EXCEL-VBA

Rezeptesammlung

Stefan Wenninger
www.simplesoft.at

VERSION 12.02.2024

EINLEITUNG + DOKUMENTATION

Liebe VBA-Programmierfreunde,

die meisten von uns sammeln über die Jahre VBA-Code.

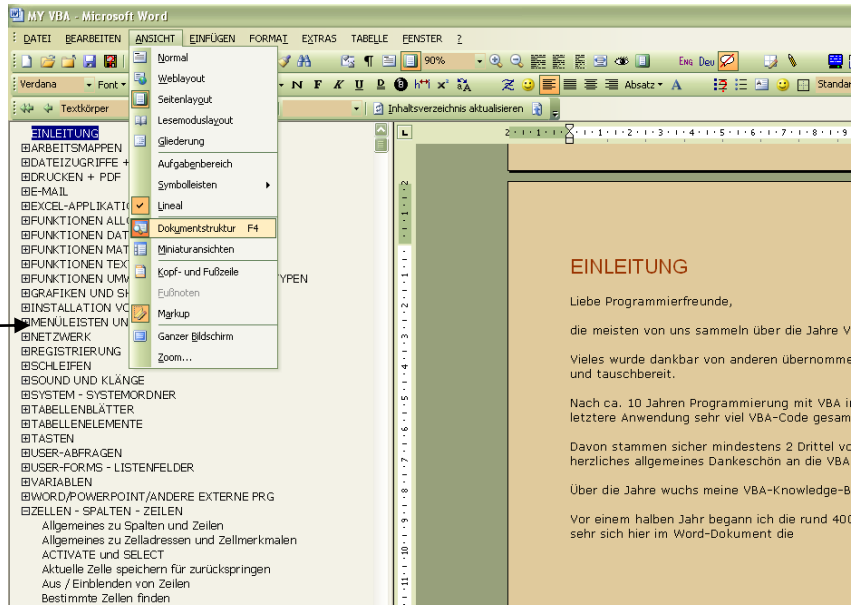
Vieles wurde dankbar von anderen übernommen, selber weiterentwickelt … - die ProgrammiererInnen-Community ist sehr hilfs- und tauschbereit.

Nach ca. 10 Jahren Programmierung mit VBA in den Office-Produkten Word, Outlook, Powerpoint und Excel habe ich vor allem für letztere Anwendung sehr viel VBA-Code gesammelt.

Davon stammen sicher mindestens 75% von anderen VBA-ProgrammiererInnen und an dieser Stelle möchte ich hier einmal ein herzliches Dankeschön an die VBA-Community richten.

Über die Jahre wuchs meine VBA-Knowledge-Base und ist jetzt bei etwa 1.300 Seiten angekommen.

2006 begann ich die rund 400 (enstehungsgeschichtlich unsortierten) Einzelfragen zu sortieren und ich empfehle sehr sich hier im Word-Dokument die Dokumentenstruktur einzublenden. So kommt man rasch über die Hauptkapitel zu den einzelnen VBA-Themen. (Auch die Volltextsuche verwende ich sehr gerne, um etwas Bestimmtes zu finden.)



Die vorliegende VBA-Code-Sammlung ist kostenlos und darf an jedermann weitergegeben - allerdings nur unentgeltlich.

Ich wünsche ein fröhliches Copy & Paste und Weiterentwickeln,

Stefan Wenninger
1050 Wien

PS: auf meiner Homepage www.simplesoft.at gibt es weitere kostenlose Downloads.

Dokumentation

Wenn ich zu existierenden Projekten eingeladen werde, um Code zu ergänzen oder zB für neuere Excel-Versionen kompatibel zu machen, werden meine Ergänzungen / Veränderungen im Code natürlich eindeutig gekennzeichnet. Ich verwende dazu folgende Kommentare:

```
' ----------- TEMPORARYLY - BEGIN -------------------------------
' ----------- TEMPORARYLY - END ---------------------------------

' ----------- TEMPORARYLY DEACTIVATED - BEGIN -------------------
' ----------- TEMPORARYLY DEACTIVATED - END ---------------------

' Just for testing - the following 1 line/s can be removed

' -----------------------------------------------------
' NEW: 2012_07_02_008 Wenninger
' WHAT: Declaring new variables (objects) FS_Folder and FS_Files
' WHY:  they are the needed to replace Application.Filesearch

' NEW-END
' -----------------------------------------------------

' -----------------------------------------------------
' INACTIVATE: 2012_07_02_00 Wenninger
' WHY: FileSearch is no more available since Excel 2007
' REPLACEMENT: is straight below NEW 2012_02_02_003
   '. . .
' INACTIVATE-END
' -----------------------------------------------------

' -----------------------------------------------------
' CHANGE: 2012_07_02_00 Wenninger
' WHAT: new "loop end"

' original:
    ' . . .
' new:
    ' . . .
' CHANGE-END
' -----------------------------------------------------
```

Wichtig: Befindet sich VBA-Code direkt in einem Tabellenblatt, bezieht sich Range("A1") IMMER auf die Zelle A1 in diesem Tabellenblatt, SELBST WENN DER VBA-CODE auf eine andere Tabelle umgeschalten hat und diese aktiv wäre. Nur in Modulen greifen Range und Cells-Befehle OHNE Tabellenbezug auf die aktuelle Tabelle.

MEIN BASIS CODE

Dies ist der Code, den ich gerne in neuen VBA-Projekten per default zur Verfügung stelle

Modul ALLGEMEIN

```
' ____________________________
'|                            |
'| MODUL - BESCHREIBUNG       |
'|____________________________|

' Dieses Modul enthält alle Prozeduren, die ALLGEMEIN für andere Module von Bedeutung sind

' ____________________________
'|                            |
'| ALLGEMEINE VARIABLEN       |
'|____________________________|

' KEINE

' ____________________________________________________________________________________

' ____________________________
'|                            |
'| ÖFFENTLICHE FUNKTIONEN     |
'|____________________________|
'
Public Function SPALTENTEXT(SPALTENNUMMER As Integer) As String
```

```
' Diese Funktion wandelt Spaltennummern in Spaltentext um: 5=>E

    If SPALTENNUMMER > 26 Then
      SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 2)
    Else
      SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 1)
    End If

End Function

Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' ermittelt die letzte Zelle einer Tabelle mit Inhalt oder Zellformat

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row
```

```
   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function

Public Function FileFolderExists(strFullPath As String) As Boolean

    ' Diese Funktion überprüft ob ein Ordner oder eine Datei existiert

    On Error GoTo EarlyExit
    If Not Dir(strFullPath, vbDirectory) = vbNullString Then FileFolderExists = True
EarlyExit:
    On Error GoTo 0

End Function

' _____________________________
'|                             |
'| ÖFFENTLICHE PROZEDUREN      |
'|_____________________________|


Public Sub AUS()

' Schaltet Bildschirmaktualisierung und Events ab

    Application.ScreenUpdating = False
    Application.EnableEvents = False
    ' nachfolgender Code Calculation Manual macht Sinn, wenn die Tabelle sehr viele Formeln mit Zellbezügen hat
    ' Application.Calculation = xlCalculationManual

End Sub
Public Sub EIN()
```

```
' Schaltet Bildschirmaktualisierung und Events ein

   Application.ScreenUpdating = True
   Application.EnableEvents = True
   ' nachfolgender Code Calculation Manual macht Sinn, wenn die Tabelle sehr viele Formeln mit Zellbezügen hat
   ' Application.Calculation = xlCalculationAutomatic

End Sub



Public Function OrdnerAuswahl() As String
    Dim bInfo As BROWSEINFO
    Dim strPath As String
    Dim R As Long
    Dim X As Long
    Dim pos As Integer

    ' Der Ausgangsordner ist der Desktop :
    bInfo.pidlRoot = 0& ' 5& wären die Eigenen Dateien
    ' Dialogtitel
    bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
    ' Rückgabe des Unterverzeichnisses
    bInfo.ulFlags = &H1
    ' Dialog anzeigen
    X = SHBrowseForFolder(bInfo)
    ' Ergebnis gliedern
    strPath = Space$(512)
    ' Ausgewähltes Verzeichnis einlesen
    R = SHGetPathFromIDList(ByVal X, ByVal strPath)
    If R Then
    pos = InStr(strPath, Chr$(0))
    OrdnerAuswahl = Left(strPath, pos - 1)
    Else
    OrdnerAuswahl = ""
    End If
End Function


'  _______________________________
'|                               |
'|  ALLGEMEINE PROZEDUREN        |
```

```
'|_____________________________|


Sub VBA_Test()
' Code des Buttons in der Anleitung/VBA-Tabelle
MsgBox "Gratulation - bei Ihnen ist der VBA-Code aktiv. " & vbCrLf & vbCrLf & _
"Sollten Sie bei jedem Öffnen der Zeiterfassungsdatei in der eingeblendeten Infozeile oben auf 'INHALTE AKTIVIEREN'
klicken müssen," & _
"damit der VBA-Code aktiv ist, können Sie In der Tabelle ANLEITUNG ganz oben unter 1.a nachlesen, wie man den VBA-Code
dauerhaft einschalten kann."

End Sub

Sub VBA_EIN()
' Code in der Parametertabelle oben der Button VBA-Test
' er schaltet verlässlich den VBA-Code ein, wenn das Enableevent auf FALSE geschaltet blieb durch zB einen Programmfehler
und Abbruch.
EIN
MsgBox "VBA-Code ist bei Ihnen aktiv"
End Sub



Sub WAEHLE_ORDNER()
Dim neuOrdner

neuOrdner = OrdnerAuswahl
If neuOrdner = "" Then
    Exit Sub
  Else
    ChDir neuOrdner
End If
MsgBox neuOrdner
End Sub


Sub CURSORUNTEN()
    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlDown
    End With
End Sub
```

```
Sub CURSORRECHTS()
    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlToRight
    End With
End Sub

Sub CURSORNO()
    Application.MoveAfterReturn = False
End Sub

Sub Zoom_groesser()
     ActiveWindow.Zoom = ActiveWindow.Zoom + 5
End Sub

Sub Zoom_kleiner()
     ActiveWindow.Zoom = ActiveWindow.Zoom - 5
End Sub


' --------------------------------
' --- AKTUELLE TABELLE SCHÜTZEN ---
' --------------------------------

Public Sub AUF()
' Diese Hilfsprozedur dient für andere Prozeduren, um den Tabellenblattschutz für die aktuelle Tabelle zu öffnen

    ' Schutz aufheben
    ActiveSheet.Unprotect Password:="" ' oder :=Tabelle24.Range("J12")

End Sub

Public Sub ZU()

' Diese Hilfsprozedur dient für andere Prozeduren, um den Tabellenblattschutz für die aktuelle Tabelle zu setzen

    ActiveSheet.Protect Password:="", DrawingObjects:=True, Contents:=True, Scenarios:=True, AllowFormattingCells:=True,
AllowSorting:=True, AllowFiltering:=True

End Sub


' ------------------------------
' --- ALLE TABELLEN SCHÜTZEN ---
```

```
' ------------------------------

Sub ALLES_AUF()

' --- Hilfsprozedur, die in allen Tabellen den Tabellenblattschutz entfernt ---

    ' Schleife durch alle Tabellenblätter

    For n = 1 To Worksheets.Count
        Worksheets(n).Unprotect Password:=""    ' oder Passwort:=Tabelle7.Range("AJ7") :In dieser Zelle ist das Passwort
für den Tabellenblattschutz
    Next

End Sub

Sub ALLES_ZU()

' --- Hilfsprozedur, die in allen Tabellen den Tabellenblattschutz setzt ---

    ' Schleife durch alle Tabellenblätter

    For n = 1 To Worksheets.Count
        Worksheets(n).Protect Password:="", DrawingObjects:=True, Contents:=True, Scenarios:=True,
AllowFormattingCells:=True, AllowSorting:=True, AllowFiltering:=True    ' oder Password:=Tabelle7.Range("AJ7") In dieser
Zelle ist das Passwort für den Tabellenblattschutz
                Worksheets(n).EnableSelection = xlUnlockedCells ' nur ungeschützte Zellen dürfen angewählt werden
    Next

End Sub
```

ABSICHERN VON MAPPEN + ADMIN-MODUS

Eine Excelmappe soll für normale User nur zum Ausfüllen sein und weite Teile sollen für die User nicht sichtbar sein.

Ein Controller (Schaltfläche CONTROLLING) soll alle Auswertungs-Spalten SEHEN, aber (durch Zellschutz und Blattschutz geschützt) nicht bearbeiten können.

Ein Administrator (Schaltfläche FORMELN) soll alle Auswertungs-Spalten sehen und bearbeiten können (Blattschutz weg) und auch ein eigenes ADMIN-Tabellenblatt sehen, welches sonst immer ausgeblendet ist.

Auch rechte Mausklicks auf Tabellenblätter und in der Tabelle sollen gemeinsam mit STRG-X, STRG-C, STRG-V, STRG +/- unterbunden werden.

Alle sensiblen Schaltflächen (CONTROLLER und FORMELN) sind mit einer Passwort-Userform abgesichert. Direkt in der Erfassungstabelle E schreibt jede der beiden Schaltflächen ihre Nummer hinein (FORMELN: 1 und CONTROLLER: 2), sodass die Passwort-Userform weiß, wer sie aufruft und die jeweils verschiedenen Passwörter (in der Admintabelle in weißer Schrift nicht sofort sichtbar ausgefüllt) überprüfen kann.

1.) DIESEARBEITSMAPPE

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

Dim MONAT As Integer

' Erfassungstabelle aufräumen, Controlling-Bereich mit Formeln und Vormonate ausblenden

    AUS
    E.Activate
    AUF

    ' Spalten mit Formeln ausblenden
    E.Columns("O:T").Hidden = True

    ' Admin/Formelbereich oben ausblenden
    E.Rows("10:50").Hidden = True

    ' oberste Zeile des Erfassungbereiches (ca. ab Zeile 1000) fixieren
    E.Rows("1008:1008").Select
    ActiveWindow.FreezePanes = True

    ' Vormonatszeilen ab dem 5.Tag des Folgemonats ausblenden, sodass man nur das aktuelle Monat sieht
    LZ = Range("B65536").End(xlUp).Row
    MONAT = Month(Date)
    If Day(Date) < 5 Then MONAT = MONAT - 1
```

```
    If MONAT < 2 Then GoTo WEITER
    For Z = 1008 To LZ
        If Cells(Z, 7) <> "" And Cells(Z, 7) < MONAT Then Rows(Z).Hidden = True
    Next Z

WEITER:
    ZU

' Admintabelle A ausblenden

    MAPPE_AUF
    A.Visible = xlSheetVeryHidden
    MAPPE_ZU

    Application.CommandBars("Ply").Enabled = False ' kein rechter Mausklick auf Registerkarten der Tabellenblätter
    Kopieren Deaktivieren
    EIN

End Sub


Private Sub Workbook_Open()

' Zellsprungrichtung auf rechts stellen
Application.MoveAfterReturnDirection = xlToRight

Application.CommandBars("Ply").Enabled = False ' kein rechter Mausklick auf Registerkarten der Tabellenblätter

Kopieren_Deaktivieren ' STRG-C, Strg-V etc deaktivieren

End Sub

Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)

    If A.Visible = xlSheetVisible Then Exit Sub ' wenn wir im Adminmodus sind, soll rechter Mausklick möglich sein
    Cancel = True ' Man kann nirgendwo einen rechten Mausklick machen

End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)

    Application.CommandBars("Cell").Reset ' Rechten Mausklick wieder aktivieren
    Application.CommandBars("Cell").Enabled = True ' Rechten Mausklick wieder aktivieren
```

```
    Application.CommandBars("Ply").Enabled = True ' rechter Mausklick auf Blattregisterkarten ist wieder aktiviert

    Kopieren_Aktivieren

End Sub
```

2.) Code der Erfassungstabelle E

```
Private Sub Worksheet_Change(ByVal Target As Range)

Dim STANDORT As String
Dim MERKER As Integer

' Wenn man bei einer Datenzeile die Spalte F ändert wollen wir die Formeln automatisch übernehmen

If Selection.Count = 1 Then ' nur wenn nur eine Zelle markiert ist
    If Target.Column = 6 And Target.Row > 1007 And Target.Value <> "" Then ' wenn in Spalte F, ab Datenerfassungsbereich
1007 und es wurde ein Wert eingegeben

        STANDORT = UCase(Target.Value)
        AUS
        MERKER = 0

        For Z = 12 To 29

            If UCase(Cells(Z, 6)) = STANDORT Then
                AUF
                For S = 15 To 19
                    If Cells(ActiveCell.Row, S) <> "" And MERKER = 0 Then
                        If MsgBox("Achtung - diese Zeile " & ActiveCell.Row & " enthält bereits Formelwerte - sollen
diese überschrieben werden", vbYesNoCancel, "ÜBERSCHREIBEN") <> vbYes Then
                            MsgBox "Abbruch"
                            EIN
                            Exit For
                        End If
                        MERKER = 1
                    End If
                    Cells(ActiveCell.Row, S).FormulaR1C1 = Cells(Z, S).FormulaR1C1
                Next S
                ZU
```

```
            End If

        Next Z

    ' löschen von Umsätze in Spalte I, wenn man umschaltet auf Bahnhof oder No1
    If Target.Row > 1007 Then
        If UCase(Cells(Target.Row, 6)) = "BAHNHOF" Or UCase(Cells(Target.Row, 6)) = "NO1" Then
            Cells(Target.Row, 9).ClearContents
            If Target.Column = 9 Then Target.Offset(0, 1).Select
        End If
    End If

    ' löschen von Listenpreisen in Spalte H, wenn man umschaltet auf etwas anderes als Bahnhof oder No1
    If Target.Row > 1007 Then
        If UCase(Cells(Target.Row, 6)) <> "BAHNHOF" And UCase(Cells(Target.Row, 6)) <> "NO1" Then
            Cells(Target.Row, 8).ClearContents
            If Target.Column = 8 Then Target.Offset(0, 1).Select
        End If
    End If

        EIN
    End If
End If


End Sub

Private Sub test()
EIN
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)


    ' Rabatte und MV und Listenpreisspalte nicht anwählen können, wenn kein Bahnhof und wenn kein No1
    If Target.Row > 1007 And UCase(Cells(Target.Row, 6)) <> "BAHNHOF" And UCase(Cells(Target.Row, 6)) <> "NO1" And
Cells(Target.Row, 6) <> "" Then
        If Target.Column = 11 Or Target.Column = 12 Then Target.Offset(0, 1).Select ' Rabatte und MV
        If Target.Column = 8 Then Target.Offset(0, 1).Select  ' Listenpreos
    End If

    ' bei Bahnhof und No1 keine Umsatzspalte I ausfüllen
```

```
    If Target.Row > 1007 Then
        If UCase(Cells(Target.Row, 6)) = "BAHNHOF" Or UCase(Cells(Target.Row, 6)) = "NO1" Then
            Cells(Target.Row, 9).ClearContents
            If Target.Column = 9 Then Target.Offset(0, 1).Select
        End If
    End If

    ' Weiterspringen wenn wir in Spalte M sind
    If Target.Column = 13 Then Cells(Target.Row + 1, 2).Select


End Sub
```

3.) Code der Passwort-Userform

```
Private Sub CommandButton1_Click()
    E.Range("P1") = 0
    Unload UserForm1

End Sub

Private Sub CommandButton2_Click()

    E.Range("P1") = 0
    Unload UserForm1

End Sub

Private Sub TextBox1_Change()

    Dim AKTIVEZELLE As Range

    ' Diese Funktion / Userform hat mehrere Anwendungen:
    ' 1) Klickt man auf die Schaltfläche FORMELN wird das Passwort dafür angefragt
    ' 2)
    ' 3)

    ' 1) Code für FORMELN

    If E.Range("P1") = 1 Then
        ' Schauen ob Passwort stimmt
```

```
        If TextBox1.Value = A.Range("C6").Value Then
            E.Range("P1") = "ok"
            Unload UserForm1
            Exit Sub
        End If
    End If

    ' 2) Code für CONTROLLING

    If E.Range("P1") = 2 Then
        ' Schauen ob Passwort stimmt
        If TextBox1.Value = A.Range("C7").Value Then
            E.Range("P1") = "ok"
            Unload UserForm1
            Exit Sub
        End If
    End If

End Sub
```

4.) ALLGEMEINER CODE

```
Public Sub EIN()

' aktiviert Screenupdating und EnableEvents

Application.EnableEvents = True
Application.ScreenUpdating = True

End Sub
Public Sub AUS()

' deaktiviert Screenupdating und EnableEvents

Application.EnableEvents = False
Application.ScreenUpdating = False

End Sub
Public Sub AUF()
```

```
' Hilfsprozedur, die das aktuelle Tabellenblatt öffnet

            PW = A.Range("C4").Value
            ActiveSheet.Unprotect (PW)


End Sub
Public Sub ZU()
' Hilfsprozedur, die das aktuelle Tabellenblatt schützt

            PW = A.Range("C4").Value
            ActiveSheet.Protect Password:=PW, DrawingObjects:=True, Contents:=True, Scenarios:=False
            ActiveSheet.EnableSelection = xlNoRestrictions

End Sub

Sub MAPPE_AUF()

    ' Hilfsprozedur um Arbeitsmappenschutz zu öffnen
    ActiveWorkbook.Unprotect Password:=A.Range("C5").Value

End Sub
Sub MAPPE_ZU()

    ' Hilfsprozedur um Arbeitsmappenschutz zu setzen
    ActiveWorkbook.Protect Password:=A.Range("C5").Value, Structure:=True

End Sub



Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' Neue Version 2015 - ermittelt die letzte Zelle einer Tabelle mit Inhalt

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER
```

```
'   If CDate(Now) >= CDate("01.07.2018") Then
'       MsgBox "Die Frist der Demoversion ist leider abgelaufen - bitte setzen Sie sich bei Interesse mit uns in
Verbindung über unsere Webseite www.simplesoft.at - vielen Dank für Ihre Geduld."
'       End
'   End If

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function
Sub ZEIGE_ALLES()
AUF
Rows("1008:5000").Hidden = False
Range("A1008").Select
```

```
ActiveCell.Offset(0, 1).Select
ZU
End Sub

Sub Kopieren_Aktivieren()
'Tastenkombinationen einschalten
Application.OnKey "^x"
Application.OnKey "^c"
Application.OnKey "^v"
Application.OnKey "+{DEL}"
Application.OnKey "+{INSERT}"

'Drag & Drop wieder erlauben
Application.CellDragAndDrop = True

'Schaltflaechen in Menüleiste => Bearbeiten aktivieren
procControlEnableDisable 21, True ' Ausschneiden
procControlEnableDisable 19, True 'Kopieren
procControlEnableDisable 22, True 'Einfuegen
procControlEnableDisable 755, True 'Inhalte einfuegen
procControlEnableDisable 809, True 'Office-&Zwischenablage

End Sub

Sub Kopieren_Deaktivieren()
'Tastenkombinationen deaktivieren
Application.OnKey "^x", ""
Application.OnKey "^c", ""
Application.OnKey "^v", ""
Application.OnKey "+{DEL}", ""
Application.OnKey "+{INSERT}", ""

'Drag & Drop ausschalten
Application.CellDragAndDrop = False

'Schaltflaechen in Menüleiste => Bearbeiten deaktivieren
procControlEnableDisable 21, False ' Ausschneiden
procControlEnableDisable 19, False 'Kopieren
procControlEnableDisable 22, False 'Einfuegen
procControlEnableDisable 755, False 'Inhalte einfuegen
procControlEnableDisable 809, False 'Office-&Zwischenablage
```

```
End Sub

Sub procControlEnableDisable(intId As Integer, bolStatus As Boolean)
    Dim cmbSuche As CommandBar
    Dim cmbcSteuerelement As CommandBarControl
    On Error Resume Next
    For Each cmbSuche In Application.CommandBars
    Set cmbcSteuerelement = _
    cmbSuche.FindControl(ID:=intId, recursive:=True)
    If Not cmbcSteuerelement Is Nothing Then
    cmbcSteuerelement.Enabled = bolStatus
    End If
    Next
End Sub
```

5.) CODE DES HAUPTMODULS

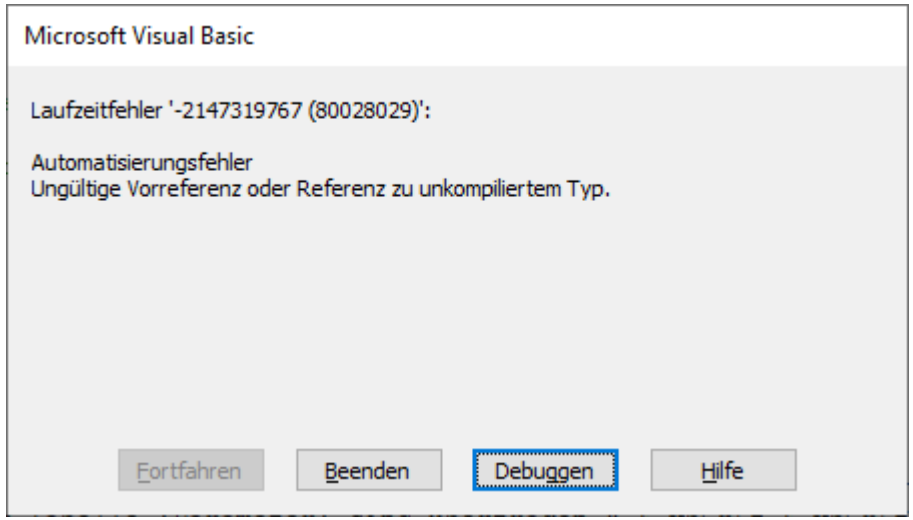
```
Sub FORMELN()

' Schaltfläche FORMELN: sie blendet die Zeilen ein, in denen die Formeln vorhanden sind

' Prüfen des Passwortes

    E.Range("P1") = 1
    UserForm1.Show
    If E.Range("P1") <> "ok" Then
        MsgBox "Das Passwort war leider nicht korrekt - bitte versuchen Sie es erneut"
        Exit Sub
    End If
    E.Range("P1") = ""

    AUF
    E.Columns("O:T").Hidden = False
    E.Rows("10:30").Hidden = False

    MAPPE_AUF
    A.Visible = xlSheetVisible
    MAPPE_ZU

    ActiveWindow.FreezePanes = False ' keine Fenster Fixieren-Funktion
```

```
    Application.CommandBars("Ply").Enabled = True ' rechter Mausklick auf Blattregisterkarten ist wieder aktiviert
    Application.CommandBars("Cell").Enabled = True ' Rechten Mausklick wieder aktivieren

    Kopieren_Aktivieren

    EIN



End Sub

Sub CONTROLLING()

' Schaltfläche CONTROLLING: sie blendet die Spalten O:T ein, aber entfernt nicht den Blattschutz

' Prüfen des Passwortes

    E.Range("P1") = 2
    UserForm1.Show
    If E.Range("P1") <> "ok" Then
        MsgBox "Das Passwort war leider nicht korrekt - bitte versuchen Sie es erneut"
        Exit Sub
    End If
    E.Range("P1") = ""

    AUF
    E.Columns("O:T").Hidden = False
    ZU

    ActiveWindow.FreezePanes = False
    EIN


End Sub

Sub SORTIERUNG()

' Schaltfläche SORTIERUNG, die alles nach der Monatsspalte sortiert

    MAPPE_AUF
    AUF
    ActiveWorkbook.Worksheets("ERFASSUNG").Sort.SortFields.Clear
```

```
    ActiveWorkbook.Worksheets("ERFASSUNG").Sort.SortFields.Add Key:=Range("G1008:G6000"), SortOn:=xlSortOnValues,
Order:=xlAscending, DataOption:= _
        xlSortNormal
    With ActiveWorkbook.Worksheets("ERFASSUNG").Sort
        .SetRange Range("B1008:V5000")
        .Header = xlGuess
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
    ZU
    MAPPE_ZU

End Sub
```

ALLGEMEINES

Automatisierungsfehler ungültige Vorreferenz oder Refrenz zu unkompliertem Typ



1. Auftreten (2023) bei diesem Code:

Worksheets(S).Activate

Lösung (Tabelle direkt ansprechen)

M01.activate

2.Auftreten bei diesem Code (es werden Werte aus anderer, offener Mappe (ALT) ausgelesen

Msgbox Workbooks(ALT).Sheets("1 JAN").Range("F6")

Excel mag mit dem neuesten Update von Office 365 keine solchen Bezüge mehr direkt auf andere Arbeitsmappen. Es geht eigentlich nur noch, indem man zuerst die andere Mappe aktiviert – darin ev. Daten ausliest oder kopiert in die Zwischenablage – dann in die Hauptmappe zurückspringt und dort weiterarbeitet

3.Lösung vom Internet

Problemzeile war:

Workbooks("Mappe2").Tabelle3.Range("A1").Value="Huhu"

Lösung (Mit Hilfsprozedur TabellenIndex liest man vom Tabellennamen die Indexnummer aus (daher: die wievielte Tabelle in der Arbeitsmappe diesen Namen trägt) und übergibt in der Worksheets-Tabelle die entsprechende Indexnummer:

```
Sub Test()
Workbooks("Mappe2").Worksheets(TabellenIndex(Workbooks("Mappe2"), "Tabelle1")).Cells(1, 4).Value = "huhu"
End Sub
Function TabellenIndex(ByRef wkb As Workbook, ByVal strCodename As String) As Integer
Dim wks As Worksheet
For Each wks In wkb.Worksheets
If wks.CodeName = strCodename Then
TabellenIndex = wks.Index
Exit Function
End If
Next wks
End Function
```

Fehler "Das Bild ist zu groß und wird abgeschnitten" beim Speichern

Dieser Fehler hat in der Regel mit einem Datenmüll in der Zwischenablage zu tun (wenn man zuvor viel kopiert hat).

Er sollte nicht kommen, wenn man die Zwischenablage vor dem Speichern leert.

Variante 1: Einfach nur eine Zelle kopieren - hat bei mir nicht geholfen

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

    ' Die Zwischenablage leeren (Fehler "Das Bild ist zu groß und wird abgeschnitten) beheben), indem man einfach eine
Zelle kopiert und damit die Zwischenablage klein macht
    ActiveSheet.Range("A1").Copy
    Application.CutCopyMode = False

End Sub
```

Variante 2:

Wenn Sie größere Datenmengen über die Zwischenablage in Excel per VBA kopieren, kann dies zu Problemen beim Schließen der Datei führen. Excel gibt beispielsweise die Meldung "***Das Bild ist zu groß und wird abgeschnitten***" aus. Häufig friert Excel dabei ein und kann nur noch über den Task-Manger beendet werden.

Um dieses Problem zu vermeiden sollten Sie vor dem Beenden der Datei die Zwischenablage (Clipboard) leeren.
Dazu können Sie folgenden VBA-Code in einem Code-Modul verwenden:

Die ersten drei Zeilen (Private Declare Function…) müssen in einem Modul ganz oben stehen (außerhalb / vor der ersten Prozedur oder Public Function)

```
Private Declare Function OpenClipboard& Lib "user32" (ByVal hwnd As Long)
Private Declare Function EmptyClipboard Lib "user32" () As Long
Private Declare Function CloseClipboard& Lib "user32" ()
```

Und hier die eigentliche Prozedur zum Löschen der Zwischenablage

```
Public Sub ClearClipboard()
OpenClipboard 0&
```

```
EmptyClipboard
CloseClipboard
End Sub
```

Aufgerufen kann diese Prozedur beispielsweise vor dem Beenden der Datei oder vor dem Speichern aufgerufen werden. Tragen Sie dazu folgenden Code in das Modul "DieseArbeitsmappe" ein.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
    ' Die Zwischenablage leeren (Fehler "Das Bild ist zu groß und wird abgeschnitten) behebenn
    ClearClipboard
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
      ClearClipboard
End Sub
```

Prozeduren "unsichtbar" machen

Hilfsprozeduren sollte man, wenn Sie nur im aktuellen Modul gebraucht werden, als PRIVATE deklarieren - denn dadurch werden Sie in der Liste der zuweisbaren Makros (Schnellzugriffsleiste oder Tabellenelemente mit Makro verknüpfen) dort nicht angezeigt

RIESENGROSSE EXCELDATEIEN

Wenn eine Exceldatei beim Speichern plötzlich riesengroß wird - z.B. 40 MB - dann ist in der Regel Schuld, dass man leeren Zellen ein Zellformat zugewiesen hat und Excel die Zellen zwar mit leerem Inhalt, aber eben doch mit Zellformat speichert.

So harmlos mein Befehl Range("A1:IV65000").clearcontents ist, weil er wirklich nur Inhalt löscht, so gegenteilig wäre es z.B. Rahmenlinien auf diese Weise zu löschen - denn KEINE Rahmenlinie ist auch eine Information und bläht das Dateivolumen auf.

Daher z.B. Rahmenlinien immer nur auf den tatsächlich mit Inhalt befüllten Zellbereich entfernen:

```
LZ = LETZTEZELLE(Tabelle3.Name).Row
Tabelle3.Range("L11:O" & LZ).Borders(xlInsideVertical).LineStyle = xlNone
Tabelle3.Range("L11:O" & LZ).Borders(xlInsideHorizontal).LineStyle = xlNone
```

Wie findet man Zellen, die ein Zellformat beinhalten und die Excel groß machen ? STRG-ENDE springt immer zur letzten Zelle, die noch eine Information enthält - und sei es nur LineStyle=xlNone - Lösung: markiere die ganzen Zeilen, die ohnedies leer sind und entferne sie z.B. mit STRG-Minus.

WENN VBA-EDITOR IMMER ZU USERFORM SPRINGT

Sowohl in Excel 2003 als auch in 2010 kommt es vor, dass ein zuvor im Editor mit Doppelclick (oder Rechter Mausklick im Projektexplorer und CODE ANZEIGEN) geöffneter Code einer Userform bei weiteren Starts von anderen Prozeduren in anderen Modulen bei deren Start direkt im VBA-Editor dazu führt, dass der VBA-Editor den zuletzt angezeigten Code verlässt und die Userform im VBA-Editor anzeigt.

Dieses automatische Anzeigen der Userform im Editor nach Ausführen von VBA-Code des Editors nervt natürlich, weil es immer vom zuletzt gesehenen / geänderten VBA-Code wegspringt.

Lösung: die "nervige" immer angezeigte Userform einfach EXPORTIEREN - dann LÖSCHEN - dann IMPORTIEREN.

Unterschied Code in Modul oder direkt in Tabellenblatt

Wichtig: Befindet sich VBA-Code direkt in einem Tabellenblatt, bezieht sich Range("A1") IMMER auf die Zelle A1 in diesem Tabellenblatt, SELBST WENN DER VBA-CODE auf eine andere Tabelle umgeschalten hat und diese aktiv wäre. Nur in Modulen greifen Range und Cells-Befehle OHNE Tabellenbezug auf die aktuelle Tabelle.

Ansonsten immer die betreffende Tabelle mitübergeben (Sheets("Jan").Range...) oder - wenn man die Tabelle zuvor aktiviert hat - mit ActiveSheet.Range... arbeiten

DEFAULT CODE

Dies ist der Code, den ich gerne in neuen VBA-Projekten per default zur Verfügung stelle#

Modul ALLGEMEIN

```
' _________________________
'|                         |
'| MODUL - BESCHREIBUNG    |
```

```
'|_____________________|

' Dieses Modul enthält alle Prozeduren, die ALLGEMEIN für andere Module von Bedeutung sind

' _____________________
'|                     |
'| ALLGEMEINE VARIABLEN  |
'|_____________________|

' KEINE

' ___________________________________________________________________________________

' _____________________________
'|                             |
'| ÖFFENTLICHE FUNKTIONEN  |
'|_____________________________|
'
Public Function SPALTENTEXT(SPALTENNUMMER As Integer) As String

' Diese Funktion wandelt Spaltennummern in Spaltentext um: 5=>E

    If SPALTENNUMMER > 26 Then
      SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 2)
    Else
      SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 1)
    End If

End Function

Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' ermittelt die letzte Zelle einer Tabelle mit Inhalt oder Zellformat

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER
```

```
   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function

Public Function FileFolderExists(strFullPath As String) As Boolean

    ' Diese Funktion überprüft ob ein Ordner oder eine Datei existiert

    On Error GoTo EarlyExit
    If Not Dir(strFullPath, vbDirectory) = vbNullString Then FileFolderExists = True
EarlyExit:
    On Error GoTo 0

End Function
```

```
'  _________________________________
'|                                 |
'| ÖFFENTLICHE PROZEDUREN          |
'|_________________________________|


Public Sub AUS()

' Schaltet Bildschirmaktualisierung und Events ab

    Application.ScreenUpdating = False
    Application.EnableEvents = False
    ' Calculation Manual macht Sinn, wenn die Tabelle sehr viele Formeln mit Zellbezügen hat
    ' Application.Calculation = xlCalculationManual

End Sub
Public Sub EIN()

' Schaltet Bildschirmaktualisierung und Events ein

   Application.ScreenUpdating = True
   Application.EnableEvents = True
   ' Calculation Manual macht Sinn, wenn die Tabelle sehr viele Formeln mit Zellbezügen hat
   ' Application.Calculation = xlCalculationAutomatic

End Sub



Public Function OrdnerAuswahl() As String
    Dim bInfo As BROWSEINFO
    Dim strPath As String
    Dim R As Long
    Dim X As Long
    Dim pos As Integer

    ' Der Ausgangsordner ist der Desktop :
    bInfo.pidlRoot = 0& ' 5& wären die Eigenen Dateien
    ' Dialogtitel
    bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
    ' Rückgabe des Unterverzeichnisses
```

```
    bInfo.ulFlags = &H1
    ' Dialog anzeigen
    X = SHBrowseForFolder(bInfo)
    ' Ergebnis gliedern
    strPath = Space$(512)
    ' Ausgewähltes Verzeichnis einlesen
    R = SHGetPathFromIDList(ByVal X, ByVal strPath)
    If R Then
    pos = InStr(strPath, Chr$(0))
    OrdnerAuswahl = Left(strPath, pos - 1)
    Else
    OrdnerAuswahl = ""
    End If
End Function


' _________________________________
'|                                 |
'|  ALLGEMEINE PROZEDUREN          |
'|_________________________________|


Sub VBA_Test()
' Code des Buttons in der Anleitung/VBA-Tabelle
MsgBox "Gratulation - bei Ihnen ist der VBA-Code aktiv. " & vbCrLf & vbCrLf & _
"Sollten Sie bei jedem Öffnen der Zeiterfassungsdatei in der eingeblendeten Infozeile oben auf 'INHALTE AKTIVIEREN'
klicken müssen," & _
"damit der VBA-Code aktiv ist, können Sie In der Tabelle ANLEITUNG ganz oben unter 1.a nachlesen, wie man den VBA-Code
dauerhaft einschalten kann."

End Sub

Sub VBA_EIN()
' Code in der Parametertabelle oben der Button VBA-Test
' er schaltet verlässlich den VBA-Code ein, wenn das Enableevent auf FALSE geschaltet blieb durch zB einen Programmfehler
und Abbruch.
EIN
MsgBox "VBA-Code ist bei Ihnen aktiv"
End Sub
```

```
Sub WAEHLE_ORDNER()
Dim neuOrdner

neuOrdner = OrdnerAuswahl
If neuOrdner = "" Then
    Exit Sub
  Else
    ChDir neuOrdner
End If
MsgBox neuOrdner
End Sub


Sub CURSORUNTEN()
    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlDown
    End With
End Sub

Sub CURSORRECHTS()
    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlToRight
    End With
End Sub

Sub CURSORNO()
    Application.MoveAfterReturn = False
End Sub

Sub Zoom_groesser()
     ActiveWindow.Zoom = ActiveWindow.Zoom + 5
End Sub

Sub Zoom_kleiner()
     ActiveWindow.Zoom = ActiveWindow.Zoom - 5
End Sub
```

Modul DIESEARBEITSMAPPE

VERSION 1

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    ' Ausblenden Symbolleiste
    On Error Resume Next
    Application.CommandBars("VBA-PROGRAMME").Delete

End Sub

Private Sub Workbook_Open()

    Dim STUNDE
    Dim MINUTEN
    Dim SEKUNDE
    Dim WAITTIME
    Dim EXCELVERSION As Integer
    Dim EV As Integer ' Excelversion




        ' 1.) ANZEIGE USERFORM9
        EXCELVERSION = Val(Application.Version)
        If EXCELVERSION > 8 Then
            UserForm9.Show vbModeless ' Ungebundene Userforms erst ab Excel 2000
            ' Eine Sekunde warten, damit USERFORM9 korrekt angezeigt wird
            STUNDE = Hour(Now())
            MINUTEN = Minute(Now())
            SEKUNDE = Second(Now()) + 1
            WAITTIME = TimeSerial(STUNDE, MINUTEN, SEKUNDE)
            Application.Wait WAITTIME
        End If


        UserForm9.Label1.Caption = "ZEITENRECHNER Meta Communciations GmbH"

   ' Druckbereich automatisch einstellen

    Sheets("Anleitung").PageSetup.PrintArea = "$D$4:$M$" & Range("D65536").End(xlUp).Row
```

```
On Error Resume Next
Columns("A:Q").Select
ActiveWindow.Zoom = True

Range("P2").Select

            STUNDE = Hour(Now())
            MINUTEN = Minute(Now())
            SEKUNDE = Second(Now()) + 2
            WAITTIME = TimeSerial(STUNDE, MINUTEN, SEKUNDE)
            Application.Wait WAITTIME

Unload UserForm9

End Sub
```

VERSION 2

Private Sub Workbook_Open()

```
Dim VERSIONBISHER As String



' hier trägt man die aktuelle Versionnummer ein, wenn man das Tool neu ausrollt auf
' andere PC's

' ************************************************
' **                                            **
               VERSIONAKTUELL = "2009-05-16"
' **                                            **
' ************************************************


On Error Resume Next

' Auslesen der Menüleisten-Option (ob breite oder schmale Menüleiste)
'If GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Menueleiste") <> "2" Then
```

```
'    Application.CommandBars("Powertools").Visible = True
'    Application.CommandBars("Powertools_kompakt").Visible = False
'  Else
'    Application.CommandBars("Powertools").Visible = False
'    Application.CommandBars("Powertools_kompakt").Visible = True
'End If


' Auslesen der letzten Powertools-Version
  VERSIONBISHER = GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Versionbisher")

  ' wenn es bereits eine bisherige Version gab (daher keine Erstinstallation) und
  ' wenn die bisherige Version nicht gleich der aktuellen Version ist, gib Hinweis über Neustart

  If VERSIONBISHER <> "" And VERSIONBISHER <> VERSIONAKTUELL Then

    MsgBox "Sie haben von den Powertools die neue Version " & VERSIONAKTUELL & " im Einsatz." & vbCrLf & vbCrLf & _
    "Damit alle neuen Funktionen korrekt in der Symbolleiste angezeigt werden," & vbCrLf & _
    "müssen Sie bitte einmal Excel schließen und dann erneut starten." & vbCrLf & vbCrLf & _
    "(Grund: erst beim nächsten Schließen wird die alte Symbolleiste gelöscht" & vbCrLf & _
    "und durch die neue ersetzt.)" & vbCrLf & vbCrLf & "Vielen Dank und ein erfolgreiches Arbeiten gewünscht !"

  End If

' Setzen des Wertes VERSIONAKTUELL
           SaveSetting "Simplesoft", "Powertools", "Versionaktuell", VERSIONAKTUELL

' Zuweisen der Funktion für die Taste ENDE und POS1

If GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Endetaste") = "1" Then
    Application.OnKey "{END}", "EXTRA_SPRING_ZUR_LETZTEN_ZEILE"
  Else
    Application.OnKey "{END}", "EXTRA_SPRING_ZUR_RECHTESTEN_ZELLE"
End If
If GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Pos1taste") = "1" Then
    Application.OnKey "{POS1}", "EXTRA_SPRING_ZUR_OBERSTEN_ZEILE"
  Else
    Application.OnKey "{POS1}", "EXTRA_SPRING_ZUR_LINKESTEN_ZELLE"
End If

' Zuweisen der Funktion für die F12-Taste
```

```
If GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="F12") <> "2" Then
    Application.OnKey "{F12}", "EXTRA_F12_WERTEINFUEGEN"
End If

' Zuweisen der Funktion für die F11-Taste

    Application.OnKey "{F11}", "EXTRA_FORMELCHECK"


On Error GoTo 0

End Sub
```

Private Sub Workbook_BeforeClose(Cancel As Boolean)

```
Dim VERSIONBISHER As String
Dim VERSIONAKTUELL As String

    On Error Resume Next

  VERSIONBISHER = GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Versionbisher")
  VERSIONAKTUELL = GetSetting(appname:="Simplesoft", section:="Powertools", KEY:="Versionaktuell")

  ' wenn bisherige Version gleich der aktuellen Version ist, dann Abbruch
  If VERSIONBISHER = VERSIONAKTUELL Then Exit Sub

  ' sonst speichern der aktuellen Versionsnummer und löschen der Symbolleiste
  SaveSetting "Simplesoft", "Powertools", "Versionbisher", VERSIONAKTUELL
  Application.CommandBars("Powertools").Delete
  Application.CommandBars("Powertools_kompakt").Delete

    On Error GoTo 0

End Sub
```

CODE-OPTIMIERUNG

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Druckversion&action=edit&section=15Code-Optimierung

Zellen nicht einzeln mit Schleife durchlaufen sondern mit FOR EACH

```
Dim ZELLE As Range
On Error GoTo FEHLER
For Each ZELLE In ActiveSheet.UsedRange.Cells
   If IsDate(ZELLE) = True And ZELLE.Column <> 62 Then ZELLE = ZELLE + 1462
Next ZELLE
```

Die folgende Grundsätze verhelfen zu einer optimalen Ablaufgeschwindigkeit Ihres VBA-Programms:

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-1**Konstanten**

Deklarieren Sie, wo immer möglich, Konstanten statt Variablen.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-2**Objektindex**

Wenn es die Klarheit des Codes nicht stört, verwenden Sie bei Objekt-Schleifen den Index des Objektes, nicht den Namen.

```
Worksheets(intCounter)
```

ist schneller als

```
Worksheets("Tabelle1")
```

Allerdings gehen For-Each-Schleifen vor, denn

```
For Each wksData In Worksheets:Next
```

ist schneller als

```
Worksheets(intCounter)
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-3

Direkte Objektzuweisungen

Verwenden Sie nach Möglichkeit keine allgemeinen Objektzuweisungen wie:

```
Dim wksData As Object
```

Deklarieren Sie so genau wie möglich:

```
Dim wksData As Worksheet
```

Die zweite Art der Deklaration ist nicht immer möglich, denn sie setzt voraus, dass die Bibliothek über die Verweise (References) eingebunden ist. Die zweite Art der Deklaration hat auch den Vorteil, dass IntelliSense nach Eingabe eines Punktes Vorschläge machen kann, welche Eigenschaften und Methoden zu dem Objekt passen. Wenn die Objekte einer anderen Anwendung entstammen (z.B. Word oder Access), muss zunächst der Verweis auf die Objektbibliothek eingefügt werden, damit Intellisense funktioniert.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-4

Selektieren

Wählen Sie keine Arbeitsmappen, Blätter, Bereiche oder andere Objekte aus:

```
Workbooks("Test.xls").Activate
Worksheets("Tabelle1").Select
Range("A1").Select
ActiveCell.Value = 12
```

Referenzieren Sie stattdessen exakt:

```
Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1").Value = 12
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-5

Keine eckigen Klammern

Verwenden Sie für Zellbereiche nicht die Schreibweise in eckigen Klammern:

```
b3= d4]
```

Schreiben Sie stattdessen (Ausführungszeit ca. 66% von vorigem):

```
Range("B3").Value = Range("D4").Value
```

Noch etwas schneller (Ausführungszeit ca. 90% von vorigem bzw. 60% von ersterem):

```
Cells(3,2).Value = Cells(4,4).Value   ' Cells(ZeilenNr, SpaltenNr)
```

Hinweis: Beachten Sie, dass bei Angabe des Zellbezug als String die Range-Eigenschaft verwendet werden muss, wohingegen bei der Angabe als Zahlen die Cells-Eigenschaft verwendet werden muss.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-6

Direkte Referenzierung

Referenzieren Sie - wenn der Programmablauf es nicht erforderlich macht - nicht hierarchieweise:

```
Set wkbData = Workbooks("Test.xls")
Set wksData = wkbData.Worksheets("Tabelle1")
Set rngData = wksData.Range("A1:F16")
```

Referenzieren Sie stattdessen direkt das Zielobjekt:

```
Set rngData = Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1:F16")
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-7

Dimensionierung

Dimensionieren Sie die Variablen nicht allgemeiner als dies erforderlich ist:

```
Dim intCounter As Integer
ist schneller als:
Dim varCounter as Variant
```

Hinweise:

- Wenn eigentlich der Datentyp Byte ausreichen sollte, kann eine Subtraktion manchmal einen Unterlauf verursachen. Die Gefahr besteht vor allem bei FOR-Schleifen mit einem negativen Argument für STEP. In diesem Falle bei INTEGER bleiben.
- In bestimmten Fällen kann man den Datentyp `Variant` nicht vermeiden, beispielsweise hier:
  - Der Rückgabewert einer Funktion soll bei Fehlern auch einen Fehlerwert der Funktion `CVErr()` ausgeben
  - VBA hat keinen eigenen Datentyp für lange Dezimalzahlen vom Typ `Dec` (Umwandlung mit `CDec()`)
  - Bei optionalen Argumenten (mit `Option MyVar` einer Sub/Function kann mit `IsMissing()` nur auf ausgelassene Argumente geprüft werden, wenn der Datentyp `Variant` ist. Andernfalls erhält man immer die default-Belegung des Datentyps von MyVar, wenn das Argument ausgelassen wird
  - Wenn eine Variable ein Array aufnehmen soll, muss sie vom Typ Variant sein

Tipp: Noch etwas schneller als der Integer ist der Datentyp Long! Das liegt vermutlich daran, dass Integer 16-bittig ist während Long 32-bittig ist und alle neueren Prozessoren für 32-Bit optimiert sind.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-8

With-Rahmen

Verwenden Sie With-Rahmen. Langsam ist:

```
Worksheets("Tabelle1").Range("A1:A16").Font.Bold = True
Worksheets("Tabelle1").Range("A1:A16").Font.Size = 12
Worksheets("Tabelle1").Range("A1:A16").Font.Name = "Arial"
Worksheets("Tabelle1").Range("A1:A16").Value = "Hallo!"
```

Schneller ist:

```
With Worksheets("Tabelle1").Range("A1:A16")
    With .Font
        .Bold = True
        .Size = 12
        .Name = "Arial"
    End With
    .Value = "Hallo!"
End With
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-9

Excel-Funktionen

Ziehen Sie Excel-Funktionen VBA-Routinen vor. Langsam ist:

```
For intCounter = 1 To 20
    dblSum = dblSum + Cells(intCounter, 1).Value
Next intCounter
```

Schneller ist:

```
dblSum = WorksheetFunction.Sum(Range("A1:A20"))
```

Wenn Sie große, zusammenhängende Zellbereich berechnen müssen, setzen Sie zur eigentlichen Berechnung Excel-Formeln ein. Die Formeln können Sie danach in absolute Werte umwandeln:

```
Sub Berechnen()
    Dim intRow As Integer
    intRow = Cells(Rows.Count, 1).End(xlUp).Row
    Range("C1").Formula = "=A1+B1/Pi()"
    Range("C1:C" & intRow).FillDown
    Columns("C").Copy
    Columns("C").PasteSpecial Paste:=xlValues
    Application.CutCopyMode = False
    Range("A1").Select
End Sub
```

Dasselbe Ergebnis hat folgende Prozedur, die auch With-Klammern verwendet und bei der Ersetzung der Formeln durch Werte ohne Copy/PasteSpecial auskommt:

```
Sub Berechnen2()
    Dim lngRow As Long
    lngRow = Cells(Rows.Count, 1).End(xlUp).Row
    With Range("C1:C" & lngRow)
      .Formula = "=A1+B1/Pi()" ' trägt die Formeln ein
      .Formula = .Value        ' ersetzt die Formeln durch Werte; .Value = .Value geht auch
    End With
    Range("A1").Select ' nur, wenn das nötig/erwünscht ist
End Sub
```

Tipp: Wenn Sie auf eine große Anzahl Zellen zugreifen müssen, dann ist es am Schnellsten, wenn Sie die Werte mit einem Befehl in ein Array kopieren und dann aus dem Array lesen:

```
Sub Berechne3()
    dim a
    dim i as long, j as long, sum as long
    a = me.Range("A1:H800").value
    for i=1 to 8
       for j=1 to 800
           sum=sum+a(j,i) ' a(ZeilenNr, SpaltenNr)
       next j
    next i
    debug.print sum
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Code-Optimierung&action=edit&section=T-10

Array-Formeln

Setzen Sie temporäre Excel-Array-Formeln zur Matrixberechnung ein. Wenn Sie in VBA zwei Zellbereiche auf Übereinstimmung überprüfen wollen, müssen Sie einzelne Zellvergleiche vornehmen. Mit Einsatz einer Excel-Array-Formel sind Sie schneller. Im nachfolgenden Code werden zwei große Zellbereiche auf Übereinstimmung überprüft. Über VBA müsste man jede einzelne Zelle des einen mit der des anderen Bereiches vergleichen. Die Excel-Array-Formel liefert das Ergebnis unmittelbar nach dem Aufruf:

```
Function MatrixVergleich(strA As String, strB As String) As Boolean
    Range("IV1").FormulaArray = "=SUM((" & strA & "=" & strB & ")*1)"
    If Range("IV1").Value - Range(strA).Cells.Count = 0 Then
        MatrixVergleich = True
    End If
    Range("IV1").ClearContents
End Function

Sub Aufruf()
    MsgBox MatrixVergleich("C1:D15662", "E1:F15662")
```

```
End Sub
```

32-Bit / 64-Bit API-Aufrufe

Ich hatte folgenden Funktions-Definitions-Code bei der Simplefibu im Einsatz, weil damit alle Codestellen arbeiteten, die PDF-Dateien öffneten, aber sobald man den Code unter 64-bit Excel öffnete kam ein Compilierungsfehler genau bei dieser Funktions-Definition, weil er mit der folgenden Zeile nicht zurecht kam

```
' Funktionen für die Funktion gbOpenFile
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
```

Änderte man diese Zeile jedoch um auf Folgendes – dann klappte es. (Wichtig: die #-Zeichen erlauben ein If, else, End if auch außerhalb von einer Sub !)

