

## Grundlagen der VBA-Programmierung

Visual Basic for Applications oder kurz VBA ist eine Programmiersprache, die von Microsoft entwickelt wurde. Verschiedene Programmstrukturen wurden aus dem früheren Basic übernommen. Die Entwicklungsumgebung und die Art zu programmieren hat sich aber erheblich geändert. Ein Hauptvorteil gegenüber anderen Programmiersprachen ist die leichte Erlernbarkeit und die Integration in viele Programme. Mit VBA können Sie keine ausführbaren Programme erstellen. Sie können aber VBA-Routinen in Visual Basic übertragen. Die Sprachelemente von VB sind identisch mit denen von VBA. In VBA gibt es aber zusätzliche auf die jeweilige Anwendung bezogene Elemente. Zum Beispiel finden Sie in Microsoft Excel das Objekt „Cell“, in Word das Objekt Paragraph (Absatz), das es in Excel nicht gibt. Um in VB mit diesen Objekten arbeiten zu können, müssen Verweise erstellt werden. Sie erfahren mehr im Kapitel „Modul einfügen“.

Dieses Skript ist kein allumfassendes Nachschlagewerk, sondern eine Sammlung von Übungen, mit dem Ziel „VBA - Learning by doing“.

Zum Arbeiten mit Excel-Makros gibt es unter dem folgenden Link eine interessante Einführung:

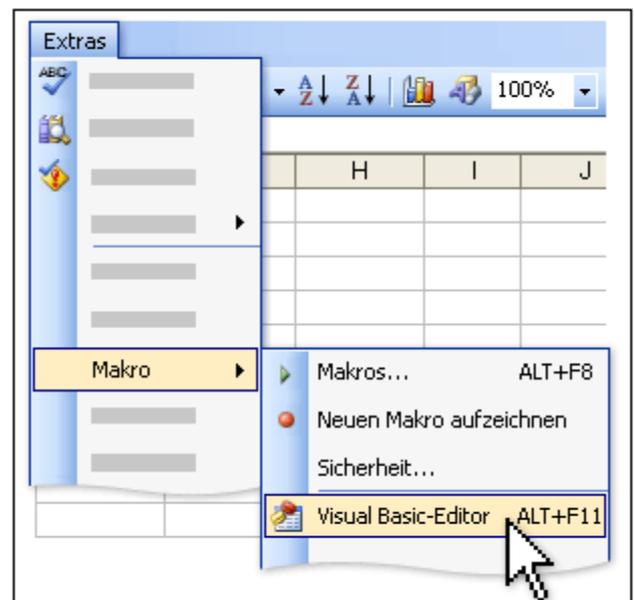
<http://office.microsoft.com/training/training.aspx?AssetID=RC011506201031&ofcresset=1>

**Makro:** Code, der einen bestimmten Effekt verursacht und einen eigenen Namen besitzt

**VBA:** Visual Basic for Applications, die Codesprache für Makros

**Modul:** ein Container zum Speichern von Makros, der an eine Arbeitsmappe angefügt wird

Sie können den VBA-Editor über ALT+F11 öffnen bzw. über Extras / Makro.



Modul erstellen



## Einführung in die VBA-Programmierung

Folgende Fragen werden in diesem Kapitel beantwortet:

- Wie funktioniert die ereignisorientierte Programmierung?
- Was sind Objekte, Klassen, Eigenschaften und Methoden?
- Welche Module gibt es?
- Was sind Formen bzw. Formulare?
- Wie werden Steuerelemente eingesetzt?
- Was ist der Unterschied zwischen Sub - Prozeduren und Function – Prozeduren?

### Ereignisgesteuerte Programmierung

VBA ist eine ereignisgesteuerte Programmiersprache. Das bedeutet, dass beim Eintritt bestimmter Ereignisse Programmcode ausgeführt werden kann.

Wenn Sie ein Benutzerformular erstellt haben, gibt es dort eine Reihe von Ereignissen, bei welchen Programmcode „ausgelöst“ werden kann.

Das Ereignis „Click“ bedeutet, dass der Programmcode aufgerufen wird, wenn auf das jeweilige Objekt (Schaltfläche, Nachschlagefeld u.a.) geklickt wird.

Ereignis	Wird ausgelöst beim
Click	Anklicken einer Schaltfläche u.a.
DbtClick	Doppelklicken einer Schaltfläche u.a.
Enter	Hingehen zu einer Schaltfläche u.a.
Exit	Verlassen einer Schaltfläche u.a.
KeyDown	Drücken einer Taste über der Schaltfläche u.a.
KeyPress	Gedrückthalten der Taste über der Schaltfläche u.a.
KeyUp	Loslassen der Taste über der Schaltfläche u.a.
MouseDown	Drücken der Maustaste über der Schaltfläche u.a.
MouseMove	Bewegen der Maus über die Schaltfläche u.a.
MouseUp	Lösen der Maustaste über der Schaltfläche u.a.

## Ereignisse auf Workbookebene

Es gibt auch eine Reihe von anwendungsspezifischen Ereignissen

Activate	SheetActivate
AddinInstall	SheetBeforeDoubleClick
AddinUninstall	SheetBeforeRightClick
BeforeClose	SheetCalculate
BeforePrint	SheetChange
BeforeSave	SheetDeactivate
Deactivate	SheetFollowHyperlink
NewSheet	SheetPivotTableUpdate
Open	SheetSelectionChange
PivotTableCloseConnection	WindowActivate
PivotTableOpenConnection	WindowDeactivate
	WindowResize

### Open

Der Benutzer öffnet eine Arbeitsmappe. Dabei kann in Excel das Ereignis **Open** ausgelöst

```
Private Sub Workbook_Open()
    Application.WindowState = xlMaximized
End Sub
```

### Eigenschaften

Im Inhaltsverzeichnis des VB-Editors (ALT+F11 / Visual Basic Hilfe) finden Sie eine Auflistung der Ereignisse, Methoden, Eigenschaften (Properties) u.a.

Objekte besitzen bestimmte Eigenschaften.

Hier ein paar Eigenschaften eines Cells-Objektes (Zelle):

```
Worksheets("Sheet1").Cells(5, 3).Font.Size = 14
```

Mehrere Eigenschaften gleichzeitig einstellen:

```
With Worksheets("Sheet1").Cells.Font
    .Name = "Arial"
    .Size = 8
End With
```

### Methoden

Auf Objekte können Methoden angewendet werden.

Methoden beantworten die Frage: "Was kann ich z.B. mit dem Diagramm machen?"

Zu diesen Methoden zählt z.B. ADD.

```
ActiveWorkbook.Charts.Add
```

```
Before:=Worksheets(Worksheets.Count)
```



## Module

Module beinhalten den Programmcode. Sie können in einem Modul viele verschiedene Prozeduren speichern. Es gibt 2 Arten von Modulen. Standardmodule und Klassenmodule.

## Standardmodule

In Standardmodulen (Einfügen / Modul) wird der Programmcode abgelegt, welcher von verschiedenen Formularen aus aufgerufen werden soll. Das Standardmodul ist öffentlich und dadurch innerhalb des Projekts von überall her ansprechbar.

## Klassenmodule

Hier werden neue Objekte definiert, die Methoden und Eigenschaften besitzen. Dadurch kann die Funktionalität von AutoCAD um neue Zeichnungsobjekte erweitert werden. Die Definition wird mit dem Schlüsselwort **New** erreicht.

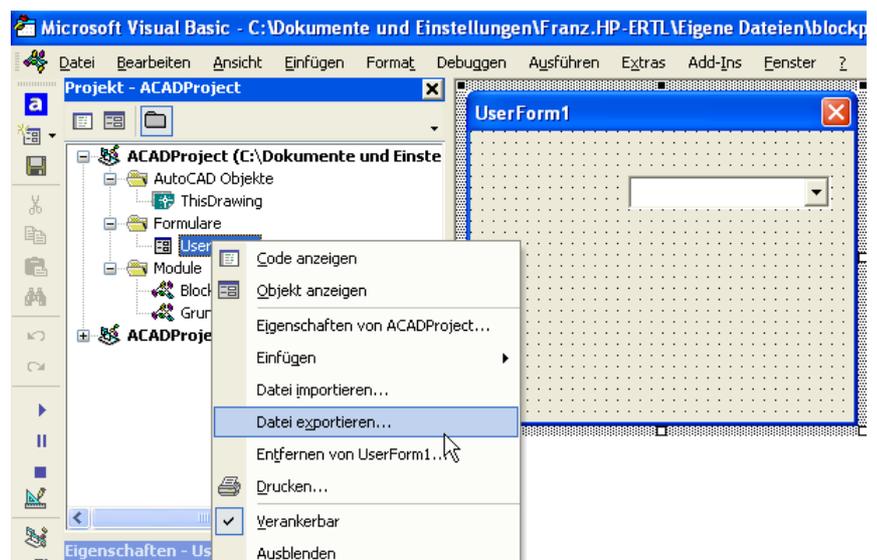
`DIM objTest as New clsMeinObjekt`

In Klassenmodulen können ActiveX-Steuerelemente definiert und API-Funktionen (Advanced Programmers Interface bzw. Programmierung der Windows-Umgebung) aufgerufen werden.

Schauen Sie mal auf der Seite [www.activevb.de](http://www.activevb.de) nach. Dort finden Sie eine tolle Sammlung von API-Funktionen.

## Userform

Es gibt eine Reihe verschiedener Bezeichnungen für Userforms, z.B. Userform, Form, Formular oder das Userformular. Hier wird die Bezeichnung „Formular“ verwendet. Es handelt sich um ein Dialogfenster, das dem Anwender zur Verfügung gestellt wird. Sie können dort Steuerelemente, wie Befehls-Schaltflächen oder Kombinationsfelder platzieren. Forms bzw. Formulare werden im Projekt gespeichert. Sie können die Formulare auch exportieren und in andere Projekte wieder importieren.



## Steuerelemente

Zum Ausführen von Befehlen oder zu Anzeige von Werten können Sie auf der Formularfläche Steuerelemente anordnen, die Sie programmieren können.



Auswahl eines Objekts.



Label: Überschriften, Textinhalte.



Ein Textfeld beinhaltet z.B. berechnete Ausdrücke wie z.B.  $=2*2$  oder berechnete Werte.



Kombinationsfeld: Nachschlagefeld, z.B. für Objekte einer Auflistung. Ein Objekt wird angezeigt.



Listenfeld: Ein Listenfeld kann Objekte einer Auflistung beinhalten. Alle Objekte werden angezeigt.



Checkbox: Das Feld kann die Zustände **Ja/Nein** annehmen = **True/False** = **-1/0**



Optionsfeld: Das Optionsfeld kann mit mehreren anderen Feldern in einem Rahmen kombiniert werden. Es kann in einer Kombination immer nur ein Feld den Wert **True** annehmen.



Umschalter: Der Schalter wird als aktiviert oder nicht aktiviert dargestellt.



Rahmen: Der Rahmen wird häufig für Optionsfelder verwendet. Fügen Sie einen Rahmen ein. Platzieren Sie innerhalb des Rahmens mehrere Optionsfelder. Dadurch werden die Optionsfelder zu einer Gruppe zusammengefasst, so dass nur eine Option aktiviert werden kann.



Befehlsschaltfläche: Zum Ausführen von Programmcode.



Register: Erzeugt Registerseiten positionieren.



Formular mit mehreren Seiten.



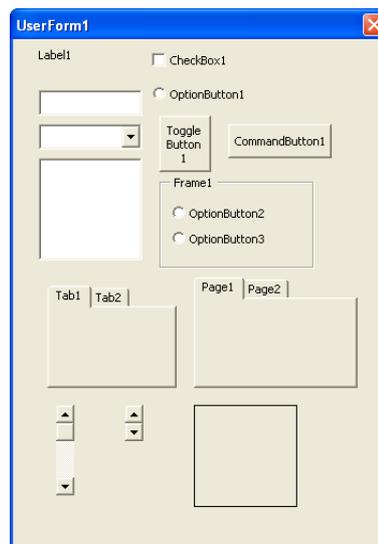
Bildlaufleiste: Rollbalken dienen zum Scrollen von Bildschirmseiten.



Drehfeld: Drehfelder werden zum Drehen von Objekten verwendet



Objekt einfügen: Zum Einfügen von Bildern oder Objekten aus anderen Anwendungen.



## Prozeduren

Es gibt zwei Arten von Prozeduren

- Function-Prozeduren
- Sub-Prozeduren

Der Unterschied liegt hauptsächlich darin, dass von einer Function-Prozedur ein Wert zurückgegeben werden kann, von einer Sub-Prozedur nicht.

### Aufbau einer Sub – Prozedur

Diese Prozedur ist mit dem Schlüsselwort *Public* definiert. Wurde dieses Schlüsselwort innerhalb eines Formulars erstellt, kann die Prozedur innerhalb des gesamten Formulars von allen darin gespeicherten Prozeduren aufgerufen werden. Wurde sie in einem Standardmodul erstellt, kann sie in allen Formularen bzw. aus anderen Modulen heraus aufgerufen werden.

```
Public Sub Prozedurname (Optional Argument1, Optional Argument2 ...)  
    Prozedurcode  
End Sub
```

Die Alternative wäre die Definition mit dem Schlüsselwort *Private*, dann ist die Prozedur nur innerhalb des Moduls oder der Prozedur im Formular aufrufbar, in welcher Sie definiert wurde.

```
Private Sub Prozedurname (Optional Argument1, Optional Argument2 ...)  
    Prozedurcode  
End Sub
```

### Aufbau einer Function – Prozedur

```
Public Function Prozedurname (Optional Argument1, Optional Argument2 ...) as Datentyp  
    Prozedurcode  
End Function
```

Alternativ, wie bei der Sub – Prozedur die Private - Deklaration

```
Private Function Prozedurname (Optional Argument1, Optional Argument2..) as Datentyp  
    Prozedurcode  
End Function
```

## Die Entwicklungsumgebung

Inhalte dieses Kapitels:

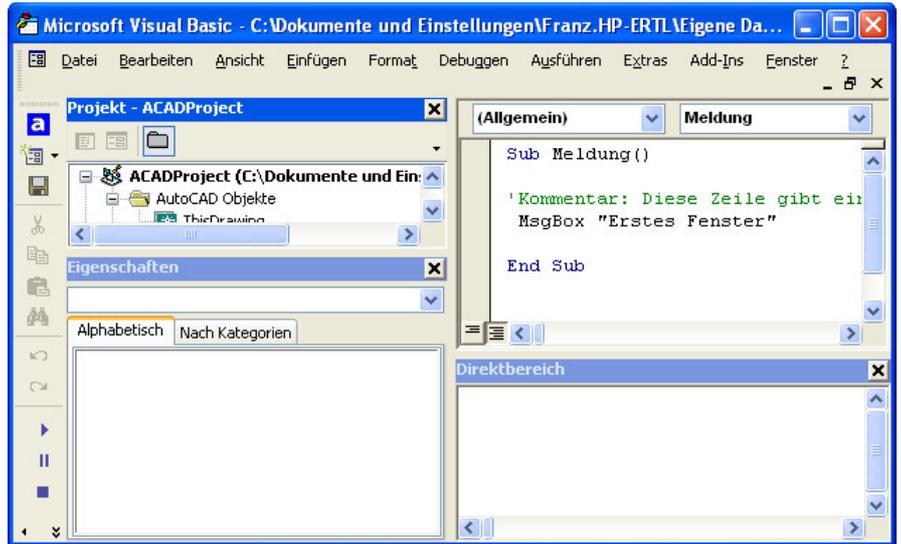
- Bedienung der VBA – Entwicklungsumgebung.
- Was sind Projekte und wie werden sie gespeichert?
- Bedienung des VBA – Editors.
- Umgang mit der Hilfe.
- Die grafischen Objekte in VBA.
- Tricks zum Umgang mit dem Codefenster.
- Wo kann man die Grundeinstellungen ändern?
- Aufrufen von Klassenbibliothek.

