRabatt.Value = [a2] * 0.04
  Case Else
  Me.txtRabatt.Value = 0
 End Select
End Sub
```

5.1.3 For... Next

Um eine Reihe gleichartiger Befehle eine bestimmte Anzahl mal anzuwenden, kann man mit **`For ... Next`** arbeiten.

```
For Zähler = Anfang To Ende [Step Schritt]
 [Anweisungen]
 [Exit For]
 [Anweisungen]
Next [Zähler]
```

Auch hier wieder ein einfaches Beispiel:

```
Sub BeispielForNext()
Dim i As Integer
Dim Ende As Integer
Ende = 5
For i = 1 To Ende
 MsgBox "i hat jetzt den Wert " & i
Next
End Sub
```

Die Anweisung `Step` ist praktisch, wenn man nicht jede Zahl der Reihe braucht, sondern z. B. nur die geraden Zahlen.

```
Sub BeispielForNextStep()
Dim i As Integer
Dim Ende As Integer
Ende = 10
For i = 2 To Ende Step 2
 MsgBox "i hat jetzt den geraden Wert " & i
Next
```

5.1.4 Do... Loop

Wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert **True** hat oder bis eine Bedingung den Wert **True** erhält. Je nachdem muss man mit `Until` (bis) oder `While` (solange) arbeiten.

```
Do [{While | Until} Bedingung]
 [Anweisungen]
Loop
```

Auch hier ein Beispiel

```
Sub BeispielDoLoop()
Dim i As Integer
i = 2
Do While i <= 10
 MsgBox "i hat jetzt den geraden Wert " & i
 i = i + 2
Loop
End Sub
```

Hier handelt es sich um eine sogenannte *kopfgesteuerte* Do... Loop-Schleife, bei der das Kriterium im Kopf der Anweisungen steht. Es gibt aber auch *fußgesteuerte* Do... Loop-Schleifen, die auf jeden Fall einmal abgearbeitet werden müssen.

Ein Beispiel

```
Sub Abfrage()
Do Passwort = InputBox(„Passswort“)
Loop Until Pasowrt = „Geheim“
End Sub
```

Es gibt dann noch die **`While... Wend`**-Variante, die solange ausgeführt wird, wie der Wert TRUE ist, aber Microsoft weist zurecht darauf hin, dass **`Do ... Loop`** die strukturierten Möglichkeiten bietet.

5.2 Ein kleiner Einschub für die Übersichtlichkeit beim Programmieren

Für die Übersichtlichkeit bieten sich kurze Module an, die sich gegenseitig aufrufen. So hat man schneller Fehler gefunden und auch notwendige Änderungen gehen einfacher. Gerade für das Initialisieren von Formularen, die viele Daten aus einer Tabelle auslesen, da verschiedene Felder gefüllt werden müssen, bietet sich die modale Struktur an:

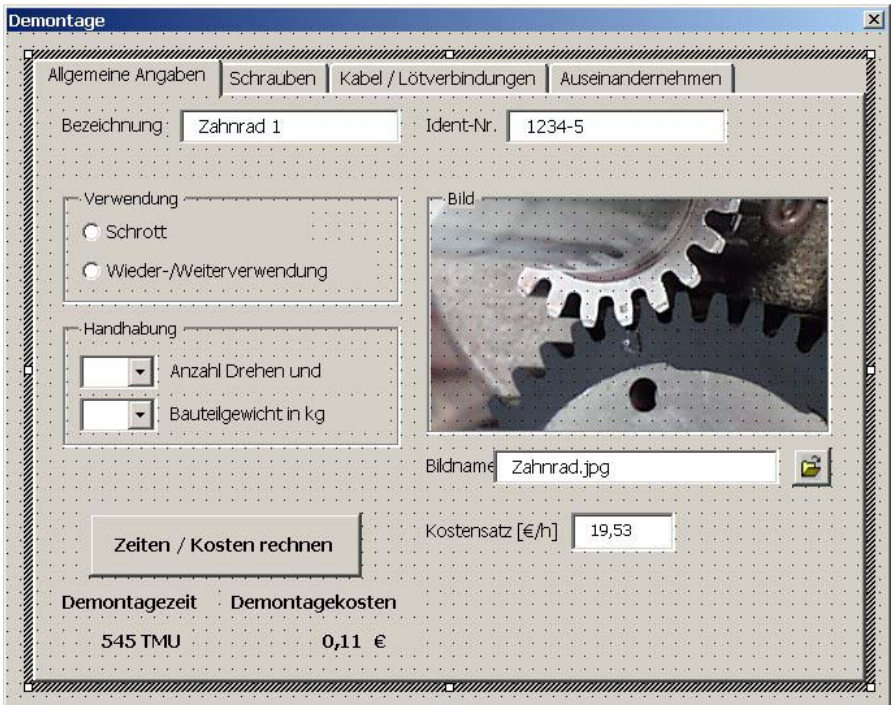
```
Private Sub UserForm_Initialize()
BearbeiterFuellen
End Sub