```
#If VBA7 Then
    Private Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
        (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal
nShowCmd As Long) As Long
#Else
    Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
        (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal
nShowCmd As Long) As Long
#End If
```

Jemand schrieb, dass die korrekte Unterscheidung zwischen 32 bit und 64 nicht nur mit VBA7 erfolgen sollte, sondern besser so

```
#If Win64 Then
' Win64=true, Win32=true, Win16= false
#ElseIf Win32 Then
' Win32=true, Win16=false
#Else
' Win16=true
#End If
```

Und noch jemand schrieb, dass es besser so geht:

Corrected typo from the book "Microsoft Excel 2010 Power Programming with VBA".

```
#If vba7 and win64 then
```

```
  declare ptrsafe function ....
#Else
  declare function ....
#End If
```

Bei mir klappte aber gleich die erste Lösung in Simplefibu unter Exc el 2013 64 bit

Hier noch ein Artikel mit mehreren Lösung zu diesem Problem für verschiedene API-Aufrufe

Declaring API functions in 64 bit Office
Introduction

With the introduction of Windows 7 and Office 2010 VBA developers face a new challenge: ensuring their applications work on both 32 bit and 64 bit platforms.

This page is meant to become the first stop for anyone who needs the proper syntax for his API declaration statement in Office VBA.

Many of the declarations were figured out by Charles Williams of www.decisionmodels.com when he created the 64 bit version of our Name Manager.

Links

Of course Microsoft documents how to do this. There is an introductory article on Microsoft MSDN:

Compatibility Between the 32-bit and 64-bit Versions of Office 2010

That article describes the how-to's to properly write the declarations. What is missing is which type declarations go with which API function or sub.

Microsoft has provided an updated version of the Win32API.txt with all proper declarations available for download here:

Office 2010 Help Files: Win32API_PtrSafe with 64-bit Support

When you run the installer after downloading the file form the link above, it does not tell you where it installed the information. Look in this -new- folder on your C drive:

C:\Office 2010 Developer Resources\Documents\Office2010Win32API_PtrSafe

You can find a list of the old Win32 API declarations here:

Visual Basic Win32 API Declarations

Microsoft also published a tool to check your code for 64 bit related problems, called the Microsoft Office Code Compatibility inspector addin.

API functions that were added/modified in 64-bit Windows: http://msdn.microsoft.com/en-us/library/aa383663(VS.85).aspx

API Functions by Windows release:

http://msdn.microsoft.com/en-us/library/aa383687(VS.85).aspx

Utter Access API declarations (a comprehensive list of many declarations)

Declarations by API function

Function name | **Declarations (32 bit followed by 64 bit)**

CreateProcess

We start off with a complicated one because it has a lot of arguments. A fully functional example is included below the example declaration lines.
Courtesy: The example code was taken from this page

```
Declare Function CreateProcess Lib "kernel32" _
        Alias "CreateProcessA" (ByVal lpApplicationName As String, _
                ByVal lpCommandLine As String, _
                lpProcessAttributes As SECURITY_ATTRIBUTES, _
                lpThreadAttributes As SECURITY_ATTRIBUTES, _
                ByVal bInheritHandles As Long, _
                ByVal dwCreationFlags As Long, _
                lpEnvironment As Any, _
                ByVal lpCurrentDriectory As String, _
                lpStartupInfo As STARTUPINFO, _
                lpProcessInformation As PROCESS_INFORMATION) As Long


Declare PtrSafe Function CreateProcess Lib "kernel32" _
        Alias "CreateProcessA" (ByVal lpApplicationName As String, _
                ByVal lpCommandLine As String, _
                lpProcessAttributes As SECURITY_ATTRIBUTES, _
                lpThreadAttributes As SECURITY_ATTRIBUTES, _
                ByVal bInheritHandles As Long, _
                ByVal dwCreationFlags As Long, _
                lpEnvironment As Any, _
                ByVal lpCurrentDriectory As String, _
                lpStartupInfo As STARTUPINFO, _
                lpProcessInformation As PROCESS_INFORMATION) As LongPtr
```

```
'Full example shown below, including the necessary structures
#If VBA7 Then
   Declare PtrSafe Function CreateProcess Lib "kernel32" _
                            Alias "CreateProcessA" (ByVal lpApplicationName As String, _
                                                    ByVal lpCommandLine As String, _
                                                    lpProcessAttributes As SECURITY_ATTRIBUTES, _
                                                    lpThreadAttributes As SECURITY_ATTRIBUTES, _
                                                    ByVal bInheritHandles As Long, _
                                                    ByVal dwCreationFlags As Long, _
                                                    lpEnvironment As Any, _
                                                    ByVal lpCurrentDriectory As String, _
                                                    lpStartupInfo As STARTUPINFO, _
                                                    lpProcessInformation As PROCESS_INFORMATION) As LongPtr

   Const INFINITE = &HFFFF
   Const STARTF_USESHOWWINDOW = &H1
Private Enum enSW
   SW_HIDE = 0
   SW_NORMAL = 1
   SW_MAXIMIZE = 3
   SW_MINIMIZE = 6
End Enum

Private Type PROCESS_INFORMATION
   hProcess As LongPtr
   hThread As LongPtr
   dwProcessId As Long
   dwThreadId As Long
End Type

Private Type STARTUPINFO
   cb As Long
   lpReserved As String
   lpDesktop As String
   lpTitle As String
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
```

```
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Byte
    hStdInput As LongPtr
    hStdOutput As LongPtr
    hStdError As LongPtr
End Type

Private Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As LongPtr
    bInheritHandle As Long
End Type

Private Enum enPriority_Class
    NORMAL_PRIORITY_CLASS = &H20
    IDLE_PRIORITY_CLASS = &H40
    HIGH_PRIORITY_CLASS = &H80
End Enum
#Else
    Declare Function CreateProcess Lib "kernel32" _
                            Alias "CreateProcessA" (ByVal lpApplicationName As String, _
                                                    ByVal lpCommandLine As String, _
                                                    lpProcessAttributes As SECURITY_ATTRIBUTES, _
                                                    lpThreadAttributes As SECURITY_ATTRIBUTES, _
                                                    ByVal bInheritHandles As Long, _
                                                    ByVal dwCreationFlags As Long, _
                                                    lpEnvironment As Any, _
                                                    ByVal lpCurrentDriectory As String, _
                                                    lpStartupInfo As STARTUPINFO, _
                                                    lpProcessInformation As PROCESS_INFORMATION) As Long

    Const INFINITE = &HFFFF
    Const STARTF_USESHOWWINDOW = &H1
Private Enum enSW
    SW_HIDE = 0
    SW_NORMAL = 1
    SW_MAXIMIZE = 3
    SW_MINIMIZE = 6
End Enum

Private Type PROCESS_INFORMATION
```

```
    hProcess As Long
    hThread As Long
    dwProcessId As Long
    dwThreadId As Long
End Type

Private Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Byte
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

Private Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type

Private Enum enPriority_Class
    NORMAL_PRIORITY_CLASS = &H20
    IDLE_PRIORITY_CLASS = &H40
    HIGH_PRIORITY_CLASS = &H80
End Enum
#End If

Private Function SuperShell(ByVal App As String, ByVal WorkDir As String, dwMilliseconds As Long, _
                            ByVal start_size As enSW, ByVal Priority_Class As enPriority_Class) As Boolean
```

```
    Dim pclass As Long
    Dim sinfo As STARTUPINFO
    Dim pinfo As PROCESS_INFORMATION
    'Not used, but needed
    Dim sec1 As SECURITY_ATTRIBUTES
    Dim sec2 As SECURITY_ATTRIBUTES
    'Set the structure size
    sec1.nLength = Len(sec1)
    sec2.nLength = Len(sec2)
    sinfo.cb = Len(sinfo)
    'Set the flags
    sinfo.dwFlags = STARTF_USESHOWWINDOW
    'Set the window's startup position
    sinfo.wShowWindow = start_size
    'Set the priority class
    pclass = Priority_Class

    'Start the program
    If CreateProcess(vbNullString, App, sec1, sec2, False, pclass, _
                0&, WorkDir, sinfo, pinfo) Then
        'Wait
       ' WaitForSingleObject pinfo.hProcess, dwMilliseconds
        SuperShell = True
    Else
        SuperShell = False
    End If
End Function

Sub Test()
    Dim sFile As String
    'Set the dialog's title
    sFile = Application.GetOpenFilename("Executables (*.exe), *.exe", , "")
    SuperShell sFile, Left(sFile, InStrRev(sFile, "\")), 0, SW_NORMAL, HIGH_PRIORITY_CLASS
End Sub
```

FindWindow

```
Private Declare Function FindWindow Lib "USER32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName
As String) As Long

Private Declare PtrSafe Function FindWindow Lib "USER32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal
lpWindowName As String) As LongPtr
```

FindWindowEx

```
Private Declare Function FindWindowEx Lib "USER32" _
                        Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, _
                        ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
```

```
                    Private Declare PtrSafe Function FindWindowEx Lib "USER32" _
                                    Alias "FindWindowExA" (ByVal hWnd1 As LongPtr, ByVal hWnd2 As LongPtr, _
                                    ByVal lpsz1 As String, ByVal lpsz2 As String) As LongPtr
GdipCreateBitmapFromFile    Private Declare Function GdipCreateBitmapFromFile Lib "GDIPlus" (ByVal filename As Long, bitmap As Long) As Long

                    Private Declare PtrSafe Function GdipCreateBitmapFromFile Lib "GDIPlus" (ByVal filename As LongPtr, bitmap As LongPtr) As
                    LongPtr
GdipCreateHBITMAPFromBitmap Private Declare Function GdipCreateHBITMAPFromBitmap Lib "GDIPlus" (ByVal bitmap As Long, hbmReturn As Long, ByVal
                    background As Long) As Long

                    Private Declare PtrSafe Function GdipCreateHBITMAPFromBitmap Lib "GDIPlus" (ByVal bitmap As LongPtr, hbmReturn As
                    LongPtr, ByVal background As Long) As LongPtr
GdipDisposeImage    Private Declare Function GdipDisposeImage Lib "GDIPlus" (ByVal image As Long) As Long

                    Private Declare PtrSafe Function GdipDisposeImage Lib "GDIPlus" (ByVal image As LongPtr) As LongPtr
GdiplusShutdown     Private Declare Function GdiplusShutdown Lib "GDIPlus" (ByVal token As Long) As Long

                    Private Declare PtrSafe Function GdiplusShutdown Lib "GDIPlus" (ByVal token As LongPtr) As LongPtr
GdiplusStartup      Private Declare Function GdiplusStartup Lib "GDIPlus" (token As Long, inputbuf As GdiplusStartupInput, Optional ByVal
                    outputbuf As Long = 0) As Long
                    Private Type GdiplusStartupInput
                       GdiplusVersion As Long
                       DebugEventCallback As Long
                       SuppressBackgroundThread As Long
                       SuppressExternalCodecs As Long
                    End Type

                    Private Declare PtrSafe Function GdiplusStartup Lib "GDIPlus" (token As LongPtr, inputbuf As GdiplusStartupInput, Optional
                    ByVal outputbuf As LongPtr = 0) As LongPtr

                    Private Type GdiplusStartupInput
                       GdiplusVersion As Long
                       DebugEventCallback As LongPtr
                       SuppressBackgroundThread As Long
                       SuppressExternalCodecs As Long
                    End Type
GetClassName        Public Declare Function GetClassName Lib "USER32" Alias "GetClassNameA" _
                                        (ByVal hWnd As Long, ByVal lpClassName As String, _
                                         ByVal nMaxCount As Long) As Long
```

```
Public Declare PtrSafe Function GetClassName Lib "USER32" Alias "GetClassNameA" _
                                (ByVal hWnd As LongPtr, ByVal lpClassName As String, _
                                 ByVal nMaxCount As LongPtr) As Long
```

GetDiskFreeSpaceEx

```
Private Declare Function GetDiskFreeSpaceEx Lib "kernel32" _
   Alias "GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, _
   lpFreeBytesAvailableToCaller As Currency, _
   lpTotalNumberOfBytes As Currency, _
   lpTotalNumberOfFreeBytes As Currency) As Long
Private Declare PtrSafe Function GetDiskFreeSpaceEx Lib "kernel32" Alias _
   "GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, _
   lpFreeBytesAvailableToCaller As Currency, lpTotalNumberOfBytes As _
   Currency, lpTotalNumberOfFreeBytes As Currency) As LongPtr
```

getDC

```
Private Declare Function GetDC Lib "USER32" (ByVal hWnd As Long) As Long

Private Declare PtrSafe Function GetDC Lib "USER32" (ByVal hWnd As LongPtr) As LongPtr
```

GetDesktopWindow

```
Public Declare Function GetDesktopWindow Lib "USER32" () As Long

Public Declare PtrSafe Function GetDesktopWindow Lib "USER32" () As LongPtr
```

getDeviceCaps

```
Private Declare Function GetDeviceCaps Lib "gdi32" (ByVal hDC As Long, ByVal nIndex As Long) As Long

Private Declare PtrSafe Function GetDeviceCaps Lib "gdi32" (ByVal hDC As LongPtr, ByVal nIndex As Long) As Long
```

GetDriveType

```
Private Declare Function GetDriveType Lib "kernel32" Alias _
                         "GetDriveTypeA" (ByVal sDrive As String) As Long

Private Declare PtrSafe Function GetDriveType Lib "kernel32" Alias _
                               "GetDriveTypeA" (ByVal sDrive As String) As LongPtr
```

GetExitCodeProcess

```
#If VBA7 Then
   Declare PtrSafe Function GetExitCodeProcess Lib "kernel32" (ByVal _
      hProcess As LongPtr, lpExitCode As Long) As Long
#Else
   Declare Function GetExitCodeProcess Lib "kernel32" (ByVal _
      hProcess As Long, lpExitCode As Long) As Long
#End If
```

GetForegroundWindow

```
Declare Function GetForegroundWindow Lib "user32.dll" () As Long

Declare PtrSafe Function GetForegroundWindow Lib "user32.dll" () As LongPtr
```

getFrequency

```
Declare Function getFrequency Lib "kernel32" Alias "QueryPerformanceFrequency" (cyFrequency As Currency) As Long

Private Declare PtrSafe Function getFrequency Lib "kernel32" Alias "QueryPerformanceFrequency" (cyFrequency As Currency)
As Long
```

GetKeyState

```
Declare Function GetKeyState Lib "USER32" (ByVal vKey As Long) As Integer
```

```
Declare PtrSafe Function GetKeyState Lib "USER32" (ByVal vKey As Long) As Integer
```

GetLastInputInfo

```
#If VBA7 Then
   Private Type LASTINPUTINFO
      cbSize As LongPtr
      dwTime As LongPtr
   End Type
   Private Declare PtrSafe Sub GetLastInputInfo Lib "USER32" (ByRef plii As LASTINPUTINFO)
#Else
   Private Type LASTINPUTINFO
      cbSize As Long
      dwTime As Long
   End Type
   Private Declare Sub GetLastInputInfo Lib "USER32" (ByRef plii As LASTINPUTINFO)
#End If
```

GetOpenFileName

```
Option Explicit

#If VBA7 Then
   Public Declare PtrSafe Function GetOpenFileName Lib "comdlg32.dll" Alias _
         "GetOpenFileNameA" (pOpenfilename As OPENFILENAME) As Long

   Public Type OPENFILENAME
      lStructSize As Long
      hwndOwner As LongPtr
      hInstance As LongPtr
      lpstrFilter As String
      lpstrCustomFilter As String
      nMaxCustFilter As Long
      nFilterIndex As Long
      lpstrFile As String
      nMaxFile As Long
      lpstrFileTitle As String
      nMaxFileTitle As Long
      lpstrInitialDir As String
      lpstrTitle As String
      flags As Long
      nFileOffset As Integer
      nFileExtension As Integer
      lpstrDefExt As String
      lCustData As Long
      lpfnHook As LongPtr
      lpTemplateName As String
   End Type
```

```
#Else

   Public Declare Function GetOpenFileName Lib "comdlg32.dll" Alias _
          "GetOpenFileNameA" (pOpenfilename As OPENFILENAME) As Long

   Public Type OPENFILENAME
      lStructSize As Long
      hwndOwner As Long
      hInstance As Long
      lpstrFilter As String
      lpstrCustomFilter As String
      nMaxCustFilter As Long
      nFilterIndex As Long
      lpstrFile As String
      nMaxFile As Long
      lpstrFileTitle As String
      nMaxFileTitle As Long
      lpstrInitialDir As String
      lpstrTitle As String
      flags As Long
      nFileOffset As Integer
      nFileExtension As Integer
      lpstrDefExt As String
      lCustData As Long
      lpfnHook As Long
      lpTemplateName As String
   End Type
#End If
'////////////////////////////////
'// End code GetOpenFileName    //
'////////////////////////////////


Public Function GetMyFile(strTitle As String) As String

   Dim OpenFile    As OPENFILENAME
   Dim lReturn     As Long

   OpenFile.lpstrFilter = ""
   OpenFile.nFilterIndex = 1
   OpenFile.hwndOwner = 0
   OpenFile.lpstrFile = String(257, 0)
```

```
    #If VBA7 Then
       OpenFile.nMaxFile = LenB(OpenFile.lpstrFile) - 1
       OpenFile.lStructSize = LenB(OpenFile)
    #Else
       OpenFile.nMaxFile = Len(OpenFile.lpstrFile) - 1
       OpenFile.lStructSize = Len(OpenFile)
    #End If
    OpenFile.lpstrFileTitle = OpenFile.lpstrFile
    OpenFile.nMaxFileTitle = OpenFile.nMaxFile
    OpenFile.lpstrInitialDir = "C:\"
    OpenFile.lpstrTitle = strTitle
    OpenFile.flags = 0
    lReturn = GetOpenFileName(OpenFile)

    If lReturn = 0 Then
       GetMyFile = ""
    Else
       GetMyFile = Trim(Left(OpenFile.lpstrFile, InStr(1, OpenFile.lpstrFile, vbNullChar) - 1))
    End If

End Function
```

GetSystemMetrics

```
Private Declare Function GetSystemMetrics Lib "USER32" (ByVal nIndex As Long) As Long

Private Declare PtrSafe Function GetSystemMetrics Lib "USER32" (ByVal nIndex As Long) As Long
```

GetTempPath

```
Declare Function GetTempPath Lib "kernel32" _
                        Alias "GetTempPathA" (ByVal nBufferLength As Long, _
                                              ByVal lpbuffer As String) As Long

Declare PtrSafe Function GetTempPath Lib "kernel32" _
                        Alias "GetTempPathA" (ByVal nBufferLength As longptr, _
                                              ByVal lpbuffer As String) As Long
```

getTickCount

```
Private Declare Function getTickCount Lib "kernel32" Alias "QueryPerformanceCounter" (cyTickCount As Currency) As Long

Private Declare PtrSafe Function getTickCount Lib "kernel32" Alias "QueryPerformanceCounter" (cyTickCount As Currency) As
Long
    '
```

getTime

```
Private Declare Function timeGetTime Lib "winmm.dll" () As Long

Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
```

GetWindow

```
Public Declare Function GetWindow Lib "USER32" _
                        (ByVal hWnd As Long, ByVal wCmd As Long) As Long
```

```
Public Declare PtrSafe Function GetWindow Lib "USER32" _
                (ByVal hWnd As LongPtr, ByVal wCmd As LongPtr) As LongPtr
```

GetWindowLong

```
This is one of the few API functions that requires the Win64 compile constant:
#If VBA7 Then
   #If Win64 Then
      Private Declare PtrSafe Function GetWindowLongPtr Lib "USER32" Alias "GetWindowLongPtrA" (ByVal hWnd As LongPtr,
ByVal nIndex As Long) As LongPtr
   #Else
      Private Declare PtrSafe Function GetWindowLongPtr Lib "USER32" Alias "GetWindowLongA" (ByVal hWnd As LongPtr,
ByVal nIndex As Long) As LongPtr
   #End If
#Else
   Private Declare Function GetWindowLongPtr Lib "USER32" Alias "GetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As
Long) As Long
#End If
```

GetWindowsDirectory

```
Declare Function GetWindowsDirectory& Lib "kernel32" Alias _
                "GetWindowsDirectoryA" (ByVal lpbuffer As String, _
                        ByVal nSize As Long)

Declare PtrSafe Function GetWindowsDirectory& Lib "kernel32" Alias _
                "GetWindowsDirectoryA" (ByVal lpbuffer As String, _
                        ByVal nSize As LongPtr)
```

GetWindowText

```
Public Declare Function GetWindowText Lib "USER32" Alias "GetWindowTextA" _
                (ByVal hWnd As Long, ByVal lpString As String, _
                 ByVal cch As Long) As Long

Public Declare PtrSafe Function GetWindowText Lib "USER32" Alias "GetWindowTextA" _
                (ByVal hWnd As LongPtr, ByVal lpString As String, _
                 ByVal cch As LongPtr) As Long
```

InternetGetConnectedState

```
Public Declare Function InternetGetConnectedState _
        Lib "wininet.dll" (lpdwFlags As Long, _
        ByVal dwReserved As Long) As Boolean

Public Declare PtrSafe Function InternetGetConnectedState _
        Lib "wininet.dll" (lpdwFlags As LongPtr, _
        ByVal dwReserved As long) As Boolean
```

IsCharAlphaNumericA

```
Private Declare Function IsCharAlphaNumericA Lib "USER32" (ByVal byChar As Byte) As Long

Private Declare PtrSafe Function IsCharAlphaNumericA Lib "USER32" (ByVal byChar As Byte) As Long
```

OleCreatePictureIndirect

```
Private Declare Function OleCreatePictureIndirect Lib "oleaut32.dll" (PicDesc As PICTDESC, RefIID As GUID, ByVal
fPictureOwnsHandle As Long, IPic As IPicture) As Long
```

```
Private Type PICTDESC
   Size As Long
   Type As Long
   hPic As Long
   hPal As Long
End Type

Private Declare PtrSafe Function OleCreatePictureIndirect Lib "oleaut32.dll" (PicDesc As PICTDESC, RefIID As GUID, ByVal
fPictureOwnsHandle As LongPtr, IPic As IPicture) As LongPtr

Private Type PICTDESC
   Size As Long
   Type As Long
   hPic As LongPtr
   hPal As LongPtr
End Type
```

OpenProcess

```
#If VBA7 Then
   Declare PtrSafe Function OpenProcess Lib "kernel32" (ByVal _
      dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal _
      dwProcessId As Long) As LongPtr
#Else
   Declare Function OpenProcess Lib "kernel32" (ByVal _
      dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal _
      dwProcessId As Long) As Long
#End If
```

ReleaseDC

```
Private Declare Function ReleaseDC Lib "USER32" (ByVal hWnd As Long, ByVal hDC As Long) As Long

Private Declare PtrSafe Function ReleaseDC Lib "USER32" (ByVal hWnd As LongPtr, ByVal hDC As LongPtr) As Long
```

SendMessage

```
Public Declare Function SendMessageA Lib "user32" (ByVal hWnd As Long, ByVal wMsg As Long, _
                                                   ByVal wParam As Long, lParam As Any) As Long
Public Declare PtrSafe Function SendMessageA Lib "user32" (ByVal hWnd As LongPtr, ByVal wMsg As Long, _
                                                           ByVal wParam As LongPtr, lParam As Any) As LongPtr
```

SetActiveWindow

```
Declare Function SetActiveWindow Lib "user32.dll" (ByVal hWnd As Long) As Long

Declare PtrSafe Function SetActiveWindow Lib "user32.dll" (ByVal hWnd As LongPtr) As LongPtr
```

SetCurrentDirectory

```
Private Declare Function SetCurrentDirectoryA Lib "kernel32" (ByVal lpPathName As String) As Long

Private Declare PtrSafe Function SetCurrentDirectoryA Lib "kernel32" (ByVal lpPathName As String) As Long
```

SetWindowLongPtr

This is one of the few API functions that requires the Win64 compile constant:

```
#If VBA7 Then
   #If Win64 Then
```

```
        Private Declare PtrSafe Function SetWindowLongPtr Lib "USER32" Alias "SetWindowLongPtrA" (ByVal hWnd As LongPtr,
ByVal nIndex As Long, ByVal dwNewLong As LongPtr) As LongPtr
    #Else
        Private Declare Function SetWindowLongPtr Lib "USER32" Alias "SetWindowLongA" (ByVal hWnd As LongPtr, ByVal
nIndex As Long, ByVal dwNewLong As LongPtr) As LongPtr
    #End If
#Else
    Private Declare Function SetWindowLongPtr Lib "USER32" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As
Long, ByVal dwNewLong As Long) As Long
#End If
```

SHBrowseForFolder

```
#If VBA7 Then
    Private Type BROWSEINFO
        hOwner As LongPtr
        pidlRoot As LongPtr
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As LongPtr
        lParam As LongPtr
        iImage As Long
    End Type

    Private Declare PtrSafe Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" _
        (lpBrowseInfo As BROWSEINFO) As LongPtr
#Else
    Private Type BROWSEINFO
        hOwner As Long
        pidlRoot As Long
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As Long
        lParam As Long
        iImage As Long
    End Type

    Private Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" _
        (lpBrowseInfo As BROWSEINFO) As Long
#End If
Private Const BIF_RETURNONLYFSDIRS = &H1
```

ShellExecute

```
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( _
        ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
        ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
```

```
Private Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( _
      ByVal hwnd As LongPtr, ByVal lpOperation As String, ByVal lpFile As String, _
      ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As LongPtr
```

SHFileOperation

```
#If VBA7 Then
   Type SHFILEOPSTRUCT
      hWnd As LongPtr
      wFunc As Long
      pFrom As String
      pTo As String
      fFlags As Integer
      fAborted As Boolean
      hNameMaps As Longptr
      sProgress As String
   End Type
   Declare PtrSafe Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
                              (lpFileOp As SHFILEOPSTRUCT) As LongPtr
#Else
   Type SHFILEOPSTRUCT
      hWnd As Long
      wFunc As Long
      pFrom As String
      pTo As String
      fFlags As Integer
      fAborted As Boolean
      hNameMaps As Long
      sProgress As String
   End Type
   Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
                              (lpFileOp As SHFILEOPSTRUCT) As Long
#End If
```

SHGetPathFromIDList

```
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" _
      (ByVal pidl As Long, ByVal pszPath As String) As Boolean

Private Declare PtrSafe Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" _
      (ByVal pidl As LongPtr, ByVal pszPath As String) As Boolean
```

SHGetSpecialFolderLocation

```
   Private Declare Function SHGetSpecialFolderLocation Lib _
      "shell32.dll" (ByVal hwndOwner As Long, ByVal nFolder As Long, _
      pidl As ITEMIDLIST) As Long

   Private Declare PtrSafe Function SHGetSpecialFolderLocation Lib _
      "shell32.dll" (ByVal hwndOwner As LongPtr, ByVal nFolder As Long, _
```

```
    pidl As ITEMIDLIST) As LongPtr

Private Type SHITEMID
   cb As Long
   abID As Byte
End Type
Private Type ITEMIDLIST
   mkid As SHITEMID
End Type
```

timeGetTime

```
Private Declare Function timeGetTime Lib "winmm.dll" () As Long

Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
```

Which Longs should become LongPtr?

It's actually pretty easy to determine what requires LongPtr and what can stay as Long. The only things that require LongPtr are function arguments or return values that represent addresses in memory. This is because a 64-bit OS has a memory space that is too large to hold in a Long data type variable. Arguments or return values that represent data will still be declared Long even in 64-bit.

The SendMessage API is a good example because it uses both types:

32-bit:

```
Public Declare Function SendMessageA Lib "user32" (ByVal hWnd As Long, ByVal wMsg As Long, _
                                    ByVal wParam As Long, lParam As Any) As Long
```

64 bit:

```
Public Declare PtrSafe Function SendMessageA Lib "user32" (ByVal hWnd As LongPtr, ByVal wMsg As Long, _
                                       ByVal wParam As Long, lParam As Any) As LongPtr
```

The first argument -hWnd- is a window handle, which is an address in memory. The return value is a pointer to a function, which is also an address in memory. Both of these must be declared LongPtr in 64-bit VBA. The arguments wMsg and wParam are used to pass data, so they can be Long in both 32-bit and 64-bit.

How to determine what is a memory address and what is data? You just have to read the MSDN documentation for the API functions (the C++ version) and it will tell you. Anything called a handle, pointer, brush or any other object type will require a LongPtr in 64-bit. Anything that is strictly data can stay as Long.

Conditional compiling

If your code needs to run on both 32 bit and 64 bit Excel, then another thing to do is add conditional compilation to your VBA.

Microsoft devised two compile constants to handle this:

VBA7: True if you're using Office 2010, False for older versions

WIN64: True if your Office installation is 64 bit, false for 32 bit.

Since the 64 bit declarations also work on 32 bit Office 2010, all you have to test for is VBA7:

```
#If VBA7 Then
   Private Declare PtrSafe Function GetDeviceCaps Lib "gdi32" (ByVal hDC As LongPtr, ByVal nIndex As Long) As Long
#Else
   Private Declare Function GetDeviceCaps Lib "gdi32" (ByVal hDC As Long, ByVal nIndex As Long) As Long
#End If
```

And then in the routine where this function is put to use:

```
#If VBA7 Then
   Dim hDC As LongPtr
#Else
   Dim hDC As Long
#End If
Dim lDotsPerInch As Long
'Get the user's DPI setting
lDotsPerInch = GetDeviceCaps(hDC, LOGPIXELSX)
```

ARBEITSMAPPEN

\\\-ASP-Pfadproblem

Auf BMD-ASP-Servern wird beim Auslesen des Speicherpfades der aktuellen Arbeitsmappe der Pfad immer DOPPELT ausgegeben. Statt \\servername\Unterordner\Datei.xlsx kommt \\servername\Unterordner\\\\servername\Unterordner\Datei.xlsx

Lösung: einfach schauen, ob im Speicherpfad \\\ vorkommt - und wenn, dann nur den Pfad ab dem abschließenden \\ vom \\\ weg verwenden. Folgender Code geht also bei beiden Varianten: sowohl bei normalen Speicherpfaden wie auch bei ASP-Speicherpfaden, weil der Code nur letztere korrigiert

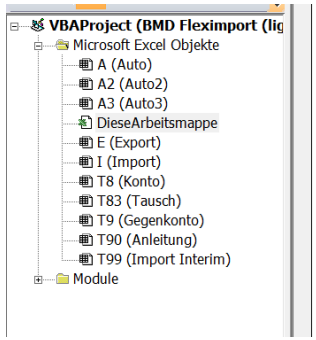
```
If Instr(strDateiname,"\\\")>0 then
        strDateiname = Mid(strDateiname, InStr(strDateiname, "\\\") + 1, 255)
End if
```

! F9-Taste Daten zwischen Mappen austauschen

VERSION 2023

Mit F9-Taste Daten aus anderer Excelmappe (mit den Quelldaten) kopieren und in die Excel-Zieldatei (Schnittstelle) einfügen über die Zwischenablage

In der Schnittstelle (Zielmappe) muss unter DIESEARBEITSMAPPE der nachfolgende Code eingefügt werden



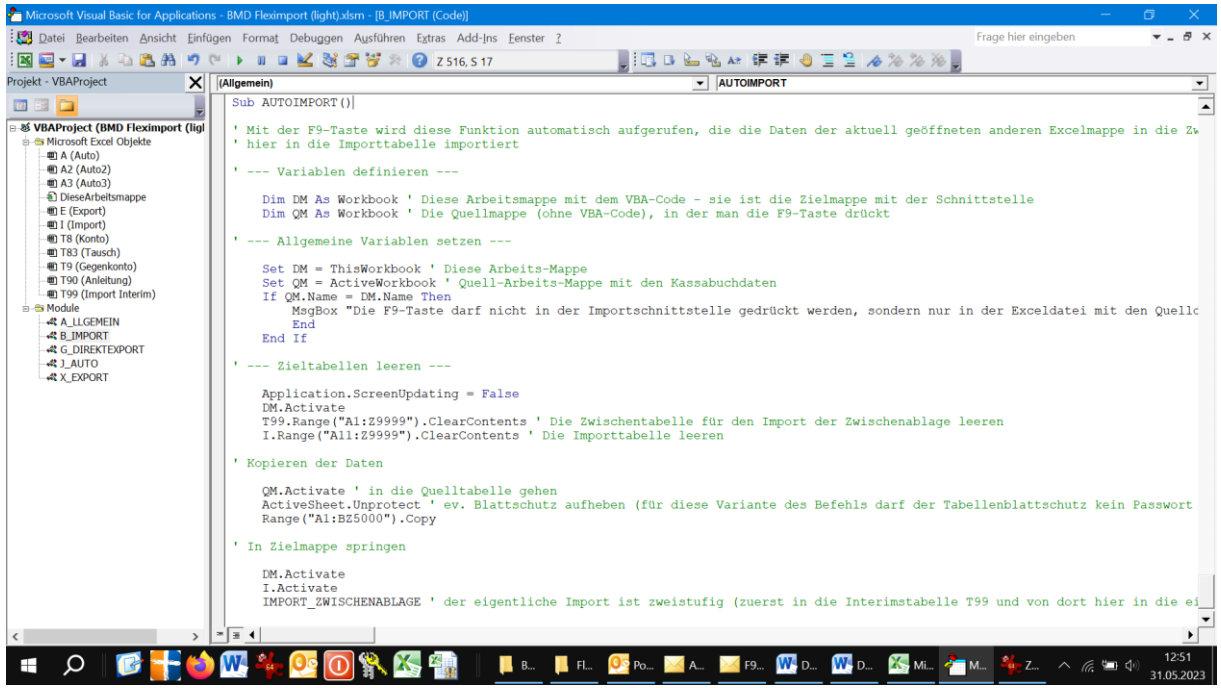

```
Private Sub Workbook_Open()


    ' F9-Taste aktivieren
    If UCase(I.Range("K5")) = "JA" Then Application.OnKey "{F9}", "AUTOIMPORT"


End Sub
```

Private Sub Workbook_Open()

    ' F9-Taste aktivieren
    If UCase(I.Range("K5")) = "JA" Then Application.OnKey "{F9}", "AUTOIMPORT"

End Sub

In ein normales Modul wird die Prozedur AUTOIMPORT eingefügt



Hier der Code

```
Sub AUTOIMPORT()

' Mit der F9-Taste wird diese Funktion automatisch aufgerufen, die die Daten der aktuell geöffneten anderen Excelmappe in die Zwischenablage kopiert und
' hier in die Importtabelle importiert

' --- Variablen definieren ---

    Dim DM As Workbook ' Diese Arbeitsmappe mit dem VBA-Code - sie ist die Zielmappe mit der Schnittstelle
    Dim QM As Workbook ' Die Quellmappe (ohne VBA-Code), in der man die F9-Taste drückt

' --- Allgemeine Variablen setzen ---

    Set DM = ThisWorkbook ' Diese Arbeits-Mappe
    Set QM = ActiveWorkbook ' Quell-Arbeits-Mappe mit den Kassabuchdaten
    If QM.Name = DM.Name Then
        MsgBox "Die F9-Taste darf nicht in der Importschnittstelle gedrückt werden, sondern nur in der Exceldatei mit den Quelldaten, die Sie hier in die Importtabelle übertragen
wollen."
        End
    End If
```

```
' --- Zieltabellen leeren ---

    Application.ScreenUpdating = False
    DM.Activate
    T99.Range("A1:Z9999").ClearContents ' Die Zwischentabelle für den Import der Zwischenablage leeren
    I.Range("A11:Z9999").ClearContents ' Die Importtabelle leeren

' Kopieren der Daten

    QM.Activate ' in die Quelltabelle gehen
    ActiveSheet.Unprotect ' ev. Blattschutz aufheben (für diese Variante des Befehls darf der Tabellenblattschutz kein Passwort haben)
    Range("A1:BZ5000").Copy

' In Zielmappe springen

    DM.Activate
    I.Activate
    IMPORT_ZWISCHENABLAGE ' der eigentliche Import ist zweistufig (zuerst in die Interimstabelle T99 und von dort hier in die eigentliche Import-Tabelle) und in eine eigene
Prozedur ausgelagert

End Sub
```

Und hier noch der Code der Prozedur IMPORT_ZWISCHENABLAGE, der auch hinter der normalen Schaltfläche IMPORT ZWISCHENABLAGE liegt und der die Daten aus der Zwischenablage einfügt in die Zieltabelle I (Import):

```
Sub IMPORT_ZWISCHENABLAGE()

' Fügt die Zwischenablage ein - über die ausgeblendete Zwischentabelle "Import Interim"

Dim ERSTE_QUELLZEILE As Integer ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
Dim ERSTE_QUELLSPALTE As Integer ' was ist die erste Spalte von der Zwischenablage
Dim ERSTE_ZIELZEILE As Integer ' was ist die erste Zeile in der empfangenden Zieltabelle, ab der die Daten eingefügt werden sollen
Dim ERSTE_ZIELSPALTE As Integer ' was ist die erste Spalte in der empfangenden Zieltabelle, ab der die Daten eingefügt werden sollen
Dim AUFTRENNUNG_TRENNZEICHEN As Integer ' 0=kein Trennzeichen, 1=Semikolon, 2=Beistrich, 3=Tabulator
Dim TRENNZEICHEN ' Parameter in der Parametertabelle für das Trennzeichen


' ************************************************************************************************************
' -----------------
' Variablen setzen
' -----------------

' Hier bitte die Einstellungen für die Prozedur machen

    ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
    ERSTE_QUELLZEILE = Range("E5")
```

```
    ' was ist die erste Spalte von der Zwischenablage, die kopiert werden soll
    ERSTE_QUELLSPALTE = Columns(Range("E6").Value).Column

    ' was ist die erste Zeile in der empfangenden Zielzeile, ab der die Daten eingefügt werden sollen
    ERSTE_ZIELZEILE = 11

    ' was ist die erste Spalte in der empfangenden Zielzeile, ab der die Daten eingefügt werden soll - in der Regel ab Spalte 1
    ERSTE_ZIELSPALTE = 1

    AUFTRENNUNG_TRENNZEICHEN = 0 ' Default: kein Trennzeichen für Import direkt aus Excel
    TRENNZEICHEN = Range("E4")
    If TRENNZEICHEN = ";" Then AUFTRENNUNG_TRENNZEICHEN = 1 ' ;
    If TRENNZEICHEN = "," Then AUFTRENNUNG_TRENNZEICHEN = 2 ' ,
    If Left(TRENNZEICHEN, 3) = "Tab" Then AUFTRENNUNG_TRENNZEICHEN = 3 ' Tabulator

' ************************************************************************************************************************

    AUS

    ' Zwischenablage_Tabelle einblenden
    T99.Visible = xlSheetVisible

    ' Eigentliche Fehlerabfangroutine, wenn Zwischenablage leer ist
    On Error GoTo FEHLER

    ' Einfügen der Daten aus der Zwischenablage in der Interimstabelle
    T99.Activate
    Range("A1").Select ' wir fügen die Daten immer ab der ersten Zeile in Spalte A ein - eine eventuell andere erste Zelle in der Zieltabelle wird unten eingestellt
    ActiveSheet.Paste

    ' Falls aus Zwischenablage keine Excelzellen-Daten kommen, sondern aus zB dem Editor die Werte einer TXT-Datei
    ' die noch aufgetrennt werden müssen - zB 0;100002377;20150629;4000 ...
    ' Hier gemäß dem Trennzeichen erfolgt die Auftrennung: 1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, (0=keine Trennung)

   ' If AUFTRENNUNG_TRENNZEICHEN > 0 Then ' falls Auftrennung gewünscht, markiere die Spalte, in der alle Daten sind
   '     Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE), Cells(ActiveSheet.UsedRange.Rows.Count, ERSTE_ZIELSPALTE)).Select
   ' End If

    Application.DisplayAlerts = False ' nachfolgender Code überschreibt Zellen und Excel würde fragen
    On Error Resume Next
    Select Case AUFTRENNUNG_TRENNZEICHEN
        Case 1 ' Semikolon
            Selection.TextToColumns DataType:=xlDelimited, Semicolon:=True
        Case 2 ' Beistrich
            Selection.TextToColumns DataType:=xlDelimited, Comma:=True
        Case 3 ' Tabulator
            Selection.TextToColumns DataType:=xlDelimited, Tab:=True
        Case 4 ' Leerzeichen
            Selection.TextToColumns DataType:=xlDelimited, Space:=True
```

```
    End Select
    Application.DisplayAlerts = True
    On Error GoTo 0

    ' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten)
    I.Activate
    I.Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE), Cells(I.UsedRange.Rows.Count, I.UsedRange.Columns.Count)).ClearContents

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    T99.Activate
    Range(Cells(ERSTE_QUELLZEILE, ERSTE_QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, LETZTEZELLE(ActiveSheet.Name).Column)).Copy

    ' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    I.Activate
    Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE).Select

    ' Wir fügen immer nur Werte ein, damit die Formatierung im Tabellenblatt Import nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False


    ' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
    T99.Activate
    T99.Range("A1:IV64000").ClearContents
    T99.Visible = xlSheetHidden

    ' Rückkehr zur aktuellen Tabelle mit den eingefügten Daten
    I.Activate

    ' Übertragen der Daten aus Zeile 11 für die Formelzeile 9

    Range("A11:GR11").Select
    Selection.Copy
    Range("A9").Select
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

    EIN
    Application.CutCopyMode = False
    Range("A11").Select

    MsgBox "Die Daten wurden eingefügt."

' ------------------------
' Ende des normalen Codes
' ------------------------

Exit Sub

' ------------------------------------------------------
' Fehlerroutine 1, wenn keine Daten in Zwischenablage
```

```
' ------------------------------------------------------

FEHLER:
    I.Activate
    EIN
    MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte die Daten erneut mit STRG-A und STRG-C kopieren." & vbCrLf & vbCrLf & _
    "Wenn die Daten aus einer anderen Exceltabelle kopiert werden und es wieder nicht klappt, schließen Sie bitte die andere Exceldatei und öffnen Sie sie erneut hier in der Importvorlage mit dem Excelmenüpunkt DATEI ÖFFNEN. Dann sind beide Excelarbeitsmappen verlässlich in derselben Excel-Instanz."
    Exit Sub

End Sub
```

ALTE VERSION

Beim Kassabuch Gattringer oder beim Transfer von Völker-Filialcontrollings in die zentrale Monatsübersicht arbeite ich gerne damit

- dass die Zielmappe beim Öffnen die F9-Taste für sich registriert und auch dass der VBA-Code für den Datenaustausch in ihr hinterlegt ist

- dann geht man in die Quellmappe und drückt dort F9 und der VBA-Code erkennt die Quellmappe als aktuelle Mappe und die Zielmappe (mit dem VBA-Code für die F9-Taste) als ThisWorkbook.

Die Quellmappe kann ganz ohne VBA-Code auskommen – was für den Emailversand und die VBA-Sicherheitseinstellungen sehr angenehm ist.

In der Zielmappe muss die F9-Taste definiert werden:

```
Private Sub Workbook_Open()


    Application.EnableAutoComplete = False ' Autovervollständigen abdrehen

    Application.OnKey "{F9}", "AUTOIMPORT"


End Sub
```

Und in einem Modul der Zielmappe wird auch die mit er F9-Taste verknüpfte Prozedur AUTOIMPORT hinterlegt sein

```
Sub AUTOIMPORT()

' Mit der F9-Taste wird diese Funktion automatisch aufgerufen, die die Daten der aktuell geöffneten anderen Excelmappe in
die Zwischenablage kopiert und
' hier in die Importtabelle importiert


' --- Variablen definieren ---


    Dim MAINTABELLE As Worksheet ' Objekt für die Main-Tabelle (Übersicht)
    Dim DM As Workbook ' Diese Arbeitsmappe mit dem VBA-Code – sie ist die Zielmappe
    Dim QM As Workbook ' Die Quellmappe (ohne VBA-Code), in der man die F9-Taste drückt

' --- Allgemeine Variablen setzen ---

    Set DM = ThisWorkbook ' Diese Arbeits-Mappe
    Set QM = ActiveWorkbook ' Quell-Arbeits-Mappe mit den Kassabuchdaten

' - Prüfen, ob wir in der richtigen Kassabuch-Datei mit einem Kassabuchtabellenblatt sind

    If Range("R2") <> "Kassastand" Then
```

```
        MsgBox "Sie dürften sich in keiner geöffneten KASSA-BUCH-Datei befinden. Bitte gehen Sie in die Kassabuchdatei
und drücken erst dann F9"
        End
    End If

' --- Zieltabellen leeren ---

    Application.ScreenUpdating = False
    DM.Activate
    T99.Range("A1:Z9999").ClearContents
    I.Range("A11:Z9999").ClearContents

' Kopieren der Kassabuchdaten

    QM.Activate
    ActiveSheet.Unprotect
    Range("B10:S500").Copy

' In Zielmappe springen

    DM.Activate
    I.Activate
    Range("A11").Select
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
    Range("A11").Select
    QM.Activate
    ActiveSheet.Protect DrawingObjects:=True, Contents:=True, Scenarios:=True
    DM.Activate
    Application.ScreenUpdating = True

    MsgBox "Die Daten wurden übertragen."


End Sub
```

! Quellmappe und Zielmappe

Nachfolgend ein Beispiel, wie eine Quelldaten-Mappe (mit VBA-Code und einer aus NTCS importierten Saldenliste) eine bereits geöffnete Zielmappe mit Reportingtabelle

automatisch befüllt.