### Aufrufen des VBA-Editors

Tastenkombination ALT + F11 oder Extras / Macros



## VBA-IDE (Integrated Developer Environment)



### Projekt speichern

Zum Speichern eines Projekts klicken Sie im Menü Datei "Speichern von..." bzw. die Tastenkombination STRG+S.

Das Projekt umfasst alle Bestandteile des Programms. Die Dateiendung lautet "DVB".

Wenn der Programmcode geändert wurde, fragt AutoCAD beim Beenden, ob das VBA-Projekt ebenfalls gespeichert werden soll. Es geht nicht automatisch.

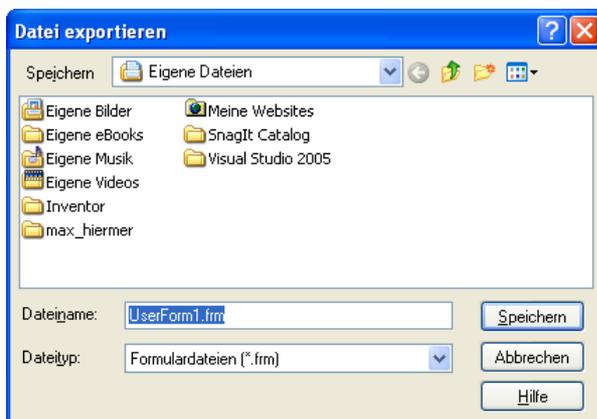


Folgende Objekte können in einem Projekt vorhanden sein:

- AutoCAD Objekte
- Formulare
- Standardmodule
- Klassenmodule

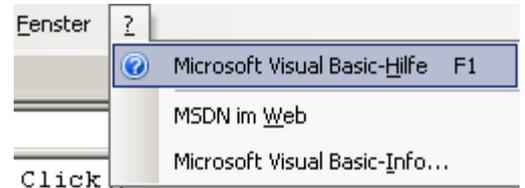
### Datei exportieren/importieren

Mit der Option "Datei importieren..." bzw. exportieren können z. B. einzelne Module oder Formulare einzeln gespeichert oder eingefügt werden.

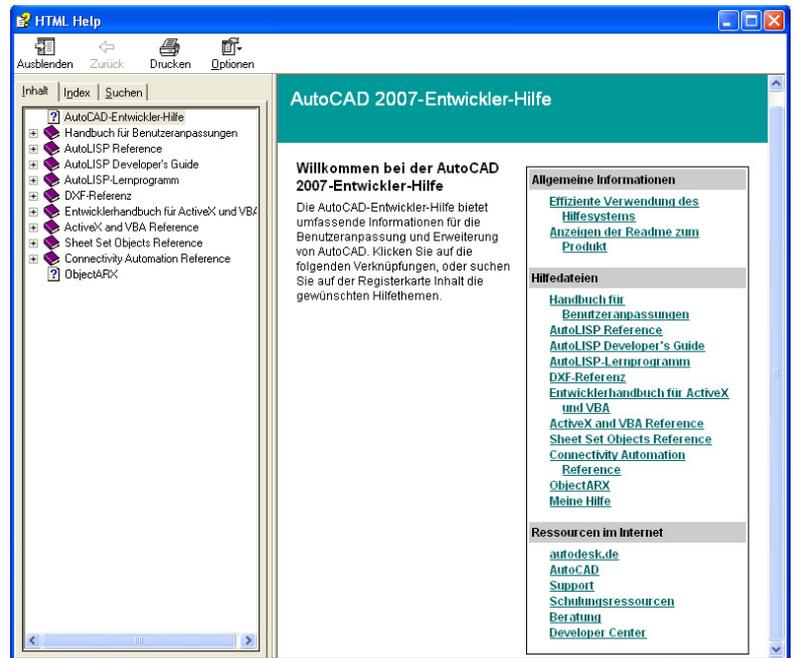


## Hilfe

Im VBA-Editor wählen Sie Microsoft Visual Basic Hilfe F1 oder klicken Sie auf das Objekt oder den Funktionsnamen, zu welchem Sie Hilfe benötigen und drücken Sie die Taste F1. Sie können sich viel Tipparbeit ersparen, indem Sie Programmcode aus der Hilfe in Ihre Prozedur kopieren und Ihren Wünschen entsprechend abändern.



Die Entwicklerhilfe ist sehr gut. Sie können nach Kategorien oder Begriffen suche. Sie finden darin auch eine Reihe fertiger Programmierbeispiele.



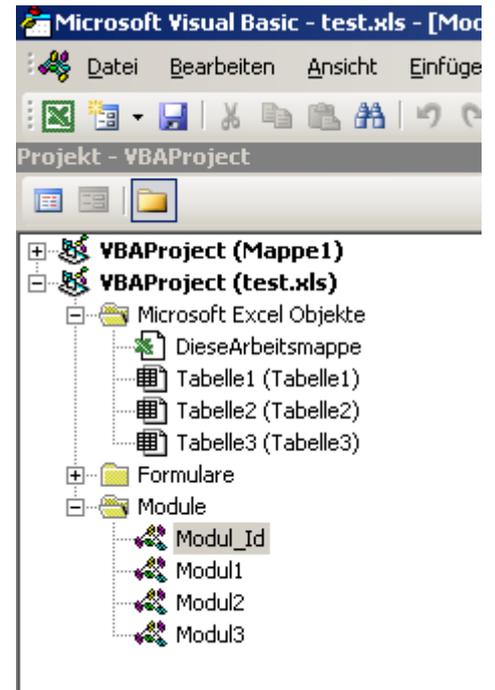
Wenn Sie in der linken Auswahl auf „Entwicklerhandbuch für ActiveX und VBA“ klicken, öffnet sich die rechts dargestellte Auflistung.

- [-] Entwicklerhandbuch für ActiveX und VBA
- [-] ActiveX and VBA Reference
  - [?] Object Model
  - [+] Events
  - [+] Methods
  - [+] Properties
  - [+] Objects
  - [+] Code Examples

## Der Projektextplorer

Menü Ansicht / Projektextplorer. Hier werden alle geladenen Projekte und deren Objekte (Module, Formulare...) angezeigt.

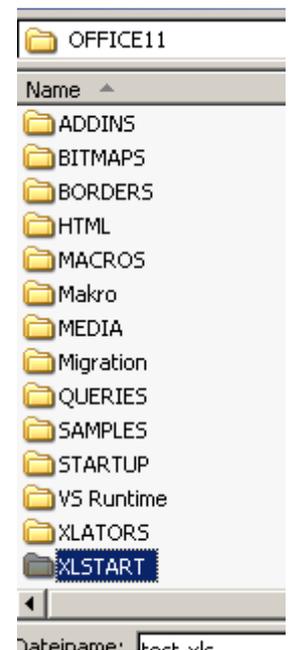
Zum Anzeigen oder Ausblenden des Projektextplorers wählen Sie im Menü Ansicht die Option "Projektextplorer anzeigen" oder drücken Sie STRG+R.



## Projekt automatisch laden

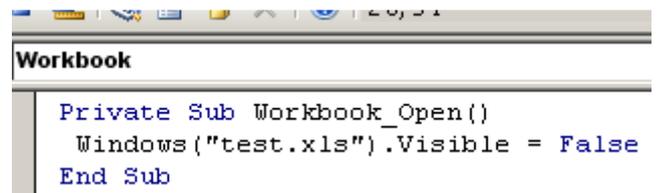
Speichern Sie die Arbeitsmappe im Ordner AutoStart im Programmverzeichnis., dann wird es beim Start automatisch geladen.

Eine kleine Programmroutine versteckt die Arbeitsmappe nach dem Öffnen.



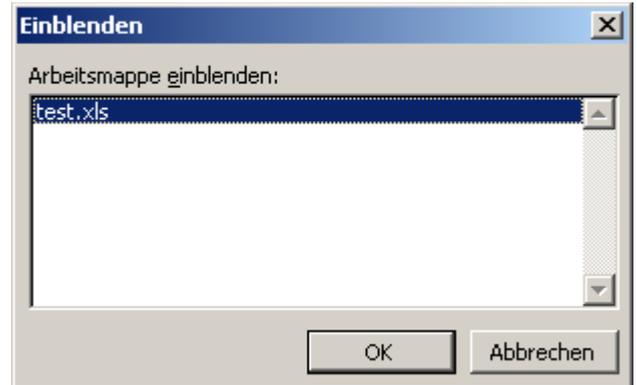
## Arbeitsmappe ausblenden

Windows("Test.xls").Visible = False



## Arbeitsmappe einblenden

Menü Fenster / Arbeitsmappe einblenden.



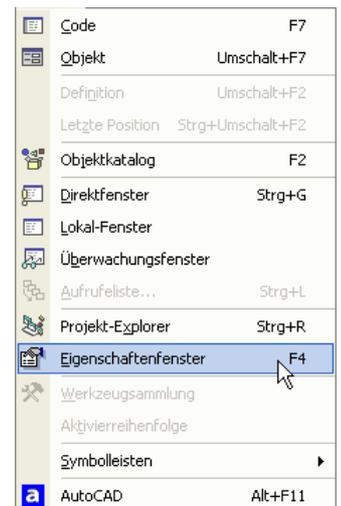
## Das Eigenschaften-Fenster

Die Eigenschaften eines Objekts können über Programmcode während der Laufzeit eingestellt werden, oder über das Eigenschaftenfenster in der Entwurfsansicht des Objekts.

Sollten das Fenster ausgeschaltet sein, kann es über das Menü Ansicht bzw. mit der Taste F4 wieder aktiviert werden.

Das Eigenschaftenfenster zeigt nur Eigenschaften an, die für das gewählte Objekt verfügbar sind.

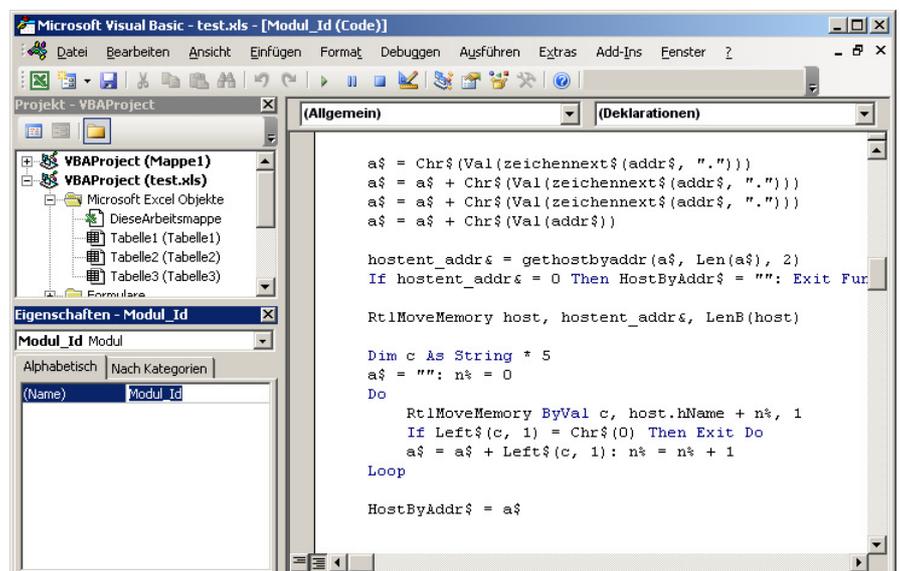
Um z.B. den Formularnamen zu ändern, klicken Sie auf das Formular, und dann auf "UserForm1".



## Das Code-Fenster

Im Codefenster können Sie Ihren Programmcode eintippen. Sie gelangen über das Kontextmenü (Klick mit der rechten Maustaste auf das gewünschte Objekt) oder über das Menü Ansicht in das Codefenster.

Durch Doppelklick auf das Dokument-Objekt „Diese Arbeitsmappe“ oder Formular-Steurelemente gelangen Sie ebenfalls in das Codefenster.

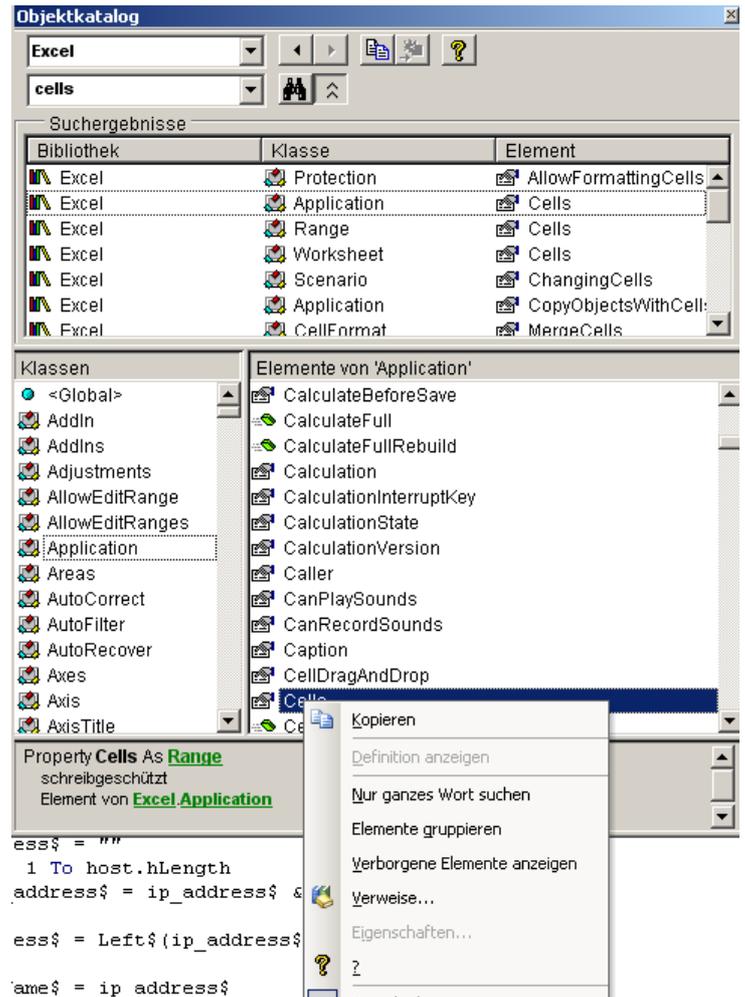


### Funktionskatalog

Drücken Sie in der Code-Ansicht die Taste F2, dann werden die Funktionen aufgelistet.

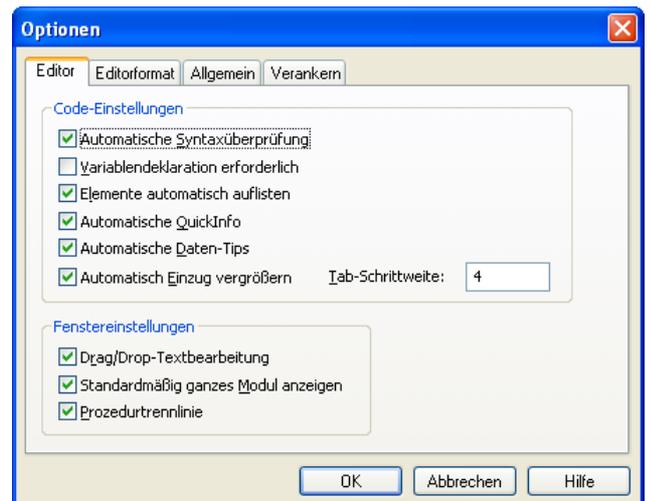
Wählen Sie die Bibliothek „Excel“. Geben Sie darunter den Suchbegriff ein. Klicken Sie in der Ergebnisliste mit der RMT auf „Cells“ und wählen Sie die Hilfe.