Sub BearbeiterFuellen()
Me.cboBearbeiter.RowSource = "$A:$A"
Me.cboBearbeiter.ListIndex = 0
End Sub
```

Man kann beim Laden des Formulars beliebig viele dieser Prozeduren ausführen lassen und weiß immer genau, wo man gerade ist.

5.3 Und nun endlich die gestellte Aufgabe

Als Vorlage gibt es schon das Userform:



Datei Software.xls

5.3.1 Steuerelemente umbenennen

Allerdings sind die Werte, die man dort sieht, von Hand in die Steuerfelder geschrieben worden. Außerdem muss die Benennung neu geregelt werden. Dazu überschreibt man in den Eigenschaften einfach den alten Namen.

Vorher:

Nachher:

Neue Namen für die Elemente

Element	*Alter Name*	*Neuer Name*
Userform	Userform1	usrDemontage
Multipage Element	Multipage1	mplDemontage
Registerkarte1	Page1	pgAllgemein
Registerkarte2	Page2	pgSchraube
Registerkarte3	Page3	pgKabel
Registerkarte4	Page4	pgAuseinandernehme
Beschriftungsfeld	Label11	lblId
Textbox	TextBox3	txtId
Rahmen	Frame3	frmVerwendung
Optionsbutton	Optionsbutton3	optSchrott
Optionsbutton	Optionsbutton4	optWeiter

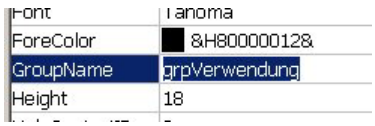
Na ja, und so weiter... Das sollte zum Üben genügen.

5.3.2 Steuerelemente ändern ... geht nicht

Als nächstes wollen wir das Steuerelement *txtBezeichnung* ändern, weil wir dort nämlich eine Combobox möchten, bei der wir aus der Anzahl der Artikel, die in der Datenbank stehen, auswählen können. Und hier stoßen wir an eine Grenze. Im Nachhinein ist es nicht möglich, ein solches Element zu ändern. Man muss das falsche Element tatsächlich löschen und das neue einfügen. In diesem Fall, da die Tabelle ein bisschen anders werden soll, fügt man eine neue ComboBox ein, die den Namen *cboBearbeiter* bekommt. Eine Combobox nehmen wir deshalb, weil wir dort sowohl Werte eingeben können, als auch aus einer Liste auswählen, falls es diese Werte schon gibt. So muss man sie dann später nicht zweimal schreiben.

5.3.3 Die Werte in einem Optionsbuttonpaar setzen

Nach dem Umbenennen des Rahmens und er beiden Buttons sowie der Eigenschaft Gruppe auf *grpVerwendung*,



möchte man, dass beim Laden des Forms der erste Button auf „Ja“ gesetzt wird. Also wird dem Initialisierungsmakro ein weiteres Modul hinzugefügt, in dem wir mit Value einen Wert übertragen. Dabei steht die ***1*** für ***an*** und die ***0*** für ***aus***:

```
Private Sub UserForm_Initialize()
BearbeiterFuellen
VerwendungAnschalten
End Sub
```

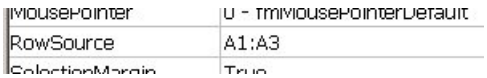
Und so funktioniert das neue Modul