```
Sub REPORT()

' --- Variablendefinition ---

    Dim QM As Workbook ' Aktuelle Arbeitsmappe mit Quelldaten
    Dim ZM As Workbook ' Ziel-Arbeitsmappe
    Dim MONAT As Integer ' Nummer des auszuwertenden Monats

    Dim KONTO As Single ' Kontonummer
    Dim LZ As Single ' Letztezeile der Quelldatentabelle
    Dim REPORTINGCODE ' der Dropdownlisten-Eintrag des jeweiligen Kontos in der Sachkontentabelle in den 4
Auswertungspalten D-G - zB "9 -  1. Umsatzerlöse"
    Dim ZZ As Single ' Zielzeilennummer in der Zieltabelle, in der die Daten übertragen werden sollen - aus dem
Reportingcode "9 -  1. Umsatzerlöse" wird die Auswertungszeile 9

    Dim FUNDZELLE As Range ' hier wurde es gefunden
    Dim SALDO As Single ' zu übertragender Betrag

' --- Vorbereitungen ---

    ' Prüfen ob importierte Daten den richtigen Satzaufbau haben

    If Range("A11") <> "Kontoklasse" Or Range("B11") <> "KontoNr" Or Range("D11") <> "EB" Or Left(Range("E11"), 3) <>
"Lfd" Or Left(Range("G11"), 5) <> "Monat" Or Left(Range("I11"), 5) <> "Saldo" Then
        MsgBox "Die importierten Daten scheinen nicht die richtigen Spalteninhalte zu haben. Stellen Sie bitte sicher,
dass die Spaltenüberschriften der importierten Daten (erkennbar in Zeile 11) gleich sind mit den Spaltenüberschriften der
Zeile 10."
        End
    End If

    ' Merken der aktuellen Quell-Arbeitsmappe
    Set QM = ActiveWorkbook

    ' Prüfen ob die Zielmappe schon offen ist
    If IsWorkbookOpen(Range("H3")) = False Then
          MsgBox "Die Datei   " & Range("H3") & "   scheint noch nicht offen zu sein." & vbCrLf & vbCrLf & _
          "- Bitte öffnen Sie sie " & vbCrLf & vbCrLf & _
          "- oder tragen ihren Namen richtigen Namen hier in der Zelle H3 ein."
```

```
    End If

    AUS

    ' Abfragen des aktuellen Monats, das übertragen werden soll

    Range("H2") = InputBox("Welches Monat soll übertragen werden ? (Bitte Zahl eingeben):", "Monat ?", Range("H2"))
    MONAT = Range("H2")

    ' Springen in die Zielmappe
    Workbooks(Range("H3").Value).Activate
    Set ZM = ActiveWorkbook

    ' ------------------------------------------------------------------------
    ' --- ÜBERTRAGEN DER P&L
    ' ------------------------------------------------------------------------

    Sheets("V Profit&Loss Statement LW").Activate ' Zieltabelle in Zielmappe anspringen

    ' Löschen der Zellen des aktuellen Monats, deren Zeilennummer in der Tabelle P&L hier in dieser Quellmappe aktiv
gesetzt ist

    For Z = 9 To 50
        If QM.Sheets(T71.Name).Cells(Z, 1).Value <> "" Then Cells(Z, 1 + MONAT * 2).ClearContents
    Next Z

    ' Rückkehr zur aktuellen Mappe
    QM.Activate

    LZ = LETZTEZELLE(ActiveSheet.Name).Row

    ' Schleife durch alle Zeilen der Quelldaten
    For Z = 12 To LZ
        If Cells(Z, 2) = "" Then GoTo WEITER ' In Summenzeilen (sie haben keine Kontonummer) ist nichts zu bertragen
        KONTO = Cells(Z, 2)
        If KONTO < 500000 Or KONTO > 899999 Then GoTo WEITER ' wir werten jetzt nur GuV-Konten aus - sie gehen bei dieser
Buchhaltung ausnahmsweise erst ab 500000 los (Umsatzerlöse)
        T8.Activate ' In die Tabelle mit den Sachkonten und deren Auswertungs-Zuordnung-Logik springen

        Range("B1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung - gewählte Zelle darf NICHT
verbunden sein !!!
```

```
        Set FUNDZELLE = Columns("B").Find(What:=KONTO, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole)  ' rows
würde in Zeile suchen, LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
        If FUNDZELLE Is Nothing Then
            MsgBox ("Achtung, das Konto " & KONTO & " wurde in der Sachkontentabelle noch nicht gefunden. Bitte legen Sie
es dort an und erfassen noch die Reporting-Logik" & vbCrLf & vbCrLf & "Das Programm bricht ab")
            EIN
            End
        End If
        REPORTINGCODE = Cells(FUNDZELLE.Row, 6) ' aus der Spalte F lesen wir den Reportingcode aus
        If REPORTINGCODE = "" Then
            Cells(FUNDZELLE.Row, 6).Select
            EIN
            MsgBox ("Achtung, das Konto " & KONTO & " hat hier in der Sachkontentabelle keine gültige Zuordnung - wählen
Sie diese bitte aus oder wählen den letzten Eintrag '- NICHT AUSWERTEN -' aus.")
        End If
        If InStr(REPORTINGCODE, "NICHT AUSWERTEN") > 0 Then GoTo WEITER ' wenn ein Konto als "nicht auswerten" definiert
ist, springen wir weiter
        ZZ = CDbl(Left(REPORTINGCODE, InStr(REPORTINGCODE, " ") - 1)) ' aus dem Reportingcode "9 -  1. Umsatzerlöse" wird
die Auswertungszeile 9

        I.Activate ' Rückkehr zur Quelltabelle
        SALDO = Cells(Z, 7) + Cells(Z, 8) ' Auslesen des Monats-SOLL- und des -HABEN-Betrags
        If ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).Formula = "" Then ' wenn die Zielzelle noch keinen Inhalt hat
                If SALDO < 0 Then
                        ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).FormulaR1C1 = "=" & Replace(SALDO, ",", ".") ' bei
negativen Salden muss kein Vorzeichen eingegeben werden
                    Else
                        ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).FormulaR1C1 = "=" & "+" & Replace(SALDO, ",", ".")
                End If
            ' Wenn es schon eine Formel gibt
                If SALDO < 0 Then
                        ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).Formula = ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).Formula
& Replace(SALDO, ",", ".")  ' bei negativen Salden muss kein Vorzeichen eingegeben werden
                    Else
                        ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).Formula = ZM.ActiveSheet.Cells(ZZ, 1 + MONAT * 2).Formula
& "+" & Replace(SALDO, ",", ".")
                End If
        End If
WEITER:
    Next Z
```

```
EIN
End Sub
```

! UPDATE-Import Daten von Alter Mappe in Neue Mappe (Versionsupdate)

Immer wieder kommt es vor, dass man in einer neuen Version einer Mappe die Daten aus einer anderen Mappe übernehmen möchte. Bei der Simplefibu aus 2004 habe ich das noch mit zwei bereits offenen Dateien gemacht - bei der Zeiterfassung schon nicht mehr.

Ich habe drei Versionen:

1. Völker Reporting 2024
2. Hansa Reporting 2021
3. Zeiterfassung 2019

VERSION 1 - VÖLKER Planwerte ins Controlling laden

Sie braucht meine normale Funktion LETZTEZELLE !

```
Sub UPDATEN()

' Dies ist der Code des Buttons "UPDATE"

Dim Wahl        ' was will der User
Dim VORLAGE     ' neue Vorlagendatei
Dim ALT         ' Name der Alten Datei mit den Planwerten

VORLAGE = ActiveWorkbook.Name

' Frage an User, ob er wirklich will

   Wahl = MsgBox("Sie können nun die Planwerte aus einer Exceldatei mit den Planwerten hier in diese Datei übertragen " & vbCrLf & vbCrLf & _
      "Möchten Sie fortfahren?", vbYesNoCancel, "UPDATE")

   If Wahl <> vbYes Then
      MsgBox "Das Programm bricht den Updatevorgang nun wie gewünscht ab - es wurden keine Daten verändert."
      End
   End If


' Auswählen und Öffnen der Datei mit den bisherigen Zeiterfassungsdaten

   If MsgBox("Im nächsten Schritt wählen Sie nun die Datei aus, in der die Planwerte erfasst wurden.", vbOKCancel, "Dateiwahl") <> vbOK Then
```

```
        MsgBox "Das Programm bricht den Updatevorgang nun wie gewünscht ab - es wurden keine Daten verändert."
        End
    End If

    Application.EnableEvents = False

    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            ALT = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With

    Workbooks.Open ALT ' die Variable Alt enthält noch den gesamten Dateipfad und den Dateinamen !
    ' Reduzieren der Variable Alt auf den Dateinamen (ohne Pfad)
    ALT = ActiveWorkbook.Name

    T5.Activate ' wir gehen ins Controlling

' Zurückwechseln in die neue Vorlagendatei

    Workbooks(VORLAGE).Activate

' ------------------------------------------
' Start des eigentlichen Kopiervorganges
' ------------------------------------------

    Application.ScreenUpdating = False

' Fehler abfangen
'    On Error Resume Next

' Übernahme der Planwerte

        Z = 3: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
        Z = 9: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
        Z = 20: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
```

```
    Z = 23: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 30: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 34: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 38: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 42: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 46: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 48: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 105: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 109: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 111: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 119: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 121: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 123: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 169: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 171: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 173: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 197: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 205: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 207: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 211: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 213: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 219: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 221: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
```

```
    Z = 223: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 252: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte einfügen
    Z = 258: Workbooks(ALT).Sheets(T5.INDEX).Range("B" & Z & ":M" & Z).Copy: Range("B" & Z).Select: Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte einfügen

  Range("A2").Select

  Workbooks(ALT).Close

  Application.EnableEvents = True

  MsgBox "Die Übernahme der Planwerte ist abgeschlossen." & vbCrLf & vbCrLf & _
    "Bitte vergessen Sie nicht anschließend die Datei hier noch zu speichern."


  Exit Sub


FEHLER:



End Sub
```

VERSION 2 - HANSA CONTROLLING

Nachfolgender Code stammt vom Kunden HANSA, wo die Werte einer Budget-Plantabelle aus einer alten Version in die neue übertragen werden sollen. Beide Tabellen heißen gleich und haben den gleichen Aufbau

Wir brauchen zwei allgemeine Funktionen

```
Public Function IsWorkbookOpen(strWB As String) As Boolean
   On Error Resume Next
   IsWorkbookOpen = Not Workbooks(strWB) Is Nothing
End Function

Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' Neue Version 2015 - ermittelt die letzte Zelle einer Tabelle mit Inhalt
```

```
   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

'   If CDate(Now) >= CDate("01.07.2018") Then
'        MsgBox "Die Frist der Demoversion ist leider abgelaufen - bitte setzen Sie sich bei Interesse mit uns in
Verbindung über unsere Webseite www.simplesoft.at - vielen Dank für Ihre Geduld."
'        End
'   End If

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col
```

```
   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function
```

Und hier der eigentliche Code

```
Sub IMPORTIERE_PLAN_HANSA()

Dim A_MAPPE As Workbook ' aktuelle Mappe
Dim A_DATEI ' Pfad und Dateiname der aktuellen Mappe
Dim Q_MAPPE As Workbook ' Quelldaten-Mappe
Dim Q_DATEI ' Pfad und Dateiname der ausgewählten Mappe
Dim Q_DATEINAME As String ' Nur der Dateiname

Dim LZ As Single ' Letztezeile in aktueller Tabelle
Dim KONTO ' Kontonummer
Dim Z As Single ' Schleife durch alle Zeilen
Dim ZZ As Single ' Zielzeile in der Zieltabelle



' Sicherheitsfrage

    If MsgBox("Möchten Sie die Planwerte hier in dieser Tabelle PLAN H(ANSA) aus einer anderen Datei importieren ? " &
vbCrLf & vbCrLf & _
        "Dadurch werden die aktuellen Werte in dieser Datei hier nur in dieser Tabelle PLAN H mit den Werten der anderen
Datei überschrieben." & vbCrLf & vbCrLf & _
        "Wichtig: die andere Datei darf noch nicht offen sein und wird im nächsten Schritt ausgewählt und automatisch
geöffnet", vbYesNoCancel) <> vbYes Then

        MsgBox "Programm bricht wunschgemäß ab"
        Exit Sub

    End If

' Auswahl Quellmappe
```

```
    Set A_MAPPE = ActiveWorkbook
    A_DATEI = ActiveWorkbook.FullName

    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            Q_DATEI = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With

    If Q_DATEI = A_DATEI Then
        MsgBox "Sie können nicht diese bereits offene Arbeitsmappe hier auswählen, um die Daten zu übernehmen. Es muss
sich um eine andere Arbeitsmappe handeln."
        Exit Sub
    End If

    Q_DATEINAME = Right(Q_DATEI, Len(Q_DATEI) - InStrRev(Q_DATEI, "\"))

    If IsWorkbookOpen(Q_DATEINAME) Then ' Wir zwingen den User, dass die Quelldatei zuvor geschlossen sein muss - damit
er nicht irrtümlich in der falschen Datei mit dem Import beginnt
        MsgBox "Die Datei " & Q_DATEINAME & " ist schon geöffnet." & vbCrLf & vbCrLf & "Sie muss geschlossen sein, wenn
Sie die Daten übernehmen wollen."
        Exit Sub
    End If

' Öffnen der Quellmappe

    Application.ScreenUpdating = False

    On Error GoTo FEHLER1
    Workbooks.Open Filename:=Q_DATEI

    On Error GoTo FEHLER2
    Set Q_MAPPE = ActiveWorkbook
    Q_MAPPE.Sheets(PH.Name).Activate ' In der Quellmappe in die PLAN H-Tabelle gehen - Achtung - nicht nur PH.Activate
eingeben - das würde immer die erste Mappe (A_Datei) aktivieren
```

```
    On Error GoTo 0


' Leeren der Zieltabelle

    MsgBox "Das Leeren Tabelle kann einige Minuten dauern - danke für Ihre Geduld"

    Application.EnableEvents = False


    A_MAPPE.Activate
    LZ = LETZTEZELLE(ActiveSheet.Name).Row
    For Z = 12 To LZ

        If Cells(Z, 4) <> "" And Val(Cells(Z, 4)) > 0 Then ' wenn wir verlässlich in einer Zeile mit einem Konto sind
            Range(Cells(Z, 6), Cells(Z, 17)).ClearContents
            Cells(Z, 18).FormulaR1C1 = "=SUM(RC[-12]:RC[-1])" ' Reparatur der Summenformel
        End If

    Next Z

    MsgBox "Die Daten wurden geleert - nun werden Sie aus der anderen Mappe geladen"

' Übertragen der WERTE der Quelltabelle

    Application.DisplayAlerts = False ' keine Warnung zum Überschreiben der früheren Daten

    ' Schleife durch alle Zeilen in der Quelltabell
        Q_MAPPE.Activate
        Q_MAPPE.Sheets(PH.Name).Activate ' Verlässlich erneut in die richtige Tabelle wechseln
        LZ = LETZTEZELLE(ActiveSheet.Name).Row

        For Z = 12 To LZ

            Q_MAPPE.Activate
            Q_MAPPE.Sheets(PH.Name).Activate ' Verlässlich erneut in die richtige Tabelle wechseln

            If Cells(Z, 4) <> "" And Val(Cells(Z, 4)) > 0 Then ' wenn wir verlässlich in einer Zeile mit einem Konto sind

                KONTO = Cells(Z, 4)

                ' Suchen der Kontonummer in der Zieltabelle
```

```
                A_MAPPE.Activate ' Wechseln in Zielmappe
                A_MAPPE.Sheets(PH.Name).Activate



                Range("D12").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung - gewählte Zelle
darf NICHT verbunden sein !!!
                Set FUNDZELLE = Columns("D").Find(What:=KONTO, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) '
rows würde in Zeile suchen, LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
                If FUNDZELLE Is Nothing Then
                    MsgBox "Das Konto " & KONTO & " wurde in der Zieltabelle leider nicht gefunden - das Programm fährt
dennoch fort."
                    GoTo SCHLEIFENENDE
                End If
                ZZ = FUNDZELLE.Row

                Q_MAPPE.Activate ' Rückkehr zur Quellmappe
                Q_MAPPE.Sheets(PH.Name).Activate ' Verlässlich erneut in die richtige Tabelle wechseln

                Range(Cells(Z, 6), Cells(Z, 17)).Copy ' Kopieren der ganzen Zeile
                A_MAPPE.Activate ' Wechsel in die Zieltabelle
                A_MAPPE.Sheets(PH.Name).Activate

                Cells(ZZ, 6).Select
                Selection.PasteSpecial Paste:=xlPasteValues

            End If


SCHLEIFENENDE:
        Application.ScreenUpdating = False

        Next Z

    Application.DisplayAlerts = True ' Warnung wieder aufdrehen
    Application.ScreenUpdating = True
    Application.EnableEvents = True

    MsgBox "Die Daten wurden übernommen."
    End
Exit Sub
```

```
 ' ---------------------------------------------------------------------

FEHLER1:

    MsgBox "Die andere Datei konnte nicht geöffnet werden - vermutlich ist ein anderer User in dieser Datei. Bitte
ersuchen Sie ihn aus der Datei auszusteigen."
    Application.ScreenUpdating = True
    Exit Sub

FEHLER2:

    MsgBox "Das Tabellenblatt konnte nicht gefunden sein, aus dem die Daten importiert werden sollen."
    Application.ScreenUpdating = True
    Exit Sub

End Sub
```

VERSION 3 - ZEITERFASSUNG

```
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
```

```
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function


Sub UPDATEN()

' Dies ist der Code des Buttons "UPDATE" in der Tabelle Parameter

Dim Wahl         ' was will der User
Dim VORLAGE      ' neue Vorlagendatei
Dim ALT          ' Name der Alten Zeiterfassungsdatei
Dim MONATSTABELLE As String ' die aktuelle Monattabelle
Dim SCHUTZ       ' ist die Tabelle Parameter aktuell geschützt
Dim JAHR         ' Jahr der Quelldatei
Dim ALTEVERSION ' Diese Variable merkt sich, ob es noch eine alter Vorlage ist, wo die Tabellenblätter anders heißen

If UCase(P.Range("IV10")) = "DEMO" Then
    ' Abbruch bei Demo
    MsgBox "Diese Funktion ist leider nur in der Vollversion möglich. Bitte besuchen Sie den Bereich
EXCELTOOLS/ZEITERFASSUNG auf unserer Webseite www.simplesoft.at" & vbCrLf & vbCrLf & "Danke"
    End
End If

VORLAGE = ActiveWorkbook.Name

' Frage an User, ob er wirklich will
```

```
    Wahl = MsgBox("Sie können nun erfasste Zeiten aus einer alten Version der Zeiterfassung hier in diese Datei übertragen " & _
        "mit folgenden Schritten:" & vbCrLf & vbCrLf & _
        "1.) Benennen Sie die geschlossene, bisherige Datei des Mitarbeiters um, indem Sie zB das Wort 'ALT' beim Dateinamen anhängen." & vbCrLf & _
        "(Datei umbenennen: mit rechtem Mausklick und Umbenennen)" & vbCrLf & vbCrLf & _
        "2.) Speichern Sie die neue Vorlagen-Version unter dem Namen des Mitarbeiters in den Jahresordner wo auch bisher die Datei des Mitarbeiters war." & vbCrLf & vbCrLf & _
        "3.) Klicken Sie dann in dieser neuen Datei hier auf den Button 'Update' um den Kopiervorgang zu starten." & vbCrLf & vbCrLf & _
        "Haben Sie diese 3 Schritte bereits gemacht und Sie befinden sich in der neuen Version und möchten nun die Daten der alten Zeiterfassung übernehmen ?" & vbCrLf & vbCrLf & _
        "(Im nächsten Schritt muss nun das allgemeine Tabellenblattschutz-Passwort eingegeben werden.)", vbYesNoCancel, "UPDATE")

    If Wahl <> vbYes Then
        MsgBox "Das Programm bricht den Updatevorgang nun wie gewünscht ab - es wurden keine Daten verändert."
        End

    End If

' Prüfen, ob wir in einer leeren Vorlagendatei sind

    If M01.Range("I45") <> 0 Or M2.Range("I45") <> 0 Or M3.Range("I45") <> 0 Or M4.Range("I45") <> 0 Or M5.Range("I45") <> 0 Or M6.Range("I45") <> 0 Or M7.Range("I45") <> 0 Or M8.Range("I45") <> 0 Or M9.Range("I45") <> 0 Or M10.Range("I45") <> 0 Or M11.Range("I45") <> 0 Or M12.Range("I45") <> 0 Then
        If MsgBox("Achtung: in mindestens einem Tabellenblatt in dieser Datei sind Zeiten erfasst. Sind Sie sicher, dass Sie sich in einer neuen Vorlagenversion befinden und alle Daten hier mit den Zeitbuchungen der alten Zeiterfassungsdatei des Mitarbeiters überschreiben wollen ? " & vbCrLf & vbCrLf & _
        "Der Updatevorgang wird normalerweise immer nur in einer leeren Vorlagenversion gestartet !" & vbCrLf & vbCrLf & _
        "Wollen Sie den Updatevorgang dennoch fortsetzen ?", vbYesNoCancel, "ACHTUNG") <> vbYes Then
            MsgBox "Das Programm bricht den Updatevorgang nun wie gewünscht ab - es wurden keine Daten verändert."
            End
        End If
    End If

' Auswählen und Öffnen der Datei mit den bisherigen Zeiterfassungsdaten

    If MsgBox("Im nächsten Schritt wählen Sie nun die alte Datei aus, in der die Zeiten des Mitarbeiters bisher erfasst wurden.", vbOKCancel, "Dateiwahl") <> vbOK Then
```

```
            MsgBox "Das Programm bricht den Updatevorgang nun wie gewünscht ab - es wurden keine Daten verändert."
            End
    End If

    Application.EnableEvents = False

    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            ALT = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With

    Workbooks.Open ALT ' die Variable Alt enthält noch den gesamten Dateipfad und den Dateinamen !
    ' Reduzieren der Variable Alt auf den Dateinamen (ohne Pfad)
    ALT = ActiveWorkbook.Name

' Zurückwechseln in die neue Vorlagendatei

    Workbooks(VORLAGE).Activate

' Prüfen, ob Kalenderjahre übereinstimmen

    On Error GoTo FEHLER
    If Year(M01.Range("F6")) <> Year(Workbooks(ALT).M01.Range("F6")) Then
        MsgBox "Das Jahr " & Year(M01.Range("F6")) & " in der neuen Vorlage entspricht nicht dem Jahr " & _
        Year(Workbooks(ALT).M01.Range("F6")) & " in der alten Stundenliste" & vbCrLf & vbCrLf & _
        "Bitte stellen Sie zuerst in der JAN-Tabelle der neuen Vorlage in der Zelle J6 mit dem grünen (hochschalten) bzw
dem blauen (zurückschalten) Button das korrekte Jahr wie in der alten Stundenliste ein."
        End
    End If

FORTFAHREN:

' ----------------------------------------
' Start des eigentlichen Kopiervorganges
```

```
' ----------------------------------------

    Application.ScreenUpdating = False
    VBA_SCHREIBT = 1

' Öffnen der 12 Monatstabellen-Blattschutz

    M01.Activate
    AUF
    M2.Activate
    AUF
    M3.Activate
    AUF
    M4.Activate
    AUF
    M5.Activate
    AUF
    M6.Activate
    AUF
    M7.Activate
    AUF
    M8.Activate
    AUF
    M9.Activate
    AUF
    M10.Activate
    AUF
    M11.Activate
    AUF
    M12.Activate
    AUF

' Öffnen der Parametertabelle (falls geschützt)

    P.Activate
    SCHUTZ = P.ProtectContents ' den Tabellenschutz der Parametertabelle merken
    AUF

' Fehler abfangen
    On Error Resume Next

' Übernahme des Passwortes der alten Stundenliste
```

```
    Workbooks(VORLAGE).Sheets("Parameter").Range("IV3") = Workbooks(ALT).Sheets("Parameter").Range("IV3")

' Übernahme der sichtbaren Parameter-Daten

        ' die Zeiten der 7 Tage
        Workbooks(ALT).Sheets("Parameter").Range("D6:F12").Copy '
        Range("D6").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Automatische Mittagspausenverbuchung
        Range("G4").Value = Workbooks(ALT).Sheets("Parameter").Range("G4").Value

        ' Gültigkeitsdatum Wochenmodell ab
        Range("F13").Value = Workbooks(ALT).Sheets("Parameter").Range("F13").Value

        ' Email-Adresse
        Range("C16").Value = Workbooks(ALT).Sheets("Parameter").Range("C16").Value

        ' Keine automatische Mittagspause, wenn weniger als 6 h gearbeitet
        If Workbooks(ALT).Sheets("Parameter").Range("G18") <> falsch And Workbooks(ALT).Sheets("Parameter").Range("G18")
<> True And Workbooks(ALT).Sheets("Parameter").Range("G18").Value <> "" Then
            Range("G18").Value = Workbooks(ALT).Sheets("Parameter").Range("G18").Value
        End If

        ' Zeiterfassung ohne MP-Logik
        Range("P18").Value = Workbooks(ALT).Sheets("Parameter").Range("P18").Value

        ' Automatisches Zeitformatieren
        Range("G21").Value = Workbooks(ALT).Sheets("Parameter").Range("G21").Value

        ' Kontrollsummen anzeigen
        Range("L21").Value = Workbooks(ALT).Sheets("Parameter").Range("L21").Value

        ' Wegzeit
        Range("F24").Value = Workbooks(ALT).Sheets("Parameter").Range("F24").Value

        ' Zellen mit Formeln NICHT anwählen können
        Range("L25").Value = Workbooks(ALT).Sheets("Parameter").Range("L25").Value

        ' Autom. Speichern nach AN/AB-Melden
```

```
        Range("D25").Value = Workbooks(ALT).Sheets("Parameter").Range("D25").Value

        ' Autom. Schließen nach AN/AB
        Range("F25").Value = Workbooks(ALT).Sheets("Parameter").Range("F25").Value

        ' Zeit bis autom. Schließen
        Range("K26").Value = Workbooks(ALT).Sheets("Parameter").Range("K26").Value

        ' Tage ohne Buchung gelten als voll gearbeitet
        Range("G29").Value = Workbooks(ALT).Sheets("Parameter").Range("G29").Value

        ' IST-h ROT anzeigen
        Range("K29").Value = Workbooks(ALT).Sheets("Parameter").Range("K29").Value

        ' Bei Neuanlagen nach Mittagspausenlogik fragen
        Range("L32").Value = Workbooks(ALT).Sheets("Parameter").Range("L32").Value

        ' Stundensaldovortrag
        Range("F34").Value = Workbooks(ALT).Sheets("Parameter").Range("F34").Value

        ' Bei Neuanlagen nach Stundensaldovortrag fragen
        Range("L34").Value = Workbooks(ALT).Sheets("Parameter").Range("L34").Value

        ' Frage ob Mittagspause durchgearbeitet wurde (arbeitsrechtlich dubiose Endzeitverlängerung wenn durchgearbeitet)
        Range("K35").Value = Workbooks(ALT).Sheets("Parameter").Range("K35").Value

        ' Urlaubswarnungen
        Range("L36").Value = Workbooks(ALT).Sheets("Parameter").Range("L36").Value

        ' Maximale Stunden laut KV
        Workbooks(ALT).Sheets("Parameter").Range("C39:G40").Copy '
        Range("C39").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Urlaubswerte
        Range("B36").Value = Workbooks(ALT).Sheets("Parameter").Range("B36").Value
        Range("C36").Value = Workbooks(ALT).Sheets("Parameter").Range("C36").Value
        ' Range("D36").Value = Workbooks(ALT).Sheets("Parameter").Range("D36").Value
        ' Range("E36").Value = Workbooks(ALT).Sheets("Parameter").Range("E36").Value
        ' Range("F36").Value = Workbooks(ALT).Sheets("Parameter").Range("F36").Value
        Range("G36").Value = Workbooks(ALT).Sheets("Parameter").Range("G36").Value
```

```
' MONATSABSCHLUSS

        ' Passwort
        Range("K44").Value = Workbooks(ALT).Sheets("Parameter").Range("K44").Value

        ' manuelle Zeiten übertragen
        Range("L46").Value = Workbooks(ALT).Sheets("Parameter").Range("L46").Value

        ' Stundensaldo auszahlen Frage OB
        Range("D47").Value = Workbooks(ALT).Sheets("Parameter").Range("D47").Value

        '  Stundensaldo auszahlen Frage WIEVIEL
        Range("E47").Value = Workbooks(ALT).Sheets("Parameter").Range("E47").Value

        ' Stundensaldo auszahlen-Rhythmus
        Range("L47").Value = Workbooks(ALT).Sheets("Parameter").Range("L47").Value

        ' Automatisches OK-setzen
        Range("L48").Value = Workbooks(ALT).Sheets("Parameter").Range("L48").Value

        ' Automatismus 1
        Range("E50").Value = Workbooks(ALT).Sheets("Parameter").Range("E50").Value
        Range("F50").Value = Workbooks(ALT).Sheets("Parameter").Range("F50").Value
        Range("I50").Value = Workbooks(ALT).Sheets("Parameter").Range("I50").Value
        Range("J50").Value = Workbooks(ALT).Sheets("Parameter").Range("J50").Value
        Range("K50").Value = Workbooks(ALT).Sheets("Parameter").Range("K50").Value
        ' Ab 2019 sind die Werte der Automatismen in Spalte L abweichend und können nur ab Versionen nach (20)18
übernommen werden
        If ALTEVERSION = 0 Then Range("L50").Value = Workbooks(ALT).Sheets("Parameter").Range("L50").Value

        ' Automatismus 2
        Range("E52").Value = Workbooks(ALT).Sheets("Parameter").Range("E52").Value
        Range("F52").Value = Workbooks(ALT).Sheets("Parameter").Range("F52").Value
        Range("I52").Value = Workbooks(ALT).Sheets("Parameter").Range("I52").Value
        Range("J52").Value = Workbooks(ALT).Sheets("Parameter").Range("J52").Value
        Range("K52").Value = Workbooks(ALT).Sheets("Parameter").Range("K52").Value
        If ALTEVERSION = 0 Then Range("L52").Value = Workbooks(ALT).Sheets("Parameter").Range("L52").Value ' Parameter
erst ab 2019

        ' Automatismus 3
```

```
        Range("E54").Value = Workbooks(ALT).Sheets("Parameter").Range("E54").Value
        Range("F54").Value = Workbooks(ALT).Sheets("Parameter").Range("F54").Value
        Range("I54").Value = Workbooks(ALT).Sheets("Parameter").Range("I54").Value
        Range("J54").Value = Workbooks(ALT).Sheets("Parameter").Range("J54").Value
        Range("K54").Value = Workbooks(ALT).Sheets("Parameter").Range("K54").Value
        If ALTEVERSION = 0 Then Range("L54").Value = Workbooks(ALT).Sheets("Parameter").Range("L54").Value ' Parameter
erst ab 2019

        ' Automatismus 4
        Range("F56").Value = Workbooks(ALT).Sheets("Parameter").Range("F56").Value
        Range("I56").Value = Workbooks(ALT).Sheets("Parameter").Range("I56").Value
        Range("J56").Value = Workbooks(ALT).Sheets("Parameter").Range("J56").Value
        Range("K56").Value = Workbooks(ALT).Sheets("Parameter").Range("K56").Value
        If ALTEVERSION = 0 Then Range("L56").Value = Workbooks(ALT).Sheets("Parameter").Range("L56").Value ' Parameter
erst ab 2019

        ' Automatismus 5
        Range("F58").Value = Workbooks(ALT).Sheets("Parameter").Range("F58").Value
        Range("I58").Value = Workbooks(ALT).Sheets("Parameter").Range("I58").Value
        Range("J58").Value = Workbooks(ALT).Sheets("Parameter").Range("J58").Value
        Range("K58").Value = Workbooks(ALT).Sheets("Parameter").Range("K58").Value
        If ALTEVERSION = 0 Then Range("L58").Value = Workbooks(ALT).Sheets("Parameter").Range("L58").Value ' Parameter
erst ab 2019

        ' Automatismus 6
        Range("E60").Value = Workbooks(ALT).Sheets("Parameter").Range("E60").Value
        Range("F60").Value = Workbooks(ALT).Sheets("Parameter").Range("F60").Value
        Range("I60").Value = Workbooks(ALT).Sheets("Parameter").Range("I60").Value
        Range("J60").Value = Workbooks(ALT).Sheets("Parameter").Range("J60").Value
        Range("K60").Value = Workbooks(ALT).Sheets("Parameter").Range("K60").Value
        If ALTEVERSION = 0 Then Range("L60").Value = Workbooks(ALT).Sheets("Parameter").Range("L60").Value ' Parameter
erst ab 2019

        ' Automatismus 7
        Range("F62").Value = Workbooks(ALT).Sheets("Parameter").Range("F62").Value
        Range("I62").Value = Workbooks(ALT).Sheets("Parameter").Range("I62").Value
        Range("J62").Value = Workbooks(ALT).Sheets("Parameter").Range("J62").Value
        Range("K62").Value = Workbooks(ALT).Sheets("Parameter").Range("K62").Value
        If ALTEVERSION = 0 Then Range("L62").Value = Workbooks(ALT).Sheets("Parameter").Range("L62").Value ' Parameter
erst ab 2019
        Range("J63").Value = Workbooks(ALT).Sheets("Parameter").Range("J63").Value ' der optional verwendbare
Morgenzeitpunkt
```

```
        ' Übernahme automatische SOLL=>IST-Buchungen
        Range("K81").Value = Workbooks(ALT).Sheets("Parameter").Range("K81").Value
        Workbooks(ALT).Sheets("Parameter").Range("C84:F90").Copy '
        Range("C84").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme bisherige Wochenmodelle
        Workbooks(ALT).Sheets("Parameter").Range("B301:J400").Copy '
        Range("B301").Select
        On Error Resume Next ' nachfolgender Code führte selten bei alten Versionen zu Fehler - daher kurz abfangen eines
Fehlers
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen
        On Error GoTo 0

' Übernahme der Bewertung mit Dienstplandaten
        If UCase(Workbooks(ALT).Sheets("Parameter").Range("G68")) = "AKTIV" Then
            Range("E69").Value = Workbooks(ALT).Sheets("Parameter").Range("E69").Value ' Zuschlag DP-Beginn
            Range("E74").Value = Workbooks(ALT).Sheets("Parameter").Range("E74").Value ' Zuschlag DP-Ende
            ' restliche Zuschläge
            Workbooks(ALT).Sheets("Parameter").Range("E70:F73").Copy '
            Range("E70").Select
            Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur




' Übernahme der unsichtbaren Parameter-Daten

        ' Beim Speichern keinen Blattschutz setzen
        Range("N14").Value = Workbooks(ALT).Sheets("Parameter").Range("N14").Value

        ' Demomodus (inkl nicht automatisch beenden beim Schließen)
        Range("I16").Value = Workbooks(ALT).Sheets("Parameter").Range("I16").Value ' rein kosmetische Anzeife
        Range("N18").Value = Workbooks(ALT).Sheets("Parameter").Range("N18").Value ' fix gespeicherte Interimsvariable
        Range("N15").Value = Workbooks(ALT).Sheets("Parameter").Range("N15").Value ' Demomodus beim Speichern nicht
beenden
```

```
        ' Version
        Range("A1").Value = Workbooks(ALT).Sheets("Parameter").Range("A1").Value ' welche Art von Vollversion
        Range("IV10").Value = Workbooks(ALT).Sheets("Parameter").Range("IV10").Value ' welche Art von Vollversion:
STANDART oder PREMIUM

 ' Übernahme der Fortbildung

        Fortbildung.Activate
        Workbooks(ALT).Sheets("Fortbildung").Range("C5:D16").Copy '
        Range("C5").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen
        Range("M4").Value = Workbooks(ALT).Sheets("Fortbildung").Range("M4").Value

' Übernahme der Jännererfassung

        MONATSTABELLE = "1 JAN"
        Sheets(MONATSTABELLE).Activate

        ' Übernahme richtiges Jahr
        If ALTEVERSION = 0 Then ' wenn Quellmappe bereits eine neue Version ist
            JAHR = Year(Workbooks(ALT).M01.Range("F6"))
          Else
            JAHR = Year(Workbooks(ALT).Sheets("1 JAN").Range("F6"))
        End If
        Range("F6") = CDate("01.01." & JAHR)

        ' Übernahme Mitarbeiter-Name + Nummer
        Range("E8").Value = Workbooks(ALT).Sheets(MONATSTABELLE).Range("E8").Value
        Range("E9").Value = Workbooks(ALT).Sheets(MONATSTABELLE).Range("E9").Value

        ' Übernahme Vorjahressaldo
        Range("R9").Value = Workbooks(ALT).Sheets(MONATSTABELLE).Range("R9").Value

        ' Übernahme Zeiten in Spalten E-H
        Workbooks(ALT).Sheets(MONATSTABELLE).Range("E14:H44").Copy '
        Range("E14").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme Überstunden in Spalten K-Q
```

```
        Workbooks(ALT).Sheets(MONATSTABELLE).Range("K14:Q44").Copy '
        Range("K14").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme Überstunden in Spalte AR-AY
        Workbooks(ALT).Sheets(MONATSTABELLE).Range("AR14:AY44").Copy '
        Range("AR14").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme Urlaub und Feiertage in Spalten T-U
        Workbooks(ALT).Sheets(MONATSTABELLE).Range("T14:U44").Copy '
        Range("T14").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme manuelle Zeiten in Spalten W-AA
        Workbooks(ALT).Sheets(MONATSTABELLE).Range("W14:AA44").Copy '
        Range("W14").Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False ' nur Werte
einfügen

        ' Übernahme ausbezahlte Stunden
        Range("R48").Value = Workbooks(ALT).Sheets(MONATSTABELLE).Range("R48").Value
        Range("E14").Select

        ' Übernahme Formeln der SOLL-Stunden
            For Z = 14 To 44
                Cells(Z, 10).FormulaR1C1 = "=IF(RC[11]="""",IF(LEN(RC[-7])>0," &
Replace(CStr(Workbooks(ALT).Sheets(MONATSTABELLE).Cells(Z, 10).Value), ",", ".") & ",0),0)"
            Next Z


        ZU

' ----------- Abschluss des UPDATES ---------------


' Setzen des Tabellenblattschutzes der Tabelle Parameter
    P.Activate
```

```
    ' zum leeren von großer Menge von Daten in Zwischenablage
    Range("D6").Copy
    Application.CutCopyMode = False
    Range("C5").Select

    If SCHUTZ = True Then
        ZU
    End If

' Schließen der alten Datei
    Application.DisplayAlerts = False ' keine Frage zu großer Menge von Daten in Zwischenablage
    Workbooks(ALT).Close savechanges:=False
    Application.DisplayAlerts = True

    Application.ScreenUpdating = True
    VBA_SCHREIBT = 0

    ' Fehler wieder aktivieren
    On Error GoTo 0

    Application.EnableEvents = True

' Aktivieren der Überstunden-Automatismen durch zweimal hin und herschalten
    EIN
    P.Range("N19") = 1 ' Interim die Variable für das Updaten setzen, damit keine Hinweise beim Einstellen der
Automatismenzellen in Spalte L kommen
    Parameter_Zeile50
    Parameter_Zeile52
    Parameter_Zeile54
    Parameter_Zeile56
    Parameter_Zeile58
    Parameter_Zeile60
    Parameter_Zeile62
    P.Range("N19") = ""

    MsgBox "Die Übernahme der erfassten Zeiten aus den 12 Monatetabellen ist abgeschlossen." & vbCrLf & vbCrLf & _
        "Auch die erfassten Werte in der Tabelle 'Parameter' sind übertragen." & vbCrLf & vbCrLf & _
        "Bitte vergessen Sie nicht anschließend die Datei noch zu speichern."

    Exit Sub
```

```
FEHLER:
' Code, wenn die Tabelle M01 in der alten Mappe nicht gefunden wurde und es noch eine Version vor 1.1.2019 war

    ALTEVERSION = 1

    If Year(M01.Range("F6")) <> Year(Workbooks(ALT).Sheets("1 JAN").Range("F6")) Then
        MsgBox "Das Jahr " & Year(M01.Range("F6")) & " in der neuen Vorlage entspricht nicht dem Jahr " &
Year(Workbooks(ALT).Sheets("Parameter").Range("F6")) & " in der alten Stundenliste" & vbCrLf & vbCrLf & _
        "Bitte stellen Sie zuerst in der JAN-Tabelle der neuen Vorlage in der Zelle J6 mit dem grünen (hochschalten) bzw
dem blauen (zurückschalten) Button das korrekte Jahr wie in der alten Stundenliste ein."
        End
    End If

GoTo FORTFAHREN

End Sub
```

! Jahreswechsel und Sicherungskopie der aktuellen Mappe

```
Sub JAHRESWECHSEL

   If MsgBox("Beim Jahreswechsel werden alle erfassten Eingangsrechnungen in den 12 Monats-Tabellen entfernt und die Formeln zurückgesetzt." & vbCrLf & vbCrLf & _
      "Sie sollten den Jahreswechsel erst dann machen, wenn Sie die Datei unter einem neuen Namen für das neue Jahr gespeichert haben und Sie nicht mehr in der Datei mit den
Rechnungen des alten Jahres sind." & vbCrLf & vbCrLf & _
      "Möchten Sie fortfahren und diese Datei hier leeren ?", vbYesNoCancel, "Jahreswechsel") <> vbYes Then

      MsgBox "Das Programm bricht wunschgemäß ab"
      End

   End If

   ' Auslesen Dateinamen
```

```
    PFADAKTUELL = ThisWorkbook.Path ' Pfad ohne Dateiname
    DATEIAKTUELL = ThisWorkbook.Name ' Nur der Dateiname inkl Dateiendung
    ALTJAHR = T01.Range("P2")
    If Dir(PFADAKTUELL & "\Backup", vbDirectory) = "" Then
       MkDir (PFADAKTUELL & "\Backup")
    End If

    Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
    Application.EnableEvents = False ' keine Events automatisch ausführen lassen

    ActiveWorkbook.SaveCopyAs PFADAKTUELL & "\Backup\" & ALTJAHR & " " & DATEIAKTUELL

    Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
    Application.EnableEvents = True ' Events wieder automatisch ausführen lassen

    End

' ----------------------------------------------------
' --- Teil 2: Leeren der Vorlage und Speichern    ---
' ----------------------------------------------------

    Application.EnableEvents = False
    Application.ScreenUpdating = False

' --- der eigentliche Jahreswechsel erfolgt ab hier ---
```

ALLGEMEINE GRUNDLAGEN ZU ARBEITSMAPPEN

BSP 1

```
MsgBox ThisWorkbook.Name ' Die Mappe, in der der Code läuft
MsgBox ActiveWorkbook.Name ' die aktuelle Mappe
Workbooks("BOOK4.XLS").Activate ' die Mappe mit dem Namen Book4.xls zur aktiven Mappe machen

' Zellen aus einer Tabelle in einer Arbeitsmappe auslesen
Workbooks("MeineDaten.xls").Sheets("Tabelle1").Range("A1:C1").Copy
```

```
Workbooks("DeineDaten.xls").Sheets(1).Cells(3, 5).PasteSpecial xlpastevalues

' Wenn man die Tabelle mit ihrem VBA-Namen (hier: nur P) ansprechen möchte
Workbooks("DeineDaten.xls").Sheets(P.Name).Cells(3, 5).Copy
```

BSP 2

```
' --- Variablen definieren ---

    Dim DM As Workbook ' Objekt D-ieser-M-appe in der dieser Code hier ist
    Dim SM As Workbook ' Objekt der aktuell geöffneten S-chleusen-M-appe
    Dim DMN As String ' N-ame D-ieser-M-appe in der dieser Code hier ist
    Dim SMN As String ' N-ame der aktuell geöffneten S-chleusen-M-appe, die ausgelesen werden soll

' --- Allgemeine Variablen setzen ---

    Set DM = ThisWorkbook
    Set SM = ActiveWorkbook
    DMN = ThisWorkbook.Name
    SMN = ActiveWorkbook.Name

    MsgBox DM.Name
    MsgBox SM.Name
    DM.Activate
    MsgBox ActiveWorkbook.Name
    SM.Activate
    MsgBox ActiveWorkbook.Name
```

BSP 3 - Aktuelle Mappe merken, neue erzeugen+speichern, Rückkehr zur aktuellen Mappe

```
Dim DieseMappe As String ' the name of the active workbook with this code
DieseMappe = ActiveWorkbook.Name
' Neue Arbeitsmappe öffnen
Workbooks.Add
' Ihren Anzeigename einstellen, wenn man das möchte - Alternativ: NeueMappe = ActiveWorkbook.Name und dann die neue Mappe
aufrufen mit Workbooks(NeueMappe)
ActiveWindow.Caption = "test.xls"
```

```
' Falls man den Zielnamen gleich anzeigen und verwenden möchte, dann muss man die Mappe interim speichern, damit man sie
ansprechen kann mit Workbooks("test.xls")
ActiveWorkbook.SaveAs "C:\test.xls"
' Rückkehr zu dieser Mappe hier mit dem Code
Workbooks(DieseMappe).Activate
```

Alles zu Pfad, Dateiname, Dateiendung

Folgender Code wandelt einen Gesamtpfad um in die einzelnen Details

```
Dim PFADGESAMT As String            ' "C:\Unterordner\Testdatei.xlsm"
Dim PFADOHNEDATEI As String         ' "C:\Unterordner\
Dim DATEINAMEMITENDUNG As String    ' Testdatei.xlsm
Dim DATEINAMEOHNEENDUNG As String   ' Testdatei
Dim DATEIENDUNG As String           ' xlsm

' Automatisches Auslesen
PFADGESAMT= ActiveWorkbook.FullName
' Manueller Pfad zum Testen
PFADGESAMT = "C:\Unterordner\Unterordner2\Testdatei.xls"

PFADOHNEDATEI = Left(PFADGESAMT, InStrRev(PFADGESAMT, "\"))
DATEINAMEMITENDUNG = Right(PFADGESAMT, Len(PFADGESAMT) - Len(PFADOHNEDATEI))
DATEINAMEOHNEENDUNG = Left(DATEINAMEMITENDUNG, InStrRev(DATEINAMEMITENDUNG, ".") - 1) ' -1, weil wir den Punkt nicht wollen
DATEIENDUNG = Right(DATEINAMEMITENDUNG, Len(DATEINAMEMITENDUNG) - Len(DATEINAMEOHNEENDUNG) - 1) ' -1, weil wir den Punkt nicht wollen

MsgBox "PFADGESAMT: " & PFADGESAMT & vbCrLf & _
       "PFADOHNEDATEI: " & PFADOHNEDATEI & vbCrLf & _
       "DATEINAMEMITENDUNG: " & DATEINAMEMITENDUNG & vbCrLf & _
       "DATEINAMEOHNEENDUNG: " & DATEINAMEOHNEENDUNG & vbCrLf & _
       "DATEIENDUNG: " & DATEIENDUNG
```

Aktuelle Arbeitsmappe mit und ohne Dateiendung

siehe auch ARBEITSMAPPEN / ALLES ZU PFAD, DATEINAME, DATEIENDUNG

MsgBox "The name of the active workbook is " & **ActiveWorkbook**.Name

MsgBox "The name of the active workbook is " & Left(ThisWorkbook.Name, Len(ThisWorkbook.Name) - 5) ' bzw 4 bei .xls bis 2003

Aktuelle Arbeitsmappe – Namen Speichern für Zurückkehren

Aktuelle Arbeitmappe speichern, um zB nach neuem Erzeugen einer anderen Excelmappe in die Originalmappe zurückzuspringen

```
Dim ThisWkb As Workbook ' the active workbook with this code

' saving the name of the active workbook
Set ThisWkb = ActiveWorkbook

' creating a new workbook
Workbooks.Add
ActiveWindow.Caption = "test.xls"

' Interim-Save-Location (otherwise Workbooks("test.xls").-Command would not work
ActiveWorkbook.SaveAs "C:\test.xls"

' Returning to this workbook
ThisWkb.Activate
```

Alternativ geht auch

```
Dim ThisWkb As String ' the name of the active workbook with this code

' saving the name of the active workbook
ThisWkb = ActiveWorkbook.Name

' creating a new workbook
Workbooks.Add
ActiveWindow.Caption = "test.xls"

' Interim-Save-Location (otherwise Workbooks("test.xls").-Command would not work
ActiveWorkbook.SaveAs "C:\test.xls"

' Returning to this workbook
Workbooks(ThisWkb).Activate
```

Aktuelle Mappe Temp-Kopie mit Passwort vermailen, umbenennen, löschen

```
' Variablen für Datei
    Public KOPIE As Workbook
    Public DATEINAME As String ' nur der Dateiname - zB Test.xlsx
    Public DATEIORIG As String ' der Dateiname im Tempordner inkl. Pfad - zB C:\Users\Me\AppData\Local\Temp\Test.xlsx
    Public DATEIKOPIE As String ' gleich wie DATEIORIG aber mit dem Interims-Unterstrich - zB C:\Users\Me\AppData\Local\Temp\Test_.xlsx
    Public EMAIL As String ' Die Emailadresse des Empfängers
    Public PASSWORT As String ' Das Passwort für die Exceldatei
    Public PAUSENMODUS ' 1=Pausenmodus =Bearbeitenmodus ist an - 0 oder leer: Standardmodus für Emailversand


' Allgemeine Funktion zum Finden der letzten Zelle einer Tabelle
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row
```

```
    ' letzte Spalte mit Daten herausfinden
    LastColWithData = ExcelLastCell.Column
    Col = ExcelLastCell.Column
    Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
       Col = Col - 1
    Loop
    LastColWithData = Col

    Set LETZTEZELLE = TheSheet.Cells(Row, Col)
    Application.ScreenUpdating = MERKER

End Function

Sub Mail_I()

' Dies ist der Code, wenn man auf den MAILEN-Button klickt, um die aktuelle Mappe per Mail zu versenden
' Der Code macht den ersten Teil des Emailversandes: er speichert die aktuelle Datei als temporäre Kopie in den TEMP-Ordner
' Dann übergibt er an das Fenster mit den Kunden und den Emailadressen und dem Passwort der Klienten

' Fehlerüberprüfung (wenn man zB auf das Mail-Icon klickt, aber gar keine Arbeitsmappe offen ist)
    On Error GoTo FEHLER

' Festlegen des reinen Dateinamens
    DATEINAME = ActiveWorkbook.Name

' Festlegen der originalen Datei im Temp-Ordner
    DATEIORIG = Environ("TEMP") & "\" & ActiveWorkbook.Name

' Prüfen, ob Datei schon gespeichert worden ist als Exceldatei
    If InStr(DATEIORIG, ".xl") = 0 Then DATEIORIG = DATEIORIG & ".xlsx" ' falls nicht, mach sie zu einer Standard-Exceldatei

' Festlegen der temporären Datei (sie muss einen Unterstrich haben, da man nicht zwei Dateien mit gleichem Namen gleichzeitig öffnen kann)
    DATEIKOPIE = Replace(DATEIORIG, ".xl", "_.xl")

' Löschen eventueller alten Dateien im TEMP-Ordner von früheren Emailversandvorgängen
    If Len(Dir(DATEIORIG)) > 0 Then Kill DATEIORIG
    If Len(Dir(DATEIKOPIE)) > 0 Then Kill DATEIKOPIE

' Speichern der Datei als temporäre Kopie
    ActiveWorkbook.SaveCopyAs Filename:=DATEIKOPIE

' DSGVO-Mailer-Fenster einblenden
    Windows("Excel-DSGVO-Mailer.xlsm").Visible = True
    Workbooks("Excel-DSGVO-Mailer.xlsm").Activate
```

```
    Exit Sub

FEHLER:
    MsgBox "Der Befehl kann nicht ausgeführt werden - vermutlich gibt es keine Arbeitsmappe"

End Sub

Sub MAIL_II()

' Dies ist der zweite Teil zum Vermailen
' er wird automatisch dann ausgeführt, wenn man in der Liste der Kunden eine Zeile mit einem Kunden angeklickt hat
' Achtung: Der Code muss unterscheiden, ob die Mandantenauswahl in Excel für eine Exceldatei erfolgt, oder ob der Aufruf auf Word erfolgte

' Variablen für WORD-Datei
    Dim WORDDATEIKOPIE As String ' Pfad und Name der Word-Dateikopie
    Dim WORDDATEIORIG As String 'Pfad und Name der originalen Word-Datei
    Dim WORDDATEINAME As String ' Der reine Dateiname für das Emailbetreff
    Dim MYWORD As Object ' Word-Application

' Variablen für Email
    Dim App, Itm

' Auslesen der Daten des gewählten Klienten

    EMAIL = Cells(ActiveCell.Row, 6)
    PASSWORT = Cells(ActiveCell.Row, 7)

' Prüfen, ob der Aufruf von Word aus erfolgte => dann zum ganz unten befindlichen Code für Word verzweigen
    If GetSetting(appname:="DSGVO", section:="Excel", Key:="Word") = 1 Then GoTo WORDAUFRUF

' Bildschirmaktualisierung aufheben
    Application.ScreenUpdating = False

' Fehler abfangen
    On Error GoTo FEHLER

' Öffnen der temporären Kopie
    Set KOPIE = Workbooks.Open(DATEIKOPIE, False, False) ' Verknüpfungen nicht aktualisieren (false) und Datei nicht nur lesend öffnen (false)

' Speichern der Kopie mit Passwort
    Application.DisplayAlerts = False ' Nicht warnen beim Überschreiben
    KOPIE.SaveAs Filename:=DATEIKOPIE, Password:=PASSWORT, WriteResPassword:="", ReadOnlyRecommended:=False, CreateBackup:=False
    KOPIE.Close
```

```
    Application.DisplayAlerts = True ' Wieder normal warnen beim Überschreiben

' Umbenennen der Kopie auf Namen ohne Unterstrich
    Name DATEIKOPIE As DATEIORIG

' E-Mailprogramm ansteuern
    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        With Itm
            .Subject = DATEINAME
            .to = EMAIL
            '.Cc =
            .Attachments.Add (DATEIORIG)
            .display
        End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunden warten
    Itm.display

    Set App = Nothing
    Set Itm = Nothing

' Ausblenden des DSGVO-Mailers

    Windows("Excel-DSGVO-Mailer.xlsm").Visible = False
    Application.ScreenUpdating = True

    Exit Sub

FEHLER:
    Application.ScreenUpdating = True
    Exit Sub

KEINOUTLOOK:
    MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
    Application.ScreenUpdating = True
    Exit Sub


' -------------------
' --- WORD-AUFRUF ---
```

```
' -------------------

' Dies ist der Code, der abgearbeitet wird, wenn die Excelpasswortliste von Word aufgerufen wurde

WORDAUFRUF:

' Zurücksetzen des Aufrufparameters in der Registrierung
    SaveSetting "DSGVO", "Excel", "Word", "0"

' Auslesen der Dateipfade
    WORDDATEIKOPIE = GetSetting(appname:="DSGVO", section:="Excel", Key:="Worddatei")
    WORDDATEIORIG = Replace(WORDDATEIKOPIE, "_.docx", ".docx")
    WORDDATEINAME = Mid(WORDDATEIORIG, InStrRev(WORDDATEIORIG, "\") + 1, 99)

' Öffnen der Kopie der Worddatei
    Set MYWORD = CreateObject("Word.Application") ' starte MS Word
    With MYWORD
      .Visible = True
      .Documents.Open Filename:=WORDDATEIKOPIE
      .Activedocument.SaveAs2 Filename:=WORDDATEIORIG, Password:=PASSWORT
      .Activedocument.Close
      .Application.Quit
    End With

' Umbenennen der Worddateikopie in den originalen Dateinamen
'    Name WORDDATEIKOPIE As WORDDATEIORIG

' E-Mailprogramm ansteuern
    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        With Itm
            .Subject = WORDDATEINAME
            .to = EMAIL
            '.Cc =
            .Attachments.Add (WORDDATEIORIG)
            .display
        End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunde warten
    Itm.display
```

```
    Set App = Nothing
    Set Itm = Nothing


' Schließen des DSGVO-Mailers
    Application.WindowState = -4137
    Application.Quit

End Sub
```

Alle Arbeitsmappen in einem Ordner zusammenfassen in neue gemeinsame Arbeitsmappe – und retour

Der nachfolgende Code läuft nur unter Excel 2003 und danach nicht mehr. Ev. anpassen für Excel ab 2007.

VERSION 1

```
Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer

On Error GoTo ErrExit

With Application
  .ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
  .Cursor = xlWait
End With

strPath = "F:\Temp" 'Pfad - Anpassen!