Dort finden Sie Beispielprogrammcode.



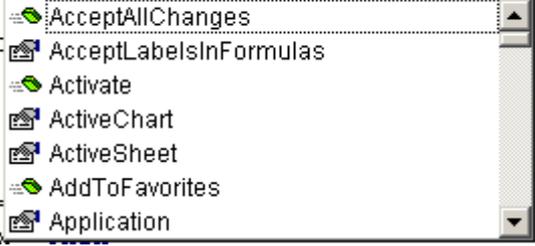
### Einstellungen der Entwicklungsumgebung

Die Optionen des Codefensters, der Formulare, des Projektextplorers und des Eigenschaftsfensters (usw.) können im Menü Extras/Optionen eingestellt werden. In den verschiedenen Registern finden Sie die weiteren Einstellungen.



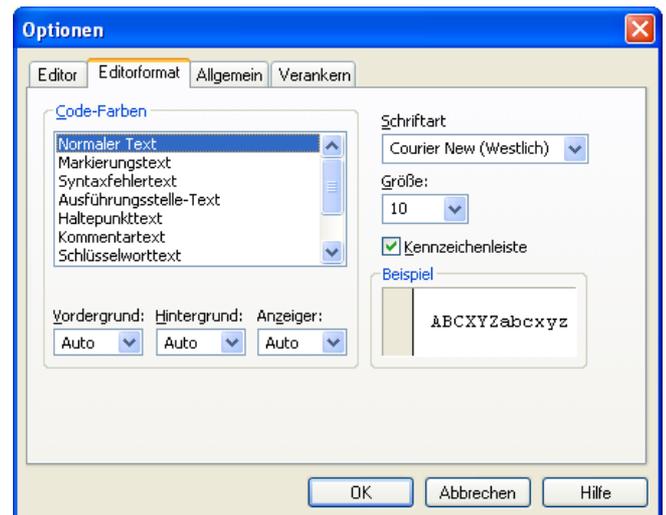
### Editoreinstellungen

Automatische Syntaxprüfung	Prüft auf Schreibfehler im Programmcode
Variablendeklaration erforderlich	Jede Variable, die verwendet wird, muss vorher deklariert werden. Im Modul wird die Zeile „Option explicit“ eingefügt.

<p>Elemente automatisch auflisten</p>	<p>Elemente, die für ein Objekt verfügbar sind, werden während des Schreibens aufgelistet. Ist die Option ausgeschaltet, wird die Auflistung nicht gezeigt.</p> <pre>debug.Print activeworkbook.</pre> <pre> If rss.RecordCount &gt;   MsgBox "Es gibt mehr Exit Sub End If  If rss.RecordCount   If Cells(i, 1) = "</pre> 
<p>Automatische Quickinfo</p>	<p>Beim Einfügen von Funktionen werden die erforderlichen bzw. verfügbaren Argumente angezeigt.</p> <pre>msgbox : MsgBox(<b>Prompt</b>, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult</pre>
<p>Automatische Datentipps</p>	<p>Wenn Sie den Mauscursor während des Programmtests auf eine Variable setzen, wird der Inhalt der Variablen angezeigt.</p> <pre>Function HochDrei(EineZahl) As Double</pre> <pre><b>HochDrei = EineZahl * EineZahl * EineZahl</b></pre> <pre>End Function</pre> <p>EineZahl = 7</p>
<p>Drag and Drop bei der Textbearbeitung</p>	<p>Markierter Text kann bei gedrückter linker Maustaste an eine andere Stelle verschoben werden</p>
<p>Standardmäßig ganzes Modul anzeigen</p>	<p>Alle Funktionen innerhalb eines Moduls werden fortlaufend angezeigt</p>
<p>Prozedurtrennlinie</p>	<p>Zwischen den Prozeduren wird eine Linie eingefügt</p>
<p>Tab-Schrittweite</p>	<p>Stellt den Abstand ein, um den der Cursor nach rechts springt, wenn die Tab-Taste einmal gedrückt wird.</p>

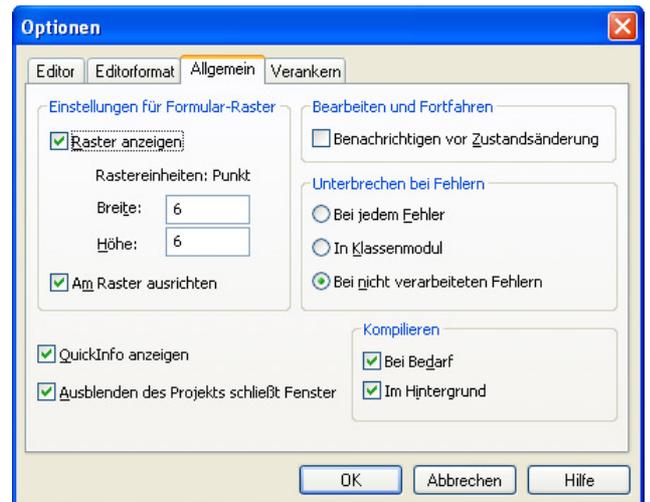
### Editorformat

Hier kann die Vorder- bzw. Hintergrundfarbe, sowie die Schriftart- und -größe für die einzelnen Textarten eingestellt werden



### Allgemeine Einstellungen

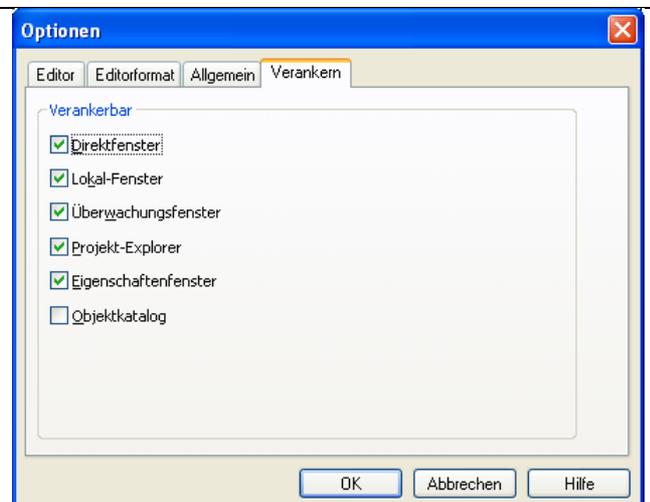
Einstellungen für Raster, Quickinfo, Fehlerbenachrichtigung und Kompilerverhalten.



Raster anzeigen	Das Raster ist als Hilfe beim Anordnen der Steuerelemente auf Formularen gedacht. Sie können einstellen, ob das Raster angezeigt werden soll und wie groß die Rasterabstände sind
Steuerelemente am Raster ausrichten	Sie können einstellen, ob am Raster, ob am Raster ausgerichtet werden soll
Quickinfo anzeigen	Zeigt die Quickinfo für die Symbolleistenflächen. 
Projektausblendung schließt Fenster	Legt fest, ob die Projekt-, UserForm-, Objekt- oder Modulfenster automatisch geschlossen werden, wenn ein Projekt im Projekt-Explorer ausgeblendet wird.
Benachrichtigung vor Zustandsänderung	Legt fest, ob eine Benachrichtigung erfolgt, dass durch die angeforderte Aktion alle Variablen auf Modulebene für ein laufendes Projekt zurückgesetzt werden.
Bei jedem Fehler unterbrechen	Bei jedem Fehler wird für das Projekt der Haltemodus aktiviert,
In Klassenmodul unterbrechen	Bei Fehler im Klassenmodul wird der Haltemodus aktiviert.
Bei nichtverarbeiteten Fehlern unterbrechen	Wenn eine Fehlerbehandlungsroutine läuft, wird der Fehler behandelt, ohne den Haltemodus zu aktivieren. Der Haltemodus wird aktiviert, wenn keine Fehlerbehandlungsroutine vorhanden ist.
Kompilieren bei Bedarf	Code wird bei Bedarf kompiliert wird, wodurch die Anwendung schneller gestartet werden kann (sonst beim Start)
Im Hintergrund kompilieren	Leerlaufzeit während der Laufzeit wird für die Kompilierung des Projekts im Hintergrund verwendet.

### Verankern der Fenster

Legt die Fenster auf der linken Seite der Entwicklungsumgebung ab und verankert sie dort.



## Makros

Inhalte dieses Kapitels:

- VBA – Befehle.
- Was ist ein Makro?
- Wie erstelle ich ein einfaches Makro?
- Wie rufe ich das Makro auf?
- Wie erstelle ich eine Schaltfläche zum Aufrufen des Makros?
- Wie rufe ich Prozeduren aus AutoCAD auf?



## Makro

Der Begriff Makro umschreibt die Aneinanderreihung verschiedener Befehle oder Funktionen zu einem neuen Funktionskomplex.

## Makrodialog

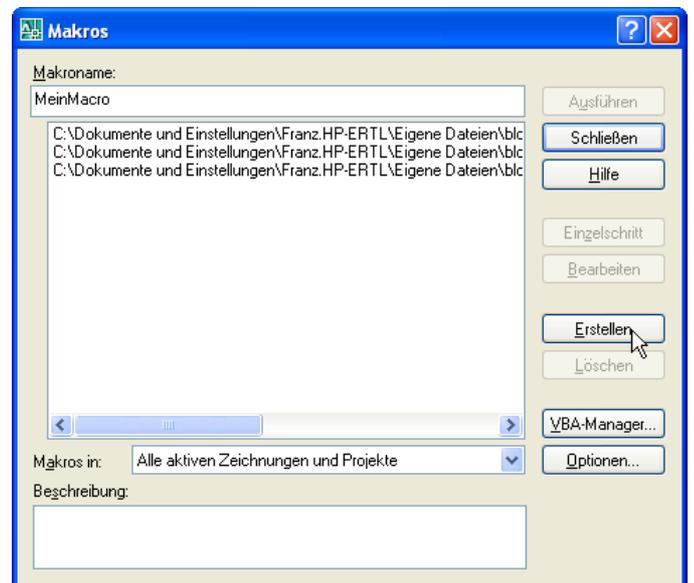
Den Makro-Dialog finden Sie im Menü Extras/Makros... in AutoCAD und Office-Programmen, die via VBA programmierbar sind (MS Word, Excel, Access, Powerpoint, Outlook ...).

## Makro erstellen

Makros werden automatisch in einem globalen Modul hinterlegt. Das Projekt, in welchem das Makro enthalten ist, wird nicht automatisch mit der Zeichnung gespeichert, sondern bedarf der manuellen Speicherung, Datei/Speichern.

Menü	Tastenkombination
Werkzeuge/Makros...	ALT+F8

1. Geben Sie dem Makro den Namen „MeinMakro“

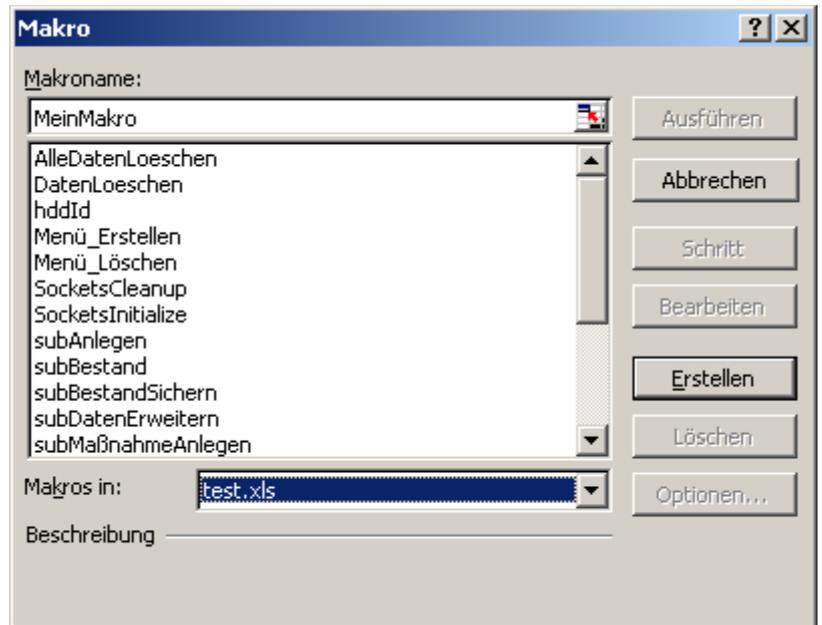


2. Klicken Sie auf „Erstellen“
3. Bestätigen Sie mit OK und bestätigen Sie, wenn Sie gefragt werden, ob die Zeichnung gespeichert werden soll

```
Sub MeinMacro()  
End Sub
```

4. Fügen Sie den folgenden Programmcode zwischen die beiden Zeilen ein:

```
Sub MeinMacro()  
    MsgBox „Das ist mein  
    erstes Programm“  
End Sub
```



5. Schließen Sie den VBA – Editor
6. Rufen Sie das Menü Werkzeuge/Makro... auf oder betätigen Sie ALT + F8
7. Wählen Sie das Makro „MeinMakro“ und klicken Sie auf „Ausführen“.



### Makro – Dialogfenster

Ausführen: Ausführen des ausgewählten Makros.

Schließen: Schließt das Dialogfenster.

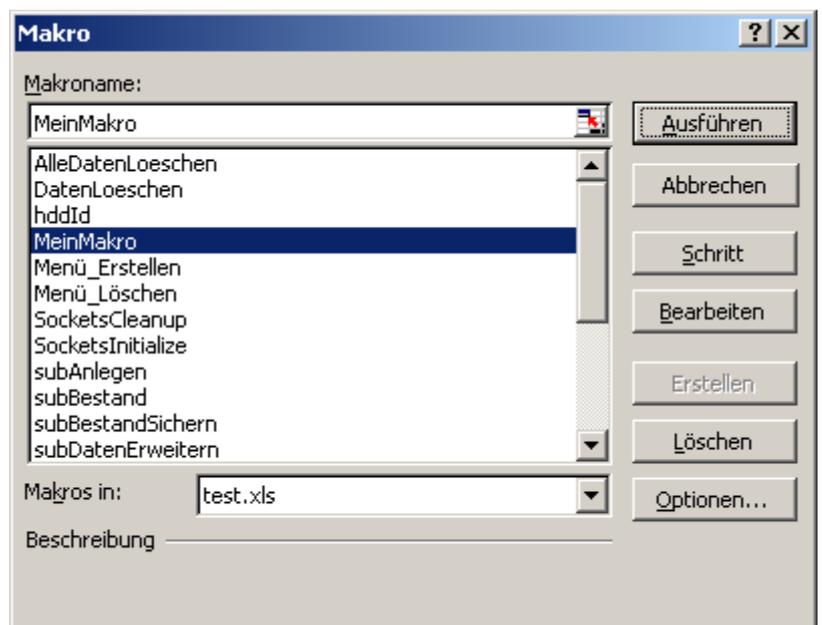
Hilfe: Ruft die Hilfe auf.

Einzelschritt: Führt das Macro zeilenweise aus.

Bearbeiten: Ändern des Macros.