```
Sub VerwendungAnschalten()
Me.optSchrott.Value = 1
End Sub
```

5.3.4 Einer Combobox einen Teil der Tabelle zuordnen

5.3.4.1 Über die Eigenschaften

Eine Methode, Daten dort hinzubekommen, ist, dass man sie direkt in den Eigenschaften des Steuerelements eingibt.



Allerdings gibt es hier keine Möglichkeit, den `ListIndex` auf 0 zu setzen.

5.3.4.2 Über Code - als Pflichtübung

Das ist jetzt eigentlich nur eine kleine Wiederholung des Kapitels 4.5 Das Dropdownfeld und das Listenfeld auf Seite 25. Mit folgende Code bekommt man den Inhalt der Spalte A in das DropdownFeld

```
Sub BearbeiterFuellen()
Me.cboBearbeiter.RowSource = "$A:$A"
Me.cboBearbeiter.ListIndex = 0
End Sub
```

5.3.4.3 Über Code - als Kür

Wenn man diesen Code benutzt, hat man ein Problem. Wenn ein Eintrag in der Spalte häufiger vorkommt, was beim Bearbeiternamen sicher der Fall ist, dann sieht man ihn auch häufiger in der Combobox. Das ist unnütz. JensF aus dem Spotlight-Forum – ein hervorwagendes Excel-Diskussions-Forum (http://spotlight.de/zforen/mse/t/forum_mse_1.html) – hat folgenden Code gebastelt, der das verhindert:

```
1   Sub BearbeiterFuellen()
2   Dim lngx As Long
3   Me.cboBearbeiter.Clear
4   For lngx = 1 To Range("A65536").End(xlUp).Row
5   If WorksheetFunction.CountIf(Range("A1:A" & lngx), Cells(lngx, 1)) _
      = 1 Then
6   Me.cboBearbeiter.AddItem Cells(lngx, 1)
7   End If
8   Next
```