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
```

```
  .NewSearch
  .LookIn = strPath
  .FileType = msoFileTypeExcelWorkbooks
  .SearchSubFolders = False

  If .Execute > 0 Then

    For intIndex = 1 To .FoundFiles.Count

      Set objWb = Workbooks.Open(.FoundFiles(intIndex))

      objWb.Sheets(1).Copy after:=objNew.Sheets(objNew.Sheets.Count)

      objNew.Sheets(objNew.Sheets.Count).Name = objFO.GetBasename(.FoundFiles(intIndex))

      objWb.Close False

      Set objWb = Nothing

    Next

  End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs strPath & "Zusammenfassung.xls"

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

VERSION 2

```
' ********************************************************************
```

```
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

'BrowseForFolder mit Extra-Funktionen
'VB -Versionen: VB5 , VB6
'Betriebssystem: Win9x , WinNT, Win2000, WinME, WinXP
'Autor: Marco Wünschmann Homepage: ohne
'Datum: 23.08.2004

Option Explicit

' == Dialog-Einstellungen ================================

' String, der vor dem aktuell ausgewählen Verzeichnis angezeigt wird,
' falls der ShowCurrentPath-Paramter True ist.
Private Const DIALOG_CURRENT_SELECTION_TEXT As String = "Auswahl: "


' == API-Deklarationen ===================================

Private Type BROWSEINFO
  hOwner As Long
  pidlRoot As Long
  pszDisplayName As String
  lpszTitle As String
  ulFlags As Long
  lpfnCallback As Long
  lParam As Long
  iImage As Long
End Type

Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type

Private Type Size
  cx As Long
  cy As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
  Alias "SHBrowseForFolderA" ( _
  lpBrowseInfo As BROWSEINFO) As Long
```

```
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
  Alias "SHGetPathFromIDListA" ( _
  ByVal lPIDL As Long, _
  ByVal pszPath As String) As Long

Private Declare Sub CoTaskMemFree Lib "ole32.dll" ( _
  ByVal pv As Long)

Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" ( _
  ByVal hwnd As Long, _
  ByVal wMsg As Long, _
  ByVal wParam As Long, _
  lParam As Any) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
  Alias "RtlMoveMemory" ( _
  pDest As Any, _
  pSource As Any, _
  ByVal dwLength As Long)

Private Declare Function ILCreateFromPath Lib "shell32" _
  Alias "#157" ( _
  ByVal sPath As String) As Long

Private Declare Function LocalAlloc Lib "kernel32" ( _
  ByVal uFlags As Long, _
  ByVal uBytes As Long) As Long

Private Declare Function LocalFree Lib "kernel32" ( _
  ByVal hmem As Long) As Long

Private Declare Function lstrcpyA Lib "kernel32" ( _
  lpString1 As Any, _
  lpString2 As Any) As Long

Private Declare Function lstrlenA Lib "kernel32" ( _
  lpString As Any) As Long

Private Declare Function FindWindowEx Lib "user32.dll" _
  Alias "FindWindowExA" ( _
  ByVal hWnd1 As Long, _
  ByVal hWnd2 As Long, _
  ByVal lpsz1 As String, _
  ByVal lpsz2 As String) As Long

Private Declare Function GetWindowDC Lib "user32.dll" ( _
```

```
  ByVal hwnd As Long) As Long

Private Declare Function GetWindowRect Lib "user32.dll" ( _
  ByVal hwnd As Long, _
  ByRef lpRect As RECT) As Long

Private Declare Function GetTextExtentPoint Lib "gdi32.dll" _
  Alias "GetTextExtentPointA" ( _
  ByVal hDC As Long, _
  ByVal lpszString As String, _
  ByVal cbString As Long, _
  ByRef lpSize As Size) As Long

Private Declare Function PathCompactPath Lib "shlwapi.dll" _
  Alias "PathCompactPathA" ( _
  ByVal hDC As Long, _
  ByVal pszPath As String, _
  ByVal dx As Long) As Long

Private Const MAX_PATH = 260

Private Const WM_USER = &H400

Private Const BFFM_INITIALIZED = 1
Private Const BFFM_SELCHANGED As Long = 2
Private Const BFFM_SETSTATUSTEXTA As Long = (WM_USER + 100)
Private Const BFFM_SETSTATUSTEXTW As Long = (WM_USER + 104)
Private Const BFFM_ENABLEOK As Long = (WM_USER + 101)
Private Const BFFM_SETSELECTIONA As Long = (WM_USER + 102)
Private Const BFFM_SETSELECTIONW As Long = (WM_USER + 103)

Private Const BIF_NEWDIALOGSTYLE As Long = &H40
Private Const BIF_RETURNONLYFSDIRS As Long = &H1
Private Const BIF_BROWSEINCLUDEFILES As Long = &H4000
Private Const BIF_STATUSTEXT As Long = &H4

Private Const LMEM_FIXED = &H0
Private Const LMEM_ZEROINIT = &H40
Private Const LPTR = (LMEM_FIXED Or LMEM_ZEROINIT)

' Zeigt den BrowseForFolder-Dialog an.
Public Function BrowseForFolder(DialogText As String, _
  DefaultPath As String, _
  OwnerhWnd As Long, _
  Optional ShowCurrentPath As Boolean = True, _
  Optional RootPath As Variant, _
  Optional NewDialogStyle As Boolean = False, _
```

```
  Optional IncludeFiles As Boolean = False) As String

' Parameter:
' o DialogText Dialogtext, der oben im Dialog angezeigt wird.
' o DefaultPath Standardmäßig ausgewähltes Verzeichnis.
' o OwnerhWnd hWnd des übergeordneten Fensters (in den meisten
' Fällen Me.hWnd).
' o ShowCurrentPath Legt fest, ob die aktuelle Verzeichnisauswahl
' angezeigt werden soll. Verfügbar ab
' Internet Explorer 4.0 (-> PathCompactPath).
' o RootPath Root-Verzeichnis. Wird es angegeben, werden nur die
' Ordner unterhalb dieses Verzeichnisses angezeigt.
' o NewDialogStyle Legt fest, ob der Dialog in der neuen Darstellung
' angezeigt werden soll (Dialog kann vergrößert/
' verkleinert werden, es ist eine Schaltfläche zum
' Anlegen eines neuen Ordners vorhanden, es können
' Dateioperationen wie löschen etc. ausgeführt
' werden, ...). Ist dieser Parameter True, hat der
' Parameter ShowCurrentPath keine Wirkung. Verfügbar
' unter WinME und Betriebsystemen ab Win2000.
' o IncludeFiles Legt fest, ob auch Dateien im Dialog angezeigt und
' ausgewählt werden können.
' Verfügbar ab Win98 und Internet Explorer 4.0 (bei
' frühreren Windowsversionen muss IE4 inkl. der
' Integrated Shell installiert sein).

Dim biBrowseInfo As BROWSEINFO
Dim lPIDL As Long
Dim sBuffer As String
Dim lBufferPointer As Long

With biBrowseInfo
  ' Handle des übergeordneten Fensters
  .hOwner = OwnerhWnd

  ' PIDL des Rootordners zuweisen
  If Not IsMissing(RootPath) Then .pidlRoot = PathToPIDL(RootPath)

  ' Dialogtext zuweisen
  If ShowCurrentPath And DialogText = "$" Then DialogText = "" ' Wird intern nicht zugelassen
  .lpszTitle = DialogText

  ' Stringbuffer für aktuell selektierten Pfad zuweisen
  If ShowCurrentPath Then .pszDisplayName = sBuffer

  ' Dialogeinstellungen zuweisen
  .ulFlags = BIF_RETURNONLYFSDIRS + _
```

```
    IIf(ShowCurrentPath, BIF_STATUSTEXT, 0) + _
    IIf(NewDialogStyle, BIF_NEWDIALOGSTYLE, 0) + _
    IIf(IncludeFiles, BIF_BROWSEINCLUDEFILES, 0)

  ' Callbackfunktion-Adresse zuweisen
  .lpfnCallback = FARPROC(AddressOf CallbackString)

  ' PIDL des vorselektierten Ordnerpfades zuweisen (wird im
  ' lpData-Parameter an die Callback-Funktion weitergeleitet)
  .lParam = PathToPIDL(DefaultPath)
End With

' BrowseForFolder-Dialog anzeigen
lPIDL = SHBrowseForFolder(biBrowseInfo)

If lPIDL Then
  ' Stringspeicher reservieren
  sBuffer = Space$(MAX_PATH)

  ' Selektierten Pfad aus der zurückgegebenen PIDL ermitteln
  SHGetPathFromIDList lPIDL, sBuffer

  ' Nullterminierungszeichen des Strings entfernen
  sBuffer = Left$(sBuffer, InStr(sBuffer, vbNullChar) - 1)

  ' Selektierten Pfad zurückgeben
  BrowseForFolder = sBuffer

  ' Reservierten Task-Speicher wieder freigeben
  Call CoTaskMemFree(lPIDL)
End If

' Stringspeicher wieder freigeben
If ShowCurrentPath Then Call LocalFree(lBufferPointer)
End Function


Private Function CallbackString(ByVal hwnd As Long, ByVal uMsg As Long, _
  ByVal lParam As Long, ByVal lpData As Long) As Long

' Callback-Funktion des BrowseForFolder-Dialogs. Wird bei
' eintretenden Ereignissen des Dialogs aufgerufen.

Dim sBuffer As String
Dim lStaticWnd As Long
Dim lStaticDC As Long
Dim sPath As String
```

```
Dim rctStatic As RECT
Dim szTextSize As Size

' Meldungen herausfiltern
Select Case uMsg
  Case BFFM_INITIALIZED
    ' Dialog wurde initialisiert

    ' Standardmäßig markierten Pfad (dessen PIDL wurde in lpData
    ' übergeben) im Dialog selektieren
    Call SendMessage(hwnd, BFFM_SETSELECTIONA, False, ByVal lpData)
  Case BFFM_SELCHANGED
    ' Selektion hat sich geändert

    ' Stringspeicher reservieren
    sBuffer = Space$(MAX_PATH)

    ' Aktuell selektierten Pfad ermitteln und anzeigen, wenn möglich
    If SHGetPathFromIDList(lParam, sBuffer) Then
      ' Temporäre Zeichenfolge an das Anzeigelabel senden, um
      ' dessen Handle anhand dieser Zeichenfolge ermitteln zu können
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal "$"

      ' Handle und DeviceContext des Anzeigelabels ermitteln
      lStaticWnd = FindWindowEx(hwnd, ByVal 0&, ByVal "Static", ByVal "$")
      lStaticDC = GetWindowDC(lStaticWnd)

      ' Abmessungen des Anzeigelabels ermitteln
      GetWindowRect lStaticWnd, rctStatic

      ' Textabmessungen der Zeichenfolge "Auswahl: " im Anzeigelabel
      ' ermitteln
      GetTextExtentPoint lStaticDC, ByVal DIALOG_CURRENT_SELECTION_TEXT, _
        ByVal Len(DIALOG_CURRENT_SELECTION_TEXT), szTextSize

      ' Anzuzeigenden Pfad auf die Abmessungen des Anzeigelabels
      ' kürzen; falls dies nicht möglich ist, gesamten Pfad anzeigen
      sPath = sBuffer
      If PathCompactPath(ByVal lStaticDC, sPath, ByVal (rctStatic.Right - _
        rctStatic.Left - szTextSize.cx + 80)) = 0 Then sPath = sBuffer

      ' Nullterminierung entfernen
      sPath = Left$(sPath, InStr(1, sPath, vbNullChar) - 1)

      ' Pfad im Dialog anzeigen
      Call SendMessage(hwnd, BFFM_SETSTATUSTEXTA, 0&, _
        ByVal DIALOG_CURRENT_SELECTION_TEXT & sPath)
```

```
    Else
      ' Pfadanzeige leeren
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal ""
    End If
End Select
End Function


Private Function FARPROC(FunctionPointer As Long) As Long
' Funktion wird benötigt, um Funktions-Adresse ermitteln
' zu können, dessen Adresse mit AddressOf übergeben und
' anschließend wieder zurückgegeben wird.

FARPROC = FunctionPointer
End Function


' Gibt die lPIDL zum übergebenen Pfad zurück.
Private Function PathToPIDL(ByVal sPath As String) As Long
Dim lRet As Long

lRet = ILCreateFromPath(sPath)
If lRet = 0 Then
  sPath = StrConv(sPath, VbStrConv.vbUnicode)
  lRet = ILCreateFromPath(sPath)
End If

PathToPIDL = lRet
End Function



Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer


strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

If strPath = "" Then Exit Sub

On Error GoTo ErrExit

With Application
```

```
  .ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
  .Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
  .NewSearch
  .LookIn = strPath
  .FileType = msoFileTypeExcelWorkbooks
  .SearchSubFolders = False

  If .Execute > 0 Then

    For intIndex = 1 To .FoundFiles.Count

      Set objWb = Workbooks.Open(.FoundFiles(intIndex))

      objWb.Sheets(1).Copy after:=objNew.Sheets(objNew.Sheets.Count)

      objNew.Sheets(objNew.Sheets.Count).Name = objFO.GetBasename(.FoundFiles(intIndex))

      objWb.Close False

      Set objWb = Nothing

    Next

  End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs strPath & "Zusammenfassung.xls"

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
```

```
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

VERSION 2-B – Anpassung von mir, damit mehr als ein Tabellenblatt pro Arbeitsmappe übernommen wird

```
Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer
Dim Tabellenname As String


'strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

strPath = "C:\test\"

If strPath = "" Then Exit Sub

' On Error GoTo ErrExit

With Application
  '.ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
 ' .Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
```

```
  .NewSearch
  .LookIn = strPath
  .FileType = msoFileTypeExcelWorkbooks
  .SearchSubFolders = False

  If .Execute > 0 Then

    For intIndex = 1 To .FoundFiles.Count

      Set objWb = Workbooks.Open(.FoundFiles(intIndex))

      For Z = 1 To objWb.Sheets.Count

            objWb.Sheets(Z).Copy after:=objNew.Sheets(objNew.Sheets.Count)
            Tabellenname = objFO.GetBasename(.FoundFiles(intIndex)) & " - " & objWb.Sheets(Z).Name
            Tabellenname = Left(Tabellenname, 31)
           ' MsgBox Tabellenname
            objNew.Sheets(objNew.Sheets.Count).Name = Tabellenname


      Next Z

      objWb.Close False

      Set objWb = Nothing

    Next

  End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs "C:\" & "Zusammenfassung.xls"

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing
```

```
End Sub
```

VERSION 3 macht das Gegenteil: alle Tabellenblätter als Arbeitsmappen wieder speichern

! Es muss zum obigen Code Version 2 dazukopiert werden, weil es ihn zum Teil benötigt

```
Public Sub TakeSheetsAndSaveAsFile()
Dim objWb As Workbook
Dim objSh As Worksheet
Dim strPath As String

strPath = BrowseForFolder("Zielverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

If strPath = "" Then Exit Sub

On Error GoTo ErrExit

With Application
  .ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
  .Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objWb = ActiveWorkbook

For Each objSh In objWb.Worksheets
  objSh.Copy
  ActiveWorkbook.SaveAs strPath & ActiveWorkbook.Sheets(1).Name & ".xls"
  ActiveWorkbook.Close True
Next

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
```

```
End With

Set objWb = Nothing


End Sub
```

Arbeitsmappe aktivieren

In diesem Beispiel wird die Arbeitsmappe MAPPE4.XLS aktiviert. Falls MAPPE4.XLS aus mehreren Fenstern besteht, wird das erste Fenster (MAPPE4.XLS:1) aktiviert.

Workbooks("BOOK4.XLS").Activate

Arbeitsmappe schließen (ohne Speichern-Abfrage)

```
' Aktive Arbeitsmappe
ActiveWorkbook.Close savechanges:=False
' irgendeine Arbeitsmappe
Workbooks("Test.xls").Close savechanges:=False
```

Automatischer Code beim Öffnen der Arbeitsmappe

```
Dass man Code im Bereich THIS WORKBOOK als Private Sub Workbook_Open() … End Sub ausführen lassen kann, ist eigentlich
allen bekannt.

Ähnlich wie in Word kann man aber auch direkt in einem Modul das Makro auto_open einfügen (hier mit Beispielcode, der auf
Vollbild umschaltet) und es wird bei jedem Öffnen ausgeführt.

Sub auto_open()
      Application.DisplayFullScreen = True
End Sub
```

--- ARBEITSMAPPEN + TABELLENBLATT-SCHUTZ ---

```
Hier sind mehrere Prozeduren
- um gesamte Arbeitsmappe zu schützen
- um alle tabellenblätter zu schützen
- das aktuelle tabellenblatt zuz schützen
```

- ein zu benennendes Tabellenblatt zu schützen

AKTUELLES TABELLENBLATT ÖFFNEN / BLATTSCHUTZ SETZEN

```
' ---------------------------------
' --- AKTUELLE TABELLE SCHÜTZEN ---
' ---------------------------------

Public Sub AUF()
' Diese Hilfsprozedur dient für andere Prozeduren, um den Tabellenblattschutz für die aktuelle Tabelle zu öffnen

    ' Schutz aufheben
    ActiveSheet.Unprotect Password:="" ' oder :=Tabelle24.Range("J12")

End Sub

Public Sub ZU()

' Diese Hilfsprozedur dient für andere Prozeduren, um den Tabellenblattschutz für die aktuelle Tabelle zu setzen

    ActiveSheet.Protect Password:="", DrawingObjects:=True, Contents:=True, Scenarios:=True, AllowFormattingCells:=True,
AllowSorting:=True, AllowFiltering:=True

End Sub
```

ARBEITSMAPPE AUF UND ZUSPERREN (2 SUB)

```
Public Sub AUF_AM()

' Diese Hilfsprozedur öffnet die gesamt Arbeitsmappe

    On Error Resume Next

    ActiveWorkbook.Unprotect Password:=""

End Sub
Public Sub ZU_AM()

' Diese Hilfsprozedur aktiviert den Arbeitsmappenschutz

    On Error Resume Next
```

```
    ActiveWorkbook.Protect Password:="", Structure:=True, Windows:=False  ' Structure true: Reihenfolge der Blätter nicht
veränderbar - Windows/Fenster false: Fenstergröße veränderbar

End Sub
```

ARBEITSMAPPE AUF UND ZUSPERREN (WECHSELSCHALTER)

```
Public Sub Arbeitsmappen_Un_Lock()

' Diese Prozedur wird über das Kontrollkästchen Arbeitsmappe der Tabelle develop in Zelle C10 aktiviert
' je nach Status des Kästchens wird die gesamte Arbeitsmappe gesperrt oder geöffnet

On Error Resume Next

If Sheets("Develop").Range("C10") = True Then ' wenn Arbeitsmappenschutz aufgehoben sein soll
    ActiveWorkbook.Unprotect Password:=Sheets("Develop").Range("J10")
    MsgBox "Die Arbeitsmappe ist nun ungeschützt und voll bearbeitbar"
   Else
    ActiveWorkbook.Protect Password:=Sheets("Develop").Range("J10"), Structure:=True , Windows:=False ' default-passwort
lockall
    MsgBox "Die Arbeitsmappe ist nun wieder geschützt"
End If
End Sub
```

TABELLENBLÄTTER VERSTECKEN / EINBLENDEN

```
Public Sub HIDE_DEVELOP()

' Dies ist die Prozedur des HIDE-Buttons im Tabellenblatt develop

On Error Resume Next

' Öffnen des Arbeitsmappenschutzes
ActiveWorkbook.Unprotect Password:=Sheets("Develop").Range("J10")

' Ausblenden der Develop-Tabellen
Sheets("Develop").Visible = xlVeryHidden
Sheets("Sprachen").Visible = xlVeryHidden
Sheets("Shapes").Visible = xlVeryHidden
Sheets("Orgashapes").Visible = xlVeryHidden
Sheets("Orga_Namen").Visible = xlVeryHidden

' Setzen des Arbeitsmappenschutzes, falls in den Einstellungen gesetzt
If Sheets("Develop").Range("C10") = False Then ' wenn Arbeitsmappenschutz aufgehoben sein soll
```

```
    ActiveWorkbook.Protect Password:=Sheets("Develop").Range("J10"), Structure:=True , Windows:=False ' default-passwort
lockall
End If

End Sub

Public Sub SHOW_DEVELOP()
' Dies ist die Prozedur um das Tabellenblatt develop einzublenden mittels STRG-SHIFT-ALT-F1

On Error Resume Next

' Öffnen des Arbeitsmappenschutzes
ActiveWorkbook.Unprotect Password:=Sheets("Develop").Range("J10")

If UCase(InputBox("Um die Entwicklungsumgebung zu öffnen, geben Sie bitte das Entwicklerpasswort ein: ")) =
UCase(Sheets("Develop").Range("J4")) Then

    Sheets("Sprachen").Visible = xlSheetVisible
    Sheets("Shapes").Visible = xlSheetVisible
    Sheets("Orgashapes").Visible = xlSheetVisible
    Sheets("Orga_Namen").Visible = xlSheetVisible
    Sheets("Develop").Visible = xlSheetVisible
    Sheets("Develop").Activate

  Else

    MsgBox "Leider nicht das richtige Passwort"

End If

' Setzen des Arbeitsmappenschutzes, falls in den Einstellungen gesetzt
If Sheets("Develop").Range("C10") = False Then ' wenn Arbeitsmappenschutz aufgehoben sein soll
    ActiveWorkbook.Protect Password:=Sheets("Develop").Range("J10"), Structure:=True , Windows:=False ' default-passwort
lockall
End If

End Sub
```

ALLE TABELLENBLÄTTER ÖFFNEN / SCHLIESSEN (VARIANTE 0 - BEST)

```
' ------------------------------
' --- ALLE TABELLEN SCHÜTZEN ---
' ------------------------------

Sub ALLES_AUF()
```

```
' --- Hilfsprozedur, die in allen Tabellen den Tabellenblattschutz entfernt ---

    ' Schleife durch alle Tabellenblätter

    For n = 1 To Worksheets.Count
        Worksheets(n).Unprotect Password:=""    ' oder Passwort:=Tabelle7.Range("AJ7") :In dieser Zelle ist das Passwort
für den Tabellenblattschutz
    Next

End Sub

Sub ALLES_ZU()

' --- Hilfsprozedur, die in allen Tabellen den Tabellenblattschutz setzt ---

    ' Schleife durch alle Tabellenblätter

    For n = 1 To Worksheets.Count
        Worksheets(n).Protect Password:="", DrawingObjects:=True, Contents:=True, Scenarios:=True,
AllowFormattingCells:=True, AllowSorting:=True, AllowFiltering:=True    ' oder Password:=Tabelle7.Range("AJ7") In dieser
Zelle ist das Passwort für den Tabellenblattschutz
                Worksheets(n).EnableSelection = xlUnlockedCells ' nur ungeschützte Zellen dürfen angewählt werden
    Next

End Sub
```

ALLE TABELLENBLÄTTER ÖFFNEN / SCHLIESSEN (VARIANTE 1)

```
Public Sub AlleTabellen_Un_Lock()

' Diese Prozedur wird über das Kontrollkästchen AlleTabellen der Tabelle develop in Zelle C12 aktiviert
' je nach Status des Kästchens werden alle User-Tabellenblätter in ihrem Tabellenblattschutz geöffnet oder gesperrt

On Error Resume Next
Application.ScreenUpdating = False


If Sheets("Develop").Range("C12") = True Then ' wenn Tabellenblattschutz aufgehoben sein soll
    Call T_Unlock(Tabelle1)
    Call T_Unlock(Tabelle2)
    Call T_Unlock(Tabelle4)
    Call T_Unlock(Tabelle5)
    Call T_Unlock(Tabelle7)
    Call T_Unlock(Tabelle8)
    Call T_Unlock(Tabelle9)
    Call T_Unlock(Tabelle10)
    Call T_Unlock(Tabelle11)
```

```
 Call T_Unlock(Tabelle12)
 Call T_Unlock(Tabelle13)
 Call T_Unlock(Tabelle14)
 Call T_Unlock(Tabelle15)
 Call T_Unlock(Tabelle16)
 Call T_Unlock(Tabelle17)
 Call T_Unlock(Tabelle18)
 Call T_Unlock(Tabelle19)
 Call T_Unlock(Tabelle20)
 Call T_Unlock(Tabelle21)
 Call T_Unlock(Tabelle22)
 Call T_Unlock(Tabelle23)
 Call T_Unlock(Tabelle26)
 Call T_Unlock(Tabelle27)
 Call T_Unlock(Tabelle28)
 Call T_Unlock(Tabelle29)
 Call T_Unlock(Tabelle30)
 Application.ScreenUpdating = True
 MsgBox "Alle User-Tabellen sind nun ungeschützt und voll bearbeitbar"
Else
 Call T_Lock(Tabelle1)
 Call T_Lock(Tabelle2)
 Call T_Lock(Tabelle4)
 Call T_Lock(Tabelle5)
 Call T_Lock(Tabelle7)
 Call T_Lock(Tabelle8)
 Call T_Lock(Tabelle9)
 Call T_Lock(Tabelle10)
 Call T_Lock(Tabelle11)
 Call T_Lock(Tabelle12)
 Call T_Lock(Tabelle13)
 Call T_Lock(Tabelle14)
 Call T_Lock(Tabelle15)
 Call T_Lock(Tabelle16)
 Call T_Lock(Tabelle17)
 Call T_Lock(Tabelle18)
 Call T_Lock(Tabelle19)
 Call T_Lock(Tabelle20)
 Call T_Lock(Tabelle21)
 Call T_Lock(Tabelle22)
 Call T_Lock(Tabelle23)
 Call T_Lock(Tabelle26)
 Call T_Lock(Tabelle27)
 Call T_Lock(Tabelle28)
 Call T_Lock(Tabelle29)
 Call T_Lock(Tabelle30)
 Application.ScreenUpdating = True
 MsgBox "Alle User-Tabellen sind nun wieder geschützt"
```

```
End If

End Sub
Public Sub T_Lock(Tabelle As Object)
    Tabelle.Protect Password:=Sheets("Develop").Range("J12"), DrawingObjects:=True, Contents:=True, Scenarios:=True
End Sub
Public Sub T_Unlock(Tabelle As Object)
    Tabelle.Unprotect Password:=Sheets("Develop").Range("J12")
End Sub
```

ALLE TABELLENBLÄTTER ÖFFNEN / SCHLIESSEN VARIANTE 2

```
Public Sub AUF_ALL()
' Gleich wie Hilfsprozedur AUF - nur werden jetzt alle User-Tabellen aufgemacht

AUF_T Tabelle1.Name
AUF_T Tabelle2.Name
AUF_T Tabelle4.Name
AUF_T Tabelle5.Name
AUF_T Tabelle7.Name
AUF_T Tabelle8.Name
AUF_T Tabelle9.Name
AUF_T Tabelle10.Name
AUF_T Tabelle11.Name
AUF_T Tabelle12.Name
AUF_T Tabelle13.Name
AUF_T Tabelle14.Name
AUF_T Tabelle15.Name
AUF_T Tabelle16.Name
AUF_T Tabelle17.Name
AUF_T Tabelle18.Name
AUF_T Tabelle19.Name
AUF_T Tabelle20.Name
AUF_T Tabelle21.Name
AUF_T Tabelle22.Name
AUF_T Tabelle23.Name
AUF_T Tabelle26.Name
AUF_T Tabelle27.Name
AUF_T Tabelle28.Name
AUF_T Tabelle29.Name
AUF_T Tabelle30.Name

End Sub

Public Sub ZU_ALL()
' Gleich wie Hilfsprozedur ZU - nur werden jetzt alle User-Tabellen zugemacht
```

```
ZU_T Tabelle1.Name
ZU_T Tabelle2.Name
ZU_T Tabelle4.Name
ZU_T Tabelle5.Name
ZU_T Tabelle7.Name
ZU_T Tabelle8.Name
ZU_T Tabelle9.Name
ZU_T Tabelle10.Name
ZU_T Tabelle11.Name
ZU_T Tabelle12.Name
ZU_T Tabelle13.Name
ZU_T Tabelle14.Name
ZU_T Tabelle15.Name
ZU_T Tabelle16.Name
ZU_T Tabelle17.Name
ZU_T Tabelle18.Name
ZU_T Tabelle19.Name
ZU_T Tabelle20.Name
ZU_T Tabelle21.Name
ZU_T Tabelle22.Name
ZU_T Tabelle23.Name
ZU_T Tabelle26.Name
ZU_T Tabelle27.Name
ZU_T Tabelle28.Name
ZU_T Tabelle29.Name
ZU_T Tabelle30.Name

End Sub

Public Sub AUF_T(Tabellenname As String)

' Gleiche Hilfsprozedur wie "AUF", nur wird hier auch ein Tabellenname übergeben

Dim Status
TN = Tabellenname

' Schutzstatus merken
Status = Sheets(TN).ProtectContents

' Schutz aufheben
Worksheets(TN).Unprotect Password:=Tabelle24.Range("J12")

' Schutzstatus speichern in Zelle IZ1
Worksheets(TN).Range("IZ1") = Status



End Sub
```

```
Public Sub ZU_T(Tabellenname As String)

' Gleiche Hilfsprozedur wie "ZU", nur wird hier auch ein Tabellenname übergeben

TN = Tabellenname

' Wenn der Status vor dem AUF-Machen gesperrt war
If Worksheets(TN).Range("IZ1") = True Then

    Worksheets(TN).Protect Password:=Tabelle24.Range("J12"), DrawingObjects:=True, Contents:=True, Scenarios:=True,
AllowFormattingCells:=False, AllowFormattingColumns:=True, AllowFormattingRows:=False

End If

End Sub
```

BACKUP der aktuellen Arbeitsmappe speichern

```
Sub BACKUP()

   Dim DATEIPFAD As String ' Gesamtpfad zur Mappe inkl Dateiname und -endung
   Dim BACKUP As String ' Der Backupdateipfad und Dateiname inkl Endung

   DATEIPFAD = ActiveWorkbook.FullName

' Anlegen einer Sicherungskopie der Datei
   Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
   Application.EnableEvents = False ' keine Events automatisch ausführen lassen
   BACKUP = Left(DATEIPFAD, InStrRev(DATEIPFAD, ".") - 1) & " Backup" & "." & Mid(DATEIPFAD, InStrRev(DATEIPFAD, ".") + 1, 4)
   ActiveWorkbook.SaveCopyAs BACKUP
   Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
   Application.EnableEvents = True ' Events wieder automatisch ausführen lassen

End Sub
```

BESTIMMTES TABELLENBLATT ÖFFNEN / BLATTSCHUTZ SETZEN

```
Public Sub AUF_T(Tabellenname As String)

' Gleiche Hilfsprozedur wie "AUF", nur wird hier auch ein Tabellenname übergeben

Dim Status
```

```
TN = Tabellenname

' Schutzstatus merken
Status = Sheets(TN).ProtectContents

' Schutz aufheben
Worksheets(TN).Unprotect Password:=Tabelle24.Range("J12")

' Schutzstatus speichern in Zelle IZ1
Worksheets(TN).Range("IZ1") = Status



End Sub

Public Sub ZU_T(Tabellenname As String)

' Gleiche Hilfsprozedur wie "ZU", nur wird hier auch ein Tabellenname übergeben

TN = Tabellenname

' Wenn der Status vor dem AUF-Machen gesperrt war
If Worksheets(TN).Range("IZ1") = True Then

    Worksheets(TN).Protect Password:=Tabelle24.Range("J12"), DrawingObjects:=True, Contents:=True, Scenarios:=True,
AllowFormattingCells:=False, AllowFormattingColumns:=True, AllowFormattingRows:=False

End If

End Sub
```

AUF-ZU-Wechselschalter zum Blattschutz

ei Bilder einfügen - z.B.

Sie - wenn man sie in mehreren Seiten braucht, kopieren und drüben einfügen. Achtung: Original_eingefügte_Bilder werden mit "Picture X" angesprochen im VBA-Code - manuell copy und paste von einer Tabelle in die andere ergibt meist Namen wie "Bild X".... - einfach im tabellenblatt das Bild markieren und schauen, wie es oben benannt ist. Man kann eingefügte Bilder übrigens auch umbenennen: einfach markieren und dann oben links in der Zelle mit dem Bildnamen einen neuen drüberschreiben - dürfte aber nur bei copy and paste - Bildern gehen - also nur bei "Bild X" und nicht bei "Picture X" - zur Not einfach Bild X-Bilder kopieren und dort einfügen, wo man sie braucht - ev. "Picture X" dann löschen.

1.) Hier das Modul "DieseArbeitsmappe" (Was tun beim Öffnen, beim Ändern und beim Speichern)

```
Private Sub Workbook_Open()

' ********************************
' ----------------
' PASSWORT-SETZEN
' ----------------

  Sheets("Parameter").Range("IV3") = 787

' ********************************


With Application
     .Calculation = xlAutomatic
     .MaxChange = 0.001
End With
```

```
ActiveWorkbook.PrecisionAsDisplayed = False

    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlToRight
    End With

Application.OnKey "^{F1}", "Alles_Auf" ' strg-F1 mit Alles Aufmachen verknüpfen

End Sub


Private Sub Workbook_BeforeClose(Cancel As Boolean)
    With Application
        .MoveAfterReturn = True
        .MoveAfterReturnDirection = xlDown
    End With
End Sub

Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)

Dim A As String ' Target.Adress

On Error Resume Next ' (wenn man mehrere Zellen markiert und löscht, kommt es zu Fehler)

If Sheets("Parameter").Range("E21") = False Then Exit Sub
If VBA_SCHREIBT = 1 Then Exit Sub ' wenn VBA-Code der An- und Abmeldetasten Zeiten schreibt, müssen diese nicht formatiert werden

If Val(Left(ActiveSheet.Name, 1)) > 0 Then ' wenn aktuelles Tabellenblatt mit einer Zahl beginnt - daher eines der 12 Monatstabellen ist

    If Target.Column = "2" And UCase(Target.Value) = "OK" Then

        Range("W14:AA" & Target.Row).Locked = True

    End If

    ' Einschränkung auf bestimmte Spalten
    A = Mid(Target.Address, 2, 1) ' die Spalte
    If A = "E" Or A = "F" Or A = "G" Or A = "H" Or A = "K" Or A = "L" Then  ' wenn ein Wert in den Spalten E-H oder K-L geändert wurde

        ' Einschränkung auf bestimmte Zellen
        A = Right(Target.Address, 2) ' Zeilennummern
        If Val(A) >= 14 And Val(A) <= 44 Then ' wenn ein Wert zwischen den Zeilen 13 und 35 geändert wurde
              If Selection.Columns.Count > 1 Or Selection.Rows.Count > 1 Then End
```

```
            If Target.Value = 1 And Len(Target.Value) > 0 Then Target.Value = "24:00": End ' die Eingabe von 24 führt ebenso zu einer 1, wie die Eingabe von
1 - daher kann 1 Uhr nachts nie eingegeben werden
            If Target.Value = 0.5 And Len(Target.Value) > 0 Then Target.Value = "12:00": End ' die Eingabe von 24 führt ebenso zu einer 1, wie die Eingabe
von 1 - daher kann 1 Uhr nachts nie eingegeben werden
         If Val(Target.Value) > 1 Then ' wenn eine Zeit ohne Doppelpunkt eingegeben wird (Doppelpunktzeiten werden ja in Teilen eines Tages gespeichert und
24:00 wäre 1)
            If Len(Target.Value) = 4 Then
               ' wird mit "ÜBERNIMM TAGESKALKULATION"-Taste Uhrzeiten der Felder X7-X10 übernommen, werden 4-stellige Zeiten wie 9:30 falsch
interpretiert
               If Left(Target.Value, 1) = "1" Or Left(Target.Value, 1) = "2" Then
                  Target.Value = Left(Target.Value, 2) & ":" & Right(Target.Value, 2)
               End If
            End If
            If Len(Target.Value) = 3 Then
               Target.Value = Left(Target.Value, 1) & ":" & Right(Target.Value, 2)
            End If
            If Len(Target.Value) = 2 Then
               Target.Value = Left(Target.Value, 2) & ":" & "00"
            End If
            If Len(Target.Value) = 1 Then
               Target.Value = Left(Target.Value, 1) & ":" & "00"
            End If
         End If
      End If
   End If
End If

End Sub
```

2.) Hier das Modul zum Auf / Zu-Machen

```
Sub OEFFNEN()
' dies ist der Code für das grüne Schloss-ZU-Icon, zum Aufmachen durch den User

   PW = Val(InputBox("Bitte Passwort zum Öffnen eingeben", "Passwort"))

   ' Das Passwort zum Vergleichen ist gespeichert in Tabelle PARAMETER Zelle IV3, welche als Zellformat STANDARD hat und mittles VBA-Prozedur bei jedem
Öffnen der Mappe mit dem Wert 787 beschrieben wird:
   ' Hier der Code im Objekt "DieseArbeitsmappe" für dieses Defaultsetzen bei jedem Öffnen der Mappe:
   '       Private Sub Workbook_Open()
   '       Sheets("Parameter").Range("IV3") = 787

   If PW = Val(Sheets("Parameter").Range("IV3")) Then
```

```
            ActiveSheet.Unprotect (Val(PW))
            ' das ZU-Icon in den Hintergrund verschieben
            For Each shShape In ActiveSheet.Shapes
                ' Wir suchen nur nach der Bildnummer 27, damit sowohl Picture 27 als auch Bild 27 gefunden wird - je nach Excelsprachversion
                If InStr(shShape.Name, "27") > 1 Then
                    shShape.Select
                    Selection.ShapeRange.ZOrder msoSendToBack
                End If
            Next shShape
            ActiveSheet.Range("C6").Select
        Else
            MsgBox "Das Passwort war leider nicht korrekt - bitte versuchen Sie es erneut"
            End
    End If

End Sub

Sub ZUMACHEN()
' dies ist der Code für das rote Schloss-AUF-Icon, zum Zumachen durch den User

    ' wenn das Tabellenblatt nicht geschützt ist
    If ActiveSheet.ProtectContents = False Then
        For Each shShape In ActiveSheet.Shapes
            If InStr(shShape.Name, "27") > 1 Then
                shShape.Select
                Selection.ShapeRange.ZOrder msoBringToFront
            End If
        Next shShape
    End If
    ActiveSheet.Range("C6").Select
    ActiveSheet.Protect Password:=Val(Sheets("Parameter").Range("IV3")), DrawingObjects:=True, Contents:=True, Scenarios:=True

End Sub
Public Sub AUF()
' dies ist der Code für das Öffnen des aktuellen Tabellenblatt-Schreibschutzes nur für den VBA-Code selbst, wenn wir An- und Abmelde-Zeiten schreiben wollen

    PW = Sheets("Parameter").Range("IV3")
    ActiveSheet.Unprotect (Val(PW))

End Sub
Public Sub ZU()
' dies ist der Code für das Schließen des aktuellen Tabellenblatt-Schutzes nur für den VBA-Code selbst, wenn wir An- und Abmelde-Zeiten schreiben wollen

    ActiveSheet.Protect Password:=Val(Sheets("Parameter").Range("IV3")), DrawingObjects:=True, Contents:=True, Scenarios:=True
```

```
End Sub
Sub Alles_Auf()
' dies ist der Code, wenn man alle Tabellenblätter mit Schreibschutz aufmachen möchte
' die Prozedure wird bei jedem Öffnen des Workbooks mit ONKEY auf die Tastenkombination STRG-F1 gelegt
' der normale User braucht diese Funktion nicht, sondern nur, wenn man über alle Tabelle als Administrator was ändern möchte

If UCase(InputBox("masterpasswort bf")) <> "JESUS" Then
       MsgBox "Das Passwort war leider nicht korrekt - bitte versuchen Sie es erneut"
       Exit Sub
End If
For i = 1 To Worksheets.Count
   Worksheets(i).Activate
   PW = Val(Sheets("Parameter").Range("IV3"))
          For Each shShape In ActiveSheet.Shapes
             ' wenn es den AUF/ZU-Button gibt
             If InStr(shShape.Name, "27") > 1 Then
                ActiveSheet.Unprotect (Val(PW))
                shShape.Select
                Selection.ShapeRange.ZOrder msoSendToBack
             End If
          Next shShape
   ActiveSheet.Range("C6").Select
Next i

End Sub

Sub Alles_Zu()
' dies ist der Code, um alle Tabellenblätter, die einen AUF/ZU-Wechselschalter haben, abzusperren
' sie wird aktuell beim Speichern der Arbeitsmappe immer automatisch ausgeführt

Aktuell = ActiveSheet.Name
Application.ScreenUpdating = False

For S = 1 To Worksheets.Count
       Worksheets(S).Activate
             For Each shShape In ActiveSheet.Shapes
             ' wenn es den AUF/ZU-Button gibt
             If InStr(shShape.Name, "27") > 1 Then
                If ActiveSheet.ProtectContents = False Then
                          ZUMACHEN
                End If
             End If
          Next shShape
```

```
Next S

Sheets(Aktuell).Activate
Application.ScreenUpdating = True

End Sub
```

Blatt-Schutz von Tabellenblättern auslesen, setzen

Blattschutz einer Tabelle auslesen und merken - Blattschutz öffnen - Tabelle bearbeiten - am Ende wieder Blattschutz setzen

Ich schütze gerne Tabellen, in denen die User arbeiten - aber wenn ich in einer Parametertabelle etwas ändere, dann muss der Blattschutz in der betroffenen Tabelle aufgehoben werden und am Ende, falls er gesetzt war, wieder gesetzt werden. So sieht der Code aus für zwei Tabellen RK und SP:

```
Private Sub Worksheet_Change(ByVal Target As Range)

Dim MERKER_RK ' ob Blattschutz gesetzt in RK-Tabelle
Dim MERKER_SP ' ob Blattschutz gesetzt in SP-Tabelle

MERKER_RK = RK.ProtectContents = True
MERKER_SP = SP.ProtectContents = True

RK.Unprotect
SP.Unprotect

Application.ScreenUpdating = False

' Ab hier kommt der Code, der die geänderten Parameterfelder umsetzt

    ' KORE-Aktivierung
    If Target.Address = "$O$10" Then
        If UCase(Target.Value = "JA") Then
                RK.Columns(6).Hidden = False
                SP.Columns(7).Hidden = False
            Else
                RK.Columns(6).Hidden = True
                SP.Columns(7).Hidden = True
        End If
    End If
….