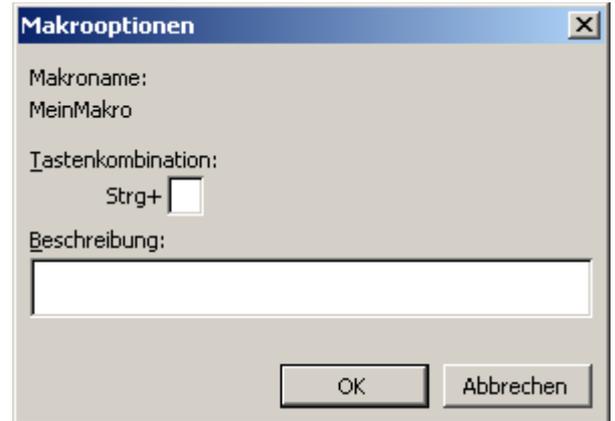
Erstellen: Erstellt neues Macro.

Löschen: Löscht das Macro.



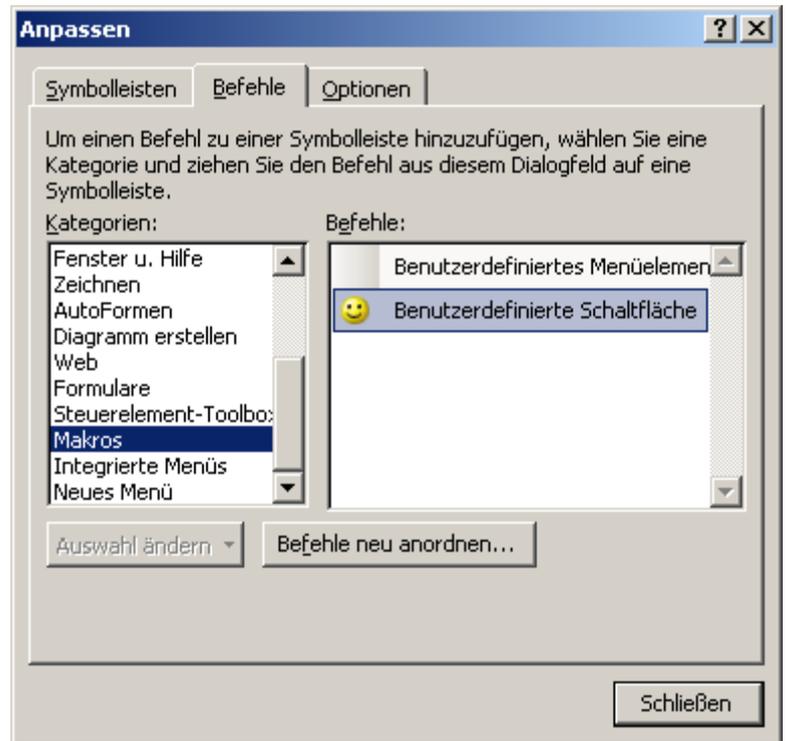
### Makro über Tastaturbefehl Aufrufen

Drücken Sie ALT+F8.  
 Wählen Sie das gewünschte Makro.  
 Klicken Sie auf „Optionen“.  
 Geben Sie ein Tastenkürzel ein.

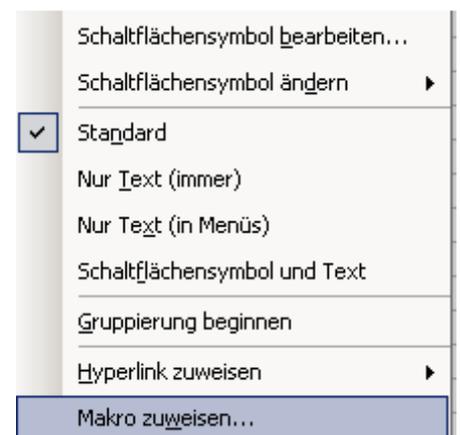
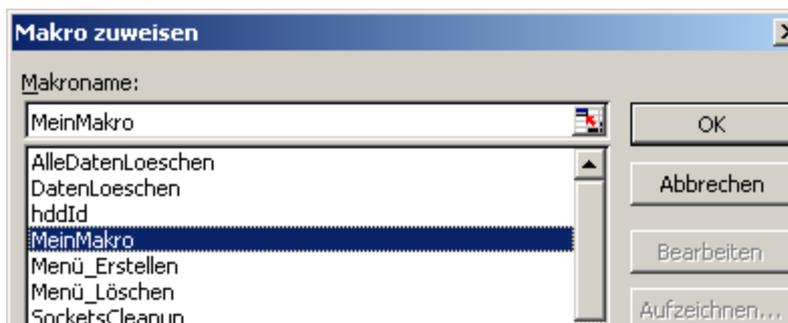


### Übung 1: Schaltfläche zum Aufrufen des Macros

1. Klicken Sie auf mit der rechten Maustaste auf ein beliebiges Werkzeugsymbol.
2. Wählen Sie die Option „Anpassen“.
3. Wählen Sie in den Kategorien „Makros“.
4. Ziehen Sie bei gedrückter linker Maustaste das Makro in die gewünschte Symbolleiste.
- 5.

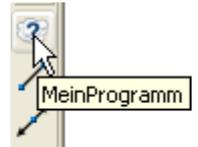


6. Klicken Sie mit der RMT auf das Symbol.
7. Wählen Sie „Makro zuweisen“.
8. Wählen Sie das eben erstellte Makro.
9. Testen Sie die Schaltfläche.



### Aufruf des Makros

Klicken Sie auf den neuen Befehl im Werkzeugkasten „Zeichnen“.



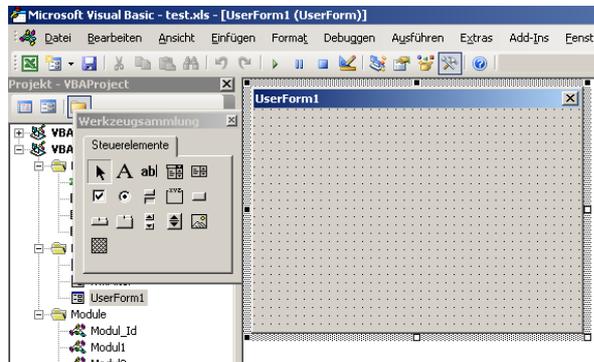
### Projekte erstellen

Betätigen Sie die in AutoCAD die Tastenkombination Alt+F11 (bei gedrückter Alt-Taste die F11-Taste drücken) oder wählen Sie im Menü Werkzeuge/Makro/Visual Basic Editor ...

Menü	Tastaturbefehl	Tastenkombination
Werkzeuge/Makro/Visual Basic Editor..	VBAIDE	ALT+F11

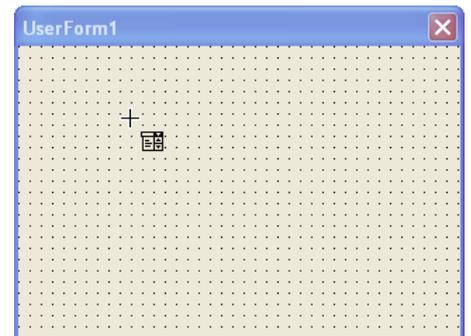
### Userform erstellen

In der Entwicklungsumgebung Menü Einfügen/Userform oder Kontextmenü des Projektextplorers/Einfügen/Userform...



### Einfügen von Steuerelementen

Um ein Steuerelement einzufügen, klicken Sie es zuerst in der Werkzeugsammlung an, um es zu aktivieren. Danach zeigen Sie im Formular an die Position, wo Sie es einfügen möchten.



## Übung 2: Platzieren von weiteren Steuerelementen

1. Klicken Sie auf die Befehlsschaltfläche in der Werkzeugsammlung 
2. Klicken Sie im Formular an die gewünschte Position (CommandButton1)
3. Klicken Sie auf das Textfeldsteuerelement in der Werkzeugsammlung 
4. Klicken Sie in den oberen Bereich es Formulars
5. Klicken Sie auf das Textfeldsteuerelement in der Werkzeugsammlung und klicken Sie unterhalb des ersten Textfeldsteuerelements auf das Formular
6. Doppelklicken Sie auf CommandButton1.
7. Fügen Sie den Programmcode ein.

```
Private Sub btnBerechnen_Click()
    TextBox2.Value = TextBox1 * TextBox1
End Sub
```

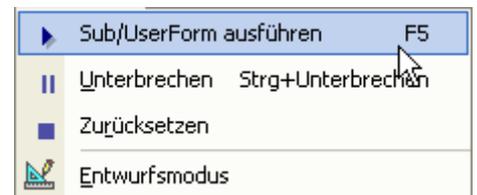
**TextBox2.Value** Wert aus TextBox2

**Val(TextBox1)** Zahlenwert aus TextBox1, falls ein Text eingegeben wird, entsteht ein Fehler.

### Ausführen des Programms

Menü Ausführen.

Zum Ausführen des Programms innerhalb des VBA-Editors, klicken Sie auf die Schaltfläche  (oder Taste F5).

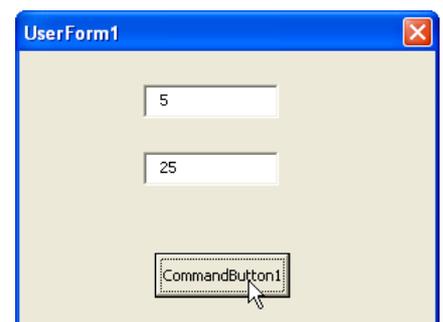
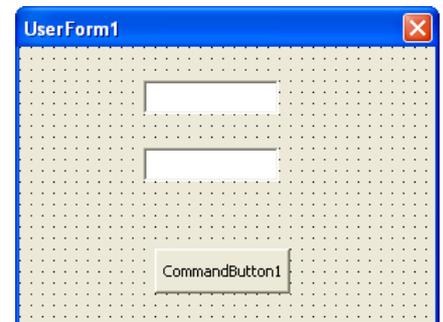


### Debuggen oder schrittweises Ausführen

Durch Betätigung der Taste F8 können Sie den Programmcode schrittweise durchlaufen (Debuggen). Sie können die Werte, welche Variablen zu dieser Zeit haben einsehen (einfach Mauscursor auf die Variable halten).

Schreiben Sie die Zahl 5 in die obere Textbox und klicken Sie auf CommandButton1.

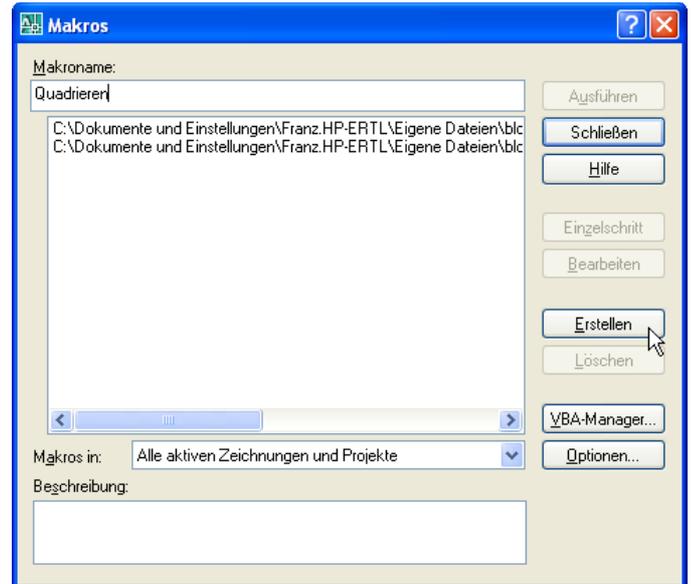
Da dieses Vorgehen nur während der Programmerstellung und zum Testen geeignet ist, finden Sie nachfolgend bzw. oben die Anleitung, wie Sie dieses Formular aus Excel aufrufen können. Klicken Sie auf Ausführen  oder drücken Sie die Funktionstaste F5



## Formulare aus Excel starten

1. Erstellen Sie in Excel ein neues Makro mit dem Namen „**Quadrieren**“
  - ALT + F8
  - Neuen Namen eingeben: Quadrieren
  - Schaltfläche Erstellen anklicken
  - mit OK bestätigen

Bestätigen Sie auch das folgende Fenster mit OK.



2. Geben Sie in der Modulzeile folgenden Code ein

Auf das Objekt UserForm1 wird die Methode „Anzeigen“ = show angewendet.

```
Sub Quadrieren()  
    userform1.show  
End Sub
```

3. Geben Sie in AutoCAD an der Befehlszeile folgende Befehle ein:

Befehl: **-vbarun** ↵

Makroname: **Quadrieren** ↵

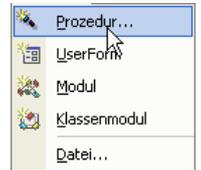
### Übung 3: Schaltfläche zum Aufrufen des Formulars

Gehen Sie so vor wie auf Seite 16 ff. beschrieben.

### Übung 4: Function-Prozedur erstellen

Wie bereits beschrieben kann eine Funktion ein Wert zurückgeben. Dies geschieht über den Funktionsnamen.

- Einfügen/Userform
- Erstellen einer Befehlsschaltfläche
- Erstellen von 2 Textfeldern
- Rufen Sie im Codefenster (F7) Einfügen/Prozedur... auf.
- Geben Sie den Funktionsnamen ein und wählen Sie „Function“.
- Übergabevariable festlegen
- Die Übergabevariable beinhaltet die zu berechnende Zahl, die von der aufrufenden Prozedur geliefert wird.



Ergänzen Sie die Function wie rechts dargestellt. An die Function HochDrei wird der Inhalt der Variablen EineZahl übergeben. Der Inhalt dieser Variablen wird 2 Mal mit sich selbst multipliziert.

```
Public Function HochDrei(EineZahl)
    HochDrei = EineZahl * EineZahl * EineZahl
End Function
```

### Festlegen von Formaten

Bei Berechnungen ist es wichtig, dass das richtige Format für die Funktion festgelegt wird. Die Funktion HochDrei wurde hier ergänzt. Das Funktionsergebnis ist eine Single-Zahl, genauso wie die Zahl, die von der aufrufenden Prozedur kommt.

```
Public Function HochDrei(EineZahl As Single) As Single
    HochDrei = EineZahl * EineZahl * EineZahl
End Function
```

### Hinweis:

Ohne Angabe eines Datentyps wird automatisch Variant verwendet. Dieser Datentyp kann alle Inhalte speichern, benötigt aber viel Speicherplatz.

## Single (Datentyp)

Ein Datentyp, der Fließkommavariablen mit einfacher Genauigkeit als 32-Bit-Fließkommazahlen (4 Bytes) speichert. Der Wertebereich ist -3,402823E38 bis -1,401298E-45 für negative Werte und von 1,401298E-45 bis 3,402823E38 für positive Werte. Das Typkennzeichen Ausrufezeichen (!) steht in Visual Basic für den Datentyp Single.