```
9   Me.cboBearbeiter.ListIndex = 0

10  End Sub
```

Hier sind ein paar Erklärungen angebracht.

Zeile 3 leert eventuelle Einträge aus der Combobox. Das funktioniert nur, wenn in den Eigenschaften kein `RowSource` festgelegt ist.

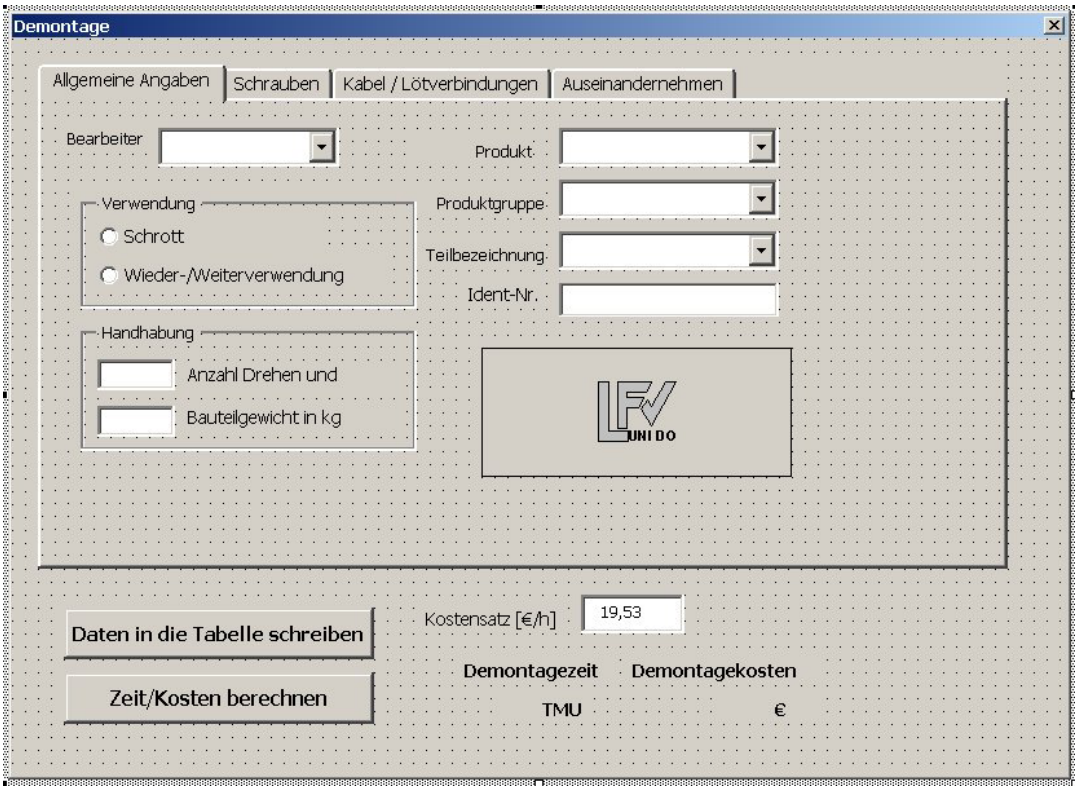
Zeile 4 zählt von 1 bis zur letzten benutzten Zeile der Tabelle. Und das ist schon ein guter Trick. Mit `Range("A65536").End(xlUp).Row` wird von der letzten Zeile `("A65536")` des Arbeitsblatts ans Ende des benutzten Bereichs gesprungen, mit Richtung nach oben `.End(xlUp)`, und dessen Zeilennummer ermittelt `.Row`.

In der *Zeile 5* wird gezeigt, wie man in VBA eine Zeile weiterführen kann, ohne dass für das Programm ein neuer Befehl beginnt. Die Kombination von `Leertaste` und `_(Unterstrich)` vermittelt VBA diesen Wunsch.

In den *Zeilen 5 bis 7* wird mit der Funktion ZÄHLENWENN ermittelt, wann der Eintrag einmal vorkommt und wird dann in die Liste geschrieben. Beim nächsten Treffen auf den Eintrag ist er schon bei zwei und wird deshalb nicht noch mal übernommen. Hört sich alles komplizierter an, als es ist. Man kann den Code auch einfach abschreiben und auf die eigenen Spalten zu schneiden.... Das soll auch noch gemacht werden, aber erst mal ein...

5.3.5 Kleines Zwischenfazit

Nach einigen Umbauten, die etwas mit der Logik der Tabelle zu tun haben die entstehen soll, und denen auch das Zuordnen der Graphik zum Opfer gefallen ist, sieht das Formular auf seiner ersten Seite inzwischen so aus:



Datei Zwischenergebnis.xls

Und auf Seite hat sich auch ein bisschen etwas geändert. So kann man jetzt nämlich von jeder Registerkarte aus die beiden Buttons drücken. Außerdem wurde an den Eigenschaften noch etwas verbessert und zwar kann man einen TAB-Index setzten, der bei 0 beginnt und damit die Reihenfolge der mit der Tabtaste angesprungenen Elemente bestimmt.

Außerdem gibt es jetzt ein Tabellenblatt, das „Datentabelle“ heißt und folgende Überschriften stehen dort.

Bearbeiter	Produktbezeichnung	Baugruppe	Teilbezeichnung	Ident-Nr	Prozesszeit	SchrottJaNein	Anzahlschrau	AnzahlVersch	TMU	Kosten

Folgender Code verbirgt sich hinter dem geänderten Userform, wobei in der ersten Zeile des Initialisierens das Arbeitsblatt „Datensammlung“ als aktives Blatt gesetzt wird, damit nichts schief geht und die Werte wirklich dort auch landen.