If MERKER_RK = True Then RK.Protect DrawingObjects:=True, Contents:=True, Scenarios:=True
```

```
If MERKER_SP = True Then SP.Protect DrawingObjects:=True, Contents:=True, Scenarios:=True

Application.ScreenUpdating = True

End Sub
```

BSP 1 über Alle Tabellenblätter mit Passwort im VBA-Code

```
Sub ALLE_BLAETTER_SCHUTZEN()

        Dim PASS_WORT

        PASS_WORT = InputBox("Passwort (leer lassen wenn Schutz ohne Passwort gewünscht):", "PASSWORT")

        For T = 1 To Sheets.Count
           Sheets(T).Protect Password:=PASS_WORT, DrawingObjects:=True, Contents:=True, Scenarios:=True
        Next T

End Sub

Sub ALLE_BLATTER_SCHUTZ_AUFHEBEN()

        Dim PASS_WORT

        PASS_WORT = InputBox("Passwort bitte (falls notwendig, sonst leer lassen):", "PASSWORT")

        For T = 1 To Sheets.Count
           Sheets(T).Unprotect (PASS_WORT)
        Next T

End Sub
```

BSP 2 über ausgewählte Tabellenblätter mittels Hilfsprozedur und Passwort in geschütztem Tabellenblatt

```
Public Sub AlleTabellen_Un_Lock()

' Diese Prozedur wird über das Kontrollkästchen AlleTabellen der Tabelle develop in Zelle C12 aktiviert
```

```
' je nach Status des Kästchens werden alle User-Tabellenblätter in ihrem Tabellenblattschutz geöffnet oder gesperrt

On Error Resume Next

If Sheets("Develop").Range("C10") = True Then ' wenn Tabellenblattschutz aufgehoben sein soll
    Call T_Lock("Optionen")
    Call T_Lock("")
    MsgBox "Alle User-Tabellen sind nun ungeschützt und voll bearbeitbar"
   Else
    Call T_Unlock("Optionen")
    Call T_Unlock("")
    MsgBox "Alle User-Tabellen sind nun wieder geschützt"
End If
End Sub
Public Sub T_Lock(Tabellenname As String)
    Sheet(Tabellenname).Protect Password:=Sheets("develop").Range("J12"), DrawingObjects:=True, Contents:=True, Scenarios:=True
End Sub
Public Sub T_Unlock(Tabellenname As String)
    Sheet(Tabellenname).Unprotect Password:=Sheets("develop").Range("J12")
End Sub
```

Tabellenblatt-Schutz setzen:

```
Worksheets(TABELLENBLATT).Protect Password:="xxxx", DrawingObjects:=True, Contents:=True, Scenarios:=True, AllowFormattingCells:=True,
AllowFormattingColumns:=True, AllowFormattingRows:=True
```

Auslesen ob Status des Tabellenblattes auf Blattschutz - und wenn ja, dann mit Passwort den Schutz aufheben

```
   If Worksheets(TABELLENBLATT).ProtectContents = True Then Worksheets(TABELLENBLATT).Unprotect Password:="xxxx"
```

Blattschutz durch Makro aus- und einschalten

```
Sub SchutzAusEin()
    ActiveSheet.Unprotect "Test"
    MsgBox "Blattschutz ist aufgehoben!"
    ActiveSheet.Protect "Test"
    MsgBox "Blattschutz ist gesetzt!"
End Sub
```

Blattschutz - Wechselschalter AUF / ZU

siehe AUF-ZU-Wechselschalter weiter oben

Blattschutz knacken Excel 97-2010

```
Sub Blattschutz_löschen()
On Error Resume Next
For i = 65 To 66
For j = 65 To 66
For k = 65 To 66
For l = 65 To 66
For m = 65 To 66
For n = 65 To 66
For o = 65 To 66
For p = 65 To 66
For q = 65 To 66
For r = 65 To 66
For s = 65 To 66
For t = 32 To 126

ActiveSheet.Unprotect Chr(i) & Chr(j) & Chr(k) & Chr(l) & Chr(m) & _
Chr(n) & Chr(o) & Chr(p) & Chr(q) & Chr(r) & Chr(s) & Chr(t)

Next t
Next s
Next r
Next q
Next p
Next o
Next n
Next m
Next l
Next k
Next j
Next i
MsgBox "Fertig"
End Sub
```

Arbeitsmappen speichern / als / verhindern / schließen ohne speichern

1.) Speichern

Activeworkbook.Save

2.) Speichern aller Offenen Arbeitsmappen und anschließendes Beenden von Excel

```
For Each w In Application.Workbooks
    w.Save
Next w
Application.Quit
```

3.) Speichern als

ActiveWorkbook.SaveAs "C:\Programme\Test.xls"

4.) Schliessen ohne Speichern

```
' Aktive Arbeitsmappe
ActiveWorkbook.Close savechanges:=False
' irgendeine Arbeitsmappe
Workbooks("Test.xls").Close savechanges:=False
```

5.) Speichern verhindern

Der eleganteste Weg (wenn man sowieso schon VBA einsetzt) ist in Verwendung der folgenden zwei Ereignisse mit den entsprechenden Code-Zeilen.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

```
    ThisWorkbook.Saved = True

End Sub



Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

    Cancel = True

End Sub
```

Beide Ereignisse müssen in das Modul “DieseArbeitsmappe” geschrieben werden.

Das Ereignis **Workbook_BeforeClose** wird immer ausgeführt, bevor die Arbeitsmappe geschlossen wird. Der verwendete Befehl suggeriert Excel, dass keine Änderungen vorgenommen wurden und damit wird die «Möchten Sie speichern» Abfrage unterdrückt.

Das Ereignis **Workbook_BeforeSave** wird immer dann ausgeführt, wenn der Benutzer bereits den Befehl zum speichern gegeben hat, aber noch bevor das Speichern ausgeführt wird. Mit der Anweisung «Cancel = True» wird der Vorgang ohne weitere Meldung abgebrochen.

Arbeitsmappen vergleichen

```
Sub CompareWorkbooks()
Dim iWB As Integer, iWS As Integer
Dim rngObj As Range

If Workbooks(1).Worksheets.Count <> Workbooks(2).Worksheets.Count Then
MsgBox "Number of worksheets differs"
Exit Sub
End If
```

```
For iWS = 1 To Workbooks(1).Worksheets.Count
If Workbooks(1).Worksheets(iWS).UsedRange.Cells.Count <>
Workbooks(2).Worksheets(iWS).UsedRange.Cells.Count Then
MsgBox "Number of used cells in sheet " & iWS & "
differs"
Exit Sub
End If

For Each rngObj In Workbooks(1).Worksheets(iWS).UsedRange
If rngObj.Value <> Workbooks(2).Worksheets(iWS).Range(rngObj.Address).Value
Then
For iWB = 1 To 2
Workbooks(iWB).Worksheets(iWS).Activate
ActiveSheet.Range(rngObj.Address).Activate
Next

MsgBox "Difference detected at sheet " & iWS & " at cell
" & rngObj.Address(False, False)
Exit For
End If
Next
Next
End Sub
```

Daten aus anderer Arbeitsmappe abrufen

SIEHE AUCH NACHFOLGENDEN EINTRAG EXCELDATEI O. CSV/TXT-DATEI ÖFFNEN UND DATEN IN AKTUELLE MAPPE KOPIEREN

MsgBox Workbooks("Countrates_Overview.xls").Sheets("Sheet2").Range("A1")

Wenn du mit Tabellen-Objekten arbeitest:

```
Dim AKTUELLEMAPPE As Workbook
Dim NEUEMAPPE as Workbook

    Set AKTUELLEMAPPE = ActiveWorkbook

    Application.SheetsInNewWorkbook = 1 ' ich will ein Tabellenblatt in der neuen Arbeitsmappe
    Workbooks.Add ' hier wird die Arbeitsmappe erstellt
    Set NEUEMAPPE = ActiveWorkbook ' festlegen der Variable

msgbox AKTUELLEMAPPE.Sheets(Tabelle1.Name).Range("O16").Value
```

Exceldatei oder CSV/TXT-Datei öffnen und Daten in aktuelle Mappe importieren

```
Sub IMPORT_DATEI_AUTOMATISCH()

' Dieser Code öffnet und importiert Daten aus
' 1.) Einer Exceldatei
' oder - lt. Parameter in T1.Range("O4") denn dort ist das Importtrennzeichen "," ";" "Tabulator" oder "Keines / Excel"
hinterlegt
' 2.) einer CSV oder TXT-Datei mit variablem Trennzeichen
' in das Zieltabellenblatt (T2) der aktuellen Arbeitsmappe

' --- Variablendefinition ---

    Dim Excel_Quelldatei As Object ' Die Excel_Quelldatei
    Dim Aktuelle_Mappe As Workbook ' Die aktuelle Mappe hier
    Dim Datei_Gesamtpfad As String ' Der Pfad zur Quelldatei inkl. Dateinamen und Endung (xls/xlsx, csv oder txt)

' --- Auslesen Quelldateipfad und Prüfen, ob es die Datei gibt ---

    Datei_Gesamtpfad = T1.Range("O5")
    If Datei_Gesamtpfad = "" Then MsgBox "Es ist kein Dateipfad und Dateiname ausgefüllt in der Zelle O5 in der
Parametertabelle": Exit Sub
    If Dir(Datei_Gesamtpfad) = "" Then MsgBox "Die Datei " & Datei_Gesamtpfad & " in der Zelle O5 in der Parametertabelle
existiert nicht.": Exit Sub
    ' Wenn Exceldatei=>richtiges Trennzeichen ? - wird aber ohnedies vom Importcode unten automatisch korrigiert, der
xls(x)-Dateien automatisch als Excel einliest
    If InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls") > 0 And T1.Range("O4").Value <> "Keines /
Excel" Then
        MsgBox "Sie haben als Quelldatei eine Exceldatei hinterlegt, aber ein falsches Importtrennzeichen eingestellt.
Bitte wählen Sie das Trennzeichen 'Keines / Excel'"
        Exit Sub
    End If
    ' Wenn keine Exceldatei=>richtiges Trennzeichen ?
    If InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls") = 0 And T1.Range("O4").Value = "Keines /
Excel" Then
        MsgBox "Sie haben als Quelldatei keine Exceldatei hinterlegt, aber als Importtrennzeichen 'Keines / Excel'
eingestellt. Bitte wählen Sie das korrekte Trennzeichen."
        Exit Sub
    End If
```

```
' ---  Löschen der Importtabelle ---

    T2.UsedRange.ClearContents ' Zellformat (Text bei uns) soll erhalten bleiben

' ---  Import der Daten ---

    If T1.Range("O4").Value = "Keines / Excel" Or InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls")
> 0 Then

       ' Wenn die Quelldatei eine Exceldatei ist

' Prüfen ob die Arbeitsmappe  bereits offen ist
    If bIsBookOpen(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "\") + 1)) Then
        MsgBox "Die Datei " & Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "\") + 1) & " ist bereits geöffnet. Sie
muss zuerst geschlossen werden, bevor die Daten übertragen werden können."
        Exit Sub
    End If

       Set Aktuelle_Mappe = ActiveWorkbook ' merken der aktuellen Mappe hier
       Set Quelldatei = Workbooks.Open(Datei_Gesamtpfad, False, True) ' Verknüpfungen nicht aktualisieren (false) und
Datei nur lesend öffnen (true)
       ActiveWorkbook.ActiveSheet.UsedRange.Copy ' Alle Daten in der Arbeitsmappe kopieren (in derem aktiven
Tabellenblatt - unsere Quelldaten haben nur ein Tabellenblatt)
       Aktuelle_Mappe.Activate ' Rückkehr in unsere aktuelle Arbeitsmappe
       T2.Range("A1").PasteSpecial ' Werte einfügen in unser Tabelle T2, die die Daten erhalten soll
       Application.DisplayAlerts = False ' kein Hinweis vor dem Schließen
       Quelldatei.Close
       Application.DisplayAlerts = True

      Else

        ' Import CSV oder TXT-Datei mit variablem Trennzeichen

       TRENNZEICHEN = T1.Range("O4")
       If Len(TRENNZEICHEN) > 1 Then TRENNZEICHEN = vbTab ' Tabulator
       iReihe = 1 ' Variable durch alle Zeilen
       iSpalte = 1 ' Variable durch alle Spalten
       Close ' Sicherheitshalber eventuell offene Dateien schließen
       T2.Activate
       ' Einspielen der Daten
       Open Datei_Gesamtpfad For Input As #1
```

```
            Do Until EOF(1)
               Line Input #1, iText
               Do While InStr(iText, TRENNZEICHEN)
                  Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
                  iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
                  iSpalte = iSpalte + 1
               Loop
               Cells(iReihe, iSpalte).Value = iText
               iReihe = iReihe + 1
               iSpalte = 1
            Loop
         Close

      End If

End Sub
```

Externe Links zu anderen Arbeitsmappen entfernen

This little macro code will go through all the external links in your workbook and break the links. Note that this will not remove those pesky (hard-to-find) external links that may be hiding inside your charts. This code only addresses links that would show up inside the **Edit Links** dialog box (Data tab > Connections Group > Edit Links).

```
Sub BreakExternalLinks()
'PURPOSE: Breaks all external links that would show up in Excel's "Edit Links" Dialog Box
'SOURCE: www.TheSpreadsheetGuru.com/The-Code-Vault

Dim ExternalLinks As Variant
Dim wb As Workbook
Dim x As Long

Set wb = ActiveWorkbook

'Create an Array of all External Links stored in Workbook
  ExternalLinks = wb.LinkSources(Type:=xlLinkTypeExcelLinks)

'Loop Through each External Link in ActiveWorkbook and Break it
  For x = 1 To UBound(ExternalLinks)
    wb.BreakLink Name:=ExternalLinks(x), Type:=xlLinkTypeExcelLinks
  Next x

End Sub
```

Kopie der aktuellen Arbeitsmappe speichern

```
Sub BACKUP()

    Dim DATEIPFAD As String ' Gesamtpfad zur Mappe inkl Dateiname und -endung
    Dim BACKUP As String ' Der Backupdateipfad und Dateiname inkl Endung

    DATEIPFAD = ActiveWorkbook.FullName

' Anlegen einer Sicherungskopie der Datei
    Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
    Application.EnableEvents = False ' keine Events automatisch ausführen lassen
    BACKUP = Left(DATEIPFAD, InStrRev(DATEIPFAD, ".") - 1) & " Backup" & "." & Mid(DATEIPFAD, InStrRev(DATEIPFAD, ".") + 1, 4)
    ActiveWorkbook.SaveCopyAs BACKUP
    Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
    Application.EnableEvents = True ' Events wieder automatisch ausführen lassen

End Sub
```

Ich verwende diesen Code in meiner LEA-Datei, um automatisch nach jedem Speichern eine Kopie der Datei in einem Backupordner zu speichern

```
Private Sub Workbook_AfterSave(ByVal Success As Boolean)
```

```
' Dieser Code dient dazu eine Arbeitsmappe nach dem Speichern gleich noch ein zweites Mal
' im Unterordner \Backup zu speichern mit dem Datum und der Uhrzeit im Dateinamen

Dim Dateiname   ' Dateiname ohne Dateiendung
Dim Dateiendung ' Dateityp
Dim Backupgesamtname    ' Backuppfad inklusive Sicherungsdatei-Gesamtname

Dim Monat As String     ' Aktuelles Monat mit führender Null, wenn <10
Dim Tag As String       ' Aktueller Tag mit führender Null, wenn <10

Dim Stunden As String    ' Aktuelle Stunde mit führender Null
Dim Minuten As String    ' Aktuelle Minute mit führender Null
Dim Sekunden As String    ' Aktuelle Sekunde mit führender Null

Dim Datumstempel As String ' YYYYMMDD HHMMSS

'   Auslesen des Dateinamens der aktuellen Datei

Dateiname = Left(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") - 1)
Dateiendung = Mid(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") + 1)

' Definieren des Datums- und Zeitstempels

If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)), 2)

Datumstempel = Year(Now) & Monat & Tag ' & "_" & Stunden & Minuten & Sekunden

' Überprüfen, ob es den Unterordner Backup bereits gibt, sonst lege ihn an

If Dir(ActiveWorkbook.Path & "\Backup", vbDirectory) = "" Then
    MkDir (ActiveWorkbook.Path & "\Backup")
End If

' Speichern der Sicherungskopie der aktuellen Datei

Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
Application.EnableEvents = False ' keine Events automatisch ausführen lassen

Backupgesamtname = ActiveWorkbook.Path & "\Backup\" & Dateiname & "_" & Datumstempel & "." & Dateiendung
ActiveWorkbook.SaveCopyAs Backupgesamtname
```

```
    Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
    Application.EnableEvents = True ' Events wieder automatisch ausführen lassen


End Sub
```

Neue Arbeitsmappe erstellen – Anzahl Tabellenblätter festlegen – Daten übertragen

```
Dim AKTUELLEMAPPE As Workbook
Dim NEUEMAPPE as Workbook
Set AKTUELLEMAPPE = ActiveWorkbook

    Application.SheetsInNewWorkbook = 1 ' ich will ein Tabellenblatt in der neuen Arbeitsmappe
    Workbooks.Add ' hier wird die Arbeitsmappe erstellt
    Set NEUEMAPPE = ActiveWorkbook ' festlegen der Variable

Schreiben von Werten zwischen den beiden Arbeitsmappen

Cells(1,1) = AKTUELLEMAPPE.Sheets(Tabelle1.Name).Range("O16").Value
```

Neue Arbeitsmappe öffnen, benennen und speichern

```
    ' creating a new workbook
    Workbooks.Add
    ActiveWindow.Caption = "test.xls"

Achtung: mit dem zweiten Befehl wird zwar der Excel-Arbeitsmappenname richtig angezeigt,
aber die Arbeitsmappe kann noch nicht mit workbooks("test.xls").-Befehl angesprochen werden

Erst mit dem Speichern kann man eine Datei ansprechen - daher:

    ' Interim-Save-Location
    ActiveWorkbook.SaveAs "D:\Test\test.xls"
```

Prüfen ob eine Exceldatei schon offen ist

Version 1

```
Public Sub TestePfad()
    Dim sPfad As String
    sPfad = "C:/Eigene Dateien/Test.xls" ' Pfad ändern für Tests
```

```
    MsgBox DateiGeoeffnet(sPfad)
End Sub

Private Function DateiGeoeffnet(DerPfad As String) As Boolean
    On Error Resume Next
    Open DerPfad For Binary Access Read Lock Read As #1
    Close #1
    If Err.Number <> 0 Then
        DateiGeoeffnet = True
        Err.Clear
    End If
End Function
```

Version 2

```
Function IsWorkbookOpen(strWB As String) As Boolean
  On Error Resume Next
  IsWorkbookOpen = Not Workbooks(strWB) Is Nothing
End Function


Sub test()
  If IsWorkbookOpen("Mappe2.xls") Then
    MsgBox "OFFEN"
  Else
    MsgBox "Nicht offen"
  End If
End Sub
```

Version 2b

```
Function bIsBookOpen(ByRef szBookName As String) As Boolean
' Funktion, die überprüft, ob die übergebene Arbeitsmappe bereits offen ist
    On Error Resume Next
    bIsBookOpen = Not (Application.Workbooks(szBookName) Is Nothing)
End Function

Sub TEST

' Prüfen ob die Arbeitsmappe  bereits offen ist - in P2 ist der gesamte Pfad und Datei: C:\Daten\Test.xlsx
    If bIsBookOpen(Mid(Range("P2"), InStrRev(Range("P2"), "\") + 1)) Then
        MsgBox "Die Datei " & Mid(Range("P2"), InStrRev(Range("P2"), "\") + 1) & " ist bereits geöffnet. Sie muss zuerst geschlossen werden, bevor die Daten
```

```
übertragen werden können."
        Exit Sub
    End If

End Sub
```

Version 3

```
Option Explicit
Function StatusExcelDat(ExcelFullDatNam As String, ExcelTabNam As String) As String
' ============
' Ab Access 97 (sollte aber auch in Excel funzen)
' ============
' Ergebnis:
' 1. Stelle: 0: kein Excel offen
'            1: excel offen
' 2: Stelle: 0: Exceldatei existiert nicht
'            1: Exceldatei existiert
' 3: Stelle: 0: Exceldatei geschlossen
'            1: Exceldatei offen
' 4: Stelle: 0 Tabellenblatt existiert nicht
'            1 Tabellenblatt existiert
' 4: Stelle: 0 Tabellenblatt existiert nicht
'            1 Tabellenblatt existiert
' 5: Stelle: 0 Name existiert nicht
'            1 Name existiert

    Dim WorkBookNam As String, Ergebnis As String
    Dim ExcelDatExist As Boolean, TabExist As Boolean, ExcelIsOpen As Boolean
    Dim NameExist As Boolean, WorkbookIsOpen As Boolean
    Dim xlApp As Object ' Excel.Application
    Dim xlBook As Object  'Excel.Workbook
    Dim xlSheet As Object ' Excel.Worksheet
    Dim i As Integer
    On Error Resume Next
    Set xlApp = GetObject(, "Excel.Application")
    If xlApp Is Nothing Then
        Set xlApp = CreateObject("Excel.Application")
    Else
        ExcelIsOpen = True
    End If
    On Error GoTo 0

    ' xls anhängen, falls user gepennt
```

```
    If Len(ExcelFullDatNam) < 5 Then ExcelFullDatNam = ExcelFullDatNam & ".xls"
    If Mid(ExcelFullDatNam, Len(ExcelFullDatNam) - 3, 4) <> ".xls" Then _
                               ExcelFullDatNam = ExcelFullDatNam & ".xls"

    If Len(Dir(ExcelFullDatNam)) > 0 Then
       ExcelDatExist = True
       ' Ermittlung des Dateinamens aus dem Namen incl. Pfad
       ' Ab AC 2000 ist dies einfacher mit InstrRev
       Dim ii As Integer, iiPos As Integer
       For ii = 1 To Len(ExcelFullDatNam)
          If Mid(ExcelFullDatNam, ii, 1) = "\" Then iiPos = ii
       Next
       WorkBookNam = Mid(ExcelFullDatNam, iiPos + 1)


       For Each xlBook In xlApp.Workbooks
          If xlBook.Name = WorkBookNam Then WorkbookIsOpen = True
       Next
       If Not WorkbookIsOpen Then
          Set xlBook = xlApp.Workbooks.Open(ExcelFullDatNam)
       Else
          Set xlBook = xlApp.Workbooks(WorkBookNam)
       End If
       For i = 1 To xlBook.names.Count
         Debug.Print xlBook.names(i)
         If xlBook.names(i).Name = ExcelTabNam Then NameExist = True
       Next
       For Each xlSheet In xlBook.Worksheets
         If xlSheet.Name = ExcelTabNam Then TabExist = True
       Next
       If Not WorkbookIsOpen Then xlBook.Close
    End If

    If Not ExcelIsOpen Then xlApp.Application.Quit
    Set xlSheet = Nothing
    Set xlBook = Nothing
    Set xlApp = Nothing
    If ExcelIsOpen Then Ergebnis = "1" Else Ergebnis = "0"
    If ExcelDatExist Then Ergebnis = Ergebnis & "1" Else Ergebnis = Ergebnis & "0"
    If WorkbookIsOpen Then Ergebnis = Ergebnis & "1" Else Ergebnis = Ergebnis & "0"
    If TabExist Then Ergebnis = Ergebnis & "1" Else Ergebnis = Ergebnis & "0"
    If NameExist Then Ergebnis = Ergebnis & "1" Else Ergebnis = Ergebnis & "0"
    StatusExcelDat = Ergebnis
End Function
```

die Anwendung der Funktion, könnte z.B. so aussehen:

Code:

```
If Mid(StatusExcelDat("D:\tar\mytest.xls", "dummy"), 2, 1) = "1" Then
  MsgBox "exceldatei D:\tar\mytest.xls geöffnet"
End If
```

VERSION 4

```
Function DateiSchonGeöffnet(ByVal str As String) As Boolean
Dim DateiSchonGeöffneted As Boolean
On Error GoTo fehler
DateiSchonGeöffneted = True
Windows(str).Activate
Exit Function

fehler:
DateiSchonGeöffneted = False
End Function

Sub ArbeitsmappeGeöffnet()
Dim B As Boolean
B = DateiSchonGeöffnet(Range("A1").Value)
If B = False Then MsgBox "Die Datei ist noch nicht geöffnet!" Else _
MsgBox "Die Datei ist bereits geöffnet!"
End Sub
```

Prozedur in anderer Arbeitsmappe aufrufen

siehe eigener Bereich FUNKTIONEN – PROZEDUREN ALLGEMEIN

Schutz von Arbeitsmappe, VBA, etc auslesen

Zum Blattschutz siehe Einträge davor "AUF / ZU-Wechselschalter" , "Blattschutz ..."

```
MsgBox ActiveWorkbook.ReadOnly


MsgBox ActiveWorkbook.ProtectStructure '  true = Arbeitsmappenschutz ist gesetzt, false: Mappe ist nicht geschützt

MsgBox ActiveWorkbook.ProtectWindows

MsgBox ActiveWorkbook.VBProject.Protection
```

Sicherungskopie der aktuellen Arbeitsmappe automatisch erstellen

```
Private Sub Workbook_AfterSave(ByVal Success As Boolean)

    ' Dieser Code dient dazu eine Arbeitsmappe nach dem Speichern gleich noch ein zweites Mal
    ' im Unterordner \Backup zu speichern mit dem Datum und der Uhrzeit im Dateinamen

    Dim Dateiname   ' Dateiname ohne Dateiendung
    Dim Dateiendung ' Dateityp
    Dim Backupgesamtname    ' Backuppfad inklusive Sicherungsdatei-Gesamtname

    Dim Monat As String     ' Aktuelles Monat mit führender Null, wenn <10
    Dim Tag As String       ' Aktueller Tag mit führender Null, wenn <10

    Dim Stunden As String    ' Aktuelle Stunde mit führender Null
    Dim Minuten As String    ' Aktuelle Minute mit führender Null
    Dim Sekunden As String    ' Aktuelle Sekunde mit führender Null

    Dim Datumstempel As String ' YYYYMMDD HHMMSS

    '   Auslesen des Dateinamens der aktuellen Datei
```

```
    Dateiname = Left(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") - 1)
    Dateiendung = Mid(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") + 1)

    ' Definieren des Datums- und Zeitstempels

    If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
    If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
    If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
    If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
    If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)), 2)

    Datumstempel = Year(Now) & Monat & Tag & "_" & Stunden & Minuten & Sekunden

    ' Überprüfen, ob es den Unterordner Backup bereits gibt, sonst lege ihn an

    If Dir(ActiveWorkbook.Path & "\Backup", vbDirectory) = "" Then
        MkDir (ActiveWorkbook.Path & "\Backup")
    End If

    ' Speichern der Sicherungskopie der aktuellen Datei

    Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
    Application.EnableEvents = False ' keine Events automatisch ausführen lassen

    Backupgesamtname = ActiveWorkbook.Path & "\Backup\" & Dateiname & "_" & Datumstempel & "." & Dateiendung
    ActiveWorkbook.SaveCopyAs Backupgesamtname

    Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
    Application.EnableEvents = True ' Events wieder automatisch ausführen lassen


End Sub
```

Überwachung SHEETCHANGE / SELECTIONCHANGE

siehe Tabellenblätter - Überwachung Sheetchange

VBA-Prozedur in anderer, offenen Arbeitsmappe aufrufen

Siehe VBA

DATEIZUGRIFFE + ORDNER

-→ Grundlagen DATEN SCHREIBEN UND LESEN

VBA already includes commands to allow data to read or write to external text files. This is more commonly known as **I/O** (Input / Output) and is used to store files in the formats such as ‘txt’, ‘csv’ and ‘ini’ files.

Example of Output Data:

```
Sub BuildTextFile()
    Dim fnum

    fnum = FreeFile()
    Open "C:\vba.txt" For Output As #fnum

    Write #fnum, "Excel VBA", "Day 1"
    Write #fnum, "Excel VBA", "Day 2"
    Write #fnum, "Excel VBA Workshop Q&A", "Day 3"

    Close #fnum
End Sub
```

The above example creates an instance of a file using the **FreeFile** function, which returns a unique number (as its handler). The **Open** method is used to locate and open the file.

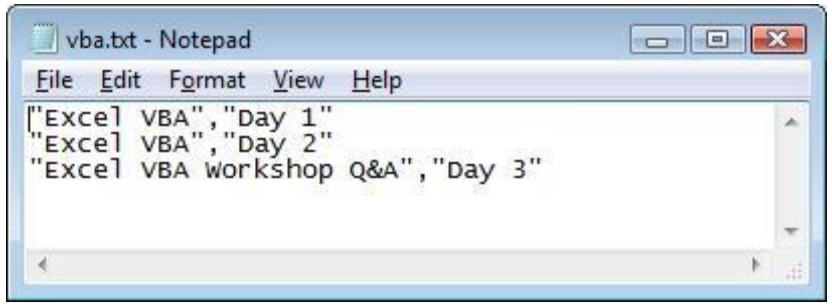
The **Output** property tells the system that data is to be written to the named file using the pointer **#fnum**.

The **Write** method adds line-by-line data to the pointer and then is lost with the **Close** method.

Even if the file name does not exist, it will create this file in the specified path but the path must exist.

If the filename already exists, this routine will overwrite (no prompt) and the previous file will be lost.

The file generated is a 'txt' file:



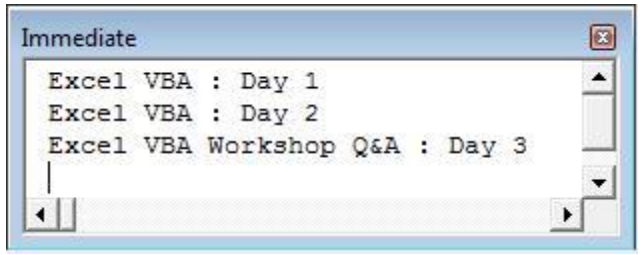
Example of Input Data:

```
Sub ReadTextFile()
    Dim fnum
    Dim strField1 As String, strField2 As String

    fnum = FreeFile()
    Open "C:\vba.txt" For Input As #fnum

    Do Until EOF(fnum)
        Input #fnum, strField1, strField2
        Debug.Print strField1 & " : " & strField2
    Loop

    Close #fnum
End Sub
```

The above example uses the **`Input`** property instead to change the direction of the flow of data (*read from*).

Using the **`EOF`** method, the procedure loops through the delimiter line break until it reaches the end of the file.

To view the results, open the **Immediate** Window (**Ctrl + G**) before running the above procedure:



The above two examples demonstrates how to read and write data to and from external files and will require a little more coding to deal with interaction and variables to make this more flexible (and practical).

Aktueller Datei-Pfad

siehe auch ARBEITSMAPPEN / ALLES ZU PFAD, DATEINAME, DATEIENDUNG

Manchmal möchte man den Pfad der aktuellen Datei wissen, um z.B. eine Datei, die im selben Ordner abgespeichert ist, direkt öffnen zu können.

PFAD der aktuellen Datei:

MsgBox ThisWorkbook.Path

GESAMTER PFAD INKLUSIVE DATEINAME:

MsgBox ActiveWorkbook.FullName

Alle Dateien in Ordner/Unterordner Excel 2007 auflisten

Filesearch gibt es ja nicht mehr unter Excel 2007 - daher folgender Code von mir

Sub Dateisuche()

```
' Variablen für Ordner und Dateien
Dim Verzeichnis As String ' das Hauptverzeichnis, das durchsucht wird
Dim objFSO As Object ' das Filesystemobjekt
Dim objDir As Object ' das Verzeichnisobjekt

Dim Unterverzeichnis As Variant ' die Unterverzeichnisse im Hauptverzeichnis
Dim Datei As Variant ' die einzelnen Dateien
Dim Datei_Gesamtpfad As String ' der gesamte Pfad zur Datei und der Dateiname : zB C:\TEST\Mappe1.xls

' Erzeugen des Ordner-Objektes

Set objFSO = CreateObject("scripting.filesystemobject")
Verzeichnis = "C:\Test"
Set objDir = objFSO.GetFolder(Verzeichnis)

' Auflisten der Dateien direkt im Hauptordner

For Each Datei In objDir.Files
   Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
   If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) wollen wir nicht anzeigen
      MsgBox Datei_Gesamtpfad & " aus dem Ordner " & objDir.Name
   End If
Next Datei

MsgBox "Das war die letzte Datei direkt im Hauptordner - nun kommen die Dateien in den Unterordnern"

' Auflisten der Dateien in den Unterordnern

For Each Unterverzeichnis In objDir.subfolders
   For Each Datei In Unterverzeichnis.Files
      Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
      If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) wollen wir nicht anzeigen
         MsgBox Datei_Gesamtpfad & " aus dem Unterordner " & Unterverzeichnis.Name
      End If
   Next Datei
Next Unterverzeichnis

' Speicher sparen

Set objFSO = Nothing
Set objDir = Nothing

End Sub
```

Alle Excel-Dateien im selben Verzeichnis öffnen und Daten übertragen

```
Sub Import()

' Variablen für Ordner und Dateien
Dim Verzeichnis As String ' das Hauptverzeichnis, das durchsucht wird
Dim Quelldatei As Object ' Die einzelnen Quelldateien
Dim Zieldatei As Object ' die Vorlagendatei hier
Dim objFSO As Object ' das Filesystemobjekt
Dim objDir As Object ' das Verzeichnisobjekt

' Dim Unterverzeichnis As Variant ' die Unterverzeichnisse im Hauptverzeichnis - aktuell nicht benötigt

Dim Datei As Variant ' die einzelnen Dateien
Dim Datei_Gesamtpfad As String ' der gesamte Pfad zur Datei und der Dateiname : zB C:\TEST\Mappe1.xls

Dim AnzahlspaltenQ As Integer ' Anzahl der verwendeten Spalten in der Quelldatei
Dim AnzahlzeilenQ As Integer ' Anzahl der verwendeten Zeilen in der Quelldatei
Dim AnzahlspaltenZ As Integer ' Anzahl der verwendeten Spalten in der Zieldatei

Dim Spaltenposition(1000) As Integer  ' die Position der Quelldateispalte in der Zieldatei maximal 1000 Spalten sind möglich
Dim QS As Integer ' Spaltennummer in der Quelldatei
Dim ZS As Integer ' Spaltennummer in der Zieldatei
Dim LetztezeileZ ' letzte Zeile in Zieltabelle mit Inhalt

Dim QSNAME As String ' Bezeichnung der Spaltenüberschrift in Quelldatei

On Error GoTo FEHLER

' Bildschirmflackern deaktivieren

    Application.ScreenUpdating = False 'Das "Flackern" ausstellen
    Application.DisplayAlerts = False 'Keine Fehlermeldungen anzeigen

' Erzeugen des Ordner-Objektes, in dem diese Vorlagendatei hier ist

Set objFSO = CreateObject("scripting.filesystemobject")
Verzeichnis = ThisWorkbook.Path
Set Zieldatei = ThisWorkbook
```

```
Set objDir = objFSO.GetFolder(Verzeichnis)

' Beginn der Schleife durch alle Excel-Dateien im Hauptordner (außer der Vorlage hier)

For Each Datei In objDir.Files
    Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String

    ' Wenn eine der nachfolgenden 3 Prüfungen ein JA ergibt, soll die betreffende Datei NICHT übernommen werden
    ' 1) Wenn es sich um keine Exceldatei handelt (sondern zB um eine TXT, PDF etc)
    ' 2) Wenn es sich um die Vorlagendatei hier selbst handelt, die ja auch im selben Ordner sein muss
    ' 3) Wenn es sich um eine bereits geöffnete Datei handelt (zB die temporäre Datei der Vorlagendatei hier - erkennbar am ~ im Dateinamen)
    If InStr(Datei_Gesamtpfad, ".xl") < 1 Or Datei_Gesamtpfad = ThisWorkbook.FullName Or InStr(Datei_Gesamtpfad, "~") > 0 Then GoTo Naechstedatei '
Code würde in einem dieser 3 Fälle zum Schleifenende springen und die nächste Datei analysieren

    ' Nur zu Demozwecken im Einsatz:
    ' MsgBox Datei_Gesamtpfad & " aus dem Ordner " & objDir.Name

    ' Auslesen der Anzahl der Spalten in der Zieldatei
    AnzahlspaltenZ = LETZTEZELLE(ActiveSheet.Name).Column

    ' Öffnen der betreffenden Datei
    Set Quelldatei = Workbooks.Open(Datei_Gesamtpfad, False, True) ' Verknüpfungen nicht aktualisieren (false) und Datei nur lesend öffnen (true)

    ' Schleife durch alle Tabellenblätter in der Quell-Datei - zur Zeit nicht aktiv, da bei allen Dateien nur das erste Tabellenblatt wichtig ist
    ' For Z = 1 To oSourceBook.Sheets.Count

    AnzahlspaltenQ = LETZTEZELLE(ActiveSheet.Name).Column
    AnzahlzeilenQ = LETZTEZELLE(ActiveSheet.Name).Row

    ' ---------------------------------------------
    ' 1) Auslesen der Spalten in Quelldatei
    ' ---------------------------------------------

    ' Schleife durch alle Spalten in der Quelldatei
    For QS = 1 To AnzahlspaltenQ

        ' Schleife durch alle Spalten in der Zieldatei
        For ZS = 1 To AnzahlspaltenZ

            ' Formatieren der Spaltenüberschrift durch Entfernen von Leerzeichen
            QSNAME = Replace(Cells(1, QS), " ", "")

            ' Wenn Quellspaltenname in Zieltabelle gefunden wird
            If QSNAME = ThisWorkbook.ActiveSheet.Cells(2, ZS) Then
```

```
            ' Merken der Zieltabellenspalte
            Spaltenposition(QS) = ZS
            GoTo WEITER
          End If

        Next ZS

        ' Wenn Code an dieser Stelle ankommt, hat er bei voriger Schleife durch Zielspalten die aktuelle Quellspalte nicht gefunden und diese muss in Zieltabelle angelegt werden
        ' merken der neuen Zielspalte
        Spaltenposition(QS) = AnzahlspaltenZ + 1
        ' Beschriften der neuen Zielspalte
        Zieldatei.ActiveSheet.Cells(2, Spaltenposition(QS)) = QSNAME
        ' Speichern, dass es nun in der Zieltabelle eine Spalte mehr gibt
        AnzahlspaltenZ = AnzahlspaltenZ + 1

WEITER:
    Next QS


    ' ------------------------------------------------
    ' 2) Übertragen der Daten
    ' ------------------------------------------------

    Zieldatei.Activate
    LetztezeileZ = LETZTEZELLE(ActiveSheet.Name).Row

    ' Übertragen Quelldateiname
    Cells(LetztezeileZ + 1, 1) = Quelldatei.Name

    For QS = 1 To AnzahlspaltenQ

        Quelldatei.Activate ' in Quelldatei springen

        ' kopiere Quelldaten in Zwischenablage
        Range(Cells(2, QS), Cells(AnzahlzeilenQ, QS)).Copy

        Zieldatei.Activate

        ' markiere oberste freie Zelle in Zieltabelle
        Cells(LetztezeileZ + 1, Spaltenposition(QS)).Select

        ' Einfügen der Daten
         Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
```

```
    Next QS


    ' aktuell geöffnete Quelldatei wieder schließen
    Quelldatei.Close False ' false = nicht speichern

Naechstedatei:
Next Datei


' Schleife durch die Dateien in den Unterordnern - aktuell nicht im Einsatz

'MsgBox "Das war die letzte Datei direkt im Hauptordner - nun kommen die Dateien in den Unterordnern"
'
'For Each Unterverzeichnis In objDir.subfolders
'    For Each Datei In Unterverzeichnis.Files
'        Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
'        If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) wollen wir nicht anzeigen
'            MsgBox Datei_Gesamtpfad & " aus dem Unterordner " & Unterverzeichnis.Name
'        End If
'    Next Datei
'Next Unterverzeichnis


' Bildschirmaktualisierung wieder aktivieren

    Application.ScreenUpdating = True 'Das "Flackern" ausstellen
    Application.DisplayAlerts = True 'Keine Fehlermeldungen anzeigen

' Speicher sparen

Set objFSO = Nothing
Set objDir = Nothing

' Fertig-Meldung

    MsgBox "Der Import der Daten ist abgeschlossen"

    End

FEHLER:

MsgBox "Leider ist ein Fehler aufgetreten - bitte kontaktieren Sie den Programmierer unter info@simplesoft.at"
```

```
End Sub
```

Alle Dateien in einem Verzeichnis öffnen

```
Sub OpenWkb()

  Dim sFile As String, sPath As String
  Application.ScreenUpdating = False
  sPath = "C:\Temp"
  If Right(sPath, 1) <> "/" Then
    sPath = sPath & "\"
  End If
  sFile = Dir(sPath & "*.xls")
  Do While sFile <> ""
    Workbooks.Open sPath & sFile
    ActiveSheet.Range("B48").FormulaLocal = "=Summe(A12:W12)"
    ActiveWorkbook.Close savechanges:=True
    sFile = Dir()
  Loop
  Application.ScreenUpdating = True
End Sub
```

Alle Dateien in einem Verzeichnis auslesen und auflisten (PDF-Dokumentenliste)

```
Sub DOKUMENTENIMPORT()


   Dim strVerzeichnis As String
   Dim strOrdner As String
   strOrdner = ThisWorkbook.Path & "\"
   With Application.FileDialog(msoFileDialogFolderPicker)
      .InitialFileName = strOrdner
      .Title = "Ordnerauswahl"
      .ButtonName = "Auswahl..."
      .InitialView = msoFileDialogViewList
       If .Show = -1 Then
         strVerzeichnis = .SelectedItems(1)
         If Right(strVerzeichnis, 1) <> "\" Then strVerzeichnis = strVerzeichnis & "\"
       Else
         MsgBox "Es wurde kein Ordner ausgewaehlt!"
```

```
        Exit Sub
      End If
    End With

    ' Variablen für Ordner und Dateien
    Dim objFSO As Object ' das Filesystemobjekt
    Dim objDir As Object ' das strVerzeichnisobjekt
    Dim Datei As Variant ' die einzelnen Dateien

    ' Erzeugen des Ordner-Objektes
    Set objFSO = CreateObject("scripting.filesystemobject")
    Set objDir = objFSO.GetFolder(strVerzeichnis)

    ' Leeren des letzten Imports
    PDF.Range("A2:Z64000").ClearContents
    Z = 2

    ' Auflisten der Dateien direkt im Hauptordner
    For Each Datei In objDir.Files
       ' MsgBox InStrRev(Datei, "\")
       Cells(Z, 1) = Mid(Datei, InStrRev(Datei, "\") + 1, 199)
       Z = Z + 1
    Next Datei


' Speicher sparen

Set objFSO = Nothing
Set objDir = Nothing

End Sub
```

Alle Dateien in einem Verzeichnis auslesen inkl. Unterverzeichnisse

```
Sub SCHLEIFE()
Dim DATEI As Variant
Const VERZEICHNIS = "C:\EIGENE DATEIEN\IT KNOWLEDGE\Excel\' VORLAGEN\LEERE VORLAGEN\"

On Error GoTo FEHLER
ChDir VERZEICHNIS
```

```
With Application.FileSearch
   .NewSearch
   .LookIn = VERZEICHNIS
   .Filename = "*.xls" ' oder csv, txt etc
   .SearchSubFolders = True

   If .Execute() > 0 Then
      For Each DATEI In .FoundFiles
         MsgBox DATEI
      Next DATEI
   End If

   MsgBox .FoundFiles.Count
End With
Exit Sub

FEHLER:
MsgBox "Das Verzeichnis " & VERZEICHNIS & " existiert nicht."

End Sub
```

Alle Excel-Dateien in einem Verzeichnis zusammenfassen

Hier meine neue Version für GTU (ab Excel 2007)

```
'Hinweise: Die Zieldatei darf nicht im gleichen Verzeichnis sein, wie die einzulesenden Dateien.
'Die einzulesenden Dateien müssen geschlossen sein.
'Wichtiger Hinweis: Die Arbeitsblätter dürfen nicht vorhanden sein!
'Alternativer Umbau: Löschen evtl. bereits vorhandener Arbeitsblätter


Sub MWSheetsAusMehrerenDateienEinlesen()
  Dim oTargetBook As Object ' Zieldatei (aktuelle Arbeitsmappe mit diesem Code hier)
  Dim oSourceBook As Object ' Quelldatei (jeweils alle Arbeitsmappen im zu durchsuchenden Verzeichnis)
  Dim sPfad As String       ' Zu durchsuchendes Quellverzeichnis
  Dim sDatei As String      ' Alle Exceldateien im Quellverzeichnis
  Dim Dateiname As String   ' Name der konkreten Datei
  Dim MaxLaenge As Integer  ' wieviele Zeichen vom Dateinamen sollen verwendet werden
  Dim Dateiteil             ' jener Teil, der vom Quelldateinamen verwendet werden soll
  Dim Blattnamenlaenge      ' Anzahl der Zeichen, die vom Quelldatei-Tabellenblattnamen verwendet werden sollen
  Dim Zufall                ' man kann aktivieren, dass ein 3-stelliger Zufallstext eingefügt wird nach dem Arbeitsmappen-Dateinamen (dieser Zusatz hat 3
Zeichen:  -a1- oder -2c-)
```

```
Dim Zufallstext          ' der Zufallstext

Application.ScreenUpdating = False 'Das "Flackern" ausstellen
Application.DisplayAlerts = False 'Keine Fehlermeldungen anzeigen

' Hier einstellen,dass interim ein Zufallstext generiert wird
Zufall = 1

' Festlegen der Zeichenanzahl, die vom Dateinamen genommen werden soll
MaxLaenge = Range("C6")
If MaxLaenge > 28 Then MaxLaenge = 28
If MaxLaenge = 0 Then MaxLaenge = 15
If Zufall = 1 Then MaxLaenge = MaxLaenge - 4

'Die aktuelle Mappe als Arbeitsmappe festlegen, in die die neuen Sheets eingefügt werden...
Set oTargetBook = ActiveWorkbook

' Festlegen des Pfades mit den Quelldateien
' sPfad = "E:\TEST\"
sPfad = Tabelle1.Range("C4")
' Ergänzen eines ev. fehlenden Bakslash
If Right(sPfad, 1) <> "\" Then sPfad = sPfad & "\"

' Filter nur auf alle Exceldateien setzen
sDatei = Dir(CStr(sPfad & "*.xl*")) 'Alle Excel Dateien

' Schleife über alle Excel Dateien im Verzeichnis
Do While sDatei <> ""

    ' öffnen der Quell-Datei
    Set oSourceBook = Workbooks.Open(sPfad & sDatei, False, True) 'nur lesend öffnen

'       Set objNew = Workbooks.Add(xlWBATWorksheet)

    ' Schleife durch alle Tabellenblätter in der Quell-Datei
    For Z = 1 To oSourceBook.Sheets.Count
        ' aktelles Tabellenblatt in Zieldatei am Schluss einfügen
            oSourceBook.Sheets(Z).Copy after:=oTargetBook.Sheets(oTargetBook.Sheets.Count)

        ' falls gewünscht: einen Zufallstext erzeugen
        If Zufall = 1 Then
              Zufallstext = "-"

              Zufallszahl = Int(35 * Rnd + 1) ' ASCI-CODE 48-57 sind die Ziffern 0-9 und ASCI-CODE  97-122 sind abc - insgesamt 36 Zeichen
```

```
        If Zufallszahl < 11 Then
           Zufallstext = Zufallstext & Chr(47 + Zufallszahl) ' die Ziffern 0-9
         Else
           Zufallstext = Zufallstext & Chr(96 + Zufallszahl - 10) ' die Buchstaben a-z
        End If

        Zufallszahl = Int(35 * Rnd + 1)
        If Zufallszahl < 11 Then
           Zufallstext = Zufallstext & Chr(47 + Zufallszahl) & "-"
         Else
           Zufallstext = Zufallstext & Chr(96 + Zufallszahl - 10) & "-"
        End If
    End If

' Festlegen des Tabellenblattnamens durch Dateinamen und Tabellenblattname (es sind max. 31 Zeichen erlaubt)

    Dateiname = sDatei
    ' Entfernen nicht benötigter Dateiendungen
    Dateiname = Replace(Dateiname, ".xlsx", "")
    Dateiname = Replace(Dateiname, ".xlsm", "")
    Dateiname = Replace(Dateiname, ".xls", "")

    If Zufall = 1 Then
        Dateiteil = Left(Dateiname, MaxLaenge)
        Blattnamenlaenge = 30 - Len(Dateiteil) - 4
        Tabellenname = Dateiteil & Zufallstext & Left(oSourceBook.Sheets(Z).Name, Blattnamenlaenge) ' drei Zeichen für den Zusatztext
      Else ' wenn kein Zufallstext
        Dateiteil = Left(Dateiname, MaxLaenge)
        Blattnamenlaenge = 30 - Len(Dateiteil)
        Tabellenname = Dateiteil & "-" & Left(oSourceBook.Sheets(Z).Name, Blattnamenlaenge)
    End If

    'Es wird versucht den Dateinamen als Arbeitsblattnamen zu setzen.
    'Ist dieser bereits vorhanden wird der Fehler abgefangen und das neue Blatt
    'bekommt keinen anderen Namen und behält den typischen Namen Tabelle x
    On Error Resume Next

    ' Benennen des neuen Tabellenblattes
    oTargetBook.Sheets(oTargetBook.Sheets.Count).Name = Tabellenname

    'Wenn ein Fehler aufgetreten ist, wird dieser resettet
    If Err.Number <> 0 Then
      Err.Number = 0
      Err.Clear
```

```
            End If
            On Error GoTo 0

        Next Z

        'Quell-Datei wieder zu machen und nächste Schleifenrunde
        oSourceBook.Close False 'nicht speichern

        'Nächste Datei
        sDatei = Dir()

    Loop

    ' Entfernen der Zufallszahlen

    On Error Resume Next

    For t = 1 To ActiveWorkbook.Sheets.Count

      a = InStr(Sheets(t).Name, "-")
      b = InStr(a + 1, Sheets(t).Name, "-")
      If b - a = 3 Then
         Tabellenname = Left(Sheets(t).Name, a) & Mid(Sheets(t).Name, b + 1, 99)
         Sheets(t).Name = Tabellenname
      End If

    Next t



    On Error GoTo 0

    Application.ScreenUpdating = True 'Das Bildschirm-Aktualisieren wieder einschalten
    Application.DisplayAlerts = True 'Fehlermeldungen wieder anzeigen

    'Kleine finale Fertig-Meldung
    MsgBox "Fertig!", vbInformation + vbOKOnly, "Hinweis!"