## Programmaufruf über CommandButton1

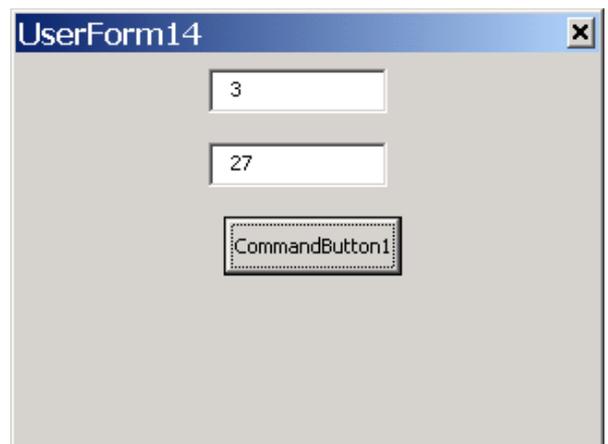
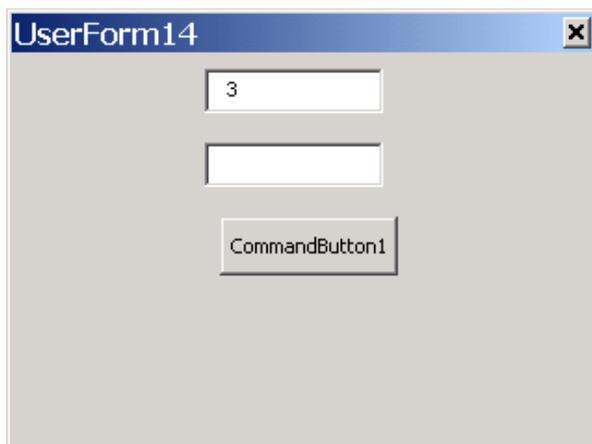
- Doppelklick auf CommandButton1
- Geben Sie den Code wie unten dargestellt ein.

```
Private Sub CommandButton1_Click()
    TextBox2 = HochDrei(Val(TextBox1.Value))
End Sub
```

Der Textbox2 wird das Ergebnis aus der Funktionsberechnung zugewiesen.

## Testen des Formulars:

1. Klicken Sie auf die Schaltfläche Ausführen  (oder F5).
2. Geben Sie in Textbox1 die Zahl 3 ein und klicken Sie auf CommandButton1.



Schaut es so aus wie oben? Super!

## Beschreibung des Programmcodes

```
TextBox2 = HochDrei(Val(TextBox1))
```

Der Wert in Textbox2 entspricht dem Ergebnis aus der Funktion HochDrei.

An die Funktion HochDrei wird der Wert aus Textbox1 übergeben.

Weil dort auch ein Text stehen könnte, wurde die Funktion „VAL“ vorangestellt. Diese Funktion wandelt Textformat in Zahlenformat um. Der Text „1.2“ wird zur Zahl 1,2. Der Text „Walter“ wird zu 0.

Achten Sie darauf, dass in VBA ein Punkt als Dezimaltrennzeichen steht, nicht ein Komma.

```
Textbox2 = Val(„2.3“)
```

Inhalt von Textbox2 = 2,3

Die Standardeigenschaft des Textbox-Steuerelements ist „Value“ also der Inhalt. Das Standardereignis muss man nicht angeben, aber man kann das Programm grundsätzlich leichter lesen, wenn die jeweilige Eigenschaft angegeben wird.

```
Textbox2.Value = HochDrei(Val(Textbox1.Value))
```

‘oder

```
Textbox2 = Hochdrei(Val(Textbox1))
```

Die Funktion HochDrei berechnet

```
HochDrei = EineZahl*EineZahl*EineZahl
```

und speichert das Ergebnis im Namen der Funktion. Dadurch kann das Ergebnis an die aufrufende Stelle zurückgegeben werden.

## Sprachelemente in VBA

Inhalt dieses Kapitels:

- Was sind Variablen?
- Welche Namen dürfen für Variablen und Objekte verwendet werden?
- Wie definiert man Kommentare?
- Wie deklariert man Variablen?
- Wie können Datentypen konvertiert werden?
- Was bedeuten die Schlüsselwörter Private und Public?
- Was sind Arrays?
- Welche Entscheidungsstrukturen gibt es?
- Was sind Schleifen?
- Wie rufe ich die Klassenbibliothek auf?

### Namen und Benennung

Befolgen Sie die folgenden Regeln, wenn Sie Prozeduren, Konstanten, Variablen und Argumente in einem Visual Basic-Modul benennen:

Die Namen dürfen nicht länger als 255 Zeichen sein, Steuerlemente, Formulare, Module und Klassen dürfen nicht mehr als 40 Zeichen haben.

*Tipp: Verwenden Sie möglichst kurze Namen.*

## Namenswahl: Einschränkungen

- kein Leerzeichen, keinen Punkt (.), Ausrufezeichen (!) oder die Zeichen @, &, \$, # am Ende des Namens weisen einen Datentyp zu
- 1. Zeichen ist ein Buchstabe
- Im Allgemeinen sollten Sie keine Namen verwenden, die bereits durch Funktionen, Anweisungen und Methoden in Visual Basic verwendet werden, da auf diese Weise die Funktionalität des entsprechenden Schlüsselworts in der Sprache beeinträchtigt wird. Wenn Sie z.B. eine Variable mit dem Namen Left verwenden, können Sie die Left-Funktion nur mit VBA.Left aufrufen.
- Sie können Namen innerhalb des gleichen Gültigkeitsbereichs nicht wiederholen. Sie können z.B. nicht zwei Variablen mit dem Namen Alter innerhalb der gleichen Prozedur deklarieren. Sie können aber eine private Variablen mit dem Namen Alter und eine Variable auf Prozedurebene mit dem Namen Alter innerhalb des gleichen Moduls deklarieren.

Visual Basic berücksichtigt die Groß-/Kleinschreibung nicht, behält jedoch die Schreibweise der Anweisung bei, mit der der Name deklariert wurde.

## Kommentare

Programmzeilen mit einem vorangestellten Hochkomma (Shift + # = ' ) werden nicht ausgeführt. Sie werden im Code grün dargestellt und beinhalten Beschreibungen.

Beschreiben Sie Programmteile, sofern nicht sofort erkennbar ist, was der Programmcode bewirkt. Sie haben es später leichter, wenn Programmänderungen erforderlich sind bzw. wenn ein anderer Programmierer Ihren Programmcode ändert.

## Namenskonventionen für Variablen

Es gibt auch bei der Benennung der Variablen Konventionen, die nicht bindend sind, an die sich aber viele Programmierer halten, um die Lesbarkeit des Programmcodes zu erleichtern:

Die ersten 3 Zeichen von Variablen sollten den Datentyp kennzeichnen. Dadurch sieht man an beliebiger Stelle des Programmcodes, welchen Datentyp die Variable hat.

Kürzel	Datentyp	Beispiel
bln	Boolean	Dim blnTest as Boolean
byt	Byte	Dim bytTest as Byte
con	Constant	Const conPi = 3.1415926
cur	Currency	Dim curBetrag as Currency
dtm	Date	Dim dtmTag as Date
dec	Dezimal	Dim decTest as Variant
dbl	Double	Dim dblPunktX as Double
int	Integer	Dim inti as Integer
lng	Long	Dim lngTest as Long
obj	Objekt	Dim objBlock as Object
sng	Single	Dim sngPunktY as Single
str	String	Dim strAttribut as String
var	Variant	Dim varTest as Variant

## Geschützte Namen

Mit folgenden Funktionen können Daten in einen anderen Typ umgewandelt werden. Die Liste ist nicht komplett. Diese und viele andere Begriffe dürfen nicht als Namen verwendet werden.

- Chr
- Format
- Lcase
- Ucase
- DateSerial
- DateValue
- CInt
- CLng
- CSng
- CStr
- Cvar
- Fix
- Int
- Day
- Month
- Weekday
- Year
- Hour
- Minute
- Second
- Val

## Variablen deklarieren

Variablen sind Platzhalter, die während des Programmablaufs mit Werten gefüllt werden. Sie werden nach Ihrer Gültigkeitsdauer und dem Typ der zu speichernden Daten unterschieden. Die Variablendeklaration kann mit unterschiedlichen Schlüsselwörtern erfolgen.

### Dim

Zur Deklaration von Variablen verwenden Sie grundsätzlich die Dim-Anweisung.

In dem folgenden Beispiel werden verschiedene Variablen erstellt und die Datentypen zugewiesen.

```
Dim inti as Integer
Dim varName as Variant
Dim dblPunkt as Double
Dim objLayer as AcadLayer
```

Oder in einer Zeile, um Platz zu sparen:

```
Dim strTxt1 as String, strTxt2 as String, strTxt3 as String
```

Im folgenden Beispiel ist die Variable dbiZahl nur innerhalb der Funktion Berechnen sichtbar. Der Variablenname kann auch in einer zweiten und dritten Prozedur eingesetzt werden und dort andere Inhalte speichern.

```
Private Sub Berechnen()
Dim dbiZahl as Double
....
End Sub
```

### Public für öffentliche Variablen

Erfolgt die Deklaration der Variablen zu Beginn eines Moduls im Deklarationsabschnitt, ist die Variable auf Modulebene sichtbar. Erfolgt die Deklaration im Standardmodul mit der Public-Anweisung, ist die Variable im gesamten Projekt sichtbar.

Z.B. Verweise auf Anwendungen oder auf Objekte in Excel, können in öffentlichen Variablen gespeichert werden.

Im folgenden Beispiel ist die Variable appWord sowohl in Prozedur1 als auch in Prozedur2 sichtbar.

```
Option explicit
Public appExcel as Word.Application

Private Sub Prozedur1()
...
End Sub

Private Sub Prozedur2()
...
End Sub
```

### Gültigkeitsbereich von Variablen

Wenn der Gültigkeitsbereich einer Variablen auf eine Prozedur beschränkt ist, nennt man sie lokal. Die Werte in Variablen sind nur so lange verfügbar, bis die Prozedur durchlaufen ist.

### Lebensdauer von Variablen

Die Lebensdauer von Variablen gibt an, wie lange der Wert in der Variablen gespeichert bleibt. Variablen, die mit der DIM- oder PRIVATE-Anweisung innerhalb von Prozeduren gespeichert wurden, verlieren Ihre Inhalte, nachdem die Prozedur beendet wurde. Der reservierte Speicherplatz wird wieder freigegeben. Benötigen Sie Variablen, die nach dem Durchlaufen der Prozedur den Wert behalten, der ihnen zugewiesen wurde, verwenden Sie die STATIC-Anweisung.

## Static

Statische Variablen werden mit dem Schlüsselwort **STATIC** definiert. Sie behalten Ihren zugewiesenen Wert während der Laufzeit des Programms.

Werte, die über öffentliche Variablen gespeichert werden, können häufig auch in **STATIC**-Anweisungen definiert werden. Sie reduzieren damit die Anzahl der öffentlichen Variablen.

## Konstanten und deren Einsatz

Konstanten sind ebenfalls Platzhalter. Ihnen werden aber während der Laufzeit keine neuen Werte zugewiesen. Der Inhalt des Platzhalters ist konstant. Soll der Wert geändert werden, ist ein Eingriff in den Programmcode erforderlich.

Z.B. die Kreiszahl **PI** sollte als Konstante definiert werden, weil sie sich kaum ändern wird. Auch die Mehrwertsteuer kann als Konstante hinterlegt werden, obwohl sie sich immer wieder ändert.

Konstanten werden in einem Modul oder einem Formularmodul definiert. Sie stehen am Anfang des Moduls, hinter dem Ausdruck *Option explicit* (Variablendeklaration erforderlich)

Kreiszahl **PI**:

```
Option Explicit
Public Const dblPi = 3.14159265358979
```

## Option explicit

**Option explicit** steht am Anfang des Moduls. Die Anweisung bewirkt, dass Variablen mit **Dim**, **Private**, **Public**, **ReDim** oder **Static explicit** deklariert werden müssen.

Wenn Sie einen nicht deklarierten Variablennamen verwenden, tritt ein Fehler auf. Wenn Sie die Anweisung **Option Explicit** nicht verwenden, haben alle nicht deklarierten Variablen, wenn nicht anders angegeben, den Typ **VARIANT**.

**Sie sollten immer mit dieser Einstellung arbeiten. Jeder Schreibfehler im Programmcode wird sonst als neue Variable verstanden.**

## Datentypen und Wertebereiche

Datentyp	Speicherbedarf	Wertebereich
Byte	1 Byte	0 bis 255
Boolean	2 Bytes	True oder False
Integer	2 Bytes	-32.768 bis 32.767
Long (lange Ganzzahl)	4 Bytes	-2.147.483.648 bis 2.147.483.647
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Bytes	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Double (Gleitkommazahl mit	8 Bytes	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte;





## Date

Variablen vom Datentyp Date werden als 64-Bit-Gleitkommazahlen (8 Bytes) nach IEEE gespeichert und können ein Datum im Bereich vom 01. Januar 100 bis zum 31. Dezember 9999 und eine Uhrzeit im Bereich von 0:00:00 bis 23:59:59 speichern. Jeder gültige Wert eines Datums- oder Zeitliterals kann einer Variablen vom Datentyp Date zugewiesen werden. Eine Datumsangabe im Programmcode muss durch das Zeichen (#) eingeschlossen sein, zum Beispiel: #01.01.1993# oder #1 Jan 93#.

Variablen vom Datentyp Date verwenden zur Darstellung des Datums das auf dem Computer eingestellte kurze Datumsformat. Zeitangaben werden mit dem auf dem Computer eingestellten Zeitformat (entweder 12- oder 24-Stunden) dargestellt.

Beim Umwandeln anderer numerischer Datentypen in Werte des Datentyps Date repräsentieren die Vorkommastellen das Datum und die Nachkommastellen die Uhrzeit. Die Zeit wird als Bruchteil eines Tages dargestellt. Mitternacht entspricht dem Wert 0, und Mittag entspricht den Nachkommawert 0,5 (0,5 Tage). Negative ganze Zahlen repräsentieren ein Datum vor dem 30. Dezember 1899.

## Object

Variablen vom Datentyp Object werden als 32-Bit-Adressen (4 Bytes) gespeichert, die auf Objekte in einer Anwendung verweisen. Einer Variablen, die als Object deklariert wurde, kann anschließend mit der Set-Anweisung ein Verweis auf jedes von der Anwendung erzeugte Objekt zugewiesen werden.

Anmerkung Obwohl eine Variable, die mit dem Datentyp Object deklariert wurde, einen Verweis auf jedes beliebige Objekt enthalten kann, erfolgt die Verbindung mit dem Objekt, auf das verwiesen wurde, immer zur Laufzeit (Binden zur Laufzeit). Sie können eine frühere Bindung (Binden zur Kompilierungszeit) erzwingen, indem Sie den Objektverweis einer Variablen zuweisen, die durch den Namen einer Klasse deklariert wurde.

## String

Es gibt zwei Arten von Zeichenfolgen: Zeichenfolgen variabler Länge und Zeichenfolgen fester Länge. Zeichenfolgen variabler Länge können bis zu 2 Milliarden (oder  $2^{31}$ ) Zeichen enthalten.

Zeichenfolgen fester Länge können 1 bis etwa 64 KB ( $2^{16}$ ) Zeichen enthalten.

Anmerkung Zeichenfolgen fester Länge mit dem Attribut Public können in Klassenmodulen nicht verwendet werden.

Die Codes für Zeichen vom Datentyp String liegen im Bereich von 0 bis 255 (einschließlich). Die ersten 128 Zeichen (0 bis 127) entsprechen den Buchstaben und Symbolen auf einer US-amerikanischen Standardtastatur. Diese ersten 128 Zeichen stimmen mit den im ASCII-Zeichensatz definierten Zeichen überein. Die zweiten 128 Zeichen (128 bis 255) sind Sonderzeichen, z.B. Buchstaben aus internationalen Alphabeten, Akzentzeichen, Währungssymbole und Symbole für mathematische Brüche. Das Typkennzeichen für String ist das Dollarzeichen (\$).

## Benutzerdefinierter Datentyp

Jeder Datentyp, den Sie mit der Type-Anweisung definieren. Benutzerdefinierte Datentypen können ein oder mehrere Elemente eines beliebigen Datentyps, eines Datenfeldes oder eines bereits bestehenden benutzerdefinierten Datentyps enthalten. Beispiel:

Type Typ1

Name1 As String ' String-Variable für Namen.

Geburtstag As Date ' Date-Variable für Geburtstag.

Geschlecht As Integer ' Integer-Variable für Geschlecht, 0 weiblich, 1 männlich.