```
Option Explicit

Private Sub UserForm_Initialize()
Worksheets("Datentabelle").Activate
BearbeiterFuellen
VerwendungAnschalten
End Sub
```

```
Sub BearbeiterFuellen()
Dim lngx As Long
```

```
Me.cboBearbeiter.Clear
For lngx = 1 To Range("A65536").End(xlUp).Row
 If WorksheetFunction.CountIf(Range("A1:A" & lngx), Cells(lngx, 1)) = 1 Then
  Me.cboBearbeiter.AddItem Cells(lngx, 1)
 End If
Next
Me.cboBearbeiter.ListIndex = 0
End Sub
```

```
Sub VerwendungAnschalten()
Me.optSchrott.Value = 1
End Sub
```

Übung

Versuchen Sie, die anderen Dropdownfelder (***Produkt***, ***Produktgruppe*** und ***Teilbezeichnung***) auch zu füllen, wie es das Beispiel ***Bearbeiter*** vorgibt. Man muss einfach die Steuerelementnamen und die Spalten anpassen.

Lösung

```
Option Explicit
```

```
Private Sub UserForm_Initialize()
BearbeiterFuellen
ProduktFuellen
GruppeFuellen
TeilFuellen
VerwendungAnschalten
End Sub
```

```
Sub BearbeiterFuellen()
Dim lngx As Long
Me.cboBearbeiter.Clear
For lngx = 1 To Range("A65536").End(xlUp).Row
 If WorksheetFunction.CountIf(Range("A1:A" & lngx), Cells(lngx, 1)) = 1 Then
  Me.cboBearbeiter.AddItem Cells(lngx, 1)
 End If
Next
Me.cboBearbeiter.ListIndex = 0
End Sub
```

```
Sub ProduktFuellen()
Dim lngx As Long
Me.cboProdukt.Clear
For lngx = 1 To Range("B65536").End(xlUp).Row
 If WorksheetFunction.CountIf(Range("B1:B" & lngx), Cells(lngx, 2)) = 1 Then
  Me.cboProdukt.AddItem Cells(lngx, 2)
 End If
Next
Me.cboProdukt.ListIndex = 0
End Sub
```

```
Sub GruppeFuellen()
Dim lngx As Long
Me.cboGruppe.Clear
For lngx = 1 To Range("C65536").End(xlUp).Row
 If WorksheetFunction.CountIf(Range("C1:C" & lngx), Cells(lngx, 3)) = 1 Then
  Me.cboGruppe.AddItem Cells(lngx, 3)
 End If
Next
Me.cboGruppe.ListIndex = 0
```

```
End Sub
```

```
Sub TeilFuellen()
Dim lngx As Long
Me.cboTeil.Clear
For lngx = 1 To Range("D65536").End(xlUp).Row
 If WorksheetFunction.CountIf(Range("D1:D" & lngx), Cells(lngx, 4)) = 1 Then
  Me.cboTeil.AddItem Cells(lngx, 4)
 End If
Next
Me.cboTeil.ListIndex = 0
End Sub
```

```
Sub VerwendungAnschalten()
Me.optSchrott.Value = 1
End Sub
```

5.3.6 Die eingetragenen Daten zurückschreiben in die Tabelle

Die Graphik auf Seite 36 zeigt es schon: es wurde ein weiterer Button hinzugefügt, der es erlaubt, die eingetragenen Werte in die Tabelle zu übernehmen. Die Überschriften dazu finden sich schon auf dem Tabellenblatt „Datensammlung“. Ein zu lösendes Problem des Makros besteht darin, dass der neue Datensatz immer an die letzte Zeile der Tabelle angehängt werden muss. Die Gretchenfrage lautet also: Wie bekomme ich heraus, wie viele Zeilen die Tabelle schon enthält?

Dazu gibt es schon den nötigen Trick. Es wird mit einem Bereich gearbeitet, der vom Ende des Blattes aufwärts zur ersten bearbeiteten Zelle springt und zwar günstiger Weise in Spalte A, weil bei Bearbeiter ziemlich sicher etwas eingetragen ist. Diesem `.Row`-Wert addiert man noch 1 dazu, und bekommt so die erste freie Reihe unterhalb der Tabelle. Voraussetzung ist, dass wirklich ein Bearbeiter eingetragen ist. Der Name des Buttons heißt „*cmdDatenschreiben“*.