    'Variablen aufräumen
    Set oTargetBook = Nothing
    Set oSourceBook = Nothing

End Sub
```

Hier meine alte Version für GTU Unitreu (läuft vermutlich nur unter Excel 2003)

```
Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWB As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer
Dim Tabellenname As String


'strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

strPath = "E:\test\"

If strPath = "" Then Exit Sub

' On Error GoTo ErrExit

With Application
  '.ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
 ' .Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
  .NewSearch
  .LookIn = strPath
  .FileType = msoFileTypeExcelWorkbooks
  .SearchSubFolders = False
```

```
  If .Execute > 0 Then

    For intIndex = 1 To .FoundFiles.Count

      Set objWB = Workbooks.Open(.FoundFiles(intIndex))

      For Z = 1 To objWB.Sheets.Count

           objWB.Sheets(Z).Copy after:=objNew.Sheets(objNew.Sheets.Count)
           Tabellenname = objFO.GetBaseName(.FoundFiles(intIndex)) & " - " & objWB.Sheets(Z).Name
           Tabellenname = Left(Tabellenname, 31)
          ' MsgBox Tabellenname
           objNew.Sheets(objNew.Sheets.Count).Name = Tabellenname


      Next Z

      objWB.Close False

      Set objWB = Nothing

    Next

  End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs "C:\" & "Zusammenfassung.xls"

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

Andere Version

```
' ********************************************************************
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

'BrowseForFolder mit Extra-Funktionen
'VB -Versionen: VB5 , VB6
'Betriebssystem: Win9x , WinNT, Win2000, WinME, WinXP
'Autor: Marco Wünschmann Homepage: ohne
'Datum: 23.08.2004


' == Dialog-Einstellungen ================================

' String, der vor dem aktuell ausgewählen Verzeichnis angezeigt wird,
' falls der ShowCurrentPath-Paramter True ist.
Private Const DIALOG_CURRENT_SELECTION_TEXT As String = "Auswahl: "


' == API-Deklarationen ===================================

Private Type BROWSEINFO
  hOwner As Long
  pidlRoot As Long
  pszDisplayName As String
  lpszTitle As String
  ulFlags As Long
  lpfnCallback As Long
  lParam As Long
  iImage As Long
End Type

Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type
```

```
Private Type Size
  cx As Long
  cy As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
  Alias "SHBrowseForFolderA" ( _
  lpBrowseInfo As BROWSEINFO) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
  Alias "SHGetPathFromIDListA" ( _
  ByVal lPIDL As Long, _
  ByVal pszPath As String) As Long

Private Declare Sub CoTaskMemFree Lib "ole32.dll" ( _
  ByVal pv As Long)

Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" ( _
  ByVal hwnd As Long, _
  ByVal wMsg As Long, _
  ByVal wParam As Long, _
  lParam As Any) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
  Alias "RtlMoveMemory" ( _
  pDest As Any, _
  pSource As Any, _
  ByVal dwLength As Long)

Private Declare Function ILCreateFromPath Lib "shell32" _
  Alias "#157" ( _
  ByVal sPath As String) As Long

Private Declare Function LocalAlloc Lib "kernel32" ( _
  ByVal uFlags As Long, _
  ByVal uBytes As Long) As Long

Private Declare Function LocalFree Lib "kernel32" ( _
  ByVal hmem As Long) As Long

Private Declare Function lstrcpyA Lib "kernel32" ( _
```

```
  lpString1 As Any, _
  lpString2 As Any) As Long

Private Declare Function lstrlenA Lib "kernel32" ( _
  lpString As Any) As Long

Private Declare Function FindWindowEx Lib "user32.dll" _
  Alias "FindWindowExA" ( _
  ByVal hWnd1 As Long, _
  ByVal hWnd2 As Long, _
  ByVal lpsz1 As String, _
  ByVal lpsz2 As String) As Long

Private Declare Function GetWindowDC Lib "user32.dll" ( _
  ByVal hwnd As Long) As Long

Private Declare Function GetWindowRect Lib "user32.dll" ( _
  ByVal hwnd As Long, _
  ByRef lpRect As RECT) As Long

Private Declare Function GetTextExtentPoint Lib "gdi32.dll" _
  Alias "GetTextExtentPointA" ( _
  ByVal hDC As Long, _
  ByVal lpszString As String, _
  ByVal cbString As Long, _
  ByRef lpSize As Size) As Long

Private Declare Function PathCompactPath Lib "shlwapi.dll" _
  Alias "PathCompactPathA" ( _
  ByVal hDC As Long, _
  ByVal pszPath As String, _
  ByVal dx As Long) As Long

Private Const MAX_PATH = 260

Private Const WM_USER = &H400

Private Const BFFM_INITIALIZED = 1
Private Const BFFM_SELCHANGED As Long = 2
Private Const BFFM_SETSTATUSTEXTA As Long = (WM_USER + 100)
Private Const BFFM_SETSTATUSTEXTW As Long = (WM_USER + 104)
Private Const BFFM_ENABLEOK As Long = (WM_USER + 101)
```

```
Private Const BFFM_SETSELECTIONA As Long = (WM_USER + 102)
Private Const BFFM_SETSELECTIONW As Long = (WM_USER + 103)

Private Const BIF_NEWDIALOGSTYLE As Long = &H40
Private Const BIF_RETURNONLYFSDIRS As Long = &H1
Private Const BIF_BROWSEINCLUDEFILES As Long = &H4000
Private Const BIF_STATUSTEXT As Long = &H4

Private Const LMEM_FIXED = &H0
Private Const LMEM_ZEROINIT = &H40
Private Const LPTR = (LMEM_FIXED Or LMEM_ZEROINIT)

' Zeigt den BrowseForFolder-Dialog an.
Public Function BrowseForFolder(DialogText As String, _
  DefaultPath As String, _
  OwnerhWnd As Long, _
  Optional ShowCurrentPath As Boolean = True, _
  Optional RootPath As Variant, _
  Optional NewDialogStyle As Boolean = False, _
  Optional IncludeFiles As Boolean = False) As String

' Parameter:
' o DialogText Dialogtext, der oben im Dialog angezeigt wird.
' o DefaultPath Standardmäßig ausgewähltes Verzeichnis.
' o OwnerhWnd hWnd des übergeordneten Fensters (in den meisten
' Fällen Me.hWnd).
' o ShowCurrentPath Legt fest, ob die aktuelle Verzeichnisauswahl
' angezeigt werden soll. Verfügbar ab
' Internet Explorer 4.0 (-> PathCompactPath).
' o RootPath Root-Verzeichnis. Wird es angegeben, werden nur die
' Ordner unterhalb dieses Verzeichnisses angezeigt.
' o NewDialogStyle Legt fest, ob der Dialog in der neuen Darstellung
' angezeigt werden soll (Dialog kann vergrößert/
' verkleinert werden, es ist eine Schaltfläche zum
' Anlegen eines neuen Ordners vorhanden, es können
' Dateioperationen wie löschen etc. ausgeführt
' werden, ...). Ist dieser Parameter True, hat der
' Parameter ShowCurrentPath keine Wirkung. Verfügbar
' unter WinME und Betriebsystemen ab Win2000.
' o IncludeFiles Legt fest, ob auch Dateien im Dialog angezeigt und
' ausgewählt werden können.
' Verfügbar ab Win98 und Internet Explorer 4.0 (bei
```

```
' frühreren Windowsversionen muss IE4 inkl. der
' Integrated Shell installiert sein).

Dim biBrowseInfo As BROWSEINFO
Dim lPIDL As Long
Dim sBuffer As String
Dim lBufferPointer As Long

With biBrowseInfo
  ' Handle des übergeordneten Fensters
  .hOwner = OwnerhWnd

  ' PIDL des Rootordners zuweisen
  If Not IsMissing(RootPath) Then .pidlRoot = PathToPIDL(RootPath)

  ' Dialogtext zuweisen
  If ShowCurrentPath And DialogText = "$" Then DialogText = "" ' Wird intern nicht zugelassen
  .lpszTitle = DialogText

  ' Stringbuffer für aktuell selektierten Pfad zuweisen
  If ShowCurrentPath Then .pszDisplayName = sBuffer

  ' Dialogeinstellungen zuweisen
  .ulFlags = BIF_RETURNONLYFSDIRS + _
    IIf(ShowCurrentPath, BIF_STATUSTEXT, 0) + _
    IIf(NewDialogStyle, BIF_NEWDIALOGSTYLE, 0) + _
    IIf(IncludeFiles, BIF_BROWSEINCLUDEFILES, 0)

  ' Callbackfunktion-Adresse zuweisen
  .lpfnCallback = FARPROC(AddressOf CallbackString)

  ' PIDL des vorselektierten Ordnerpfades zuweisen (wird im
  ' lpData-Parameter an die Callback-Funktion weitergeleitet)
  .lParam = PathToPIDL(DefaultPath)
End With

' BrowseForFolder-Dialog anzeigen
lPIDL = SHBrowseForFolder(biBrowseInfo)

If lPIDL Then
  ' Stringspeicher reservieren
  sBuffer = Space$(MAX_PATH)
```

```
  ' Selektierten Pfad aus der zurückgegebenen PIDL ermitteln
  SHGetPathFromIDList lPIDL, sBuffer

  ' Nullterminierungszeichen des Strings entfernen
  sBuffer = Left$(sBuffer, InStr(sBuffer, vbNullChar) - 1)

  ' Selektierten Pfad zurückgeben
  BrowseForFolder = sBuffer

  ' Reservierten Task-Speicher wieder freigeben
  Call CoTaskMemFree(lPIDL)
End If

' Stringspeicher wieder freigeben
If ShowCurrentPath Then Call LocalFree(lBufferPointer)
End Function


Private Function CallbackString(ByVal hwnd As Long, ByVal uMsg As Long, _
  ByVal lParam As Long, ByVal lpData As Long) As Long

' Callback-Funktion des BrowseForFolder-Dialogs. Wird bei
' eintretenden Ereignissen des Dialogs aufgerufen.

Dim sBuffer As String
Dim lStaticWnd As Long
Dim lStaticDC As Long
Dim sPath As String
Dim rctStatic As RECT
Dim szTextSize As Size

' Meldungen herausfiltern
Select Case uMsg
  Case BFFM_INITIALIZED
    ' Dialog wurde initialisiert

    ' Standardmäßig markierten Pfad (dessen PIDL wurde in lpData
    ' übergeben) im Dialog selektieren
    Call SendMessage(hwnd, BFFM_SETSELECTIONA, False, ByVal lpData)
  Case BFFM_SELCHANGED
    ' Selektion hat sich geändert
```

```
    ' Stringspeicher reservieren
    sBuffer = Space$(MAX_PATH)

    ' Aktuell selektierten Pfad ermitteln und anzeigen, wenn möglich
    If SHGetPathFromIDList(lParam, sBuffer) Then
      ' Temporäre Zeichenfolge an das Anzeigelabel senden, um
      ' dessen Handle anhand dieser Zeichenfolge ermitteln zu können
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal "$"

      ' Handle und DeviceContext des Anzeigelabels ermitteln
      lStaticWnd = FindWindowEx(hwnd, ByVal 0&, ByVal "Static", ByVal "$")
      lStaticDC = GetWindowDC(lStaticWnd)

      ' Abmessungen des Anzeigelabels ermitteln
      GetWindowRect lStaticWnd, rctStatic

      ' Textabmessungen der Zeichenfolge "Auswahl: " im Anzeigelabel
      ' ermitteln
      GetTextExtentPoint lStaticDC, ByVal DIALOG_CURRENT_SELECTION_TEXT, _
        ByVal Len(DIALOG_CURRENT_SELECTION_TEXT), szTextSize

      ' Anzuzeigenden Pfad auf die Abmessungen des Anzeigelabels
      ' kürzen; falls dies nicht möglich ist, gesamten Pfad anzeigen
      sPath = sBuffer
      If PathCompactPath(ByVal lStaticDC, sPath, ByVal (rctStatic.Right - _
        rctStatic.Left - szTextSize.cx + 80)) = 0 Then sPath = sBuffer

      ' Nullterminierung entfernen
      sPath = Left$(sPath, InStr(1, sPath, vbNullChar) - 1)

      ' Pfad im Dialog anzeigen
      Call SendMessage(hwnd, BFFM_SETSTATUSTEXTA, 0&, _
        ByVal DIALOG_CURRENT_SELECTION_TEXT & sPath)
    Else
      ' Pfadanzeige leeren
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal ""
    End If
End Select
End Function
```

```
Private Function FARPROC(FunctionPointer As Long) As Long
' Funktion wird benötigt, um Funktions-Adresse ermitteln
' zu können, dessen Adresse mit AddressOf übergeben und
' anschließend wieder zurückgegeben wird.

FARPROC = FunctionPointer
End Function


' Gibt die lPIDL zum übergebenen Pfad zurück.
Private Function PathToPIDL(ByVal sPath As String) As Long
Dim lRet As Long

lRet = ILCreateFromPath(sPath)
If lRet = 0 Then
  sPath = StrConv(sPath, VbStrConv.vbUnicode)
  lRet = ILCreateFromPath(sPath)
End If

PathToPIDL = lRet
End Function



Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer
Dim Tabellenname As String


'strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

strPath = "C:\test\"

If strPath = "" Then Exit Sub

' On Error GoTo ErrExit

With Application
```

```
  '.ScreenUpdating = False
  .EnableEvents = False
  .DisplayAlerts = False
  .Calculation = xlCalculationManual
 ' .Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
  .NewSearch
  .LookIn = strPath
  .FileType = msoFileTypeExcelWorkbooks
  .SearchSubFolders = False

  If .Execute > 0 Then

    For intIndex = 1 To .FoundFiles.Count

      Set objWb = Workbooks.Open(.FoundFiles(intIndex))

      For Z = 1 To objWb.Sheets.Count

            objWb.Sheets(Z).Copy after:=objNew.Sheets(objNew.Sheets.Count)
            Tabellenname = objFO.GetBasename(.FoundFiles(intIndex)) & " - " & objWb.Sheets(Z).Name
            Tabellenname = Left(Tabellenname, 31)
           ' MsgBox Tabellenname
            objNew.Sheets(objNew.Sheets.Count).Name = Tabellenname


      Next Z

      objWb.Close False

      Set objWb = Nothing

    Next
```

```
  End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs "C:\" & "Zusammenfassung.xls"

ErrExit:
With Application
  .ScreenUpdating = True
  .EnableEvents = True
  .DisplayAlerts = True
  .Calculation = xlCalculationAutomatic
  .Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

Version 2

Ich habe ein verzeichnis mit Excel-Dateien. Ich würde nun gerne aus allen Dateien in diesem Verzeichnis eine einzige Arbeitsmappe machen, in der dann die einzelnen Dateien in den Arbeitsblättern erscheinen. Der Titel der Arbeitsblätter soll den gleichen haben wie die ursprüngliche Datei...

Ich bin mit VBA leider noch nicht sehr bewandert, will es aber lernen! Im moment habe ich nur leider keine Zeit dafür....

könnte mir von euch vilt jemand den code für ein makro geben oder so?

Kann mir jemand helfen? Wäre wirklich sehr nett!!

Viele Grüße

Daniele

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Josef Ehrensberger

Geschrieben am: 13.01.2006 16:05:52

Hallo Daniele!

Wieviele Blätter sind in den Dateien, bzw. welches Blatt soll in die Sammelmappe kommen?

```
'******************************
'* Gruß Sepp
'*
'* Rückmeldung wäre nett!
'******************************
```

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: HansH
Geschrieben am: 13.01.2006 16:53:15

Hallo Daniele,
bist Du sicher, dass das Sinn macht? Die Datei könnte zu groß werden. Wie wärs mit ner Alternative die jeweiligen Dateien aus einer Datei aufzurufen?
http://www.herber.de/bbs/user/30006.xls
Gruß
Hans

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Daniele
Geschrieben am: 17.01.2006 10:29:45

Hallo!

Sorry, bin erst heute wieder dazugekommen reinzuschauen.

Ja, da bin ich eigentlich relativ sicher. Hintergrund:

Ich habe ca 30 Dateien mit je einem sheet. Eine Tabelle, immer exakt das selbe Format, wenn ich jetzt alle Dateien in einer Mappe öffnen könnte markiere ich unten alle Blätter und kann so das Layout auf einmal ändern....

Verstanden?

Ich würde mich über weitere Antworten freuen...!

Grüße

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Josef Ehrensberger
Geschrieben am: 17.01.2006 12:28:04

Hallo Daniele!

Viel Spass!

```
' ********************************************************************
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

Option Explicit


Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer

On Error GoTo ErrExit

With Application
.ScreenUpdating = False
.EnableEvents = False
.DisplayAlerts = False
.Calculation = xlCalculationManual
.Cursor = xlWait
End With

strPath = "F:\Temp" 'Pfad - Anpassen!

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
.NewSearch
.LookIn = strPath
.FileType = msoFileTypeExcelWorkbooks
.SearchSubFolders = False

If .Execute > 0 Then

For intIndex = 1 To .FoundFiles.Count

Set objWb = Workbooks.Open(.FoundFiles(intIndex))

objWb.Sheets(1).Copy after:=objNew.Sheets(objNew.Sheets.Count)

objNew.Sheets(objNew.Sheets.Count).Name = objFO.GetBasename(.FoundFiles(intIndex))
```

```
objWb.Close False

Set objWb = Nothing

Next

End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs strPath & "Zusammenfassung.xls"

ErrExit:
With Application
.ScreenUpdating = True
.EnableEvents = True
.DisplayAlerts = True
.Calculation = xlCalculationAutomatic
.Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

```
'******************************
'* Gruß Sepp
'*
'* Rückmeldung wäre nett!
'******************************
```

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Daniele
Geschrieben am: 17.01.2006 15:01:43

Hallo!

Erstmals 100 DANK! Wirklich wahnsinn, das es sowas noch gibt, setzt du dich hin und schreibst mir so ein MAKRO! Super!

Nun, funktionieren tuts bei mir leider nicht...

Wie bin ich vorgegangen:

Code kopiert,
in VB Editor,
MODUL einfügen,
abspeichern,
Editor schließen,
Makro öffnen.

Was passiert?

Es öffnet sich eine weiter mappe, in dieser ist EIN Tabellenblatt enthalten mit dem Titel:
"Tabelle1"

mehr passiert leider nicht....

Gruß

**Ich benutze EXCEL 2002

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Josef Ehrensberger
Geschrieben am: 17.01.2006 15:09:27

Hallo Daniele!

Hast du den Pfad im Makro auch angepasst?

```
'*****************************
'* Gruß Sepp
'*
'* Rückmeldung wäre nett!
'*****************************
```

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Daniele
Geschrieben am: 17.01.2006 16:45:22

JA!!!!!!!!!!!!!!

ES GEHT!!!!!!!!!!!!!!!!!!!!!!!!!!

Eine kleine bitte noch:

Wäre es möglich das er anfangs nach dem Pfad fragt? Statt das mans im Code ändern muss??

Wäre super!!!

DANKE! und den besten Feierabend

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Josef Ehrensberger
Geschrieben am: 17.01.2006 16:58:56

Hallo Daniele!

Sicher geht das!

Pack alles in ein Modul!

```
' ********************************************************************
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

'BrowseForFolder mit Extra-Funktionen
'VB -Versionen: VB5 , VB6
'Betriebssystem: Win9x , WinNT, Win2000, WinME, WinXP
'Autor: Marco Wünschmann Homepage: ohne
'Datum: 23.08.2004

Option Explicit

' == Dialog-Einstellungen ================================

' String, der vor dem aktuell ausgewählen Verzeichnis angezeigt wird,
' falls der ShowCurrentPath-Paramter True ist.
Private Const DIALOG_CURRENT_SELECTION_TEXT As String = "Auswahl: "


' == API-Deklarationen ===================================

Private Type BROWSEINFO
hOwner As Long
pidlRoot As Long
pszDisplayName As String
lpszTitle As String
ulFlags As Long
lpfnCallback As Long
lParam As Long
iImage As Long
End Type

Private Type RECT
Left As Long
Top As Long
Right As Long
Bottom As Long
End Type

Private Type Size
cx As Long
cy As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
Alias "SHBrowseForFolderA" ( _
lpBrowseInfo As BROWSEINFO) As Long
```

```
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
Alias "SHGetPathFromIDListA" ( _
ByVal lPIDL As Long, _
ByVal pszPath As String) As Long

Private Declare Sub CoTaskMemFree Lib "ole32.dll" ( _
ByVal pv As Long)

Private Declare Function SendMessage Lib "user32" _
Alias "SendMessageA" ( _
ByVal hwnd As Long, _
ByVal wMsg As Long, _
ByVal wParam As Long, _
lParam As Any) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
Alias "RtlMoveMemory" ( _
pDest As Any, _
pSource As Any, _
ByVal dwLength As Long)

Private Declare Function ILCreateFromPath Lib "shell32" _
Alias "#157" ( _
ByVal sPath As String) As Long

Private Declare Function LocalAlloc Lib "kernel32" ( _
ByVal uFlags As Long, _
ByVal uBytes As Long) As Long

Private Declare Function LocalFree Lib "kernel32" ( _
ByVal hmem As Long) As Long

Private Declare Function lstrcpyA Lib "kernel32" ( _
lpString1 As Any, _
lpString2 As Any) As Long

Private Declare Function lstrlenA Lib "kernel32" ( _
lpString As Any) As Long

Private Declare Function FindWindowEx Lib "user32.dll" _
Alias "FindWindowExA" ( _
ByVal hWnd1 As Long, _
ByVal hWnd2 As Long, _
ByVal lpsz1 As String, _
ByVal lpsz2 As String) As Long
```

```
Private Declare Function GetWindowDC Lib "user32.dll" ( _
ByVal hwnd As Long) As Long

Private Declare Function GetWindowRect Lib "user32.dll" ( _
ByVal hwnd As Long, _
ByRef lpRect As RECT) As Long

Private Declare Function GetTextExtentPoint Lib "gdi32.dll" _
Alias "GetTextExtentPointA" ( _
ByVal hDC As Long, _
ByVal lpszString As String, _
ByVal cbString As Long, _
ByRef lpSize As Size) As Long

Private Declare Function PathCompactPath Lib "shlwapi.dll" _
Alias "PathCompactPathA" ( _
ByVal hDC As Long, _
ByVal pszPath As String, _
ByVal dx As Long) As Long

Private Const MAX_PATH = 260

Private Const WM_USER = &H400

Private Const BFFM_INITIALIZED = 1
Private Const BFFM_SELCHANGED As Long = 2
Private Const BFFM_SETSTATUSTEXTA As Long = (WM_USER + 100)
Private Const BFFM_SETSTATUSTEXTW As Long = (WM_USER + 104)
Private Const BFFM_ENABLEOK As Long = (WM_USER + 101)
Private Const BFFM_SETSELECTIONA As Long = (WM_USER + 102)
Private Const BFFM_SETSELECTIONW As Long = (WM_USER + 103)

Private Const BIF_NEWDIALOGSTYLE As Long = &H40
Private Const BIF_RETURNONLYFSDIRS As Long = &H1
Private Const BIF_BROWSEINCLUDEFILES As Long = &H4000
Private Const BIF_STATUSTEXT As Long = &H4

Private Const LMEM_FIXED = &H0
Private Const LMEM_ZEROINIT = &H40
Private Const LPTR = (LMEM_FIXED Or LMEM_ZEROINIT)

' Zeigt den BrowseForFolder-Dialog an.
Public Function BrowseForFolder(DialogText As String, _
DefaultPath As String, _
OwnerhWnd As Long, _
Optional ShowCurrentPath As Boolean = True, _
Optional RootPath As Variant, _
```

```
Optional NewDialogStyle As Boolean = False, _
Optional IncludeFiles As Boolean = False) As String

' Parameter:
' o DialogText Dialogtext, der oben im Dialog angezeigt wird.
' o DefaultPath Standardmäßig ausgewähltes Verzeichnis.
' o OwnerhWnd hWnd des übergeordneten Fensters (in den meisten
' Fällen Me.hWnd).
' o ShowCurrentPath Legt fest, ob die aktuelle Verzeichnisauswahl
' angezeigt werden soll. Verfügbar ab
' Internet Explorer 4.0 (-> PathCompactPath).
' o RootPath Root-Verzeichnis. Wird es angegeben, werden nur die
' Ordner unterhalb dieses Verzeichnisses angezeigt.
' o NewDialogStyle Legt fest, ob der Dialog in der neuen Darstellung
' angezeigt werden soll (Dialog kann vergrößert/
' verkleinert werden, es ist eine Schaltfläche zum
' Anlegen eines neuen Ordners vorhanden, es können
' Dateioperationen wie löschen etc. ausgeführt
' werden, ...). Ist dieser Parameter True, hat der
' Parameter ShowCurrentPath keine Wirkung. Verfügbar
' unter WinME und Betriebsystemen ab Win2000.
' o IncludeFiles Legt fest, ob auch Dateien im Dialog angezeigt und
' ausgewählt werden können.
' Verfügbar ab Win98 und Internet Explorer 4.0 (bei
' frühreren Windowsversionen muss IE4 inkl. der
' Integrated Shell installiert sein).

Dim biBrowseInfo As BROWSEINFO
Dim lPIDL As Long
Dim sBuffer As String
Dim lBufferPointer As Long

With biBrowseInfo
' Handle des übergeordneten Fensters
.hOwner = OwnerhWnd

' PIDL des Rootordners zuweisen
If Not IsMissing(RootPath) Then .pidlRoot = PathToPIDL(RootPath)

' Dialogtext zuweisen
If ShowCurrentPath And DialogText = "$" Then DialogText = "" ' Wird intern nicht zugelassen
.lpszTitle = DialogText

' Stringbuffer für aktuell selektierten Pfad zuweisen
If ShowCurrentPath Then .pszDisplayName = sBuffer

' Dialogeinstellungen zuweisen
```

```
.ulFlags = BIF_RETURNONLYFSDIRS + _
IIf(ShowCurrentPath, BIF_STATUSTEXT, 0) + _
IIf(NewDialogStyle, BIF_NEWDIALOGSTYLE, 0) + _
IIf(IncludeFiles, BIF_BROWSEINCLUDEFILES, 0)

' Callbackfunktion-Adresse zuweisen
.lpfnCallback = FARPROC(AddressOf CallbackString)

' PIDL des vorselektierten Ordnerpfades zuweisen (wird im
' lpData-Parameter an die Callback-Funktion weitergeleitet)
.lParam = PathToPIDL(DefaultPath)
End With

' BrowseForFolder-Dialog anzeigen
lPIDL = SHBrowseForFolder(biBrowseInfo)

If lPIDL Then
' Stringspeicher reservieren
sBuffer = Space$(MAX_PATH)

' Selektierten Pfad aus der zurückgegebenen PIDL ermitteln
SHGetPathFromIDList lPIDL, sBuffer

' Nullterminierungszeichen des Strings entfernen
sBuffer = Left$(sBuffer, InStr(sBuffer, vbNullChar) - 1)

' Selektierten Pfad zurückgeben
BrowseForFolder = sBuffer

' Reservierten Task-Speicher wieder freigeben
Call CoTaskMemFree(lPIDL)
End If

' Stringspeicher wieder freigeben
If ShowCurrentPath Then Call LocalFree(lBufferPointer)
End Function


Private Function CallbackString(ByVal hwnd As Long, ByVal uMsg As Long, _
ByVal lParam As Long, ByVal lpData As Long) As Long

' Callback-Funktion des BrowseForFolder-Dialogs. Wird bei
' eintretenden Ereignissen des Dialogs aufgerufen.

Dim sBuffer As String
Dim lStaticWnd As Long
Dim lStaticDC As Long
```

```
Dim sPath As String
Dim rctStatic As RECT
Dim szTextSize As Size

' Meldungen herausfiltern
Select Case uMsg
Case BFFM_INITIALIZED
' Dialog wurde initialisiert

' Standardmäßig markierten Pfad (dessen PIDL wurde in lpData
' übergeben) im Dialog selektieren
Call SendMessage(hwnd, BFFM_SETSELECTIONA, False, ByVal lpData)
Case BFFM_SELCHANGED
' Selektion hat sich geändert

' Stringspeicher reservieren
sBuffer = Space$(MAX_PATH)

' Aktuell selektierten Pfad ermitteln und anzeigen, wenn möglich
If SHGetPathFromIDList(lParam, sBuffer) Then
' Temporäre Zeichenfolge an das Anzeigelabel senden, um
' dessen Handle anhand dieser Zeichenfolge ermitteln zu können
SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal "$"

' Handle und DeviceContext des Anzeigelabels ermitteln
lStaticWnd = FindWindowEx(hwnd, ByVal 0&, ByVal "Static", ByVal "$")
lStaticDC = GetWindowDC(lStaticWnd)

' Abmessungen des Anzeigelabels ermitteln
GetWindowRect lStaticWnd, rctStatic

' Textabmessungen der Zeichenfolge "Auswahl: " im Anzeigelabel
' ermitteln
GetTextExtentPoint lStaticDC, ByVal DIALOG_CURRENT_SELECTION_TEXT, _
ByVal Len(DIALOG_CURRENT_SELECTION_TEXT), szTextSize

' Anzuzeigenden Pfad auf die Abmessungen des Anzeigelabels
' kürzen; falls dies nicht möglich ist, gesamten Pfad anzeigen
sPath = sBuffer
If PathCompactPath(ByVal lStaticDC, sPath, ByVal (rctStatic.Right - _
rctStatic.Left - szTextSize.cx + 80)) = 0 Then sPath = sBuffer

' Nullterminierung entfernen
sPath = Left$(sPath, InStr(1, sPath, vbNullChar) - 1)

' Pfad im Dialog anzeigen
Call SendMessage(hwnd, BFFM_SETSTATUSTEXTA, 0&, _
```

```
ByVal DIALOG_CURRENT_SELECTION_TEXT & sPath)
Else
' Pfadanzeige leeren
SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal ""
End If
End Select
End Function


Private Function FARPROC(FunctionPointer As Long) As Long
' Funktion wird benötigt, um Funktions-Adresse ermitteln
' zu können, dessen Adresse mit AddressOf übergeben und
' anschließend wieder zurückgegeben wird.

FARPROC = FunctionPointer
End Function


' Gibt die lPIDL zum übergebenen Pfad zurück.
Private Function PathToPIDL(ByVal sPath As String) As Long
Dim lRet As Long

lRet = ILCreateFromPath(sPath)
If lRet = 0 Then
sPath = StrConv(sPath, VbStrConv.vbUnicode)
lRet = ILCreateFromPath(sPath)
End If

PathToPIDL = lRet
End Function



Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer

On Error GoTo ErrExit

With Application
.ScreenUpdating = False
.EnableEvents = False
.DisplayAlerts = False
.Calculation = xlCalculationManual
```

```
.Cursor = xlWait
End With

strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , False, False)

If strPath = "" Then GoTo ErrExit

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
.NewSearch
.LookIn = strPath
.FileType = msoFileTypeExcelWorkbooks
.SearchSubFolders = False

If .Execute > 0 Then

For intIndex = 1 To .FoundFiles.Count

Set objWb = Workbooks.Open(.FoundFiles(intIndex))

objWb.Sheets(1).Copy after:=objNew.Sheets(objNew.Sheets.Count)

objNew.Sheets(objNew.Sheets.Count).Name = objFO.GetBasename(.FoundFiles(intIndex))

objWb.Close False

Set objWb = Nothing

Next

End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs strPath & "Zusammenfassung.xls"

ErrExit:
With Application
.ScreenUpdating = True
.EnableEvents = True
.DisplayAlerts = True
```

```
.Calculation = xlCalculationAutomatic
.Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

```
'******************************
'* Gruß Sepp
'*
'* Rückmeldung wäre nett!
'******************************
```

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Daniele
Geschrieben am: 17.01.2006 17:18:19

Wirklich super!!!

Genau wie ich mir das vorgestellt habe!

Perfekt!

GEIL!

Hiermit werden mir bestimmt 2 Tage Arbeit erspart!

Ein kleiner Bug (der mich persönlich nicht stört, nur zur info)::

Das Fenster das sich öffnet, in welchem man das Verzeichnis angibt, wenn man dieses in die Mitte vom Bildschirm ziehen will, also allgemein das ganze Fenster bewegt, zieht es Schlieren.....So als wäre die Grafikkarte schrecklich langsam...

Keine Ahnung woran das liegt!

Nochmals 1000Dank!

Ciao!

Betrifft: AW: Viele Excel Dateien zu einer zusammenführen
von: Josef Ehrensberger
Geschrieben am: 17.01.2006 17:22:10

Hallo Daniele!

So ist's besser!

```
Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer


strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

If strPath = "" Then Exit Sub

On Error GoTo ErrExit

With Application
.ScreenUpdating = False
.EnableEvents = False
.DisplayAlerts = False
.Calculation = xlCalculationManual
.Cursor = xlWait
End With

If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

Set objFS = Application.FileSearch
Set objFO = CreateObject("Scripting.FileSystemObject")
Set objNew = Workbooks.Add(xlWBATWorksheet)

With objFS
.NewSearch
.LookIn = strPath
.FileType = msoFileTypeExcelWorkbooks
.SearchSubFolders = False

If .Execute > 0 Then

For intIndex = 1 To .FoundFiles.Count

Set objWb = Workbooks.Open(.FoundFiles(intIndex))

objWb.Sheets(1).Copy after:=objNew.Sheets(objNew.Sheets.Count)

objNew.Sheets(objNew.Sheets.Count).Name = objFO.GetBasename(.FoundFiles(intIndex))

objWb.Close False

Set objWb = Nothing
```

```
Next

End If

End With

objNew.Sheets(1).Delete
objNew.SaveAs strPath & "Zusammenfassung.xls"

ErrExit:
With Application
.ScreenUpdating = True
.EnableEvents = True
.DisplayAlerts = True
.Calculation = xlCalculationAutomatic
.Cursor = xlDefault
End With

Set objNew = Nothing
Set objFS = Nothing
Set objFO = Nothing

End Sub
```

Den rest des Codes unverändert lassen!

Andere Exceldatei öffnen

Mit einem Absolutpfad geht es über folgende zwei Zeilen:

ChDir "C:\OPLeXor\Schnittstelle"
Workbooks.Open Filename:="C:\OPLeXor\Schnittstelle\BMD_OP_LEXOR.xls"

Möchte man den Pfad der aktuellen Tabelle verwenden und relativ von ihr auszugreifen

Andere Excel-Datei aus voreingestelltem Pfad auswählen, öffnen, auslesen

die im nachfolgenden Unterkapitel " Andere Excelmappe wählen, öffnen, auslesen, schließen " beschriebene Variante über Application.GetOpenFilename hat den großen Nachteil, dass sie nicht erlaubt zuvor ein fixes Verzeichnis zu setzen, das beim anschließenden Dateiauswahl-Dialog vorausgewählt ist. (Dafür ist jene

Version noch Excel 2000-täuglich - die folgende braucht zumindest Excel XP)

Großer Vorteil dieser Version: wird beim Öffnen der Datei ein Passwort benötigt, so bricht der Code nicht ab, sondern fragt nach dem Passwort !

MEINE VERSION VON DER ZEITERFASSUNGSSOFTWARE

```
Dim ALT as string ' Alte Datei

    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            ALT = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With

Workbooks.Open ALT ' die Variable Alt enthält noch den gesamten Dateipfad und den Dateinamen !

' Reduzieren der Variable Alt auf den Dateinamen (ohne Pfad)
ALT = ActiveWorkbook.Name

MsgBox Workbooks(ALT).Sheets("Parameter").Range("IV3")
```

ALTE VERSION

```
' ALLGEMEINER CODE, UM NUR EINE DATEI AUSZUWÄHLEN UND ANZUZEIGEN

Public Sub Datei_Auswahl()
    Dim strDatei As String
    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
```

```
        If .Show = -1 Then
            strDatei = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With
    MsgBox strDatei
End Sub
```

Andere Excelmappe wählen, öffnen, auslesen, schließen

Wichtig: leider kann man bei dem nachfolgenden GETOPENFILENAME nicht vorher mit CHDrive und CHDir ein fixes Verzeichnis voreinstellen - etwa das der aktuellen Exceldatei. In diesem Fall bitte besser mit dem Application.Filedialog arbeiten aus dem vorigen Eintrag hier "ANDERE EXCEL-DATEI AUS VOREINGESTELLTEM PFAD AUSWÄHLEN, ÖFFNEN, AUSLESEN"

Der Vorteil dieser Version ist eigentlich nur, dass er auch für Excelversionen vor XP funktioniert

```
DATEI = Application.GetOpenFilename("EXCELDATEI (*.xls), *.xls")
DATEI = Right(DATEI, Len(DATEI) - Len(CurDir) - 1) ' wegschneiden Pfad
Workbooks.Open DATEI ' Öffnen der Datei (muss im selben Ordner sein)
'... irgendein Code - zB
Workbooks(DATEI).Sheets("UVA").Range("B5:M7").Copy
'...
Workbooks(Datei).Close SaveChanges:=False ' Schließt die Datei wieder
```

Anzahl Dateien in einem Verzeichnis bestimmen

```
Sub Datei_Anzahl()
'Long abhängig von der zu erwartenden Dateimenge,
'... Integer würde wahrscheinlich auch reichen :-))
Dim i As Long
'Variable für das Verzeichnis erstellen
Dim SuchVZ As String
SuchVZ = InputBox("Bitte Verzeichnisnamen angeben", "Dateisuche", "C:\Dein Verzeichnis\")
If SuchVZ = "" Then Exit Sub
'Fehlerroutine einschalten
On Error GoTo Error_SuchVZ
'Laufwerk wechseln
ChDrive Left(SuchVZ, 2)
'In das Suchverzeichnis wechseln
ChDir SuchVZ
With Application.FileSearch
   .NewSearch
   .LookIn = SuchVZ
   'Unterzeichnisse mit einbeziehen
   .SearchSubFolders = False '=True
   'Dateitypen angeben
   .FileType = msoFileTypeExcelWorkbooks
   'Suche ausführen
   .Execute
   NrCount = .FoundFiles.Count
End With
MsgBox NrCount & " Datei(en) gefunden"
Exit Sub

Error_SuchVZ:
MsgBox "Das Verzeichnis: " & SuchVZ & " konnte nicht gefunden werden! "
End Sub
```

Arbeitsmappe schließen ohne Änderungen speichern (siehe auch ARBEITSMAPPEN)

```
ActiveWorkbook.Close False
oder
ActiveWorkbook.Close savechanges:=False
```

Der Befehl FALSE deaktiviert die Speichern-Frage
Ohne False würde Excel im Bedarfsfall (wenn Änderungen waren) nachfragen, ob es speichern soll.

Auslesen Mappenschutz

Msgbox Activeworkbook.Readonly

Backup-Datei automatisch erstellen und ältere Backups automatisch löschen

Siehe hier Eintrag DATEIEN IM BACKUPORDNER LÖSCHEN, DIE ÄLTER ALS 10 TAGE SIND

--- DATEI-IMPORTE UND EXPORTE (CSV / TXT) ---

° CSV-Export-SCHNELL

--- CODE ---

VERSION 1 von FLEXIMPORT

```
Sub DIREKT_EXPORT_EXPORT()

    ' ------------------------------------------------------------
    ' DIES IST DIE SCHNELLE VARIANTE FÜR DEN EXPORT als CSV
    ' man muss nur sicherstellen, dass nach dem Export in der CSV-Datei das Trennzeichen richtig ist, sonst muss es in
den Windowseinstellungen
    ' REGION/SPRACHE - FORMATE - WEITERE EINSTELLUNGEN - ZAHLEN - LISTENTRENNZEICHEN richtig auf ; eingestellt werden
    ' ------------------------------------------------------------

    ' --- DATEIFORMAT ---
    ' Beim Export stehen im Wesentlichen 3 Formate zur Auswahl, die im SaveAs-Befehl gleich anschließend nach dem
Dateinamen als Flag kommt
    ' 1.) xlCSV => *.csv-Datei mit Listentrennzeichen (; bei local:=True oder , wenn ohne local:=True)
    ' 2.) xlCurrentPlatformText => *.txt-Datei mit Tabulatortrennzeichen
    ' 3.) xlUnicodeText => speichert eine Datei als Unicodetext (Tabulatortrennzeichen) aber ohne Dateiendung (also kein
.txt)
```

```
    ' --- DATEIENDUNG ---
    ' Die Dateiendung kann, aber muss nicht im SaveAs-Befehler angegeben werden => xlCSV ergänzt automatisch *.csv und
xlCurrentPlatformText *.txt
    ' man kann aber freiwillig die Dateiendung übergeben und dann ist sogar folgender Sonderfall möglich: Speichern mit
xlCSV und Dateinamen "Test.txt"
    ' => speichert eine CSV-Datei als txt-Datei aber mit normalem CSV-Trennzeichen und nicht mit TXT-Standardtrennzeichen
Tabulator

        Dim CSV_EXPORTDATEI_NAMEN As String
        Dim DATEIPFAD As String
        Dim DATUMSTEMPEL As String

        DATEIPFAD = Replace(P.Range("E402"), "/", "\")        ' Dateipfad auslesen und / ersetzen
        If DATEIPFAD = "" Then DATEIPFAD = ActiveWorkbook.Path ' wenn Dateipfad leer ist, nimm den Pfad der Vorlage
        If Right(DATEIPFAD, 1) <> "\" Then DATEIPFAD = DATEIPFAD & "\" ' wenn ein abschließendes \ fehlt, ergänze es
        If Mid(DATEIPFAD, 2, 1) <> ":" Then DATEIPFAD = ActiveWorkbook.Path & "\" & DATEIPFAD ' wenn der Pfad nur Ordner
(und ev. Unterordner) enthält, ergänze den Pfad der Vorlage
        ' Anlegen des Unterordners, falls es ihn noch nicht gibt
        If Dir(DATEIPFAD, vbDirectory) = "" Then
            On Error Resume Next
            MkDir (DATEIPFAD)
            On Error GoTo 0
        End If
        ' Soll Datumstempel auch einen bestimmten Namen haben
        If P.Range("E403") <> "" Then
            DATUMSTEMPEL = P.Range("E403")
        End If
        ' Definieren des Datums- und Zeitstempels
        If P.Range("I404") = "JA" Then
            If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
            If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
            If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
            If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
            If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)),
2)
            DATUMSTEMPEL = DATUMSTEMPEL & " " & Year(Now) & Monat & Tag & "_" & Stunden & Minuten & Sekunden
        End If

        ' Dateiname
        CSV_EXPORTDATEI_NAMEN = DATEIPFAD & DATUMSTEMPEL & "." & P.Range("E405")
```

```
        If Right(DATEINAME, 1) = "." Then DATEINAME = Left(DATEINAME, Len(DATEINAME) - 1) ' wenn keine Dateiendung,
entferne den Punkt

      ' Korrektur auf ASP-Systemen

        If InStr(CSV_EXPORTDATEI_NAMEN, "\\\") > 0 Then
            CSV_EXPORTDATEI_NAMEN = Mid(CSV_EXPORTDATEI_NAMEN, InStr(CSV_EXPORTDATEI_NAMEN, "\\\") + 1, 255)
        End If

        ' Aktuelle Tabelle in neue Interims-Mappe kopieren
        ActiveSheet.Copy ' oder: Workbooks.Add xlWBATWorksheet

        '    ' nachfolgendes Kopieren und Einfügen der Werte ist in der Regel NICHT nötig
        '    ActiveWorkbook.ActiveSheet.UsedRange.Copy: ActiveSheet.Cells(1, 1).PasteSpecial xlPasteValues
        '
        '    Reduzieren auf erwünschten Bereich durch Löschen von ev. nicht benötigen Zeilen/Spalten
        Rows("1:9").Delete
        Rows(Range("A65536").End(xlUp).Row + 1 & ":200000").Delete ' Unten alles Wegschneiden, was ";;;;;;;;"-Datenzeilen
ergeben könnte
        '    Columns("D:E").Delete

        ' Kontrolle der letzten Zelle mit Daten und darunter alle Zeilen löschen

        ' Überschreiben von ev. bestehender CSV-Datei soll ohne Nachfragen erfolgen
        Application.DisplayAlerts = False

        ' Speichern als CSV-Datei
        ActiveWorkbook.SaveAs CSV_EXPORTDATEI_NAMEN, xlCSV, local:=True ' Local:=True sorgt für deutsches
Listentrennzeichen ;

        ' Interims-Mappe schließen
        ActiveWorkbook.Close False

        ' Warnmeldungen wieder normal anzeigen
        Application.DisplayAlerts = True

        MsgBox "Die NTCS-Buchungen wurden expoprtiert nach " & CSV_EXPORTDATEI_NAMEN & " ."

End Sub
```

VERSION 2 (KURZ)

```
Sub CSV_EXPORT ()

    ' --- DATEIFORMAT ---
    ' Beim Export stehen im Wesentlichen 3 Formate zur Auswahl, die im SaveAs-Befehl gleich anschließend nach dem
Dateinamen als Flag kommt
    ' 1.) xlCSV => *.csv-Datei mit Listentrennzeichen (; bei local:=True oder , wenn ohne local:=True)
    ' 2.) xlCurrentPlatformText => *.txt-Datei mit Tabulatortrennzeichen
    ' 3.) xlUnicodeText => speichert eine Datei als Unicodetext (Tabulatortrennzeichen) aber ohne Dateiendung (also kein
.txt)
    ' --- DATEIENDUNG ---
    ' Die Dateiendung kann, aber muss nicht im SaveAs-Befehler angegeben werden => xlCSV ergänzt automatisch *.csv und
xlCurrentPlatformText *.txt
    ' man kann aber freiwillig die Dateiendung übergeben und dann ist sogar folgender Sonderfall möglich: Speichern mit
xlCSV und Dateinamen "Test.txt"
    ' => speichert eine CSV-Datei als txt-Datei aber mit normalem CSV-Trennzeichen und nicht mit TXT-Standardtrennzeichen
Tabulator

    Dim CSV_EXPORTDATEI_NAMEN As String

    ' Aktuelle Tabelle in neue Interims-Mappe kopieren
    ActiveSheet.Copy ' oder: Workbooks.Add xlWBATWorksheet

'     ' nachfolgendes Kopieren und Einfügen der Werte ist in der Regel NICHT nötig
'     ActiveWorkbook.ActiveSheet.UsedRange.Copy: ActiveSheet.Cells(1, 1).PasteSpecial xlPasteValues
'
'     ' auch nachfolgendes Reduzieren auf erwünschten Bereich durch Löschen von ev. nicht benötigen Zeilen/Spalten ist
NICHT nötig
'     Rows("5:6").Delete
'     Columns("D:E").Delete

    ' Überschreiben von ev. bestehender CSV-Datei soll ohne Nachfragen erfolgen
    Application.DisplayAlerts = False

      ' Speichername der Datei
    CSV_EXPORTDATEI_NAMEN = ThisWorkbook.Path & "\" & T1.Range("O7") & " " & Left(Now, 10) & ".csv"

' Korrektur auf ASP-Systemen

If Instr(CSV_EXPORTDATEI_NAMEN,"\\\")>0 then
      CSV_EXPORTDATEI_NAMEN = Mid(CSV_EXPORTDATEI_NAMEN, InStr(CSV_EXPORTDATEI_NAMEN, "\\\") + 1, 255)
End if
```

```
    ' Speichern als CSV-Datei
    ActiveWorkbook.SaveAs CSV_EXPORTDATEI_NAMEN, xlCSV, local:=True ' Local:=True sorgt für deutsches Listentrennzeichen
;

    ' Interims-Mappe schließen
    ActiveWorkbook.Close False

    ' Warnmeldungen wieder normal anzeigen
    Application.DisplayAlerts = True

End Sub
```

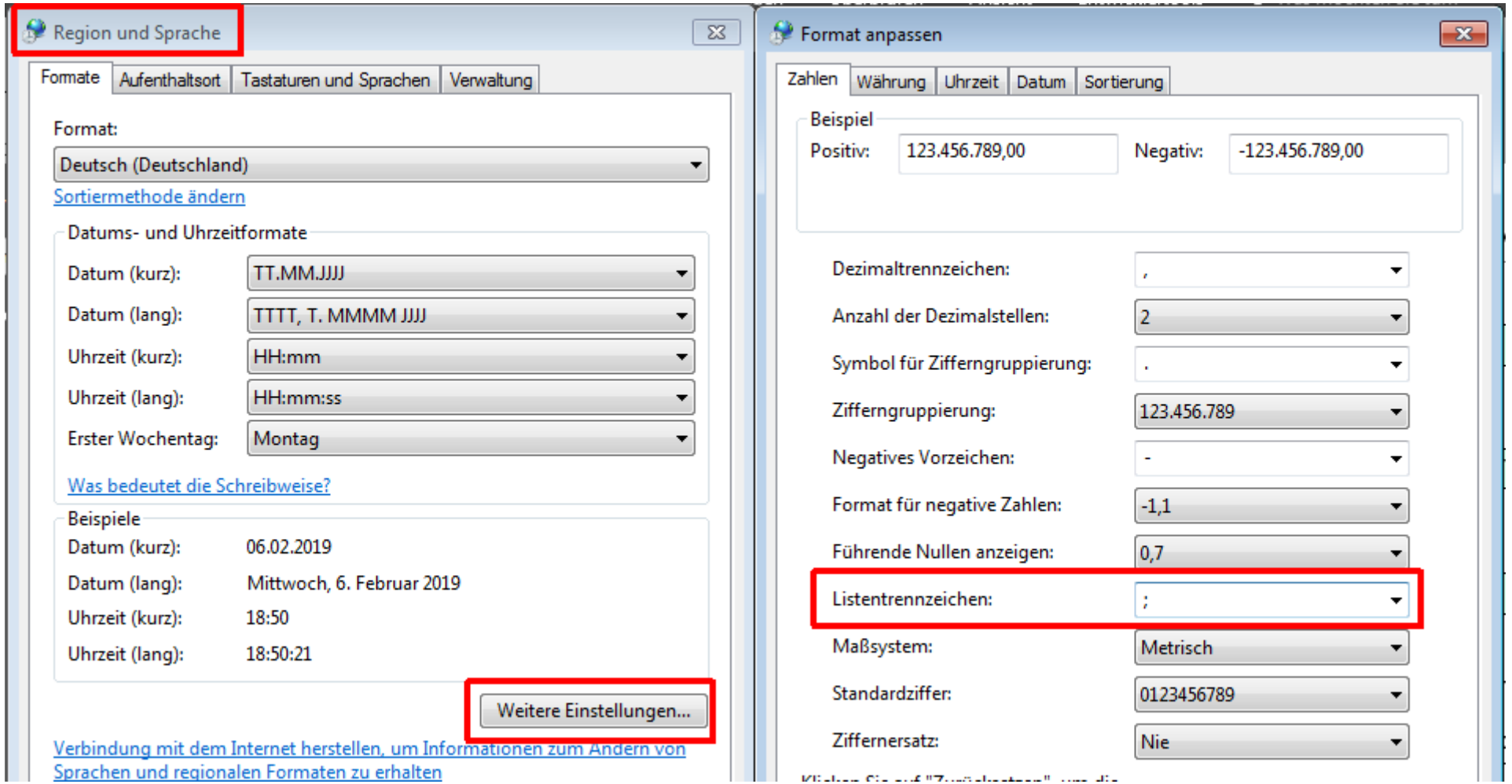
--- THEORIE ---

Mein anschließender beim nächsten Eintrag angeführter, langer Standardcode hat den Vorteil, dass er das Trennzeichen aktiv setzt und er 1000 Zeilen von der Zeit her problemslos und schnell schafft - aber bei 300.000 Zeilen 5 Minuten braucht.

Beim nachfolgenden SCHNELLEN CSV-Code geht das Speichern viel schneller - aber es ist nicht immer ganz einfach als Trennzeichen den STRICHPUNKT (Semikolon) und nicht den BEISTRICH (KOMMA) zu erhalten, weil hier das wahlweise das englische VBA-Standardtrennzeichen (,) greift oder das im Betriebssystem definierte Trennzeichen greift.

Der normale SAVEAS-Befehl speichert mit dem englischen VBA-eigenen "," und nicht mit dem gewünschten ";". Man kann aber dem SAVEAS das Flag "local:=True" übergeben, damit das lokale Listentrennzeichen aus den SPRACHEINSTELLUNGEN von Windows greift.

Dieses wird hier eingestellt …



… und kann mit folgendem Befehl abgefragt werden:

```
MsgBox Application.International(xlListSeparator)
```

Ich verwende bevorzugt einen Code, der den gewünschten Inhalt in eine neue Mappe kopiert. Das hat folgende Vorteile:

- speichert man eine Mappe direkt als CSV, dann ist das das letzte Speicherformat und mit normalen Speichern durch den User wird nicht die ursprüngliche Mappe gespeichert, sondern nur noch die aktuelle Tabelle und das als CSV

- ich kann dann auch genau den gewünschten Bereich wählen - der normale CSV-Speichernvorgang exportiert ja immer alles

° CSV-Export-Code (Mein Standardexport)

```
Sub CSV_EXPORT()

' Dies ist der Code vom Button 'CSV EXPORT'
' er exportiert vom aktuellen Tabellenblatt alles im wählbaren Bereich

    Dim DATEI As String
    Dim Z As Long, S As Long
    Dim STARTZEILE As Long, STARTSPALTENNUMMER As Long
    Dim ENDZEILE As Long, ENDSPALTENNUMMER As Long
    Dim ZELLINHALT As String
    Dim GESAMTEXT As String
    Dim TRENNZEICHEN
    Dim TEXTBEGRENZER As Boolean

    ' ------------------------------
    ' ---    PARAMETER SETZEN    ---
    ' ------------------------------

    ' Trennzeichen
    TRENNZEICHEN = P.Range("U35")  ' Alternativ vbTab
    ' Erste Export-ZEILE
    STARTZEILE = 10
    ' Erste Export-SPALTE
    STARTSPALTENNUMMER = 1
    ' Letzte Export-ZEILE
    ENDZEILE = LETZTEZELLE(ActiveSheet.Name).Row  ' Alternativ: eine bestimmte Zeilennummer eingeben
    ' Letzte Export-SPALTE
    ENDSPALTENNUMMER = 17 ' LETZTEZELLE(ActiveSheet.Name).Column oder Alternativ: eine bestimmte Spaltennummer eingeben
    ' als Text formatierte Zellen mit " " ausgeben
    ' ACHTUNG: falls aktiviert im Falle von NTCS-Buchungen die 1. Zeile mit Spalten-Definition auf Zellformat STANDARD
setzen, weil NTCS zB die Definition "Buchsymbol" nicht kennt und nur Buchsymbol braucht
    TEXTBEGRENZER = False ' o. True - bestimmt, ob beim Export " verwendet werden sollen
    ' Speichername der Datei
```

```
    DATEI = ThisWorkbook.Path & "\NTCS-Fuhrpark " & P.Range("O37") & " " & Left(Now, 10) & "." & P.Range("O35")

    ' ------------------------------
    ' ---    EXPORT DER DATEN    ---
    ' ------------------------------

    ' Warnungen übergehen (ev. Überschreiben einer existierende Datei)

    Application.DisplayAlerts = False

' Korrektur, falls ASP-System
If InStr(DATEI, "\\\") > 0 Then
    DATEI = Mid(DATEI, InStr(DATEI, "\\\") + 1, 255)
End If


' Öffnen CSV-Datei

    On Error Resume Next
    Close #1
    On Error GoTo 0

    Open DATEI For Output As #1

    ' Export

    For Z = STARTZEILE To ENDZEILE
        For S = STARTSPALTENNUMMER To ENDSPALTENNUMMER
            ' Auslesen Zellinhalt als Text
            ZELLINHALT = CStr(Cells(Z, S).Text)
            ' Ersetzen des Trennzeichens, falls es in der Zelle vorkommt
            ZELLINHALT = Replace(ZELLINHALT, TRENNZEICHEN, "_")
            ' Zellen im Textformat werden mit Textbegrenzer " " übergeben
            If TEXTBEGRENZER = True And Cells(Z, S).NumberFormat = "@" Then
                GESAMTEXT = GESAMTEXT & """" & ZELLINHALT & """" & TRENNZEICHEN
            Else
                GESAMTEXT = GESAMTEXT & ZELLINHALT & TRENNZEICHEN
            End If
        Next
        ' Entfernen des abschließenden Trennzeichens
        If Right(GESAMTEXT, 1) = TRENNZEICHEN Then GESAMTEXT = Left(GESAMTEXT, Len(GESAMTEXT) - 1)
        Print #1, GESAMTEXT
```

```
        GESAMTEXT = ""
    Next

    Close #1

' Warnungen wieder aktivieren
    Application.DisplayAlerts = True
    MsgBox "Die Daten wurden exportiert nach " & DATEI

End Sub

Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' ermittelt die letzte Zelle einer Tabelle mit Inhalt oder Zellformat

  Dim ExcelLastCell As Range
  Dim Row As Long
  Dim Col As Long
  Dim LastRowWithData As Long
  Dim LastColWithData As Long
  Dim TheSheet As Worksheet
  Dim MERKER

  Set TheSheet = Worksheets(TABELLENBLATT)

  MERKER = Application.ScreenUpdating
  Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
   Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

  Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

  ' letzte Zeile mit Daten herausfinden
  LastRowWithData = ExcelLastCell.Row
  Row = ExcelLastCell.Row
  Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
    Row = Row - 1
  Loop
  LastRowWithData = Row

  ' letzte Spalte mit Daten herausfinden
  LastColWithData = ExcelLastCell.Column
  Col = ExcelLastCell.Column
```

```
  Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
    Col = Col - 1
  Loop
  LastColWithData = Col

  Set LETZTEZELLE = TheSheet.Cells(Row, Col)
  Application.ScreenUpdating = MERKER

End Function
```

CSV-Datei importieren mit Trennzeichen in ""-Textfeldern

Bisweilen kommt das Trennzeichen (;) in TextFeldern vor und dürfen dort nicht als Trennzeichen interpretiert werden
z.B. das ; nach Susi: 1.1.18;abc;"Susi;Heinz";123

die normalen CSV-Importe von mir unterscheiden da nicht - mein nachfolgend komplizierter Code importiert zuerst die ganze Zeile und fasst dann Text zwischen den Textbegrenzungszeichen zusammen:

```
Open DATEI For Input As #1
    Do Until EOF(1)
      Line Input #1, iText
      ' Wenn zwischen zwei Anführungszeichen das Trennzeichen , vorkommt, obwohl es an dieser Stelle kein Datensatz-Trennzeichen
      ' sondern ein Beistrich zB im Firmennamen ist, dann soll dieser Beistrich ja nicht als Trennzeichen gelten.
      ' BSP 1,10,"Creative, Media Gmbh",4200,50 - alle Beistriche sind ein Trennzeichen, außer der Bestrich zwischen den beiden ".
      ' Lösung: wir wandeln zuerst alle Trennzeichen in ein neues Trennzeichen um, außer das Trennzeichen kommt zwischen zwei " vor
      For i = 1 To Len(iText)
        If Mid(iText, i, 1) = """ And WORTMODUS = 0 Then
          WORTMODUS = 1 ' beim ersten " wird der Wortmodus auf 1 gestellt - daher alle nachfolgenden Trennzeichen bleiben ein Satzzeichen
        Else
          WORTMODUS = 0 ' beim zweiten " wird der Wortmodus auf 0 zurückgesetzt - daher alle nachfolgenden Trennzeichen sind ein echtes Trennzeichen und werden umgewandelt in neues Trennzeichen
        End If
        If Mid(iText, i, 1) = TRENNZEICHEN Then
          If WORTMODUS = 1 Then
            iTextNeu = iTextNeu & Mid(iText, i, 1) ' wenn Trennzeichen zwischen 2 " vorkommt, es 1:1 übernehmen
          Else
            iTextNeu = iTextNeu & TRENNZEICHENNEU
          End If
        Else ' wenn kein Trennzeichen: es normal übernehmen
          iTextNeu = iTextNeu & Mid(iText, i, 1)
        End If
      Next i
      iText = iTextNeu
      iTextNeu = ""
      Do While InStr(iText, TRENNZEICHENNEU)
        Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHENNEU) - 1) ' Einfügen des jeweiligen Teils zwischen zwei Trennzeichen
        iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHENNEU))
        iSpalte = iSpalte + 1
      Loop
      Cells(iReihe, iSpalte).Value = iText ' Einfügen der letzten Zelle
      iReihe = iReihe + 1
      iSpalte = 1
    Loop
  Close
```

Wesentlich eleganter geht es mit folgenden 2 Varianten:

als 1b gibt es hier ein sehr gutes, fertiges Importmodul !

1. QUERYTABLES.ADD

Der "`QueryTables.Add(Connection:="Text;"`"-Teil ist immer fix und bedeutet, dass wir eine Text-/CSV-Datei importieren

```
Sub TESTIMPORT()

    ' Variablen für Trennzeichen
    Dim VTAB As Boolean ' Tabulator
    Dim VSEMIKOLON As Boolean ' ;
    Dim VKOMMA As Boolean ' ,
    Dim VLEER As Boolean ' Leerzeichen
```

```
    Dim VERBINDUNG  ' Verbindung zur Datenabfrage

    ' Da alle Trennzeichenvaribalen per Default false sind, brauchen wir nur die gewünschte auf True setzen
    VSEMIKOLON = True

    With ActiveSheet.QueryTables.Add(Connection:="Text;" & "E:\Test.txt", Destination:=Range("A1"))
        .Name = "*"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = False
        .RefreshPeriod = 0
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = xlWindows
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited ' Wichtig, dass es überhaupt Trennzeichen-Logik gibt
        .TextFileTextQualifier = xlTextQualifierDoubleQuote ' Wichtig, dass mit ""-geschützte Felder das Trennzeichen
enthalten dürfen
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimiter = VTAB
        .TextFileSemicolonDelimiter = VSEMIKOLON
        .TextFileCommaDelimiter = VCOMMA
        .TextFileSpaceDelimiter = VLEER
        .TextFileColumnDataTypes = Array(1, 1, 1, 1, 1, 1, 1, 1) ' Definiert für jede Spalte ihr Format: 1=Allgemein,
2=Text, 4 Datum, 9 Spalte wird nicht analysiert
        .Refresh BackgroundQuery:=False
    End With

    ' Da mit dem Namen der Datei eine Verbindung (siehe Excelregister DATEN/VERBINDUNGEN) aufgebaut und gleich benannt
wird (ohne Dateiendung) will man diese ev. entfernen
    ' ActiveWorkbook.Connections("Test").Delete

    ' Alternativ einfach alle Verbindungen entfernen
    For Each VERBINDUNG In ActiveWorkbook.Connections
        VERBINDUNG.Delete
    Next
```

```
    ' Alternativ alle Querys entfernen
    For Each VERBINDUNG In ActiveSheet.QueryTables
        VERBINDUNG.Delete
    Next

End Sub
```

1.b Fertiges Importmodul von Fleximport

```
Sub IMPORT_DATEIAUSWAHL()

' Er liest eine TXT oder CSV-Datei ein,

    ' Variablen für Trennzeichen
    Dim TRENNZEICHEN As String ' das in einer Parametertabelle hinterlegte Trennzeichen
    Dim VTAB As Boolean ' Tabulator
    Dim VSEMIKOLON As Boolean ' ;
    Dim VKOMMA As Boolean ' ,
    Dim VLEER As Boolean ' Leerzeichen

   Dim DATEIPFAD As String ' Der Ordner, in dem sich die einzulesende Datei befindet mit abschließendem \
   Dim DATEI As String
   Dim DATEINAME As String ' der reine Dateiname der Importdatei, wichtig für automatisches Verschieben der Datei nach
Import
   Dim DATEIPFADLAGER As String ' Pfad und Ordner fürs Verschieben der importierten Daten

   Dim LETZTEZEILE

   ' Variablen für Datumsstempel
    Dim Monat As String     ' Aktuelles Monat mit führender Null, wenn <10
    Dim Tag As String       ' Aktueller Tag mit führender Null, wenn <10
    Dim Stunden As String    ' Aktuelle Stunde mit führender Null
    Dim Minuten As String    ' Aktuelle Minute mit führender Null
    Dim Sekunden As String    ' Aktuelle Sekunde mit führender Null
    Dim DATUMSTEMPEL As String ' YYYYMMDD HHMMSS



' Prüfen ob Button freigeschalten ist

    If T1.Range("U30") = "" Then
```

```
        MsgBox "Die Schaltfläche 'IMPORT AUS DATEI' ist in der Parametertabelle in Zelle U30 deaktiviert." & vbCrLf &
vbCrLf & _
        "Vermutlich wurde der Import über die Zwischenablage eingerichtet. Bitte verwenden Sie den blauen Button 'IMPORT
ZWISCHENABLAGE' rechts."
        Exit Sub
    End If


' --- Vorbereitungen ---

    ' Defaultpfad einstellen
    DATEIPFAD = Replace(T1.Range("E30"), "/", "\") ' Dateipfad auslesen und / ersetzen
    If DATEIPFAD = "" Then DATEIPFAD = ActiveWorkbook.Path ' wenn Dateipfad leer ist, nimm den Pfad der Vorlage
    If Right(DATEIPFAD, 1) <> "\" Then DATEIPFAD = DATEIPFAD & "\" ' wenn ein abschließendes \ fehlt, ergänze es
    If Mid(DATEIPFAD, 2, 1) <> ":" Then DATEIPFAD = ActiveWorkbook.Path & "\" & DATEIPFAD ' wenn der Pfad nur Ordner (und
ev. Unterordner) enthält, ergänze den Pfad der Vorlage

    ' Löschen der Importtabelle und der Interimstabelle
    If T1.Range("O37") = "JA" Then ' wenn Zeile 10 auch überschrieben werden soll
        T2.Range("A10:AZ64000").ClearContents
      Else
        T2.Range("A11:AZ64000").ClearContents
    End If
    AUS

    ' Interim-Importtabelle einblenden
    T99.Visible = xlSheetVisible
    T99.Activate
    Range("A1:AZ64000").ClearContents

    ' wenn automatisches Direktladen aktiviert ist
    If T1.Range("O32") = "JA" Then

        ' Auslesen des automatischen Dateinamens
        DATEIPFAD = Replace(T1.Range("E30"), "/", "\") ' Dateipfad auslesen und / ersetzen
        If DATEIPFAD = "" Then DATEIPFAD = ActiveWorkbook.Path ' wenn Dateipfad leer ist, nimm den Pfad der Vorlage
        If Right(DATEIPFAD, 1) <> "\" Then DATEIPFAD = DATEIPFAD & "\" ' wenn ein abschließendes \ fehlt, ergänze es
        If Mid(DATEIPFAD, 2, 1) <> ":" Then DATEIPFAD = ActiveWorkbook.Path & "\" & DATEIPFAD ' wenn der Pfad nur Ordner
(und ev. Unterordner) enthält, ergänze den Pfad der Vorlage
        DATEI = T1.Range("E31")
        If Left(Right(DATEI, 4), 1) <> "." Then ' wenn das viertletzte Zeichen des Dateinamens keinen Punkt enthält (zb
.csv oder .txt)
```

```
            DATEI = DATEI & "." & T1.Range("E32")
        End If
        DATEI = DATEIPFAD & DATEI
        ' Prüfen ob Datei existiert
        If FileFolderExists(DATEI) = False Then
            MsgBox "Die Datei " & DATEI & " existiert leider nicht. Bitte hinterlegen Sie deren Pfad und Name korrekt in
der Parameter-Tabelle" & _
            " in den Zellen E30 bis E32 oder deaktivieren Sie in der Zelle O32 den automatischen Import."
        End If
        GoTo WEITER
    End If

' --- Auswahl der Datei ---

    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = DATEIPFAD
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            DATEI = .SelectedItems(1)
        Else
            T2.Activate
            T99.Visible = xlSheetHidden
            EIN
            MsgBox "Es wurde keine Datei ausgewählt!"
            Exit Sub
        End If
    End With

WEITER:

   If Dir(DATEI) = "" Then
      T2.Activate
      T99.Visible = xlSheetHidden
      EIN
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
   End If

    ' Auslesen Trennzeichen
```

```
    TRENNZEICHEN = T1.Range("O30")
    If Len(TRENNZEICHEN) > 1 Then TRENNZEICHEN = vbTab ' Tabulator
    If TRENNZEICHEN = vbTab Then VTAB = True
    If TRENNZEICHEN = "," Then VKOMMA = True
    If TRENNZEICHEN = ";" Then VSEMIKOLON = True


' --- Einspielen der Daten ---

    ' Neue Version, die Trennzeichen in ""-geschützten Zellen erkennt

    With ActiveSheet.QueryTables.Add(Connection:="Text;" & DATEI, Destination:=Range("A1"))
        .Name = "*"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = False
        .RefreshPeriod = 0
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = xlWindows
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited ' Wichtig, dass es überhaupt Trennzeichen-Logik gibt
        .TextFileTextQualifier = xlTextQualifierDoubleQuote ' Wichtig, dass mit ""-geschützte Felder das Trennzeichen
enthalten dürfen
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimiter = VTAB
        .TextFileSemicolonDelimiter = VSEMIKOLON
        .TextFileCommaDelimiter = VCOMMA
        .TextFileSpaceDelimiter = VLEER
        .TextFileColumnDataTypes = Array(1, 1, 1, 1, 1, 1, 1, 1) ' Definiert für jede Spalte ihr Format: 1=Allgemein,
2=Text, 4 Datum, 9 Spalte wird nicht analysiert
        .Refresh BackgroundQuery:=False
    End With

    ' Alternativ einfach alle Verbindungen entfernen
    For Each VERBINDUNG In ActiveWorkbook.Connections
        VERBINDUNG.Delete
```

```
    Next


   ' Auslesen des Dateinamens
   DATEINAME = Right(DATEI, Len(DATEI) - InStrRev(DATEI, "\"))

   ' Übertragen der Daten von Interimstabelle in Importtabelle
    Range(Cells(T1.Range("O35"), Columns(T1.Range("O36").Value).Column), Cells(LETZTEZELLE(ActiveSheet.Name).Row,
LETZTEZELLE(ActiveSheet.Name).Column)).Copy

   ' Markieren der Zelle in der Import-Tabelle, ab der die Daten eingefügt werden sollen
    T2.Activate
    If T1.Range("O37") = "JA" Then
        Cells(10, 1).Select
       Else
        Cells(11, 1).Select
    End If

   ' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
    Range("A11").Select

   ' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
    If T1.Range("O35") = "JA" Then T99.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
    T99.Visible = xlSheetHidden

    ' verschieben der Importdatei falls gewünscht
    If T1.Range("O33") = "JA" Then
        ' Festlegen Lagerpfad fürs Verschieben
        DATEIPFADLAGER = Replace(T1.Range("E33"), "/", "\")         ' Dateipfad auslesen und / ersetzen
        If DATEIPFADLAGER = "" Then DATEIPFADLAGER = ActiveWorkbook.Path ' wenn Dateipfad leer ist, nimm den Pfad der
Vorlage
        If Right(DATEIPFADLAGER, 1) <> "\" Then DATEIPFADLAGER = DATEIPFADLAGER & "\" ' wenn ein abschließendes \ fehlt,
ergänze es
        If Mid(DATEIPFADLAGER, 2, 1) <> ":" Then DATEIPFADLAGER = ActiveWorkbook.Path & "\" & DATEIPFADLAGER ' wenn der
Pfad nur Ordner (und ev. Unterordner) enthält, ergänze den Pfad der Vorlage
        ' Anlegen des Unterordners, falls es ihn noch nicht gibt
        If Dir(DATEIPFADLAGER, vbDirectory) = "" Then
            MkDir (DATEIPFADLAGER)
        End If
        ' Soll Datumstempel auch einen bestimmten Namen haben
```

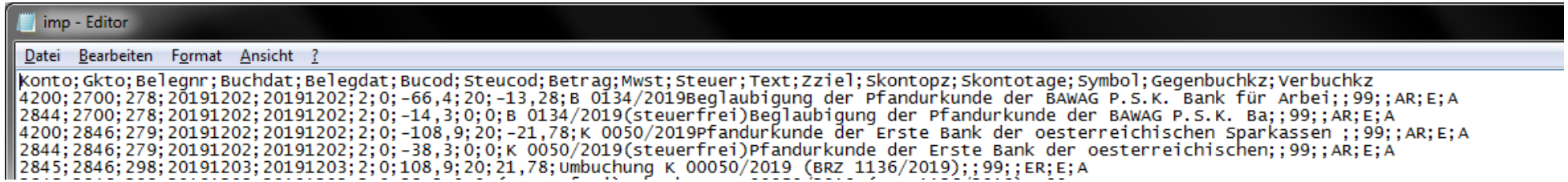
```
        If T1.Range("H34") <> "" Then
            DATUMSTEMPEL = " " & T1.Range("H34")
        End If
        ' Definieren des Datums- und Zeitstempels
        If T1.Range("I35") = "JA" Then
            If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
            If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
            If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
            If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
            If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)),
2)
            DATUMSTEMPEL = DATUMSTEMPEL & " " & Year(Now) & Monat & Tag & "_" & Stunden & Minuten & Sekunden
        End If
        ' Verschieben und umbenennen der Datei
        DATEINAME = Left(DATEINAME, Len(DATEINAME) - 4) & DATUMSTEMPEL & Right(DATEINAME, 4)
        CreateObject("Scripting.FileSystemObject").MoveFile DATEI, DATEIPFADLAGER & DATEINAME
        ' Hinweis
        If T1.Range("O34") = "JA" Then
            MsgBox "Die Importdatei wurde verschoben nach " & DATEIPFADLAGER
        End If
    End If

    ' Abschluss

    EIN

    Range("A12").Activate


' ------------------------
' Ende des normalen Codes
' ------------------------

Exit Sub


'KEINEDATEIAUSGEWÄHLT:
'
'MsgBox "Ein Fehler ist aufgetreten. Bitte kontaktieren Sie den Programmierer: Hr. Wenninger unter
stefan.wenninger@simplesoft.at"
```

```
End Sub
```

2. Workbooks.OpenText Method

Achtung: dieser Code erzeugt IMMER eine neue Arbeitsmappe, die sich öffnet mit dem Namen der CSV-Datei. Dieser Code kann nicht direkt in die aktuelle Mappe reinladen - es ginge nur über Kopieren der Daten in die Zwischenablage, nachdem man die Datei geöffnet hat und anschließend nach dem Einfügen, schließt man wieder die interim-geöffnete Datei

Für diese Datei



sieht der Code so aus (ohne Local:=True kam es zu keiner Auftrennung in einzelnes Spalten !)

Workbooks.OpenText Filename:="Y:\' KLIENT AKTUELL\Went\Zörner - Notar\imp.csv", DataType:=xlDelimited, TextQualifier:=xlTextQualifierDoubleQuote, Semicolon:=True, local:=True

Für diese Datei …

```
Employee Name,Department,Title,Salary,Grade
Apple,Risk,Rick Assistant,10000,01
Banana,Finance,Finance Assistant,20000,02
Cat,Actuarial,Actuarial Assistant,30000,03
Cathy,Actuarial,Actuarial Assistant,30000,03
David,Actuarial,Actuarial Officer,40000,04
Eva,Finance,Senior Finance Officer,50000,05
Flora,Actuarial,"Assistant Manager, Finance",60000,06
Gretta,IT,IT Manager,70000,07
Henry,Customer Fulfillment,CF Manager,80000,08
Icy,Actuarial,"Senior Manager, Actuarial",90000,09
Jerry,Compliance,"Senior Manager, Compliance",100000,10
Ken,Customer Fulfillment,"Senior Manager, CF",110000,11
Larry,Risk,"AVP, Risk",120000,12
Metthew,Admin,AVP Admin,130000,13
Nancy,Admin,"VP, Admin",140000,14
Olivia,HR,"SVP, Admin",150000,15
Peggy,Admin,"ED, Admin",160000,16
```

sieht der Code so aus:

```
Workbooks.OpenText Filename:="C:\Users\WYMAN\Desktop\staff list.csv", DataType:=xlDelimited, comma:=True
```

Und das ist der gesamte Befehl mit allen möglichen Parametern

```
Workbooks.OpenText(FileName, Origin , StartRow , DataType , TextQualifier , ConsecutiveDelimiter , Tab , Semicolon ,
Comma , Space , Other , OtherChar , FieldInfo , TextVisualLayout , DecimalSeparator , ThousandsSeparator ,
TrailingMinusNumbers , Local)
```

Name	Required/Optional	Data type	Description
FileName	Required	String	Specifies the file name of the text file to be opened and parsed.
Origin	Optional	Variant	Specifies the origin of the text file. Can be one of the following xlPlatform constants: xlMacintosh, xlWindows, or xlMSDOS. Additionally, this could be an integer representing the code page number of the desired code page. For example, "1256" would specify that the encoding of the source text file is Arabic (Windows). If this argument is omitted, the method uses the current setting of the File Origin option in the Text Import Wizard.
StartRow	Optional	Variant	The row number at which to start parsing text. The default value is 1.

<table>
<tr>
<td>DataType</td>
<td>Optional</td>
<td>Variant</td>
<td>Specifies the column format of the data in the file. Can be one of the following XlTextParsingType constants: xlDelimited or xlFixedWidth. If this argument is not specified, Microsoft Excel attempts to determine the column format when it opens the file.
<table>
<tr><th>Name</th><th>Value</th><th>Description</th></tr>
<tr><td>xlDelimited</td><td>1</td><td>Default. Indicates that the file is delimited by delimiter characters.</td></tr>
<tr><td>xlFixedWidth</td><td>2</td><td>Indicates that the data in the file is arranged in columns of fixed widths.</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td>TextQualifier</td><td>Optional</td><td>Variant</td><td><table><tr><th>Name</th><th>Value</th><th>Description</th></tr><tr><td>xlTextQualifierDoubleQuote</td><td>1</td><td>Double quotation mark (").</td></tr><tr><td>xlTextQualifierNone</td><td>-4142</td><td>No delimiter.</td></tr><tr><td>xlTextQualifierSingleQuote</td><td>2</td><td>Single quotation mark (').</td></tr></table></td></tr>
<tr><td>ConsecutiveDelimiter</td><td>Optional</td><td>Variant</td><td>True to have consecutive delimiters considered one delimiter. The default is False.</td></tr>
<tr><td>Tab</td><td>Optional</td><td>Variant</td><td>True to have the tab character be the delimiter (DataType must be xlDelimited). The default value is False.</td></tr>
<tr><td>Semicolon</td><td>Optional</td><td>Variant</td><td>True to have the semicolon character be the delimiter (DataType must be xlDelimited). The default value is False.</td></tr>
<tr><td>Comma</td><td>Optional</td><td>Variant</td><td>True to have the comma character be the delimiter (DataType must be xlDelimited). The default value is False.</td></tr>
<tr><td>Space</td><td>Optional</td><td>Variant</td><td>True to have the space character be the delimiter (DataType must be xlDelimited). The default value is False.</td></tr>
<tr><td>Other</td><td>Optional</td><td>Variant</td><td>True to have the character specified by the OtherChar argument be the delimiter (DataType must be xlDelimited). The default value is False.</td></tr>
<tr><td>OtherChar</td><td>Optional</td><td>Variant</td><td>(required if Other is True). Specifies the delimiter character when Other is True. If more than one character is specified, only the first character of the string is used; the remaining characters are ignored.</td></tr>
</table>

<table>
<tr><td>FieldInfo</td><td>Optional</td><td>Variant</td><td>An array containing parse information for individual columns of data. The interpretation depends on the value of DataType. When the data is delimited, this argument is an array of two-element arrays, with each two-element array specifying the conversion options for a particular column. The first element is the column number (1-based), and the second element is one of the XlColumnDataType constants specifying how the column is parsed.
<table>
<tr><th>Name</th><th>Value</th><th>Description</th></tr>
<tr><td>xlDMYFormat</td><td>4</td><td>DMY date format.</td></tr>
<tr><td>xlDYMFormat</td><td>7</td><td>DYM date format.</td></tr>
<tr><td>xlEMDFormat</td><td>10</td><td>EMD date format.</td></tr>
<tr><td>xlGeneralFormat</td><td>1</td><td>General.</td></tr>
<tr><td>xlMDYFormat</td><td>3</td><td>MDY date format.</td></tr>
<tr><td>xlMYDFormat</td><td>6</td><td>MYD date format.</td></tr>
<tr><td>xlSkipColumn</td><td>9</td><td>Column is not parsed.</td></tr>
<tr><td>xlTextFormat</td><td>2</td><td>Text.</td></tr>
<tr><td>xlYDMFormat</td><td>8</td><td>YDM date format.</td></tr>
<tr><td>xlYMDFormat</td><td>5</td><td>YMD date format.</td></tr>
</table></td></tr>
<tr><td>TextVisualLayout</td><td>Optional</td><td>Variant</td><td>The visual layout of the text.</td></tr>
<tr><td>DecimalSeparator</td><td>Optional</td><td>Variant</td><td>The decimal separator that Microsoft Excel uses when recognizing numbers. The default setting is the system setting.</td></tr>
<tr><td>ThousandsSeparator</td><td>Optional</td><td>Variant</td><td>The thousands separator that Excel uses when recognizing numbers. The default setting is the system setting.</td></tr>
</table>

TrailingMinusNumbers	Optional	Variant	Specify True if numbers with a minus character at the end should be treated as negative numbers. If False or omitted, numbers with a minus character at the end are treated as text.
Local	Optional	Variant	Specify True if regional settings of the machine should be used for separators, numbers and data formatting.

CSV-Datei importieren und anschließend geändert exportieren im Pfad der Vorlage mit Dateiauswahl

```
VERSION 1

Sub CSV_Import()

' Er liest eine CSV-Datei ein,
' exportiert die zu definierenden Spalten

   Dim iReihe As Long, iSpalte As Long
   Dim DATEI As String, iText As String, TRENNZEICHEN As String
   Dim i As Long
   Dim WERT
   Dim test As String
```

```
  Dim LETZTEZEILE
  Dim TEMP ' enthält die Daten einer Zeile


' Vorbereitungen

    Application.ScreenUpdating = False
'    Tabelle1.Activate
'    Tabelle1.Range("A2:AZ63999").ClearContents


' Auswahl der Datei

    With Application.FileDialog(msoFileDialogFilePicker)
       .InitialFileName = ThisWorkbook.Path & "\"
       .Title = "Dateiauswahl"
       .ButtonName = "Auswahl..."
       .InitialView = msoFileDialogViewDetails
       If .Show = -1 Then
          DATEI = .SelectedItems(1)
       Else
          MsgBox "Es wurde keine Datei ausgewählt!"
          Exit Sub
       End If
    End With

  If Dir(DATEI) = "" Then
     MsgBox "Es wurde keine Datei ausgewählt"
     Exit Sub
  End If

' Auslesen welche Kundentypdatei ausgewählt wurde

  TRENNZEICHEN = ";"
  iReihe = 3
  iSpalte = 1
  Close ' Sicherheitshalber eventuell offene Dateien schließen

' Einspielen der Daten

  Open DATEI For Input As #1
     Do Until EOF(1)
        Line Input #1, iText
        Do While InStr(iText, TRENNZEICHEN)
```

```
            Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
            iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
            iSpalte = iSpalte + 1
          Loop
          Cells(iReihe, iSpalte).Value = iText
          iReihe = iReihe + 1
          iSpalte = 1
        Loop
      Close

' Anpassen Exportdatei um freiwillige Namensergänzung


    DATEI = Replace(DATEI, ".csv", "_neu.csv")


' Exportieren der definierten Spalten

    LETZTEZEILE = LETZTEZELLE(Tabelle1.Name).Row

    Open DATEI For Output As #1

    For Z = 2 To LETZTEZEILE

        TEMP = ""
        For S = 1 To 10 ' hier einstellen wieviele Spalten zu exportieren sind

            TEMP = TEMP & Cells(Z, S) & ";"

        Next S

        ' Da nach letztem Datenfeld kein abschließendes Semikolon benötigt wird
        TEMP = Left(TEMP, Len(TEMP) - 1)
        Print #1, TEMP

    Next Z

    Close #1

    Application.ScreenUpdating = True

    MsgBox "Die Daten wurden exportiert nach " & DATEI

Exit Sub
```

KEINEDATEIAUSGEWÄHLT:

MsgBox "Ein Fehler ist aufgetreten. Bitte kontaktieren Sie den Programmierer: Hr. Wenninger unter stefan.wenninger@simplesoft.at"

End Sub

VERSION 2 mit auswählbaren Spaltenbuchstaben für den Usern

```
Sub Kundenstamm_CSV_Import()

' Dies ist der Button 'Kundenstamm_CSV_Import'
' Er liest eine NTCS-Datei mit Kundenkontenstamm aus,
' er erkennt ausgehend vom Dateinamen, um welchen Kundenstamm-Typ es sich handelt
' und exportiert die zu definierenden Spalten

' WOZU: für Zahlscheinkunden wird ein anderer Export benötigt (Inhalt) als für zB Kreditkartenkunden

   Dim iReihe As Long, iSpalte As Long
   Dim DATEI As String, iText As String, TRENNZEICHEN As String
   Dim i As Long
   Dim WERT
   Dim test As String

   Dim KUNDENTYPZEILE As Integer ' welche der drei Zeilen 7-9 in der Tabelle CSV-Kundenstamm regelt den Satzaufbau
   Dim LETZTEZEILE
   Dim TEMP ' enthält die Daten einer Zeile


' Vorbereitungen

Application.ScreenUpdating = False
Tabelle1.Activate
Tabelle1.Range("A2:AZ63999").ClearContents


' Auswahl der Datei

    With Application.FileDialog(msoFileDialogFilePicker)
       .InitialFileName = ThisWorkbook.Path & "\"
```

```
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            DATEI = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewählt!"
            Exit Sub
        End If
    End With


    If Dir(DATEI) = "" Then
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
    End If

' Auslesen welche Kundentypdatei ausgewählt wurde

    KUNDENTYPZEILE = 0
    If InStr(DATEI, Tabelle7.Range("C7")) > 0 Then KUNDENTYPZEILE = 7
    If InStr(DATEI, Tabelle7.Range("C8")) > 0 Then KUNDENTYPZEILE = 8
    If InStr(DATEI, Tabelle7.Range("C9")) > 0 Then KUNDENTYPZEILE = 9

    If KUNDENTYPZEILE = 0 Then
        Tabelle7.Activate
        Application.ScreenUpdating = True
        MsgBox "Im Dateinamen der ausgewählten Datei ist keiner der drei Dateikürzel der Tabelle 'CSV-Kundenstamm' im Bereich 'C7:C9' vorhanden"
        End
    End If

    TRENNZEICHEN = ";"
    iReihe = 3
    iSpalte = 1
    Close ' Sicherheitshalber eventuell offene Dateien schließen

' Einspielen der Daten

    Open DATEI For Input As #1
      Do Until EOF(1)
        Line Input #1, iText
        Do While InStr(iText, TRENNZEICHEN)
          Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
          iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
```

```
          iSpalte = iSpalte + 1
        Loop
        Cells(iReihe, iSpalte).Value = iText
        iReihe = iReihe + 1
        iSpalte = 1
      Loop
    Close

' Anpassen Exportdatei um freiwillige Namensergänzung

  If Tabelle7.Range("U2") <> "" Then

    DATEI = Replace(DATEI, ".csv", Tabelle7.Range("U2") & ".csv")

  End If

' Exportieren der definierten Spalten

    LETZTEZEILE = LETZTEZELLE(Tabelle1.Name).Row

    Open DATEI For Output As #1

    For Z = 3 To LETZTEZEILE

        TEMP = ""
        For S = 8 To 52 ' in der Tabelle 7 können in der Zeile KUNDENTYPZEILE von der Spalte 8 bis zur Spalte 52 Spaltenbuchstaben eingebegeben werden vom
User, um zu steuern welche Spalten exportiert werden sollen

          If Tabelle7.Cells(KUNDENTYPZEILE, S) = "" Then Exit For ' wenn die erste Zelle ohne Spalte erreicht ist in der Tabelle CSV-Kundenstamm geh zur
nächsten Zeile, um sie zu exportieren
          TEMP = TEMP & Range(Tabelle7.Cells(KUNDENTYPZEILE, S) & Z) & ";"

        Next S
        ' Da nach letztem Datenfeld kein abschließendes Semikolon benötigt wird
        TEMP = Left(TEMP, Len(TEMP) - 1)
        Print #1, TEMP
    Next Z

    Close #1

    Application.ScreenUpdating = True

    MsgBox "Die Daten wurden exportiert nach " & DATEI
```

Exit Sub

KEINEDATEIAUSGEWÄHLT:

MsgBox "Ein Fehler ist aufgetreten. Bitte kontaktieren Sie den Programmierer: Hr. Wenninger unter stefan.wenninger@simplesoft.at"

End Sub

CSV/TXT oder Exceldatei in aktuelle Mappe importieren

```
Sub IMPORT_DATEI_AUTOMATISCH()

' Dieser Code öffnet und importiert Daten aus
' 1.) Einer Exceldatei
' oder - lt. Parameter in T1.Range("O4") denn dort ist das Importtrennzeichen "," ";" "Tabulator" oder "Keines / Excel"
hinterlegt
' 2.) einer CSV oder TXT-Datei mit variablem Trennzeichen
' in das Zieltabellenblatt (T2) der aktuellen Arbeitsmappe

' --- Variablendefinition ---

    Dim Excel_Quelldatei As Object ' Die Excel_Quelldatei
    Dim Aktuelle_Mappe As Workbook ' Die aktuelle Mappe hier
    Dim Datei_Gesamtpfad As String ' Der Pfad zur Quelldatei inkl. Dateinamen und Endung (xls/xlsx, csv oder txt)

' --- Auslesen Quelldateipfad und Prüfen, ob es die Datei gibt ---

    Datei_Gesamtpfad = T1.Range("O5")
    If Datei_Gesamtpfad = "" Then MsgBox "Es ist kein Dateipfad und Dateiname ausgefüllt in der Zelle O5 in der
Parametertabelle": Exit Sub
```

```
    If Dir(Datei_Gesamtpfad) = "" Then MsgBox "Die Datei " & Datei_Gesamtpfad & " in der Zelle O5 in der Parametertabelle
existiert nicht.": Exit Sub
    ' Wenn Exceldatei=>richtiges Trennzeichen ? - wird aber ohnedies vom Importcode unten automatisch korrigiert, der
xls(x)-Dateien automatisch als Excel einliest
    If InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls") > 0 And T1.Range("O4").Value <> "Keines /
Excel" Then
        MsgBox "Sie haben als Quelldatei eine Exceldatei hinterlegt, aber ein falsches Importtrennzeichen eingestellt.
Bitte wählen Sie das Trennzeichen 'Keines / Excel'"
        Exit Sub
    End If
    ' Wenn keine Exceldatei=>richtiges Trennzeichen ?
    If InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls") = 0 And T1.Range("O4").Value = "Keines /
Excel" Then
        MsgBox "Sie haben als Quelldatei keine Exceldatei hinterlegt, aber als Importtrennzeichen 'Keines / Excel'
eingestellt. Bitte wählen Sie das korrekte Trennzeichen."
        Exit Sub
    End If

' ---  Löschen der Importtabelle ---

    T2.UsedRange.ClearContents ' Zellformat (Text bei uns) soll erhalten bleiben

' ---  Import der Daten ---

    If T1.Range("O4").Value = "Keines / Excel" Or InStr(Mid(Datei_Gesamtpfad, InStrRev(Datei_Gesamtpfad, "."), 4), "xls")
> 0 Then

       ' Wenn die Quelldatei eine Exceldatei ist

       Set Aktuelle_Mappe = ActiveWorkbook ' merken der aktuellen Mappe hier
       Set Quelldatei = Workbooks.Open(Datei_Gesamtpfad, False, True) ' Verknüpfungen nicht aktualisieren (false) und
Datei nur lesend öffnen (true)
       ActiveWorkbook.ActiveSheet.UsedRange.Copy ' Alle Daten in der Arbeitsmappe kopieren (in derem aktiven
Tabellenblatt - unsere Quelldaten haben nur ein Tabellenblatt)
       Aktuelle_Mappe.Activate ' Rückkehr in unsere aktuelle Arbeitsmappe
       T2.Range("A1").PasteSpecial ' Werte einfügen in unser Tabelle T2, die die Daten erhalten soll
       Application.DisplayAlerts = False ' kein Hinweis vor dem Schließen
       Quelldatei.Close
       Application.DisplayAlerts = True

      Else
```

```
        ' Import CSV oder TXT-Datei mit variablem Trennzeichen

        TRENNZEICHEN = T1.Range("O4")
        If Len(TRENNZEICHEN) > 1 Then TRENNZEICHEN = vbTab ' Tabulator
        iReihe = 1 ' Variable durch alle Zeilen
        iSpalte = 1 ' Variable durch alle Spalten
        Close ' Sicherheitshalber eventuell offene Dateien schließen
        T2.Activate
        ' Einspielen der Daten
        Open Datei_Gesamtpfad For Input As #1
           Do Until EOF(1)
              Line Input #1, iText
              Do While InStr(iText, TRENNZEICHEN)
                 Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
                 iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
                 iSpalte = iSpalte + 1
              Loop
              Cells(iReihe, iSpalte).Value = iText
              iReihe = iReihe + 1
              iSpalte = 1
           Loop
        Close

    End If

End Sub
```

CSV-Export markierter Bereich

```
Sub Test()

Bereich_Export_CSV ("C:\Test.csv")

End Sub
```

```
Sub Bereich_Export_CSV(Speicherpfad As String)

' www.simplesoft.at
' Dieses Makro exportiert den aktuell markierten Bereich in den übergebenen Speicherpfad
' Dieser Speicherpfad muss auch den CSV-Speicherdateinamen enthalten am Ende

Dim Bereich As Range, Zeile As Object, Zelle As Object
Dim strTrennzeichen As String
Dim blnAnfuehrungszeichen As Boolean

' Parameter setzen

   strTrennzeichen = ";" ' Trennzeichen

   blnAnfuehrungszeichen = False ' o. True - bestimmt, ob beim Export " verwendet werden sollen

' Auslesen markierter Bereich

   Set Bereich = Selection

' Warnungen übergehen (ev. Überschreiben einer existierende Datei)

   Application.DisplayAlerts = False

' Öffnen CSV-Datei

' Korrektur, falls ASP-System
If Instr(Speicherpfad,"\\\")>0 then
      Speicherpfad = Mid(Speicherpfad, InStr(Speicherpfad, "\\\") + 1, 255)
End if

   Open Speicherpfad For Output As #1

' Export

   For Each Zeile In Bereich.Rows
      For Each Zelle In Zeile.Cells
         If blnAnfuehrungszeichen = True Then
            strTemp = strTemp & """" & CStr(Zelle.Text) & """" & strTrennzeichen
         Else
            strTemp = strTemp & CStr(Zelle.Text) & strTrennzeichen
         End If
```

```
      Next
      If Right(strTemp, 1) = strTrennzeichen Then strTemp = Left(strTemp, Len(strTemp) - 1)
      Print #1, strTemp
      strTemp = ""
    Next

  Close #1
  Set Bereich = Nothing


' Warnungen wieder aktivieren

    Application.DisplayAlerts = False

End Sub
```

CSV-Export mit frei wählbaren Spalten und Reihenfolgen

```
' CSV-Export


    Speicherpfad = "N:\PR und Kommunikation\Newsservice\Supermail_Emailadressen\"
    Dateiname = "Newsletter " & Newsletter & " - " & Replace(Left(Now, 10), ".", "_") & ".csv"
    Gesamtpfad = Speicherpfad & Dateiname

    Application.DisplayAlerts = False


' Korrektur, falls ASP-System
If Instr(Gesamtpfad,"\\\")>0 then
       Gesamtpfad = Mid(Gesamtpfad, InStr(Gesamtpfad, "\\\") + 1, 255)
End if



    On Error GoTo FEHLER


    Open Gesamtpfad For Output As #1

    On Error GoTo 0
```

```
    For Z = 2 To LETZTEZELLE(ActiveSheet.Name).Row
       Name = Cells(Z, 5) & " " & Cells(Z, 6)
       If Right(Name, 1) = " " Then Name = Left(Name, Len(Name) - 1)
       Print #1, Name & ";" & Cells(Z, 7) & ";" & Cells(Z, 8) & ";" & Cells(Z, 11)
    Next Z
    Close #1

    Application.DisplayAlerts = True


MsgBox "Es gibt " & LETZTEZELLE(ActiveSheet.Name).Row - 1 & " Empfänger für diesen Newsletter und" & vbCrLf & _
       "es wurden " & Dublikate & " Dublikate entfernt." & vbCrLf & vbCrLf & _
       "Die Datei   '" & Dateiname & "'   wurde gespeichert in das Verzeichnis " & vbCrLf & Speicherpfad
Exit Sub


FEHLER:

MsgBox "Die Datei '" & Speicherpfad & "' konnte nicht gespeichert werden. Bitte kontrollieren Sie, " & vbCrLf & vbCrLf & _
"- dass das entsprechende Verzeichnis und das Laufwerk " & Left(Speicherpfad, 1) & " existieren" & vbCrLf & vbCrLf & _
"- und Sie das Recht zum Speichern von Dateien in dieses Verzeichnis haben."
```

CSV-Export mit Abfrage

```
Sub ExportCSV()

Dim Bereich As Object, Zeile As Object, Zelle As Object
Dim strTemp As String
Dim strDateiname As String
Dim strTrennzeichen As String
Dim strMappenpfad As String
Dim blnAnfuehrungszeichen As Boolean
```

```
strMappenpfad = ActiveWorkbook.FullName

strDateiname = InputBox("Bitte den Namen der CSV-Datei angeben.", "CSV-Export", strMappenpfad)
If strDateiname = "" Then Exit Sub

strTrennzeichen = InputBox("Welches Trennzeichen soll verwendet werden?", "CSV-Export", ",")
If strTrennzeichen = "" Then Exit Sub

If MsgBox("Sollen die Werte in Anführungszeichen exportiert werden?", vbQuestion + vbYesNo, "CSV-Export") = vbYes Then
   blnAnfuehrungszeichen = True
Else
   blnAnfuehrungszeichen = False
End If

Set Bereich = ActiveSheet.UsedRange

' Korrektur, falls ASP-System
If Instr(strDateiname,"\\\")>0 then
      strDateiname = Mid(strDateiname, InStr(strDateiname, "\\\") + 1, 255)
End if

Open strDateiname For Output As #1

For Each Zeile In Bereich.Rows
   For Each Zelle In Zeile.Cells
      If blnAnfuehrungszeichen = True Then
         strTemp = strTemp & """" & CStr(Zelle.Text) & """" & strTrennzeichen
      Else
         strTemp = strTemp & CStr(Zelle.Text) & strTrennzeichen
      End If
```

```
    Next
    If Right(strTemp, 1) = strTrennzeichen Then strTemp = Left(strTemp, Len(strTemp) - 1)
    Print #1, strTemp
    strTemp = ""
Next


  Close #1
  Set Bereich = Nothing
  MsgBox "Export erfolgreich. Datei wurde exportiert nach" & vbCrLf & strDateiname

End Sub
```

CSV-Export aller Tabellenblätter

```
Sub outputtotextfile()
wline = ""
For Each ws In Worksheets

With ws
For r = 1 To .Range("A65536").End(xlUp).Row
    lcol = .Cells(r, 256).End(xlToLeft).Column
    wline = wline & Join(Application.Transpose(Application.Transpose(.Range("A" & r).Resize(, lcol))), ",") & vbNewLine
Next r
End With

Next ws


    Open "C:\test2.csv" For Output As #1 'Replaces existing file
    Print #1, wline
    Close #1

End Sub
```

CSV-/TXT-Datei erzeugen und mit Daten befüllen

Mit dem File-System-Object haben Sie Zugriff auf das Dateisystem ihres Computers.
Damit haben Sie die Möglichkeit direkt aus EXCEL auf Dateien zuzugreifen, Eigenschaften abzurufen oder Dateien zu verändern bzw. zu erstellen.
Auch das suchen von Dateien und Ordnern, bzw. die Überprüfung ob diese Datei/en / Ordner vorhanden sind oder nicht ist möglich.
Damit ist eine interaktive Benutzerführung auch auf "unbekannten" Rechnern möglich.
Der Windows-Scripting-Host bietet ähnliche oder gleiche Funktionen, setzt aber voraus dass dieser mit dem Internet Explorer installiert wurden, insofern ist das File-System-Object die bessere Variante, da sie unabhängig funktioniert.
Grundsätzlich erfolgt der Aufruf immer gleich. Zuerst muss eine Variable als Object deklariert werden, und anschliessend wird dem Object etwas zugewiesen, in unserem Fall das File-System-Object.
Dim myFSO As Object, myTxt As Object
Set myFSO = CreateObject("Scripting.FileSystemObject")
Anschliessend können sie diesem Object weitere Methoden zuweisen
Set myTxt = fs.CreateTextFile("C:\DemoText.txt", True)
mit
myTxt.WriteLine("Dies ist ein Test.")
können sie nun einen Text in das File schreiben, wobei dieses auch noch gleichzeitig automatisch erstellt wird.

CSV-Daten importieren + exportieren (von BMD-Schnittstellen von mir)

Sub IMPORT_AUS_DATEI_SEMIKOLON()

```
  Dim iReihe As Long, iSpalte As Long
  Dim DATEI As String, iText As String, TRENNZEICHEN As Variant
  Dim i As Long
  Dim WERT

' Auswahl der Datei
   With Application.FileDialog(msoFileDialogFilePicker)
      .InitialFileName = ThisWorkbook.Path & "\"
      .Title = "Dateiauswahl"
      .ButtonName = "Auswahl..."
      .InitialView = msoFileDialogViewDetails
      If .Show = -1 Then
         DATEI = .SelectedItems(1)
      Else
```

```
            MsgBox "Es wurde keine Datei ausgewählt!"
            Exit Sub
        End If
    End With

    If Dir(DATEI) = "" Then
       MsgBox "Es wurde keine Datei ausgewählt"
       Exit Sub
    End If

    TRENNZEICHEN = Tabelle9.Range("F13")
    If TRENNZEICHEN = "Tabulator" Then TRENNZEICHEN = vbTab

    iReihe = 2
    iSpalte = 3
    Close ' Sicherheitshalber eventuell offene Dateien schließen

    Range("C2:IV65000").ClearContents ' löschen eventuell früherer Werte


' Einspielen der Daten

    Open DATEI For Input As #1
       Do Until EOF(1)
          Line Input #1, iText
          Do While InStr(iText, TRENNZEICHEN)
             Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
             iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
             iSpalte = iSpalte + 1
          Loop
          Cells(iReihe, iSpalte).Value = iText
          iReihe = iReihe + 1
          iSpalte = 3
       Loop
    Close

     ' Columns("A:FF").EntireColumn.AutoFit
     Cells(2, 3).Select


End Sub
```

Sub EXPORTIEREN()

```
' Speichert die Buchungen in die Exportdatei

    Dim ZELLWERT As String
    Dim REIHEN As Long
    Dim SPALTEN As Integer
    Dim DATEINAME
    Dim DATEI As String
    Dim SPEICHERORDNER As String
    Dim SPEICHERFESTPLATTE As String
    Dim BUTTONTEXT As String
    Dim Wahl As Integer
    Dim TRENNZEICHEN


    ' Abfragen des Dateinamens

        DATEI = Tabelle9.Range("F11") & Mid(Tabelle2.Range("N2"), 5, 2) & ".txt" ' Defaultnamen festlegen

        ' Defaultspeicherpfad einstellen

        SPEICHERFESTPLATTE = Tabelle9.Range("F7")
        SPEICHERORDNER = Tabelle9.Range("J7")

        On Error Resume Next
        ChDrive SPEICHERFESTPLATTE
        ChDir SPEICHERORDNER

        ' Speichern-Fenster öffnen mit Vorschlag

        DATEINAME = Application.GetSaveAsFilename(InitialFileName:=DATEI, fileFilter:="Text Files (*.txt), *.txt")

        If DATEINAME = False Then
            MsgBox "Abbruch - Es wurde nichts gespeichert !", vbInformation
            Exit Sub
        End If

    ' Speichern des Speicherpfades

        SPEICHERORDNER = DATEINAME
        While Right(SPEICHERORDNER, 1) <> "\"
            SPEICHERORDNER = Left(SPEICHERORDNER, Len(SPEICHERORDNER) - 1)
        Wend
        SPEICHERFESTPLATTE = SPEICHERORDNER
        While Right(SPEICHERFESTPLATTE, 1) <> ":"
```

```
        SPEICHERFESTPLATTE = Left(SPEICHERFESTPLATTE, Len(SPEICHERFESTPLATTE) - 1)
      Wend
      SPEICHERFESTPLATTE = SPEICHERFESTPLATTE & "\"
      ChDrive SPEICHERFESTPLATTE
      ChDir SPEICHERORDNER

  ' Das Speichern in die Datei

      AUS

     TRENNZEICHEN = Tabelle9.Range("F15")
     If TRENNZEICHEN = "Tabulator" Then TRENNZEICHEN = vbTab

      Open DATEINAME For Output As #1
      Range("A1").Select
      For REIHEN = 2 To LETZTEZELLE(ActiveSheet.Name).Row + 1
         For SPALTEN = 1 To ActiveSheet.UsedRange.Columns.Count
            ZELLWERT = ZELLWERT & ActiveCell.Value & vbTab
            ActiveCell.Offset(0, 1).Select
         Next SPALTEN
         Print #1, ZELLWERT
         ZELLWERT = ""
         Range("A" & REIHEN).Select
      Next REIHEN
      Close #1

      EIN
      MsgBox "TEXTDATEI  WURDE  ERSTELLT  IN  " & SPEICHERORDNER & "  !", vbInformation

End Sub
```

CSV- o. TXT-Datei einlesen (var.Trennzeichen, vorgeschlagener Ordner wie aktuelle Datei)

```
Sub IMPORT_AUS_DATEI_BEISTRICH()

   Dim iReihe As Long, iSpalte As Long
   Dim DATEI As String, iText As String, TRENNZEICHEN As String
   Dim i As Long
   Dim WERT
```

```
' Auswahl der Datei
    With Application.FileDialog(msoFileDialogFilePicker)
        .InitialFileName = ThisWorkbook.Path & "\"
        .Title = "Dateiauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewDetails
        If .Show = -1 Then
            DATEI = .SelectedItems(1)
        Else
            MsgBox "Es wurde keine Datei ausgewaehlt!"
            Exit Sub
        End If
    End With

   If Dir(DATEI) = "" Then
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
   End If

   TRENNZEICHEN = ","
   iReihe = 1
   iSpalte = 1
   Close ' Sicherheitshalber eventuell offene Dateien schließen

   Range("A1:IV65000").ClearContents ' löschen eventuell früherer Werte


' Einspielen der Daten

   Open DATEI For Input As #1
      Do Until EOF(1)
         Line Input #1, iText
         Do While InStr(iText, TRENNZEICHEN)
            Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
            iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
            iSpalte = iSpalte + 1
         Loop
         Cells(iReihe, iSpalte).Value = iText
         iReihe = iReihe + 1
         iSpalte = 1
      Loop
   Close

    ' Columns("A:FF").EntireColumn.AutoFit
```

```
    Cells(2, 1).Select

End Sub
```

Dateien im Ordner verschieben, umbenennen...

Nachfolgender Code stammt von meinem Lexor-Archivdokumente-Konverter, der alle Dateien in einem Ordner inkl Unterordner durchsucht und umbenennt.

Wirklich brauchbar sind am Beginn der Code, der das gesamte Verzeichnis inkl. Unterordner ausliest

```
Const VERZEICHNIS = "E:\Lexor\" ' Speicherpfad, wo die Lexor-Klientenordner kopiert worden sind

'On Error GoTo FEHLER
ChDir VERZEICHNIS

With Application.FileSearch
   .NewSearch
   .LookIn = VERZEICHNIS
   .Filename = "*.*" ' oder csv, txt etc
   .SearchSubFolders = True

   If .Execute() > 0 Then
      For Each DATEI In .FoundFiles
         Cells(zeile, 1) = DATEI
```

Und dann der Code zum Umbenennen und gleichzeitigem Verschieben

```
' Datei verschieben und umbenennen
Cells(zeile, 9) = "Name " & DATEI & " As " & SPEICHERPFAD & DATEINEU
Name DATEI As SPEICHERPFAD & DATEINEU
```

```
' ANLEITUNG:

' Wir exportieren - über die Zwischenablage - aus Lexor die drei Archivkategorien DOKUMENT, POSTEINGANG, POSTAUSGANG und
' kopieren das Ergebnis hier
' in die Tabellenblätter DOK, EIN und AUS
' wir kopieren anschließend auf einem lokalen Rechner in das Hauptverzeichnis in den Unterordner LEXOR alle Klientenordner
' im Fall der MEDPLAN war dies der Ordner C:\Lexor\

' Wir tragen diesen Pfad in die Konstante VERZEICHNIS ein (ist die erste Zeile nach der Variablen-Definition)

' Die Tabelle FUND dient nur zum Protokollieren.

' Es ist egal in welchem Tabellenblatt man gerade ist,wenn man den VBA-Code startet.
' Man startet ihn übrigens direkt hier in der VBA-Umgebung.