End Type

## Variant

Der Datentyp Variant ist der Datentyp für alle Variablen, die nicht explizit (durch eine Anweisung wie Dim, Private, Public oder Static) als anderer Datentyp deklariert werden. Für den Datentyp Variant gibt es kein Typkennzeichen.

Variant ist ein besonderer Datentyp, der beliebige Daten mit Ausnahme von String-Daten fester Länge und benutzerdefinierten Typen enthalten kann. Ein Variant kann auch die speziellen Werte Empty, Error, Nothing und Null enthalten. Mit der Funktion VarType oder TypeName können Sie festlegen, wie die Daten in einer Variablen vom Datentyp Variant interpretiert werden.

Als numerische Daten sind beliebige Ganzzahlen oder reelle Zahlen im Bereich von -1,797693134862315E308 bis -4,94066E-324 für negative Werte und von 4,94066E-324 bis 1,797693134862315E308 für positive Werte zulässig. Im Allgemeinen behalten numerische Daten vom Datentyp Variant den ursprünglich festgelegten Datentyp als Untertyp innerhalb des Variants bei. Wenn Sie zum Beispiel einem Variant einen Wert vom Datentyp Integer zuweisen, interpretieren alle nachfolgenden Operationen den Variant als Datentyp Integer. Wenn Sie jedoch mit einem Variant mit dem Typ Byte, Integer, Long oder Single eine arithmetische Operation ausführen und das Ergebnis den zulässigen Bereich für den ursprünglichen Datentyp überschreitet, wird das Ergebnis innerhalb des Variant automatisch zu dem nächstgrößeren Datentyp erweitert. Byte wird zu Integer, Integer wird zu Long, und Long bzw. Single werden zu Double umgewandelt. Werden die zulässigen Bereiche für den Datentyp Currency, Decimal oder Double in einem Variant überschritten, so tritt ein Fehler auf.

Der Datentyp Variant kann anstelle jedes anderen Datentyps verwendet werden. Enthält eine Variant-Variablen Ziffern, so können diese (je nach Kontext) entweder als Zeichenfolgendarstellung der Ziffern oder als deren tatsächlicher Wert interpretiert werden. Beispiel:

```
Dim varTest As Variant  
varTest = 44137
```

In diesem Beispiel enthält TestVar eine numerische Darstellung, nämlich den eigentlichen Wert 44137. Die arithmetischen Operatoren können auf alle Variablen vom Datentyp Variant angewendet werden, sofern sie numerische Daten enthalten oder Zeichenfolgendaten, die als numerische Daten interpretiert werden können. Mit dem Operator + kann TestVar zu einer Variant-Variablen, die eine Zahl enthält, bzw. zu einer Variablen eines numerischen Typs addiert werden; das Ergebnis ist eine arithmetische Summe

## Empty

Der Wert Empty bezeichnet eine nicht initialisierte Variant-Variable (d.h., ihr wurde noch kein Wert zugewiesen). Eine Variant-Variable mit dem Wert Empty entspricht dem Wert 0 in einem numerischen Kontext und einer Null-Zeichenfolge ("" ) im Zusammenhang mit Zeichenfolgenoperationen.

## Null

Verwechseln Sie Empty nicht mit Null. Der Wert Null zeigt an, dass die Variable vom Datentyp Variant absichtlich keine gültigen Daten enthält.

Der Sonderwert Error in einer Variablen vom Datentyp Variant wird verwendet, um das Eintreten von Fehlerbedingungen in einer Prozedur zu kennzeichnen. Im Gegensatz zu anderen Fehlern findet jedoch keine normale Fehlerbehandlung durch die Anwendung statt. Daher kann der Programmierer oder die Anwendung selbst den Fehler auswerten und nötige Maßnahmen ergreifen. Mit der CVErr-Funktion können Sie eine reelle Zahl umwandeln und einen Wert vom Typ Error erzeugen.

## Konvertierung von Datentypen

Häufig ist es nötig, die Daten in ein anderes Format umzuwandeln. Z.B. zur Ausgabe von Währungsinformationen sollte DM (EUR) als Währungsformat verwendet werden. Sind die Zahlen im Format Single oder Double vorhanden, können sie schnell in das Währungsformat gebracht werden.

```
Me!Textbox1 = CCur(dblZahl)
```

## Datentypkonvertierung

Häufig müssen Werte in einen anderen Datentyp umgewandelt werden. Das geschieht mit den folgenden Funktionen:

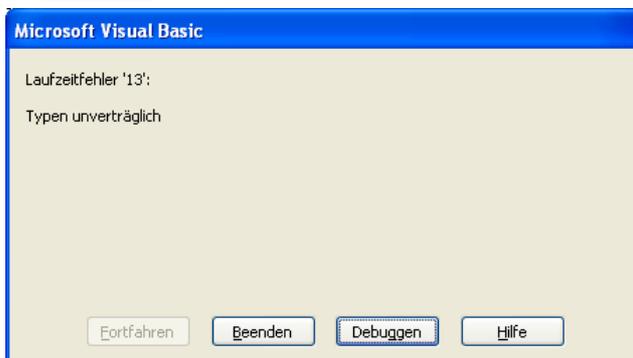
• CBool	Boolean
• CByte	Byte
• CCur	Währung
• CDate	Datum
• CDbI	Double
• CInt	Integer
• CLng	Long
• CSng	Single
• CStr	String
• CVar	Variant
• CVErr	Error

## Text in Zahl umwandeln

Umwandeln eines Textes in eine Zahl.  
In der Variablen inti steht 1234, obwohl der Wert aus einer Variablen des Typs STRING stammt.

Wenn der Textwert „Alfred“ an die Integer-Variable übergeben wird, führt dies zu einem Laufzeitfehler.

```
Sub Umwandeln()  
    Dim inti As Integer, strText As String  
    strText = "1234"  
    inti = strText  
End
```



## Fehler bei der Umwandlung

Die Funktion führt also früher oder später zu Fehlern.

Der Fehler kann wie rechts dargestellt umgangen werden, indem man die Variable StrText mit der Funktion „VAL“ in einen Zahlenwert umwandelt. Alfred ergibt dabei den Zahlenwert 0.

```
Sub Umwandeln()  
    Dim inti As Integer, strText As String  
    strText = "Alfred"  
    inti = Val(strText)  
End
```

## Verkettung von Codezeilen

Mit dem & (Et = kaufmännisches und) kann eine Textverkettung durchgeführt werden. Diese Verkettung funktioniert auch z.B. in einer Zelle in Excel:

Schreiben Sie in der Zelle A3: =A1 & A2

in der Zelle A3 stehen die Inhalte der Zellen A1 und A2 nebeneinander.

## Arrays

Variablen können nur einen Wert speichern. Arrays sind in der Lage viele Werte zu speichern. Sie sind vergleichbar mit einer fortlaufenden Liste. Sie können auch mehrdimensional sein. D.h. mehrere Dimensionen darstellen. Eine Liste ist eindimensional, ein Excel-Blatt 2-dimensional. Arrays könnten Skalare darstellen.

```
Dim varZahl(0 to 100) as Variant
```

Die folgende Schleife speichert  
101 Werte im Array.

```
Function TestArray()
    Dim dblZahl(0 To 100) As Double
    Dim intL As Integer
    Dim inti As Byte
    Dim strZahlen As String

    For inti = 0 To 100
        dblZahl(inti) = inti
    Next

    For intL = 0 To inti - 1
        strZahlen = strZahlen & vbCrLf & dblZahl(intL)
    Next

    MsgBox strZahlen

End Function
```

---

### Arrayinhalt auslesen

Zum Auslesen der  
gespeicherten  
Informationen verwenden  
Sie folgenden Code:  
Setzen Sie in der Funktion  
einen Haltepunkt (F9) und  
drücken Sie dann die Taste  
F8 zum Debuggen. Prüfen  
Sie mit dem Mauscursor  
die Werte in *inti* bzw.  
*varZahl*.

```
Function TestArray2()
    Dim dblArray(0 To 100) As Double
    Dim inti As Integer, strMeldung As String

    For inti = 0 To 100
        dblArray(inti) = inti
    Next
    For inti = 0 To 100
        strMeldung = strMeldung & dblArray(inti) & vbCrLf
    Next
    MsgBox strMeldung
End Function
```

### Dynamische Arrays

Wenn Sie Werte in einem Array speichern möchten, ist beim Programmstart nicht immer klar, wie viele Werte gespeichert werden sollen. Deshalb kann man das Array ohne Speicherreservierung deklarieren und später, wenn bekannt ist, wie viele Werte darin gespeichert werden sollen, kann mit dem Schlüsselwort **ReDim** der Bereich festgelegt werden.

```
Dim strArray () as String...
ReDim strArray (10)
....
```

Wurden im Array bereits Werte gespeichert, gehen sie verloren, wenn Sie nicht das Schlüsselwort **Preserve** verwenden.

```
ReDim Preserve strArray (UBound(strArray)+1)
```

## Kontrollstrukturen

Kontrollstrukturen ermöglichen es, auf Situationen im Programmablauf zu reagieren.

### Entscheidungsstrukturen

Häufig soll ein Programmcode nur ausgeführt werden, wenn eine oder mehrere Bedingungen erfüllt sind. Z.B. sollte geprüft werden, ob ein Layer vorhanden ist, bevor er angelegt wird.

#### If...Then...Else

Wenn eine Bedingung erfüllt ist, führe den ersten Programmcode aus, sonst (Else) führe den alternativen Programmcode aus.

```
If Bedingung Then
    Programmcode
Else
    Programmcode
End If
```

Falls nur eine Programmzeile ausgeführt werden soll, kann der Block auch folgendermaßen aussehen:

```
If Bedingung Then Programmcode
```

Die Wenn-Bedingung muss hier nicht mit End If beendet werden, weil der Code in einer Zeile steht. Die Funktion kann weitere Bedingungen enthalten.

#### Select Case

Wenn mehrere Bedingungen geprüft werden sollen, kommt die Select Case-Struktur zum Einsatz.

```
Sub SelectCase()
    Dim strEingabe As String

    strEingabe = InputBox("Geben Sie ein Zahl ein")
    Select Case strEingabe
        Case 1
            MsgBox "Sie haben 1 eingegeben"
        Case 2
            MsgBox "Sie haben 2 eingegeben"
        Case 3
            MsgBox "Sie haben 3 eingegeben"
        Case Else

            MsgBox "Sie haben weder 1, noch 2, noch 3 eingegeben"

    End Select
```

## Do While ... Loop

Kopfgesteuerte Schleifen prüfen am Schleifeneingang, ob eine Bedingung erfüllt ist. Ist die Bedingung erfüllt, wird die Schleife nicht durchlaufen.

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

## Übung 5: Kopfgesteuerte Schleife

```
Private Sub Schleife1()
    Dim byti As Byte
    Dim intZahl As Integer
    intZahl = 50
    Do While intZahl > 10
        intZahl = intZahl - 1
        byti = byti + 1
    Loop
    MsgBox "Die Schleife wurde " & byti & " mal durchlaufen."
End Sub
```

Sie können auch die folgende, ebenfalls zulässige Syntax verwenden:

## Do ... Loop Until (While)

Fußgesteuerte Schleifen prüfen am Schleifenende, ob die Bedingung erfüllt ist. Sie werden mindestens einmal durchlaufen.

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

## Übung 6: Fußgesteuerte Schleife

```
Sub Schleife2()
    Dim intZahl As Integer

    intZahl = 0
    Do
        intZahl = intZahl + 1
    Loop While intZahl < 10

    MsgBox " Die Schleife wurde " & intZahl & " mal durchlaufen."

End Sub
```

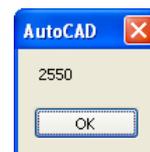
Die Schleife wird 10 mal durchlaufen.

## For...next

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

Step legt fest, wie groß die Schritte sind, mit welchen das Programm durchlaufen wird. Ein negatives Vorzeichen zählt rückwärts.

```
Private Sub SchleifeForNext()
  Dim intZähler As Integer
  Dim intSumme As Integer
  For intZähler = 0 To 100 Step 2
    intSumme = intSumme + intZähler
  Next
  MsgBox intSumme
End Sub
```



Zählt in 2er-Schritten bis 100, addiert die Werte und gibt das Ergebnis als Meldung aus (hier 2.250).

## For Each...next

```
For Each Element In Gruppe
  [Anweisungen]
[Exit For]
  [Anweisungen]
Next [Element]
```

For Each wird u.a. verwendet, um bestehende Auflistungen zu durchlaufen. Jedes Element einer Liste wird ausgelesen.

Bei jedem Schleifendurchlauf wird eine Meldung ausgegeben.

```
Private Sub Schleife2_ForNext()
  Dim objElement As AcadObject
  For Each objElement In ThisDrawing.ModelSpace
    MsgBox objElement.ObjectName
  Next
End Sub
```



Der Programmablauf wird durch die Meldung unterbrochen. Besser wäre hier die Ausgabe am Ende der Prozedur. Dazu werden die Namen der Zeichnungsobjekte in einem String gespeichert.

## Unterbrechen des Programmablaufs

Drücken Sie STRG + Pause, um den Programmablauf zu unterbrechen.

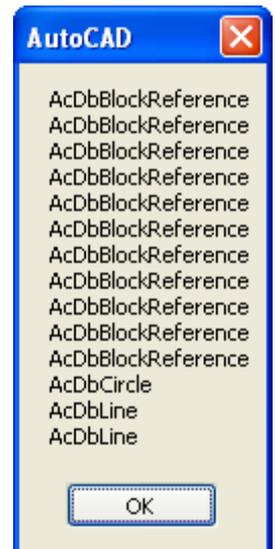
## Übung 7: Objektnamen einer Zeichnung auslesen

```
Private Sub Schleife3_ForNext()
    Dim objElement As AcadObject
    Dim strObjekte As String

    For Each objElement In ThisDrawing.ModelSpace
        strObjekte = strObjekte & objElement.ObjectName & vbCrLf
    Next

    MsgBox strObjekte

End Sub
```



Das Ergebnis, die Namen der AutoCAD-Objekte, wird in der Variable strObjekte gespeichert und am Ende in einer einzigen Meldung ausgegeben.

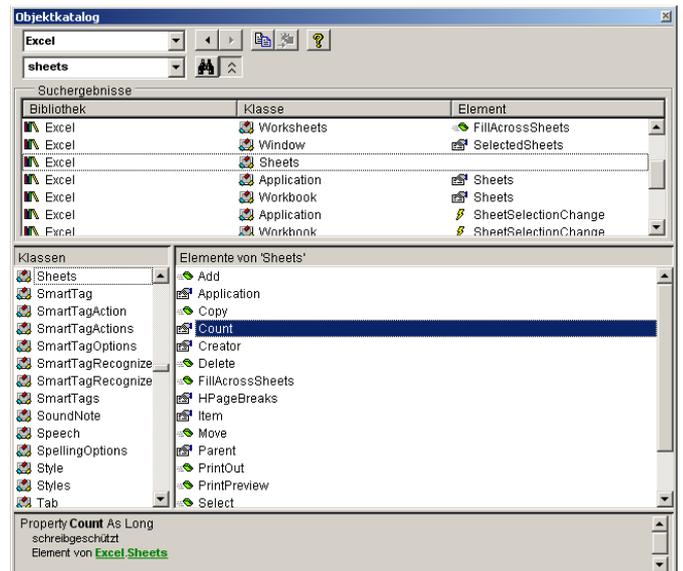
## Klassenbibliothek bzw. Objektkatalog

Der Objektkatalog beinhaltet alle Objekt. Dort sind auch die selbst erstellten Prozeduren, Formulare und Module bzw. Klassenmodule hinterlegt.