```
Private Sub cmdDatenschreiben_Click()
Dim lngNeueReihe As Long
,Aktivieren
Worksheets("Datentabelle").Activate
'Neue Reihe berechnen
lngNeueReihe = Range("A65536").End(xlUp).Row + 1
```

Der Rest ist einfach

```
'Werte eintragen
ActiveSheet.Cells(lngNeueReihe, 1).Value = Me.cboBearbeiter.Value
ActiveSheet.Cells(lngNeueReihe, 2).Value = Me.cboProdukt.Value
ActiveSheet.Cells(lngNeueReihe, 3).Value = Me.cboGruppe.Value
ActiveSheet.Cells(lngNeueReihe, 4).Value = Me.cboTeil.Value
ActiveSheet.Cells(lngNeueReihe, 5).Value = Me.txtId.Value
ActiveSheet.Cells(lngNeueReihe, 6).Value = Me.txtProzesszeit.Value
ActiveSheet.Cells(lngNeueReihe, 7).Value = Me.optSchrott.Value
End Sub
```

In dem fertigen Produkt sind das dann enstprechende mehr Steuerelemente, die man ausliest, aber um das Prinzip zu verstehen reicht das. Solange man den Button ***„Daten in Tabelle schreiben“*** nicht gedrückt hat, passiert auch nichts.

5.4 Das Ausrechnen der Zeit und damit der Kosten

Um die Aufgabe zu Ende zu bringen, fehlt jetzt nur noch der Button, der Zeit und Kosten in Abhängigkeit der eingegebenen Werte ausrechnet. Das kann man am Besten mit der Verknüpfung von WENN-Bedingung und dem Operator AND erreichen, wobei das von Fall zu Fall beliebig verschachtelt wird, weshalb hier nur zwei relativ überschaubare Beispiele gezeigt werden sollen.

Aber zuerst soll der Button benannt werden und zwar als *cmdZeitKostenRechnen*. Auf dem Registerblatt „Schrauben" werden die beiden Textfelder benannt, und zwar als: *txtAnzahlSchrauben* und *txtAnzahlSchraubenTypen*. Das Makro, dass die Werte in die Tabelle einträgt, wird um zwei Zeilen erweitert:

```
ActiveSheet.Cells(lngNeueReihe, 8).Value = Me.txtAnzahlSchrauben
ActiveSheet.Cells(lngNeueReihe, 9).Value = Me.txtSchraubenTypen
```

Außerdem binden wir die Liste der Codes, die mit den Zeiten korrespondieren in die Tabelle ein und nennen das Arbeitsblatt „Zeiten".

Eigentlich müsste man in der Abfrage bei der *Prozesszeit* beginnen, da diese bei Abweichung vom Standard die Zeiten der Tabelle ändert. Dieser Algorithmus sprengt aber jetzt ein bisschen das Seminar, so dass wir dort von einem Standardwert ausgehen.

Lässt man diese Abfrage weg, ist die erste Frage die nach Weiterverwertung oder Schrott und der Anzahl der Schrauben. Dann ist es relevant, ob es sich um mehrere verschiedene Sorten von Schrauben handelt und zum Schluss (nur in unserem Beispiel ;-)) werden eventuelle zusätzliche Vorgänge noch als Zeit addiert.

Also:

WENN Schrott= Ja UND Anzahl der Schrauben = 1 DANN ist der Code 1S1A001. Diesen Wert muss man dann in der Datentabelle nachschauen. Dort steht 144 TMU und damit haben wir unseren Zeitwert, den wir dann noch mit der entsprechenden Kostensatz multiplizieren können.

Oder:

WENN Schrott = Ja UND Anzahl der Schrauben > 1 DANN lautet der Code 1S1A001 + (Anzahl der Schrauben – 1)*1S2A001 und die Zeit errechnet sich als: 144 TMU + (Anzahl der Schrauben – 1) * 115 TMU.

5.4.1 Eine Funktion für das Durchsuchen erstellen

An dieser Stelle muss man kurz innehalten, weil man merkt, dass es einen Vorgang gibt, den man immer wieder braucht, nämlich an Hand eines bestimmten Codes einen Wert aus der Zeitentabelle auslesen. Und mit diesem Wert soll dann weitergerechnet werden. So etwas erledigt man am besten mit einer Funktion, die sich von einer Prozedur nur dadurch unterscheidet, dass man mit ihr auch einen Wert zurückgeben kann, was hier der Fall sein muss, weil wir ja den Wert TMU aus der Tabelle brauchen. Wie geht das nun?

Statt des Zauberworts **`Sub`** brauchen wir als Einleitung **`Function`** (und damit auch als Schluss **`End Function`**). Dahinter kommt in Klammern ein Platzhalter für den Wert, der übergeben wird. Bei diesem Beispiel ist es ein Code wie z. B. „1S1A001". Dann kommen die Variablen und Anweisungen und zum Schluss der Name der Funktion mit dem Ergebnis, das zurückgegeben werden soll, hier die zu dem Code gehörige Zeit in TMU (was auch immer das für eine Einheit ist ;-)).