Hier werden auch alle Klassen von Programmen aufgelistet, zu welchen ein Verweis erstellt wurde (Extras/Verweise).

Wählen Sie z.B. Sheets und klicken Sie mit der rechten Maustaste im Fenster „Elemente“ auf die Eigenschaft „Count“.

Wählen Sie das Fragezeichen, um zu diesem Objekt und der entsprechenden Eigenschaft oder Methode Hilfe zu erhalten.



```
Sub TabellenZaehlen()
    MsgBox ActiveWorkbook.Sheets.Count
End Sub
```

## Beispielcode aus der Hilfe laden

Klicken Sie auf „Beispiel“.  
 Markieren und kopieren Sie den Code.  
 Fügen Sie ihn in einem Modul ein.  
 Debuggen Sie mit der Taste F8.  
 Der Text aus der Hilfe muss lediglich eingedeutscht werden: „Sheet“ muss durch „Tabelle“ ersetzt werden. Evtl. sollten Sie auch die Meldung ändern.

```
Sub DisplayColumnCount()
    Dim iAreaCount As Integer
    Dim i As Integer

    Worksheets("Tabelle1").Activate
    iAreaCount = Selection.Areas.Count

    If iAreaCount <= 1 Then
        MsgBox "Die Auswahl enthält " & Selection.Columns.Count & " Spalten."
    Else
        For i = 1 To iAreaCount
            MsgBox "Der erste Bereich der Auswahl enthält " & i & Selection.Areas(i).Columns.Count & " Spalten."
        Next i
    End If
End Sub
```

## Dialogfenster und Meldungen

Die fest eingebaute Funktion MsgBox ermöglicht die Ausgabe von Meldungen, um Fehlermeldungen darzustellen, dem Benutzer das Ergebnis von Funktionen mitzuteilen.

```
If MsgBox("Eingabeaufforderung", vbYesNo, "Überschrift") = vbYes Then
...
End If
```

Wenn nur eine Meldung ausgegeben werden soll, benötigen Sie keine Klammern.

```
MsgBox "Sie haben NEIN geklickt"
```

Beim Eintippen der Funktion wird, sobald Sie die Klammer öffnen die Syntax der Funktion eingeblendet.

Wenn Sie ein Komma eintippen, springen Sie in das nächste Argument.

Wenn Sie dort "vby" eingeben, werden automatisch die verschiedenen Optionen eingeblendet und der Cursor auf "VbYesNo" gestellt.

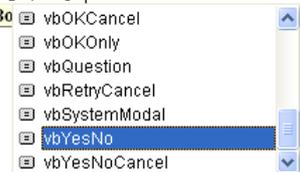
Im folgenden Beispiel sehen Sie, wie man das Ergebnis eines Meldungsfensters in einer Bedingung weiterverarbeitet.

Im ersten Fall stehen die Argumente in Klammern, weil eine Verzweigungsstruktur enthalten ist. Sonst sind für die MsgBox keine Klammern erforderlich.

```
Public Sub NurEineMeldung()
```

```
if msgbox ("Eingabeaufforderung", vbye|
```

```
MsgBox(Prompt, [Buttons As VbMsgBo
```



```
Sub Einfach()
```

```
MsgBox "Sie haben NEIN geklickt"
```

```
End Sub
```

Die Buttons können auch durch Zahlenwerte ersetzt werden. Dadurch wird die Funktion aber schwerer lesbar.

Konstante	Wert	Beschreibung
VbOKOnly	0	Nur die Schaltfläche OK anzeigen.
VbOKCancel	1	Schaltflächen OK und Abbrechen anzeigen.
VbAbortRetryIgnore	2	Schaltflächen Abbruch, Wiederholen und Ignorieren anzeigen.
VbYesNoCancel	3	Schaltflächen Ja, Nein und Abbrechen anzeigen.
VbYesNo	4	Schaltflächen Ja und Nein anzeigen.
VbRetryCancel	5	Schaltflächen Wiederholen und Abbrechen anzeigen.
VbCritical	16	Meldung mit Stop-Symbol anzeigen.
VbQuestion	32	Meldung mit Fragezeichen-Symbol anzeigen.
VbExclamation	48	Meldung mit Ausrufezeichen-Symbol anzeigen.
VbInformation	64	Meldung mit Info-Symbol anzeigen.
VbDefaultButton1	0	Erste Schaltfläche ist Standardschaltfläche.
VbDefaultButton2	256	Zweite Schaltfläche ist Standardschaltfläche.
VbDefaultButton3	512	Dritte Schaltfläche ist Standardschaltfläche.
VbDefaultButton4	768	Vierte Schaltfläche ist Standardschaltfläche.
VbApplicationModal	0	An die Anwendung gebunden. Der Benutzer muß auf das Meldungsfeld reagieren, bevor er seine Arbeit mit der aktuellen Anwendung fortsetzen kann.
VbSystemModal	4096	An das System gebunden. Alle Anwendungen werden unterbrochen, bis der Benutzer auf das Meldungsfeld reagiert.

## Inputbox

Argumente einer Funktion werden häufig direkt vom Benutzer abgefragt. Die Inputbox ist ähnlich aufgebaut, wie die MsgBox und dient zur Eingabe von Daten..

```
Public Sub NurEineEingabe()
    Dim strEingabe As String
    strEingabe = InputBox("Eingabe", "Überschrift")
    MsgBox strEingabe
End Sub
```

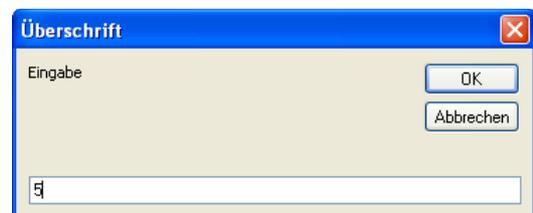
Syntax:

**InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])**

Das Ergebnis der Inputbox muss einer Variablen oder einer Funktion zugewiesen werden.

```
strEingabe = InputBox ("Eingabe", "Überschrift")
```

Die Inputbox liefert einen Wert vom Datentyp Text zurück.



Durchlaufen Sie die Prozedur mit dem Debugger. Sie sehen, dass der Übergabewert in Anführungszeichen steht.

```
Public Sub NurEineEingabe()
    Dim strEingabe As String

    strEingabe = InputBox("Eingabe", "Überschrift")
    strEingabe = "5" & "Eingabe"
End Sub
```

Falls Sie eine Zahleneingabe benötigen, verwenden Sie eine Variable mit dem Datentyp Zahl.  
 Wenn Sie Text eingeben z.B. "Das ist ein Text" und übergeben den Inhalt der Inputbox an die Variable dblEingabe, wird der Zahlenwert 0 daraus.  
 Wenn Sie die Schaltfläche "Abbrechen" betätigen, wird die leere Zeichenfolge "" zurückgegeben.

## Erzeugen von Sub-Prozeduren

Einige Punkte, die bereits früher im Skript behandelt wurden, werden hier noch einmal wiederholt.

```
[Private | Public] [Static] Sub Name [(ArgListe)]
    [Anweisungen]
    [Exit Sub]
    [Anweisungen]
End Sub
```

## Unterschied zwischen Function-Prozeduren und Sub-Prozeduren

Prozeduren führen Anweisungen durch. Es gibt – wie bereits auf Seite 5 beschrieben - zwei Typen: Die Function-Prozedur und die Sub-Prozedur.

Die Function-Prozedur kann über den Funktionsnamen einen Wert an die aufrufende Stelle zurückgeben, die Prozedur kann das nicht. Einer Function-Prozedur kann ein Datentyp zugewiesen werden.

Tippen Sie den folgenden Code in ein Modul ein. Drücken Sie dann die Taste F8, um den Code schrittweise zu durchlaufen. Sie können den Mauscursor auf die verschiedenen Variablen setzen, um die Inhalte abzufragen.

## Public und Private Prozeduren

Mit dem Schlüsselwort PUBLIC definieren Sie Prozeduren/Funktionen, die während das Formular geöffnet ist von jedem Modul/Formular der Anwendung aufgerufen werden können. Wenn Sie das Schlüsselwort PRIVATE verwenden, ist die Prozedur nur innerhalb dieses Formulars verfügbar.

## Ereignisprozeduren und Function-Prozeduren

Ereignisprozeduren reagieren auf bestimmte Ereignisse, wie z.B. beim Klicken einer Schaltfläche, beim Speichern, beim Drucken, beim Öffnen, beim Schließen einer Zeichnung.

Function-Prozeduren werden aus einer anderen Prozedur aufgerufen.

## Aufruf von Prozeduren

Eine Sub-Prozedur wird durch den Funktionsnamen, eventuell gefolgt von Argumenten aufgerufen:

## Argumente übergeben

Die Übergabewerte stehen in den Klammern hinter einer Sub- oder Function-Prozedur.

## Prozeduren beenden

### Sub-Prozeduren

```
Public Sub Prozedurname(Argumente)
    Exit Sub
End Sub
```

## Function-Prozeduren beenden

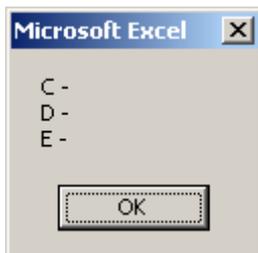
```
Public Function Funktionsname(Argumente)
    Exit Function
End Function
```

Programmcode kann in Formularmodulen oder in Standardmodulen hinterlegt werden. Der Hauptunterschied liegt in der „Sichtbarkeit“ des Codes bezogen auf andere Prozeduren / Funktionen. Wird die Prozedur im Standardmodul hinterlegt und mit dem Schlüsselwort „Public“ eingeleitet, ist sie von überall aus anzusprechen. Wird die Prozedur dagegen im Modul hinterlegt, muss das Formular geöffnet sein, um darauf zuzugreifen.

## Zugriff auf Auflistungen und Listenelemente

Der Zugriff auf Listenelemente in einer Auflistung erfolgt über Programmschleifen. Mit Programmschleifen können Sie aber auch einzelne Objekte auswählen.

### Auflistung der verfügbaren Laufwerke



```
Sub ShowDriveList()
    On Error Resume Next
    Dim fs, d, dc, s, n
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set dc = fs.Drives
    For Each d In dc
        s = s & d.DriveLetter & " - "
        If d.DriveType = Remote Then
            n = d.ShareName
        Else
            n = d.VolumeName
        End If
        s = s & n & vbCrLf
    Next
    MsgBox s
End Sub
```

## Zeilenschaltung

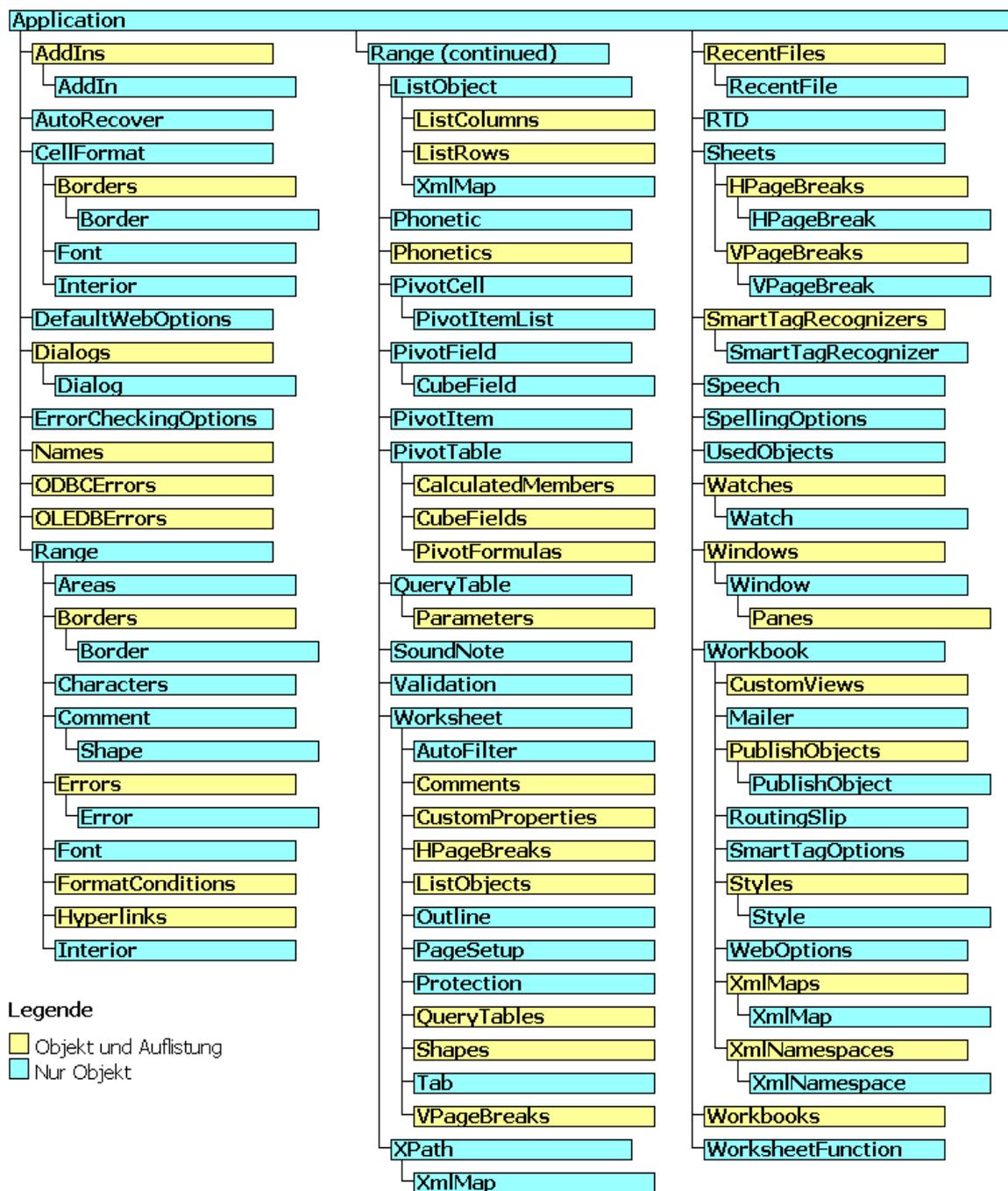
Eine Zeilenschaltung in Meldungen erhalten Sie über „& vbCr“ oder *vbLf*.

## Objektmodell (Auszug)

### Das Application-Objekt

Das Application-Objekt ist allen anderen Objekten übergeordnet. Die Workbooks-Auflistung - also die Auflistung aller geöffneten Arbeitsmappen - ist dem Application-Objekt untergeordnet.

### Objektmodell von Microsoft Excel



## Daten in Excel löschen

Diese Funktion löscht alle Daten der belegten Zellen.

```
Sub AlleDatenLoeschen()
    If MsgBox("Sollen alle Daten gelöscht werden?", vbYesNo) = vbYes Then
        Range("A1").Select
        Range(Selection, ActiveCell.SpecialCells(xlLastCell)).Select
        Selection.Clear
        Range("A1").Select
    End If
End Sub
```

## Spaltennummern in alphanumerische Bezeichnung umwandeln

Dem Großbuchstaben „A“ ist der ASCII-Wert 65 zugeordnet. Das können Sie in Excel leicht mit der Funktion =ZEICHEN(Wert) prüfen.

	A	B	C
1			
2	65	=ZEICHEN(A2)	
3	66	B	
4	67	C	
5	68	D	
6	69	E	
7	70	F	
8	71	G	
9	72	H	
10	73	I	
11	74	J	

iAlpha: Ergibt sich aus der Teilung des Spaltenwertes durch 27. Sind es 26 Spalten (= Spalte Z), ist der Wert 0.