```
Function Durchsuchen(strCode As String) As Double
```

Der Name der Funktion lautet Durchsuchen, als Wert wird der Inhalt der Variable strCode übergeben und das Ergebnis, mit dem man nachher weiterrechnen will, ist vom Werte Double. Uff...

Die gesamte Funktion sieht dann so aus:

```
Function Durchsuchen(strCode As String) As Double
`Zuererst werden die Variablen gesetzt
Dim rgZelle As Range
Dim rgBereich As Range
Dim dblZeit As Double

'Welcher Bereich soll durchsucht werden?
Set rgBereich = Worksheets("Zeiten").Range("A1:A30")

'Für jede Zelle in dem Bereich
For Each rgZelle In rgBereich

'Wenn der Wert gleich dem übergebenen Wert ist
   If rgZelle.Value = strCode Then

'Soll sich der Zeitwert daneben gemerkt werden in der Variabel dblZeit
    dblZeit = rgZelle.Offset(0, 1)
   End If
'Das soll solange sein, bis alle Zellen durchsucht sind
Next

'Im Funktionsnamen wird dann die Variable zurückgegeben
Durchsuchen = dblZeit
End Function
```

Ein Aufruf einer solcher Funktion erfolgt über eine normale Prozedur. Mal ein einfaches Beispiel, bevor es wieder zurück in das Formular geht:

```
Sub FunktionAufrufen()

Dim dblZwischenzeit1 As String

'Hier wird die Funktion aufgerufen mit dem Code "0H1A001".
'Das Ergebnis wird in die Variable dblZwischenezeit1 geschrieben
dblZwischenzeit1 = Durchsuchen("0H1A001")
MsgBox dblZwischenzeit1
End Sub
```

5.4.2 Zurück zum Formular

In der oben begonnen WENN-DANN Bedingung, die zusammen mit einem AND funktioniert, ist ein Fehler in der Logik. Ich habe ihn extra stehen lassen, damit Sie merken, wie leicht man sich auch verrennt. Es war klar, dass ein Bauteil, das nicht auf den Schrott kommt, anders behandelt wird, wenn seine Schraubenanzahl größer ist als 1. Es wird dann nämlich mit einem zweiten Code addiert. Aber eigentlich braucht man das gar nicht zu beachten, weil die Formel in dem Ausdruck "(Anzahl der Schrauben – 1)*1S2A001" dies antizipiert. Wenn nämlich die Zahl der Schrauben = 1 ist, dann wird eine 1 abgezogen, der Ausdruck wird 0 und der ganze Teil der Addition fällt sowie so weg. Also kann man diese Unterscheidung schon mal lassen. Ob es wichtig ist, dass die Schraubenanzahl nicht kleiner als 0 wird, muss man dann noch überlegen.

```
Private Sub cmdZeitKostenRechnen_Click()
Dim dblEndzeit As Double
Dim dblZwischenzeit1 As Double
Dim dblZwischenzeit2 As Double