Es gibt keinen Restwert z.B. für die Spalte AA, die nach der Spalte Z folgt.

Chr(64) ergibt das große „A“ im ASCII-Zeichensatz.

Die Zählung geht nach „Z“ mit „AA“ weiter.

Die Debug.Print-Anweisungen geben das Ergebnis im Direktfenster aus.

```
Sub testSpalte()
    MsgBox fktZahlInBuchstabe(3)
End Sub

Function fktZahlInBuchstabe(ByVal iCol As Integer) As String
    Dim iAlpha As Integer
    Dim iRemainder As Integer
    Debug.Print iCol
    'Dim iCol As Integer
    'icol = 55
    iAlpha = Int(iCol / 27)

    iRemainder = iCol - (iAlpha * 26)
    Debug.Print iRemainder
    If iAlpha > 0 Then
        fktZahlInBuchstabe = Chr(iAlpha + 64)
        Debug.Print fktZahlInBuchstabe
    End If
    Debug.Print Chr(iAlpha + 64)
    If iRemainder > 0 Then
        fktZahlInBuchstabe = fktZahlInBuchstabe & Chr(iRemainder + 64)
        Debug.Print fktZahlInBuchstabe
    End If
    Debug.Print iAlpha
End Function
```

## Umwandeln von Datumsangaben in SQL-lesbares Datum

Zur Übergabe des Datums in Abfragen muss der Datumswert im SQL-Format vorliegen. Die rechts dargestellte Routine wandelt das normale Datum in ein SQL-Datum um.

```
Function SqlDatum(datDatum As Date) As String
    Dim lngJ As Long, lngM As Long, lngT As Long
```

```
    On Error Resume Next
```

```
    lngJ = Year(datDatum)
    lngM = Month(datDatum)
    lngT = Day(datDatum)
```

```
    SqlDatum = "#" & lngJ & "/" & lngM & "/" & lngT & "#"
```

```
End Function
```

## Menü erstellen

Die rechts dargestellte Routine erstellt ein Menü.

Die Einträge sind in den Konstanten festgelegt.

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
    "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) _
    As Long
```

```
'Menü erweitern
```

```
Const MenuName = "&Patente einspielen"
```

```
Const Befehl1 = "Daten &auswerten"
```

```
Const Befehl2 = "Daten &zählen"
```

```
Public lngDp As Long
```

```
Public strLetzter As String
```

```
Sub Menü_Erstellen()
```

```
    Dim MB As Object, MeinMenü As Object, Befehl As Object
```

```
    Set MB = CommandBars.ActiveMenuBar
```

```
    Set MeinMenü = MB.Controls.Add(Type:=msoControlPopup, Temporary:=True)
    MeinMenü.Caption = MenuName
```

```
    Set Befehl = MeinMenü.Controls.Add(Type:=msoControlButton, ID:=1)
```

```
    With Befehl
```

```
        .Caption = Befehl1
```

```
        .OnAction = "subAuswerten"
```

```
    End With
```

```
    Set Befehl = MeinMenü.Controls.Add(Type:=msoControlButton, ID:=1)
```

```
    With Befehl
```

```
        .Caption = Befehl2
```

```
        .OnAction = "subZaehlen"
```

```
    End With
```

```
End Sub
```

## Patentschriften auswerten

Die rechts dargestellte Routine sucht über eine Select-Case-Verzweigung nach den Begriffen „PA“, „AD“, „PN“, „TI“ und schreibt die rechts daneben stehenden Werte in die Spalten C, D, E und F in Excel.

Die Grundwerte sind in der Spalte B:

```

----- Treffer 1 von 123 -----
-----
PN   : DE 102007025187 A1
AD   : 30.05.2007
PA   : ebm-papst St. Georgen GmbH & Co. KG,
78112 St. Georgen, DE
TI_DE: Kollektorloser Außenläufermotor
----- Treffer 2 von 123 -----
-----
PN   : DE 202007000867 U1
AD   : 20.01.2007
PA   : ebm-papst St. Georgen GmbH & Co. KG,
78112 St. Georgen, DE
TI_DE: Elektromotor
----- Treffer 3 von 123 -----
-----
PN   : DE 000004143597 C2
AD   : 22.11.1991
PA   : Baumüller Nürnberg GmbH, 90482 Nürnberg,
DE
      Baumüller Anlagen-Systemtechnik GmbH
& Co., 90482 Nürnberg, DE
TI_DE: Druckmaschine mit wenigstens einem
elektromotorisch angetriebenen,
      axial verstellbaren Zylinder oder
sonstigen Drehkörper
TI_EN: Electromotor for driving rotating
cylinders of printing machines
----- Treffer 4 von 123 -----
-----
PN   : DE 000019545326 C1
AD   : 05.12.1995
PA   : ebm Elektrobau Mulfingen GmbH & Co,
74673 Mulfingen, DE
TI_DE: Anordnung zum Anschluß eines Geräte-
Anschlußkabels
TI_EN: Connection arrangement for electromotor
cable with terminals
      pull-relief
----- Treffer 5 von 123 -----
PN   : EP 000000844723 A2
AD   : 25.11.1997
PA   : FLENDER AUSTRIA ANTRIEBSTECHNI, AT
TI_DE: Elektromotorisch angetriebene Pumpe
TI_EN: Electric motor driven pump
TI_FR: Pompe entraînée par un moteur
électrique

```

```

Sub subAuswerten()
Dim i As Integer, lngPos As Integer
Dim str As String
Dim lngErsteSpalte As Long

lngErsteSpalte = 2 'B

i = 2
Do While Cells(i, lngErsteSpalte) <> ""
  str = Cells(i, lngErsteSpalte)
  Select Case Left(str, 2)
    Case "--"
      i = i + 1
    Case "PN"
      strLetzter = "PN"
      lngPos = i
      Cells(lngPos, lngErsteSpalte + 1) = fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
    Case "AD"
      strLetzter = "AD"
      Cells(lngPos, lngErsteSpalte + 2) = fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
    Case "PA"
      strLetzter = "PA"
      Cells(lngPos, lngErsteSpalte + 3) = fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
    Case "TI"
      strLetzter = "TI"
      Cells(lngPos, lngErsteSpalte + 4) = fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
    Case " "
  End Select
  Select Case strLetzter
    Case "PA"
      Cells(lngPos, lngErsteSpalte + 3) = Cells(lngPos, lngErsteSpalte + 3) & " " & fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
    Case "TI"
      Cells(lngPos, lngErsteSpalte + 4) = Cells(lngPos, lngErsteSpalte + 4) & " " & fktDP(Cells(i, lngErsteSpalte))
      i = i + 1
  End Select
End Select
Loop
End Sub

```

## Laufende Nummer

Die rechts dargestellte Routine fügt in der Spalte A eine laufende Nummer ein, wenn die Spalte B nicht leer („“) ist.

```
Sub subZaehlen()
  Dim lngMax As Integer, i As Integer

  i = 2
  lngMax = 1

  Do While Cells(i, 2) <> ""
    If Left(Cells(i, 2), 2) = "PN" Then

      Cells(i, 1) = lngMax
      lngMax = lngMax + 1
    End If
    i = i + 1
  Loop

End Sub
```

## API-Funktion

Zum Auslesen der Festplattennummer (aus [www.activevb.de](http://www.activevb.de)). Kopieren Sie den folgenden Programmcode in den VBA-Editor.

Rufen Sie ihn über die folgende Prozedur auf:

```
Sub hddId()
  Debug.Print SerNum("c")
End Sub
```

Option Explicit

```
Private Declare Function GetVolumeInformation Lib "kernel32" _
  Alias "GetVolumeInformationA" (ByVal lpRootPathName _
  As String, ByVal pVolumeNameBuffer As String, ByVal _
  nVolumeNameSize As Long, lpVolumeSerialNumber As Long, _
  lpMaximumComponentLength As Long, lpFileSystemFlags As _
  Long, ByVal lpFileSystemNameBuffer As String, ByVal _
  nFileSystemNameSize As Long) As Long
```

```
Const MAX_FILENAME_LEN As Long = 256&
```

```
Private Const WS_VERSION_REQD = &H101
Private Const WS_VERSION_MAJOR = WS_VERSION_REQD \ &H100 And &HFF&
Private Const WS_VERSION_MINOR = WS_VERSION_REQD And &HFF&
Private Const MIN_SOCKETS_REQD = 1
Private Const SOCKET_ERROR = -1
Private Const WSADescription_Len = 256
Private Const WSASYS_Status_Len = 128
```

```
Private Type HOSTENT
  hName As Long
  hAliases As Long
  hAddrType As Integer
  hLength As Integer
  hAddrList As Long
End Type
```

Private Type WSADATA

wversion As Integer  
 wHighVersion As Integer  
 szDescription(0 To WSADescription\_Len) As Byte  
 szSystemStatus(0 To WSASYS\_Status\_Len) As Byte  
 iMaxSockets As Integer  
 iMaxUdpDg As Integer  
 lpszVendorInfo As Long

End Type

Private Declare Function WSAGetLastError Lib "WSOCK32.DLL" () As Long

Private Declare Function WSASStartup Lib "WSOCK32.DLL" (ByVal wVersionRequired&, lpWSADATA As WSADATA) As Long

Private Declare Function WSACleanup Lib "WSOCK32.DLL" () As Long

Private Declare Function gethostname Lib "WSOCK32.DLL" (ByVal hostname\$, ByVal HostLen%) As Long

Private Declare Function gethostbyname Lib "WSOCK32.DLL" (ByVal hostname\$) As Long

Private Declare Function gethostbyaddr Lib "WSOCK32.DLL" (ByVal addr\$, ByVal laenge%, ByVal typ%) As Long

Private Declare Sub RtlMoveMemory Lib "kernel32" (hpvDest As Any, ByVal hpvSource&, ByVal cbCopy&)

Public Function HostByAddr\$(ByVal addr\$)

Dim host As HOSTENT  
 Dim a\$, hostent\_addr&, n%

a\$ = Chr\$(Val(zeichennext\$(addr\$, ".")))
 a\$ = a\$ + Chr\$(Val(zeichennext\$(addr\$, ".")))
 a\$ = a\$ + Chr\$(Val(zeichennext\$(addr\$, ".")))
 a\$ = a\$ + Chr\$(Val(addr\$))

hostent\_addr& = gethostbyaddr(a\$, Len(a\$), 2)  
 If hostent\_addr& = 0 Then HostByAddr\$ = "": Exit Function

RtlMoveMemory host, hostent\_addr&, LenB(host)

Dim c As String \* 5  
 a\$ = "": n% = 0  
 Do  
 RtlMoveMemory ByVal c, host.hName + n%, 1  
 If Left\$(c, 1) = Chr\$(0) Then Exit Do  
 a\$ = a\$ + Left\$(c, 1): n% = n% + 1  
 Loop

HostByAddr\$ = a\$

End Function

Public Function HostByName\$(na\$, Optional adapter% = 0) '7

Dim host As HOSTENT  
 Dim temp\_ip\_address() As Byte  
 Dim hostent\_addr&, i%, hostip\_addr&, ip\_address\$

```

hostent_addr& = gethostbyname(na$)
If hostent_addr& = 0 Then HostByName$ = "": Exit Function

RtlMoveMemory host, hostent_addr&, LenB(host)

For i% = 0 To adapter% 'Wenn schon eher ein Eintrag 0 ist, dann ist Adapter-Wert zu groß!
    RtlMoveMemory hostip_addr&, host.hAddrList + 4 * i%, 4
    If hostip_addr& = 0 Then HostByName$ = "": Exit Function
Next

ReDim temp_ip_address(1 To host.hLength)
RtlMoveMemory temp_ip_address(1), hostip_addr&, host.hLength

ip_address$ = ""
For i = 1 To host.hLength
    ip_address$ = ip_address$ & temp_ip_address(i) & "."
Next
ip_address$ = Left$(ip_address$, Len(ip_address$) - 1)

HostByName$ = ip_address$

End Function
Public Function MyHostName$() '4

    Dim hostname As String * 256

    If gethostname(hostname, 256) = SOCKET_ERROR Then
        'MsgBox "Windows Sockets error " & Str(WSAGetLastError())
        Exit Function
    Else
        MyHostName$ = zeichennext$(Trim$(hostname), Chr$(0))
    End If

End Function
Public Sub SocketsInitialize() '1

    Dim WSAD As WSADATA
    Dim iReturn%, sHighByte$, sLowByte$, sMsg$

    iReturn% = WSASStartup(WS_VERSION_REQD, WSAD)
    If iReturn% <> 0 Then
        MsgBox "Winsock.dll is not responding."
        End
    End If

    If lobyte(WSAD.wversion) < WS_VERSION_MAJOR Or (lobyte(WSAD.wversion) = WS_VERSION_MAJOR
And _
hibyte(WSAD.wversion) < WS_VERSION_MINOR) Then
        sHighByte$ = Trim$(Str$(hibyte(WSAD.wversion)))
        sLowByte$ = Trim$(Str$(lobyte(WSAD.wversion)))
        sMsg$ = "Windows Sockets version " & sLowByte$ & "." & sHighByte$
        sMsg$ = sMsg$ & " is not supported by winsock.dll "

```

```

    MsgBox sMsg$
    End
End If

If WSAD.iMaxSockets < MIN_SOCKETS_REQD Then
    sMsg$ = "This application requires a minimum of "
    sMsg$ = sMsg$ & Trim$(Str$(MIN_SOCKETS_REQD)) & " supported sockets."
    ' MsgBox sMsg$
    End
End If

End Sub
Public Sub SocketsCleanup() '6

    Dim IReturn&

    IReturn& = WSACleanup()

    If IReturn& <> 0 Then
        'MsgBox "Socket error " & Trim$(Str$(IReturn&)) & " occurred in Cleanup "
        End
    End If

End Sub
Private Function lobyte(ByVal wParam As Integer) '2

    lobyte = wParam And &HFF&

End Function
Private Function hibyte(ByVal wParam As Integer) '3

    hibyte = wParam \ &H100 And &HFF&

End Function
Private Function zeichennext$(a$, ch$) '5

    Dim ai%

    ai% = InStr(a$, ch$)

    If ai% = 0 Then
        zeichennext$ = a$: a$ = ""
    Else
        zeichennext$ = Left$(a$, ai% - 1): a$ = Mid$(a$, ai% + Len(ch$))
    End If

End Function

Sub testlp()
    Dim hostname As String
    Dim strip_Adresse As String, netzkarte As String, varfestplattennummer As String

```

```
Dim varkartenNummer As Variant, kartenNummer
```

```
SocketsInitialize
```

```
    hostname = MyHostName$()
```

```
'End If
```

```
SocketsCleanup
```

```
If strIp_Adresse = "" Then
```

```
    SocketsInitialize
```

```
    strIp_Adresse = HostByName$(hostname)
```

```
    SocketsCleanup
```

```
End If
```

```
' IP_Adresse = strIP_Adresse
```

```
varkartenNummer = strIp_Adresse & "/" & netzkarte & "/" & varfestplattennummer
```

```
kartenNummer = varkartenNummer
```

```
End Sub
```

```
Sub hddId()
```

```
    Debug.Print SerNum("c")
```

```
End Sub
```

```
Public Function SerNum(Drive$) As Long
```

```
    Dim No As Long, s As String * MAX_FILENAME_LEN
```

```
    Call GetVolumeInformation(Drive & ":\", s, MAX_FILENAME_LEN, _  
        No, 0&, 0&, s, MAX_FILENAME_LEN)
```

```
    SerNum = No
```

```
End Function
```