 If Me.optSchrott = True Then
   dblZwischenzeit1 = Durchsuchen("1S1A001")
   dblZwischenzeit2 = Durchsuchen("1S2A001")
   dblEndzeit = dblZwischenzeit1 + (Me.txtAnzahlSchrauben - 1) * _
                dblZwischenzeit2
   MsgBox "Schrott JA und Anzahl Schrauben =" & _
          Me.txtAnzahlSchrauben & vbCr & dblEndzeit
 Else
   dblZwischenzeit1 = Durchsuchen("1S1B001")
   dblZwischenzeit2 = Durchsuchen("1S2B001")
   dblEndzeit = dblZwischenzeit1 + (Me.txtAnzahlSchrauben - 1) * _
                dblZwischenzeit2
   MsgBox "Schrott NEIN und Anzahl Schrauben =" & _
          Me.txtAnzahlSchrauben & vbCr & dblEndzeit
 End If
End Sub
```

Erweitern wir obiges Beispiel um einen Punkt, nämlich, den, dass sich die Anzahl der Schrauben auf mehrere verschiedene Typen verteilt. Der Zuschlag lautet dann *"Zeit des Codes '1S1A002' * (AnzahlSorten – 1)"*. Wenn hier für Anzahl der Sorten eine 1 eingeben wird, dann wird er ganze Ausdruck 0. Es stört also nicht, wenn der Ausdruck in der Formel steht. So kann man dann die meisten anderen Zuschläge auch verwalten.

```
Private Sub cmdZeitKostenRechnen_Click()
Dim dblEndzeit As Double
Dim dblZwischenzeit1 As Double
Dim dblZwischenzeit2 As Double
Dim dblZwischenzeit3 As Double

 If Me.optSchrott = True Then
   dblZwischenzeit1 = Durchsuchen("1S1A001")
   dblZwischenzeit2 = Durchsuchen("1S2A001")
   dblEndzeit = dblZwischenzeit1 + (Me.txtAnzahlSchrauben - 1) * _
                dblZwischenzeit2
   MsgBox "Schrott JA und Anzahl Schrauben =" & _
          Me.txtAnzahlSchrauben & vbCr & dblEndzeit
 Else
   dblZwischenzeit1 = Durchsuchen("1S1B001")
   dblZwischenzeit2 = Durchsuchen("1S2B001")
   dblZwischenzeit3 = Durchsuchen("1S1A002")
   dblEndzeit = dblZwischenzeit1 + (Me.txtAnzahlSchrauben - 1) * _
                dblZwischenzeit2 + (Me.txtSchraubenTypen - 1) * _
                dblZwischenzeit3
   MsgBox "Schrott NEIN und Anzahl Schrauben =" & _
          Me.txtAnzahlSchrauben & vbCr & dblEndzeit
 End If
End Sub
```

5.4.3 Berechnete Daten zurückschreiben ins Formular und in die Tabelle

In unserem Beispielformular sieht man keine MessageBox mit den entsprechenden Werten, sondern sie werden im Formular angezeigt. Das ist nun wiederum ganz einfach. Das Steuer-

element heißen lbldemontageZeit und lblDemontageKosten, wobei sich das zweite aus dem ersten ergibt.

```
  End If
'Im Formularsichtbar machen
Me.lbldemontageZeit = dblEndzeit
Me.lblDemontagekosten = dblEndzeit * Me.txtKostensatz.Value / 36000
End Sub
```

Um die Daten in die Tabelle zu schreiben müssen wir unser Makro `cmdDatenschreiben_Click()` um die beiden Zeilen erweitern.

```
ActiveSheet.Cells(lngNeueReihe, 10).Value = Me.lbldemontageZeit
ActiveSheet.Cells(lngNeueReihe, 11).Value = Me.lblDemontagekosten
```

Ganz zum Schluss braucht man noch einen Button, der das Formular wieder entlädt.

```
Private Sub cmdFertig_Click()
Unload Me
End Sub
```

5.4.4 Weiterführende Ideen

Es wäre nicht schlecht, wenn man bei manchen der Feldern eine Überprüfung der Logik der Eingaben macht. Z. B. ist es sicher falsch, wenn man die Anzahl der verschiedenen Sorten Schrauben größer setzt, als die Anzahl der Schrauben insgesamt. Außerdem wäre eventuell eine Abfrage bei manchen Feldern sinnvoll, die sie darauf hin überprüft, ob sie leer sind. Außerdem lohnt es sich, das Objekt ***Error*** näher zu studieren, das eine Fehlerbehandlung erlaubt. Für den Anfang soll das erst einmal reichen. Da ich auch ein Telefonnummer an der Uni habe, können Sie diese getrost benutzen. Viel Spaß bei der Umsetzung.