' Zu Testzwecken lohnt es sich nur drei, vier Klienten für den ersten Lauf in den Ordner LEXOR zu kopieren.

' Nach dem Testlauf dürfen keine Dateien mehr in den Klientenordnern sein mit den original Namen (PE12345.msg etc) sondern
' nur noch alle in den 0_BMD-Ordnern.

' Das Programm kann hier mehrfach über Ordner drüber wandern, die es schon bearbeitet hat (etwa nach Abbruch durch Fehler),
' weil es beim erneuten Lauf die Dateien im Unterordner 0_BMD ignoriert.

' Dateien, die am gleichen Tag vom selben User mit selbem Betreff gespeichert wurden, würden zu gleichen Zieldateinamen führen
' - das Programm fügt bei den Dateien mit gleichem Namen einen zweistelligen Zufalls-Hexcode ein.

Sub SCHLEIFE()
Dim DATEI As Variant ' der gesamte Speicherpfad der Datei inkl. Datei
Dim KLIENT ' KLientennummer (bei Medplan 8xxxxx)
Dim DATEINAME ' nur der reine Dateiname - zB DK123456.doc
Dim DATEINR ' nur die reine Datei-Nummer: 123456 - wichtig: führende Nulls müssen weggeschnitten werden
Dim DATEITYP ' ob doc, xls, msg etc
Dim SUCHTABELLE ' in welcher Tabelle soll das Dokument gesucht werden: DK-Dokumente in DOK,PA-Dokumente in AUS und PE in EIN
Dim DATEINEU ' der neue Dateiname: Datum DK/PE/PA Mitarbeiternamenskürzel Betreff
Dim ZUFALL ' Zufallszahl
Dim SPEICHERPFAD ' wohin wird die Datei verschoben

Const VERZEICHNIS = "E:\Lexor\" ' Speicherpfad, wo die Lexor-Klientenordner kopiert worden sind
```

```
'On Error GoTo FEHLER
ChDir VERZEICHNIS

Sheets("FUND").Activate
Range("A1:IB65000").ClearContents
zeile = 1
Application.ScreenUpdating = False


With Application.FileSearch
   .NewSearch
   .LookIn = VERZEICHNIS
   .Filename = "*.*" ' oder csv, txt etc
   .SearchSubFolders = True

   If .Execute() > 0 Then
      For Each DATEI In .FoundFiles
         Cells(zeile, 1) = DATEI

         ' Klientennummer
         KLIENT = Mid(DATEI, 10, 6)
         Cells(zeile, 2) = KLIENT
         If InStr(DATEI, "0_BMD") <> 0 Then GoTo WEITER2

         ' Dateiname
         DATEINAME = Mid(DATEI, 1 + InStrRev(DATEI, "\"), 999)
         Cells(zeile, 3) = DATEINAME
         DATEITYP = UCase(Right(DATEINAME, 4))

         ' Ungültige Datei (man kann ja auch manuell was in den Lexor-Ordner kopiert haben - kam bei Medplan vor
         If Left(DATEINAME, 2) <> "DK" And Left(DATEINAME, 2) <> "PA" And Left(DATEINAME, 2) <> "PE" Then
            Cells(zeile, 4) = "Ungültig"
            GoTo WEITER
         End If

         ' Datei-Nr
         DATEINR = Mid(DATEINAME, 3, 3 + Len(DATEI) - InStrRev(DATEI, "."))
         While Left(DATEINR, 1) = "0"
            DATEINR = Mid(DATEINR, 2, 99)
         Wend
         Cells(zeile, 5) = DATEINR

         ' Suchen der Datei
         If Left(DATEINAME, 2) = "DK" Then SUCHTABELLE = "DOK"
```

```
        If Left(DATEINAME, 2) = "PA" Then SUCHTABELLE = "AUS"
        If Left(DATEINAME, 2) = "PE" Then SUCHTABELLE = "EIN"

        Sheets(SUCHTABELLE).Activate
        Cells(1, 1).Select
WEITERSUCHEN:
        On Error Resume Next
           Cells.Find(What:=DATEINR, After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, SearchOrder:=xlByColumns, SearchDirection:=xlNext,
MatchCase:=False, SearchFormat:=False).Activate
        On Error GoTo 0

        If ActiveCell.Row = 1 Then ' wenn es nicht gefunden wurde
           Sheets("FUND").Activate
           Cells(zeile, 6) = "KEIN FUND"
           Sheets("FUND").Activate
           GoTo WEITER
        End If

        If ActiveCell.Column <> 6 Then ' wenn der Wert in einer anderen Spalte gefunden wurde => weitersuchen
           GoTo WEITERSUCHEN
        End If

        ' Wenn gefunden

        ' Speicherpfad
        SPEICHERPFAD = Left(DATEI, 16) & "0_BMD" & "\"
        If Dir(SPEICHERPFAD, vbDirectory) = "" Then MkDir (SPEICHERPFAD) ' Verzeichnis anlegen, falls es noch vorhanden ist

        If SUCHTABELLE = "DOK" Then DATEINEU = Mid(Cells(ActiveCell.Row, 18), 7, 4) & "." & Mid(Cells(ActiveCell.Row, 18), 4, 2) & "." &
Left(Cells(ActiveCell.Row, 18), 2) & " " & SUCHTABELLE & DATEITYP & " -" & Left(Cells(ActiveCell.Row, 5), 5) & "- " & Left(Cells(ActiveCell.Row, 8), 99)
        If SUCHTABELLE = "AUS" Then DATEINEU = Mid(Cells(ActiveCell.Row, 35), 7, 4) & "." & Mid(Cells(ActiveCell.Row, 35), 4, 2) & "." &
Left(Cells(ActiveCell.Row, 35), 2) & " " & SUCHTABELLE & DATEITYP & " -" & Left(Cells(ActiveCell.Row, 5), 5) & "- " & Left(Cells(ActiveCell.Row, 20), 99)
        If SUCHTABELLE = "EIN" Then DATEINEU = Mid(Cells(ActiveCell.Row, 22), 7, 4) & "." & Mid(Cells(ActiveCell.Row, 22), 4, 2) & "." &
Left(Cells(ActiveCell.Row, 22), 2) & " " & SUCHTABELLE & DATEITYP & " -" & Left(Cells(ActiveCell.Row, 5), 5) & "- " & Left(Cells(ActiveCell.Row, 16), 99)
        DATEINEU = Left(DATEINEU, 200) ' nur 200 Zeichen
        ' entfernen von / und \ (vom Betreff kommend)
        DATEINEU = Replace(DATEINEU, "/", "-")
        DATEINEU = Replace(DATEINEU, "\", "_")
        DATEINEU = Replace(DATEINEU, ";", " ")
        DATEINEU = Replace(DATEINEU, ":", " ")
        DATEINEU = Replace(DATEINEU, "?", " ")
        DATEINEU = Replace(DATEINEU, """", "'")
        DATEINEU = Replace(DATEINEU, "*", " ")
        DATEINEU = Replace(DATEINEU, "*", " ")
```

```
        DATEINEU = Replace(DATEINEU, "<", "(")
        DATEINEU = Replace(DATEINEU, ">", ")")
        If Right(DATEINEU, 1) = "." Then DATEINEU = Left(DATEINEU, Len(DATEINEU) - 1) ' entfernen eines Punktes am Schluss des Betreffs
        If Dir(SPEICHERPFAD & DATEINEU & DATEITYP) <> "" Then ' falls es die Datei schon gibt (weil am selben Tag schon mal archiviert wurde mit gleichem
Betreff)
           ZUFALL = Hex(Int(Rnd * 240) + 16) ' generiere zweistelligen Hexcode 10 - FF
           DATEINEU = DATEINEU & " (" & ZUFALL & ")"
        End If
        DATEINEU = DATEINEU & DATEITYP

        Sheets("FUND").Activate
        Cells(zeile, 8) = SPEICHERPFAD

        Cells(zeile, 7) = DATEINEU

        ' Datei verschieben und umbenennen
        Cells(zeile, 9) = "Name " & DATEI & " As " & SPEICHERPFAD & DATEINEU
        Name DATEI As SPEICHERPFAD & DATEINEU

WEITER: ' für alle Fälle, wo eine Datei gesucht wurde, aber nicht gefunden wurde
        zeile = zeile + 1
WEITER2: ' für alle Dateien, die bereits im Unterordner 0_BMD sind
      Next DATEI
   End If
   Application.ScreenUpdating = True
   MsgBox .FoundFiles.Count & " Dateien"
End With
Exit Sub

FEHLER:
MsgBox "Das Verzeichnis " & VERZEICHNIS & " existiert nicht."

End Sub
```

Dateien in Backupordner löschen, die älter als 10 Tage sind

```
Private Sub Workbook_Open()

Dim Vorname As String
Dim Nachname As String

Dim OrdnerExist As Boolean
Dim Ordner
```

```
Dim intZahl
Dim strExtension
Dim intTage
Dim objDatei
Dim objFSO
Dim objOrdner

Dim Dateiname   ' Dateiname ohne Dateiendung
Dim Dateiendung ' Dateityp
Dim Backupgesamtname    ' Backuppfad inklusive Sicherungsdatei-Gesamtname
Dim Monat As String     ' Aktuelles Monat mit führender Null, wenn <10
Dim Tag As String       ' Aktueller Tag mit führender Null, wenn <10
Dim Stunden As String    ' Aktuelle Stunde mit führender Null
Dim Minuten As String    ' Aktuelle Minute mit führender Null
Dim Sekunden As String    ' Aktuelle Sekunde mit führender Null
Dim Datumstempel As String ' YYYYMMDD HHMMSS


' --- Einblenden des Backupordners / bzw Erzeugen, falls es ihn noch nicht gibt

On Error GoTo ORDNER_ANLEGEN ' wenn nachfolgende SetAttr-Zeile zu Fehler führt, gibt es den betreffenden Ordner noch
nicht und wir lassen ihn von der Fehlerroutine ORDNER_ANLEGEN erzeugen

SetAttr ActiveWorkbook.Path & "\Backup DP-Leiter\", vbNormal      ' Backup Ordner einblenden / wenn es den Ordner gibt
wird er sichtbar - wenn es ihn nicht gibt, verzweigt VBA zur Ordner-Anlage und kehrt dann wieder zurück zur nächsten
Zeile WEITER

WEITER:

' --- Löscht Dateien im Backupordner die älter sind als 10 Tage ---

        Ordner = ActiveWorkbook.Path & "\Backup DP-Leiter"
        Set objFSO = CreateObject("Scripting.FileSystemObject")
        intTage = 9  ' Löscht Dateien die älter sind als 10 Tage

        Set objOrdner = objFSO.GetFolder(Ordner)
        intZahl = 0
        For Each objDatei In objOrdner.Files
             If DateDiff("d", objDatei.DateLastModified, Now) > intTage Then
                 objDatei.Delete
                 intZahl = intZahl + 1
```

```
            End If
        Next
        'MsgBox intZahl & " Dateien gelöscht."

' ---  Dieser Code dient dazu eine Arbeitsmappe nach dem Öffnen im Unterordner \Backup zu speichern mit dem Datum und der Uhrzeit im Dateinamen

    '   Auslesen des Dateinamens der aktuellen Datei

    Dateiname = Left(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") - 1)
    Dateiendung = Mid(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") + 1)

    ' Definieren des Datums- und Zeitstempels

    If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
    If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
    If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
    If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
    If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)), 2)

    Datumstempel = Year(Now) & Monat & Tag & "_" & Stunden & Minuten & Sekunden

    ' Speichern der Sicherungskopie der aktuellen Datei

    Application.DisplayAlerts = False ' nicht warnen bei Überschreiben
    Application.EnableEvents = False ' keine Events automatisch ausführen lassen

    Backupgesamtname = ActiveWorkbook.Path & "\Backup DP-Leiter\" & Dateiname & "_" & Datumstempel & "." & Dateiendung
    ActiveWorkbook.SaveCopyAs Backupgesamtname

    Application.DisplayAlerts = True ' wieder warnen bei Überschreiben
    Application.EnableEvents = True ' Events wieder automatisch ausführen lassen

    SetAttr ActiveWorkbook.Path & "\Backup DP-Leiter\", vbHidden      'Backup Ordner wieder ausblenden

' --- Abschluss

Exit Sub ' VBA verlässt diese Prozedur

' Hilfsprozedur zum Anlegen des Ordners
ORDNER_ANLEGEN:
```

```
    MkDir (ActiveWorkbook.Path & "\Backup DP-Leiter") ' Ordner erzeugen
    GoTo WEITER ' Zurückkehren zum normalen Code

End Sub
```

Dateipfad der aktuellen Datei für Datenimport im selben Verzeichnis festlegen

Möchte man den aktuellen Speicherpfad als aktuellen Pfad einstellen (zB um für einen Datenimport gleich im selben Verzeichnis zu sein wie die aktuelle Importvorlage):

```
ChDrive Left(ActiveWorkbook.FullName, 1)
ChDir Left(ActiveWorkbook.FullName, InStrRev(ActiveWorkbook.FullName, "\"))
```

Daten in CSV/TXT-Datei schreiben mit wählbarem Trennzeichen

- siehe auch: Datei erzeugen und mit Daten befüllen

Version 1 mit auswählbarem Speicherpfad (ein allgemeiner Typ, 3 Funktionen und die eigentliche Prozedur)

```
Public Type BROWSEINFO
   hOwner As Long
   pidlRoot As Long
   pszDisplayName As String
   lpszTitle As String
   ulFlags As Long
   Ipfn As Long
   IParam As Long
   iImage As Long
End Type

Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Function OrdnerAuswahl() As String
   Dim bInfo As BROWSEINFO
   Dim strPath As String
   Dim r As Long
   Dim x As Long
   Dim pos As Integer
```

```
    ' Der Ausgangsordner ist der Desktop :
    bInfo.pidlRoot = 0& ' 5& wären die Eigenen Dateien
    ' Dialogtitel
    bInfo.lpszTitle = "Wohin soll die Datei gespeichert werden ?"
    ' Rückgabe des Unterverzeichnisses
    bInfo.ulFlags = &H1
    ' Dialog anzeigen
    x = SHBrowseForFolder(bInfo)
    ' Ergebnis gliedern
    strPath = Space$(512)
    ' Ausgewähltes Verzeichnis einlesen
    r = SHGetPathFromIDList(ByVal x, ByVal strPath)
    If r Then
    pos = InStr(strPath, Chr$(0))
    OrdnerAuswahl = Left(strPath, pos - 1)
    Else
    OrdnerAuswahl = ""
    End If
End Function

Sub TXT_EXPORT()

Dim Z As Integer ' Zeilenvariable
Dim S As Integer ' Spaltenvariable
Dim DATEINAME
Dim PFAD
Dim Trennzeichen

Trennzeichen = vbTab ' oder ; ider , etc

' Auslesen des Speicherpfades
PFAD = Left(ActiveWorkbook.FullName, InStrRev(ActiveWorkbook.FullName, "\"))

' Speicherpfad bestätigen oder auswählen lassen
If MsgBox("Ist der aktuelle Speicherpfad " & vbCrLf & vbCrLf & PFAD & vbCrLf & vbCrLf & "korrekt ?", vbYesNo, "SPEICHERPFAD") = vbNo Then
    PFAD = OrdnerAuswahl
    If PFAD = "" Then
        MsgBox "Es wurde kein Speicherverzeichnis ausgewählt"
        Exit Sub
    End If
End If

' Festlegen dieses Speicherpfades
```

```
    ' On Error GoTo FEHLER
    ChDrive (PFAD)
    ChDir (PFAD)

Application.ScreenUpdating = False

    DATEINAME = PFAD & "/Monat " & Sheets("dvo").Range("E1") & ".txt"

    Open DATEINAME For Output As #1


    For Z = 1 To LETZTEZELLE("dvo").Row

        For S = 1 To 8
            ZELLWERT = ZELLWERT & ActiveCell.Value & Trennzeichen
            ActiveCell.Offset(0, 1).Select
        Next S
        Print #1, ZELLWERT
        ZELLWERT = ""
        Range("A" & Z).Select
    Next Z
    Close #1

Cells(2, 1).Select
Application.ScreenUpdating = True


    Exit Sub

FEHLER:
MsgBox "Es ist folgender Fehler aufgetreten :" & vbCrLf & vbCrLf & "' Error " & Err.Number & " - " & Err.Description & " '" & vbCrLf & vbCrLf & _
"Bitte überprüfen Sie, ob das ausgewählte Verzeichnis zum Speichern auch wirklich existiert."

End Sub
```

Version 2

```
Sub Write_Csv()
F = FreeFile(0)
fname = InputBox("Enter the filename with Path:", _
"Please Enter Output File Name:")
```

```
MsgBox "File Selected is: " & fname
If fname <> False Then
Open fname For Output As #F
Set Rng = ActiveCell.CurrentRegion
Debug.Print Rng.Address
FCol = Rng.Columns(1).Column
LCol = Rng.Columns(Rng.Columns.Count).Column
Frow = Rng.Rows(1).Row
Lrow = Rng.Rows(Rng.Rows.Count).Row
For i = Frow To Lrow
outputLine = ""
For j = FCol To LCol
If j <> LCol Then
'Semikolon als Texttrennzeichen, kann geändert werden
outputLine = outputLine & Cells(i, j) & ";"
Else
outputLine = outputLine & Cells(i, j)
End If
Next j
Print #F, outputLine
Next i
Close #F
End If
End Sub
```

Version 3

```
Sub schreibeCSV()
F = FreeFile(0)
fname = InputBox("Bitte Pfad und Dateinamen der Zieldatei eingeben (z.B.
c:\tmp\text.csv):", _
"Eingabe Pfad und Dateiname")
MsgBox "Der Name der Ausgabedatei lautet: " & fname
fseparator = InputBox("Bitte das Trennzeichen eingeben:", _
"Eingabe Trennzeichen")
MsgBox "Das gewählte Trennzeichen ist: " & fseparator
If fname <> False Then
Open fname For Output As #F
Set Rng = ActiveCell.CurrentRegion
Debug.Print Rng.Address
FCol = Rng.Columns(1).Column
LCol = Rng.Columns(Rng.Columns.Count).Column
Frow = Rng.Rows(1).Row
Lrow = Rng.Rows(Rng.Rows.Count).Row
For i = Frow To Lrow
outputLine = ""
For j = FCol To LCol
If j <> LCol Then
outputLine = outputLine & Cells(i, j) & fseparator
Else
outputLine = outputLine & Cells(i, j)
End If
```

```
Next j
Print #F, outputLine
Next i
Close #F
End If
MsgBox "Vorgang abgeschlossen!"
End Sub
```

Version 4 siehe Powertools

Daten aus einer CSV auslesen

```
Sub IMPORT_IB_GESAMT()

' Mit der Import-Taste in der Tabelle IB-GESAMT werden die Daten aus der CSV-Datei IB.csv importiert,
' die sich optimalerweise im gleichen Ordner befinden soll, oder in einem Ordner, der in der Tabelle
' Einstellungen eingetragen ist - ansonsten öffnet sich ein Dateiauswahlfenster
' Der Name der Datei (IB.csv) kann ebenfalls in den Einstellungen G114 geändert werden

   Dim iReihe As Long, iSpalte As Long
   Dim DATEI As String, iText As String, Trennzeichen As String
   Dim I As Long
   Dim T As Integer
   Dim WERT


' Auswahl der Datei

' 1.) suchen, ob sie existiert im gleichen Verzeichnis
' 2.) suchen, ob sie existiert im Verzeichnis, das in Tabelle Einstellungen eingegeben ist in Zelle G115
' 3.) suchen mittels Dateiauswahlsfenster



' 1.) im aktuellen Verzeichnis suchen
If Dir(Sheets("Einstellungen").Range("G114")) <> "" Then 'zuerst im aktuellen Verzeichnis suchen, wo Importfile ist
    'MsgBox "Gibts im aktuellen Verzeichnis"
    DATEI = Dir(Sheets("Einstellungen").Range("G114"))
  Else
```

```
    ' 2.) dann nach der Datei im angegebenen Verzeichnis suchen, das in Tabelle "Einstellungen" eingegeben ist
    ' fehlendes \ am Ende wird dort eingefügt
    If Right(Sheets("Einstellungen").Range("G115"), 1) <> "\" Then Sheets("Einstellungen").Range("G115") = Sheets("Einstellungen").Range("G115") & "\"

    If Dir(Sheets("Einstellungen").Range("G115") & Sheets("Einstellungen").Range("G114")) <> "" Then
        'MsgBox "Gibts im eingestellten Verzeichnis"
        DATEI = Dir(Sheets("Einstellungen").Range("G115") & Sheets("Einstellungen").Range("G114"))
        ChDrive (Left(Sheets("Einstellungen").Range("G115"), 1))
        ChDir (Sheets("Einstellungen").Range("G115"))

      Else

        ' 3.) sonst die Auswahldialogbox anbieten zum Öffnen
        DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)
        ' Speichern des neuen Pfades für die Zukunft in Tabelle "Einstellungen"
        For T = 0 To Len(DATEI) - 1
            If Mid(DATEI, Len(DATEI) - T, 1) = "\" Then
                Sheets("Einstellungen").Range("G115") = Mid(DATEI, 1, Len(DATEI) - T)
                GoTo Fertig
            End If
        Next T
    End If
End If

Fertig:




   If Dir(DATEI) = "" Then
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
   End If

Sheets("IB-Gesamt").Range("G6") = Date

   Trennzeichen = Sheets("Einstellungen").Range("G140")
   If Left(UCase(Trennzeichen), 3) = "TAB" Then Trennzeichen = Chr$(9)
   iReihe = 1
   iSpalte = 1
   Close ' Sicherheitshalber eventuell offene Dateien schließen

   Sheets("IB-Gesamt").Range("D10:S9000").ClearContents
   Sheets("Rohdaten").Range("A1:FF65000").ClearContents
```

```
Application.ScreenUpdating = False:  ' Bild nicht aktualisieren
```

' Einspielen der Daten - Variante 1 unverändert

```
  Open DATEI For Input As #1
    Do Until EOF(1)
      Line Input #1, iText
      Do While InStr(iText, Trennzeichen)
        Sheets("Rohdaten").Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, Trennzeichen) - 1)
        iText = Right(iText, Len(iText) - InStr(iText, Trennzeichen))
        iSpalte = iSpalte + 1
      Loop
      Sheets("Rohdaten").Cells(iReihe, iSpalte).Value = iText
      iReihe = iReihe + 1
      iSpalte = 1
    Loop
  Close

   'Sheets("Rohdaten").Columns("A:FF").EntireColumn.AutoFit
```

ALTERNATIV:

' Einspielen der Daten - Variante 2 mit entfernen etwaiger " in der Quelldatei (wenn csv-Datei bereits "xy"-Werte hat statt xy)

```
  Open DATEI For Input As #1
    Do Until EOF(1)
      Line Input #1, iText
      Do While InStr(iText, TRENNZEICHEN)
        Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHEN) - 1)
        If Left(Cells(iReihe, iSpalte), 1) = """" And Right(Cells(iReihe, iSpalte), 1) = """" Then
           Cells(iReihe, iSpalte) = Mid(Cells(iReihe, iSpalte), 2, Len(Cells(iReihe, iSpalte)) - 2)
        End If
        iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHEN))
        iSpalte = iSpalte + 1
      Loop
      If Left(iText, 1) = """" And Right(iText, 1) = """" Then iText = Mid(iText, 2, Len(iText) - 2)
      Cells(iReihe, iSpalte).Value = iText
      iReihe = iReihe + 1
      iSpalte = 1
    Loop
```

Close

ALTERNATIVE 2

' wenn das Trennzeichen (,) in Textteilen vorkommen darf, die in "" stehen
' z.B. wenn beim Datensatz 10,201001,"Meier, Hansen und Co Gmbh",50
' der Beistrich zwischen den beiden " kein Datensatztrennzeichen ist

```
  Open DATEI For Input As #1
    Do Until EOF(1)
      Line Input #1, iText
      ' Wenn zwischen zwei Anführungszeichen das Trennzeichen , vorkommt, obwohl es an dieser Stelle kein Datensatz-Trennzeichen
      ' sondern ein Beistrich zB im Firmennamen ist, dann soll dieser Beistrich ja nicht als Trennzeichen gelten.
      ' BSP 1,10,"Creative, Media Gmbh",4200,50 - alle Beistriche sind ein Trennzeichen, außer der Bestrich zwischen den beiden ".
      ' Lösung: wir wandeln zuerst alle Trennzeichen in ein neues Trennzeichen um, außer das Trennzeichen kommt zwischen zwei " vor
      For i = 1 To Len(iText)
        If Mid(iText, i, 1) = """" And WORTMODUS = 0 Then
          WORTMODUS = 1 ' beim ersten " wird der Wortmodus auf 1 gestellt - daher alle nachfolgenden Trennzeichen bleiben ein Satzzeichen
        Else
          WORTMODUS = 0 ' beim zweiten " wird der Wortmodus auf 0 zurückgesetzt - daher alle nachfolgenden Trennzeichen sind ein echtes Trennzeichen
und werden umgewandelt in neues Trennzeichen
        End If
        If Mid(iText, i, 1) = TRENNZEICHEN Then
          If WORTMODUS = 1 Then
            iTextNeu = iTextNeu & Mid(iText, i, 1) ' wenn Trennzeichen zwischen 2 " vorkommt, es 1:1 übernehmen
          Else
            iTextNeu = iTextNeu & TRENNZEICHENNEU
          End If
        Else ' wenn kein Trennzeichen: es normal übernehmen
          iTextNeu = iTextNeu & Mid(iText, i, 1)
        End If
      Next i
      iText = iTextNeu
      iTextNeu = ""
      Do While InStr(iText, TRENNZEICHENNEU)
        Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHENNEU) - 1) ' Einfügen des jeweiligen Teils zwischen zwei Trennzeichen
        iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHENNEU))
        iSpalte = iSpalte + 1
      Loop
      Cells(iReihe, iSpalte).Value = iText ' Einfügen der letzten Zelle
```

```
      iReihe = iReihe + 1
      iSpalte = 1
    Loop
  Close
```

Daten aus einer Textdatei mit Trennzeichen , einlesen

Dies ist die Importprozedur, die die Lexorhonorarnoten importiert, die mit Beistrich als Trennzeichen aus Lexor kommen.

Besonderheit, Textfelder werden mit Anführungszeichen übergeben und sollen das Trennzeichen , enthalten dürfen:
Beispiel: die ersten beiden Beistriche sind außerhalb von "" und sind daher echte Trennzeichen - der Beistrich in "IneoQuest Technologies, Inc" hingegen ist Teil des Firmennamens und soll erhalten bleiben.

10,1591180,"IneoQuest Technologies, Inc","IneoQuest","",03112006,

```
Sub IMPORTIEREN()

  Dim iReihe As Long, iSpalte As Long
  Dim DATEI As String, iText As String, TRENNZEICHEN As String
  Dim TRENNZEICHENNEU As String
  Dim WORTMODUS As Integer
  Dim iTextNeu
  Dim i As Long
  Dim WERT
  Dim Test As String

Dim VONPFAD As String
Dim NACHPFAD As String
Dim DATEINAME As String
Dim DATEINAMENEU As String
Dim BMDORDNER As String

  Sheets("Import").Range("A1:IV65000").ClearContents
  Sheets("Import").Activate

  VONPFAD = Sheets("Stamm").Range("K16")
  If Right(VONPFAD, 1) <> "/" And Right(VONPFAD, 1) <> "\" Then VONPFAD = VONPFAD & "/"
  NACHPFAD = Sheets("Stamm").Range("K18")
  If Right(NACHPFAD, 1) <> "/" And Right(NACHPFAD, 1) <> "\" Then NACHPFAD = NACHPFAD & "/"

'Aktuelles Verzeichnis dieser Vorlage festlegen als aktives Verzeichnis

  ChDrive Left(VONPFAD, 1)
```

```
   ChDir (VONPFAD)

' Auswahl der Datei

AUSWAHL:
    DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)

    'On Error GoTo KEINEDATEIAUSGEWÄHLT

    ' nicht notwendiges Herausfiltern des Dateinamens zu Testzwecken

    Wertlänge = Len(WERT)
    For i = 1 To Wertlänge
        Wertrechts = Right(WERT, k)
        Slash = Left(Wertrechts, 1)
        Select Case Slash
            Case Is = "\"
            Wertname = Right(WERT, k - 1)
            'MsgBox Wertname
            GoTo WEITER
        End Select
    Next i
WEITER:


   If Dir(DATEI) = "" Then
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
   End If

   TRENNZEICHEN = ","
   TRENNZEICHENNEU = "~"
   iReihe = 1
   iSpalte = 1
   Close ' Sicherheitshalber eventuell offene Dateien schließen

   DATEINAME = Mid(DATEI, InStrRev(DATEI, "\") + 1)

' Einspielen der Daten

   Open DATEI For Input As #1
      Do Until EOF(1)
         Line Input #1, iText
         ' Wenn zwischen zwei Anführungszeichen das Trennzeichen , vorkommt, obwohl es an dieser Stelle kein Datensatz-Trennzeichen
```

```
        ' sondern ein Beistrich zB im Firmennamen ist, dann soll dieser Beistrich ja nicht als Trennzeichen gelten.
        ' BSP 1,10,"Creative, Media Gmbh",4200,50 - alle Beistriche sind ein Trennzeichen, außer der Bestrich zwischen den beiden ".
        ' Lösung: wir wandeln zuerst alle Trennzeichen in ein neues Trennzeichen um, außer das Trennzeichen kommt zwischen zwei " vor
        For i = 1 To Len(iText)
          If Mid(iText, i, 1) = """" Then
            If WORTMODUS = 0 Then
              WORTMODUS = 1 ' beim ersten " wird der Wortmodus auf 1 gestellt - daher alle nachfolgenden Trennzeichen bleiben ein Satzzeichen
              ' MsgBox "wortmodus1" & i
            Else
              WORTMODUS = 0 ' beim zweiten " wird der Wortmodus auf 0 zurückgesetzt - daher alle nachfolgenden Trennzeichen sind ein echtes Trennzeichen
und werden umgewandelt in neues Trennzeichen
              ' MsgBox "wortmodus0" & i
            End If
          End If
          If Mid(iText, i, 1) = TRENNZEICHEN Then
            ' MsgBox i & " - " & WORTMODUS
            If WORTMODUS = 1 Then
              iTextNeu = iTextNeu & Mid(iText, i, 1) ' wenn Trennzeichen zwischen 2 " vorkommt, es 1:1 übernehmen
              'MsgBox Mid(iText, i, 1) & " - " & i & WORTMODUS
            Else
              iTextNeu = iTextNeu & TRENNZEICHENNEU
              'MsgBox TRENNZEICHENNEU & " - " & i & WORTMODUS
            End If
          Else ' wenn kein Trennzeichen: es normal übernehmen
            iTextNeu = iTextNeu & Mid(iText, i, 1)
          End If
          ' MsgBox i & " - - " & WORTMODUS
        Next i
        ' MsgBox iText & vbCrLf & iTextNeu
        iText = iTextNeu
        iTextNeu = ""
        Do While InStr(iText, TRENNZEICHENNEU)
          Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, TRENNZEICHENNEU) - 1) ' Einfügen des jeweiligen Teils zwischen zwei Trennzeichen
          iText = Right(iText, Len(iText) - InStr(iText, TRENNZEICHENNEU))
          iSpalte = iSpalte + 1
        Loop
        Cells(iReihe, iSpalte).Value = iText ' Einfügen der letzten Zelle
        iReihe = iReihe + 1
        iSpalte = 1
      Loop
    Close

    Columns("A:FF").EntireColumn.AutoFit
    Cells(1, 1).Select
```

```
    If CDbl(Cells(1, 2)) <> CDbl(Sheets("Stamm").Range("K24")) Then
        MsgBox "Die importierten Daten stammen vom Betrieb " & Cells(1, 2) & " und die Vorlage hier ist für den Betrieb " & Sheets("Stamm").Range("K24") &
vbCrLf & vbCrLf & _
        "Bitte den Vorgang mit den richtigen Daten bzw der richtigen Vorlage wiederholen"
        End
    End If

' Verschieben der Importdatei
DATEINAMENEU = Left(DATEINAME, Len(DATEINAME) - 4) & "-" & Date & "-" & Hour(Now) & "_" & Minute(Now) & "_" & Second(Now) & Right(DATEINAME, 4)

If Sheets("Stamm").Range("K4") = True Then
    CreateObject("Scripting.FileSystemObject").MoveFile VONPFAD & DATEINAME, NACHPFAD & DATEINAMENEU
End If

' BMD-Ordner entmisten

If Sheets("Stamm").Range("K26") = True Then

   VONPFAD = Sheets("Stamm").Range("K6")
   If Right(VONPFAD, 1) <> "/" And Right(VONPFAD, 1) <> "\" Then VONPFAD = VONPFAD & "\"

   ChDrive Left(VONPFAD, 1)
   ChDir (VONPFAD)

   Shell VONPFAD & "AUSMISTN.bat"
End If

Exit Sub

KEINEDATEIAUSGEWÄHLT:

MsgBox "Es ist folgender Fehler aufgetreten " & vbCrLf & vbCrLf & "(" & Err.Number & " " & Err.Description & ")"

End Sub
```

Textdatei importieren in Excel

1. Beispiel
Wenn ich eine *.txt-Datei mit Semikolon als Feldtrenner manuell öffne, wird dieser erkannt, öffne ich die Datei mit Makro, wird das Semikolon nicht erkannt.
Wie kann ich das ändern?

StandardModule: Modul1

```
Sub OpenText()
```

```
  Dim sFile As String
  sFile = Application.Path & "\texttest.txt"
  Open sFile For Output As #1
  Print #1, "Text 1; Text 2; Text 3"
  Close
  Workbooks.OpenText _
    FileName:=sFile, _
    DataType:=xlDelimited, _
    semicolon:=True
  MsgBox "Weiter"
  ActiveWorkbook.Close savechanges:=False
  Kill sFile
End Sub
```

2. Beispiel:

Die in Zelle P1 genannte Textdatei soll importiert werden. Es handelt sich um eine Texttdatei mit Semikoli als Feldtrenner.

StandardModule: Modul1

```
Sub TextImport()
  Dim iRow As Integer, iCol As Integer
  Dim sFile As String, sTxt As String
  sFile = Range("P1").Value
  If Dir(sFile) = "" Then
    Beep
    MsgBox "Datei wurde nicht gefunden!"
    Exit Sub
  End If
  iRow = 1
  iCol = 1
  Close
  Open sFile For Input As #1
    Do Until EOF(1)
      Line Input #1, sTxt
      Do While InStr(sTxt, ";")
        Cells(iRow, iCol).Value = Left(sTxt, InStr(sTxt, ";") - 1)
        sTxt = Right(sTxt, Len(sTxt) - InStr(sTxt, ";"))
        iCol = iCol + 1
      Loop
      Cells(iRow, iCol).Value = sTxt
      iRow = iRow + 1
      iCol = 1
    Loop
```

```
  Close
End Sub
```

3. Beispiel

Werte aus Textdatei importieren und umwandeln

Aus einer Textdatei sollen Werte importiert und umgewandelt werden. Handelt es sich bei dem rechten Zeichen im Wert um ein H, ist der Wert positiv, bei einem S ist er negativ. Die Werte werden in die Tabelle eingetragen.

StandardModule: Modul1

```
Sub Importieren()
  Dim arr As Variant
  Dim vValue As Variant
  Dim iRow As Integer, iCol As Integer
  Dim sFile As String, sTxt As String
  sFile = ThisWorkbook.Path & "\setminus.txt"
  If Dir(sFile) = "" Then
    Beep
    MsgBox "Kann die Textdatei " & sFile & " nicht finden!"
    Exit Sub
  End If
  Close
  Open sFile For Input As #1
  Do Until EOF(1)
    iRow = iRow + 1
    Line Input #1, sTxt
    arr = fncSplit(sTxt, vbTab)
    For iCol = 0 To UBound(arr)
      vValue = arr(iCol)
      If Right(vValue, 1) = "H" Then
        vValue = CDbl(Left(vValue, Len(vValue) - 1))
      ElseIf Right(vValue, 1) = "S" Then
        vValue = CDbl(Left(vValue, Len(vValue) - 1)) * -1
      End If
      Cells(iRow, iCol + 1).Value = vValue
    Next iCol
  Loop
  Close
End Sub

Function fncSplit( _
  SplitText As String, _
```

```
  Delimiter As Variant _
  ) As Variant
  Dim iFile As Integer, iCounter As Integer
  Dim arr() As String
  Dim sTxt As String
  Do While SplitText <> ""
    ReDim Preserve arr(iCounter)
    If InStr(SplitText, Delimiter) Then
      sTxt = Left(SplitText, _
        InStr(SplitText, Delimiter) - 1)
      arr(iCounter) = sTxt
      SplitText = Right(SplitText, Len(SplitText) - _
        InStr(SplitText, Delimiter))
    Else
      arr(iCounter) = SplitText
      SplitText = ""
    End If
    iCounter = iCounter + 1
  Loop
  fncSplit = arr
End Function
```

Meine Prozedur zum Import, wobei in Tabelle "parameters" in Zelle E115 das Trennzeichen steht, in G114 der Dateiname und in G115 der Speicherpfad

```
' Mit der Import-Taste in der Tabelle IB-GESAMT werden die Daten aus der CSV-Datei HH.csv importiert,
' die sich optimalerweise im gleichen Ordner befinden soll, oder in einem Ordner, der in der Tabelle
' parameters eingetragen ist - ansonsten öffnet sich ein Dateiauswahlfenster

  Dim iReihe As Long, iSpalte As Long
  Dim DATEI As String, iText As String, Trennzeichen As String
  Dim I As Long
  Dim T As Integer
  Dim WERT


' Auswahl der Datei - 3 Suchmöglichkeiten

' 1.) suchen, ob sie existiert im gleichen Verzeichnis
' 2.) suchen, ob sie existiert im Verzeichnis, das in Tabelle parameters eingegeben ist in Zelle G115
' 3.) suchen mittels Dateiauswahlsfenster
```

```
' 1.) im aktuerllen Verzeichnis suchen
If Dir(Sheets("parameters").Range("G114")) <> "" Then 'zuerst im aktuellen Verzeichnis suchen, wo Importfile ist
    MsgBox "Gibts im aktuellen Verzeichnis"
    DATEI = Dir(Sheets("parameters").Range("G114"))
  Else

    ' 2.) dann nach der Datei im angegebenen Verzeichnis suchen, das in Tabelle "parameters" eingegeben ist
    ' fehlendes \ am Ende wird dort eingefügt
    If Right(Sheets("parameters").Range("G115"), 1) <> "\" Then Sheets("parameters").Range("G115") = Sheets("parameters").Range("G115") & "\"

    If Dir(Sheets("parameters").Range("G115") & Sheets("parameters").Range("G114")) <> "" Then
       MsgBox "Gibts im eingestellten Verzeichnis"
       DATEI = Dir(Sheets("parameters").Range("G115") & Sheets("parameters").Range("G114"))
       ChDrive (Left(Sheets("parameters").Range("G115"), 1))
       ChDir (Sheets("parameters").Range("G115"))

      Else

       ' 3.) sonst die Auswahldialogbox anbieten zum Öffnen
       DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)
       ' Speichern des neuen Pfades für die Zukunft in Tabelle "parameters"
       For T = 0 To Len(DATEI) - 1
          If Mid(DATEI, Len(DATEI) - T, 1) = "\" Then
             Sheets("parameters").Range("G115") = Mid(DATEI, 1, Len(DATEI) - T)
             GoTo Fertig
          End If
       Next T
    End If
End If

Fertig:



   If Dir(DATEI) = "" Then
      MsgBox "Es wurde keine Datei ausgewählt"
      Exit Sub
   End If

   Trennzeichen = Sheets("parameters").Range("E115")
   If Left(UCase(Trennzeichen), 3) = "TAB" Then Trennzeichen = Chr$(9)
   iReihe = 1
   iSpalte = 1
   Close ' Sicherheitshalber eventuell offene Dateien schließen
```

```
  Sheets("Rohdaten").Range("A1:FF65000").ClearContents


' Einspielen der Daten

  Open DATEI For Input As #1
    Do Until EOF(1)
      Line Input #1, iText
      Do While InStr(iText, Trennzeichen)
        Sheets("Rohdaten").Cells(iReihe, iSpalte).Value = Left(iText, InStr(iText, Trennzeichen) - 1)
        iText = Right(iText, Len(iText) - InStr(iText, Trennzeichen))
        iSpalte = iSpalte + 1
      Loop
      Sheets("Rohdaten").Cells(iReihe, iSpalte).Value = iText
      iReihe = iReihe + 1
      iSpalte = 1
    Loop
  Close

   Sheets("Rohdaten").Activate
   Columns("A:FF").EntireColumn.AutoFit

Exit Sub
```

WEITERE VERSIONEN

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Textimport&action=edit&section=T-1

Import zur Anzeige in MsgBoxes

Beim Import mit der Funktion Line Input sucht Excel nach Zeichen, die das Zeilenende ankündigen. Wurde eine Datei unter Windows geschrieben, endet eine Zeile üblicherweise mit zwei Zeichen: CHR(13) und CHR(10), also Wagenrücklauf (CR = Carriage Return) und Zeilenvorschub (LF = LineFeed). Mac-Dateien enden üblicherweise mit CHR(13) und Unix-Dateien enden üblicherweise mit CHR(10). 'Üblicherweise' meint, dass dies für Textdateien gilt, die das Betriebssystem schreibt und die als Konvention auch so von vielen Anwendungen von ihrem jeweiligen Betriebssystem übernommen wird. Es gibt aber auch Anwendungen, die auf mehreren Betriebssystemen laufen und andere oder überall die gleiche Konvention für das Zeilenende verwenden.

Excel gibt es für Windows und Mac, daher werden von Line Input sowohl CR+LF als auch CR als Zeilenendzeichen erkannt. Ein einfaches LF oder andere Symbole werden versteht Excel nicht als Zeilenende und liest dann so lange ein, bis der Puffer voll ist – die eingelesene Zeichenfolge kann in diesem Falle mehrere zehntausend Byte lang werden.

```
Sub WriteInMsgBoxes()
   Dim cln As New Collection
   Dim arrAct As Variant
   Dim intNo As Integer, intCounter As Integer
   Dim txt As String, strMsg As String
   Dim bln As Boolean
   intNo = FreeFile
   Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intNo
   Do Until EOF(intNo)
      If bln = False Then
         Line Input #intNo, txt
         arrAct = SplitString(txt, ",")
         For intCounter = 1 To UBound(arrAct)
            cln.Add arrAct(intCounter)
         Next intCounter
      Else
         Line Input #intNo, txt
         arrAct = SplitString(txt, ",")
         For intCounter = 1 To UBound(arrAct)
            strMsg = strMsg & cln(intCounter) & ": " & _
               arrAct(intCounter) & vbLf
         Next intCounter
      End If
      If bln Then MsgBox strMsg
      bln = True
      strMsg = ""
   Loop
   Close intNo
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Textimport&action=edit&section=T-2

Import zur Konvertierung in eine HTML-Seite

```
Sub WriteInHTML()
   Dim arrAct As Variant
   Dim intSource, intTarget, intCounter As Integer
   Dim txt, strTag As String
   Dim bln As Boolean
   intTarget = FreeFile
   Open ThisWorkbook.Path & "\TextImport.htm" For Output As #intTarget
```

```
   Print #intTarget, "<html><body><table>"
   intSource = FreeFile
   Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource
   Do Until EOF(intSource)
      If bln Then strTag = "td" Else strTag = "th"
      Line Input #intSource, txt
      arrAct = SplitString(txt, ",")
      Print #intTarget, "<tr>"
      For intCounter = 1 To UBound(arrAct)
         Print #intTarget, "<" & strTag & ">" & arrAct(intCounter) & "</" & strTag & ">"
      Next intCounter
      Print #intTarget, "</tr>"
      bln = True
   Loop
   Close intSource
   Print #intTarget, "</table></body></html>"
   Close intTarget
   Shell "hh " & ThisWorkbook.Path & "\TextImport.htm", vbMaximizedFocus
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Textimport&action=edit&section=T-3

Import zur Anzeige in einem Arbeitsblatt

```
Sub WriteInWks()
   Dim cln As New Collection
   Dim arrAct As Variant
   Dim intSource As Integer, intRow As Integer, intCol As Integer
   Dim txt As String
   Workbooks.Add
   intSource = FreeFile
   Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource
   Do Until EOF(intSource)
      Line Input #intSource, txt
      arrAct = SplitString(txt, ",")
      intRow = intRow + 1
      For intCol = 1 To UBound(arrAct)
         Cells(intRow, intCol).Value = arrAct(intCol)
      Next intCol
   Loop
   Close intSource
   Rows(1).Font.Bold = True
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Textimport&action=edit&section=T-4

Import zur Übernahme in UserForm-Controls

In einem Standardmodul:

```
Public garr() As String
Public gint As Integer
```

Im Klassenmodul der UserForm:

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

Private Sub cmdWeiter_Click()
   Dim intCounter As Integer
   If gint <= 4 Then gint = gint + 1 Else gint = 1
   For intCounter = 1 To 5
      Controls("TextBox" & intCounter).Text = garr(gint, intCounter)
   Next intCounter
End Sub

Private Sub UserForm_Initialize()
   Dim arrAct As Variant
   Dim intSource As Integer, intCounter As Integer, intRow As Integer
   Dim txt As String
   Dim bln As Boolean
   gint = 0
   intSource = FreeFile
   Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource
   Do Until EOF(intSource)
      Line Input #intSource, txt
      arrAct = SplitString(txt, ",")
      If bln = False Then
         For intCounter = 1 To UBound(arrAct)
            Controls("Label" & intCounter).Caption = _
               arrAct(intCounter) & ":"
         Next intCounter
         ReDim garr(1 To 5, 1 To UBound(arrAct))
      Else
         intRow = intRow + 1
         For intCounter = 1 To UBound(arrAct)
            garr(intRow, intCounter) = arrAct(intCounter)
         Next intCounter
      End If
      bln = True
   Loop
   Close intSource
End Sub
```

Für alle vorstehende Routinen wird die folgende benutzerdefinierte Funktion in einem Standardmodul benötigt (Die Funktion macht unabhängig von der erst ab XL2000 verfügbaren VBA-Funktion **Split**:

```
Function SplitString(ByVal txt As String, strSeparator As String)
   Dim arr() As String
   Dim intCounter As Integer
   Do
      intCounter = intCounter + 1
      ReDim Preserve arr(1 To intCounter)
      If InStr(txt, strSeparator) Then
         arr(intCounter) = Left(txt, InStr(txt, strSeparator) - 1)
         txt = Right(txt, Len(txt) - InStr(txt, strSeparator))
      Else
         arr(intCounter) = txt
         Exit Do
      End If
   Loop
   SplitString = arr
End Function
```

How to Rename a File or Directory

Did you know there is a built-in statement in VB6/VBA/Office/Access/Excel that allows you to do this without using API calls and without referencing the File System?

From the VB6, VBA, Office/Access Help File

The Name statement moves the file to the new directory or folder and renames the file, if necessary. Here's the syntax:

```
Name OldPathName As NewPathName
```

Name can move a file across drives, but it can only rename an existing directory or folder when both OldPathName and NewPathName are located on the same drive. Name cannot create a new file, directory, or folder.

If OldPathName and NewPathName have different paths, and the same file name, the Name statement moves the file to the new location and leaves the file name unchanged.

Using Name, you can move a file from one directory or folder to another, but you cannot move a directory or folder.

Using Name on an open file produces an error. You must close an open file before renaming it. Name arguments cannot include multiple-character (*) and single-character (?) wildcards.

Note

Our example below only provides support for files or simple folder renaming (e.g. the path folder does not exist). Renaming a directory to a name that already exists is more complicated and may be included in a future tip.

Sample Code

```
Public Function TestNameStatement()
Dim fOK As Boolean
' Folders must exist for Source, but do not need to exist _
' for destination
fOK = RenameFileOrDir("C:\TestFolder\test.txt", _
"C:\TestFolder\test_NEWNAME.txt")
fOK = RenameFileOrDir("C:\TestFolder\test.txt", _
"D:\TestFolder\test_NEWNAME.txt")
' Folder must exist for source
fOK = RenameFileOrDir("C:\TestFolder", _
"C:\TestFolder_NEWNAME")
' Folders only will fail across drives
fOK = RenameFileOrDir("C:\TestFolder", "D:\TestFolder")
End Function
Public Function RenameFileOrDir( _
ByVal strSource As String, _
ByVal strTarget As String, _
Optional fOverwriteTarget As Boolean = False) As Boolean
On Error GoTo PROC_ERR
Dim fRenameOK As Boolean
Dim fRemoveTarget As Boolean
Dim strFirstDrive As String
Dim strSecondDrive As String
Dim fOK As Boolean
If Not ((Len(strSource) = 0) Or _
(Len(strTarget) = 0) Or _
(Not (FileOrDirExists(strSource)))) Then
' Check if the target exists
If FileOrDirExists(strTarget) Then
If fOverwriteTarget Then
fRemoveTarget = True
```

```
Else
If vbYes = MsgBox("Do you wish to overwrite the " & _
"target file?", vbExclamation + vbYesNo, _
"Overwrite confirmation") Then
fRemoveTarget = True
End If
End If
If fRemoveTarget Then
' Check that it's not a directory
If ((GetAttr(strTarget) And vbDirectory)) <> _
vbDirectory Then
Kill strTarget
fRenameOK = True
Else
MsgBox "Cannot overwrite a directory", vbOKOnly, _
"Cannot perform operation"
'FUTURE CODE FOR DIRECTORIES
End If
End If
Else
' The target does not exist
' Check if source is a directory
If ((GetAttr(strSource) And vbDirectory) = _
vbDirectory) Then
' Source is a directory, see if drives are the same
strFirstDrive = Left(strSource, InStr(strSource, ":\"))
strSecondDrive = Left(strTarget, InStr(strTarget, ":\"))
If strFirstDrive = strSecondDrive Then
fRenameOK = True
Else
MsgBox "Cannot rename directories across drives", _
vbOKOnly, "Cannot perform operation"
'FUTURE CODE FOR DIRECTORIES ON DIFFERENT DRIVES
End If
Else
'It's a file, ok to proceed
fRenameOK = True
End If
End If
```

```
If fRenameOK Then
Name strSource As strTarget
fOK = True
End If
End If
RenameFileOrDir = fOK
PROC_EXIT:
Exit Function
PROC_ERR:
MsgBox "Error: " & Err.Number & ". " & Err.Description, , _
"RenameFileOrDir"
Resume PROC_EXIT
End Function
Public Function FileOrDirExists(strDest As String) As Boolean
Dim intLen As Integer
Dim fReturn As Boolean

fReturn = False

If strDest <> vbNullString Then
On Error Resume Next
intLen = Len(Dir$(strDest, vbDirectory + vbNormal))
On Error GoTo PROC_ERR
fReturn = (Not Err And intLen > 0)
End If

PROC_EXIT:
FileOrDirExists= fReturn
Exit Function

PROC_ERR:
MsgBox "Error: " & Err.Number & ". " & Err.Description, , _
"FileOrDirExists"
Resume PROC_EXIT
End Function
```

SAMPLE CODE 2

```
Option Explicit
```

```
Sub ToggleStuff(ByVal x As Boolean)
Application.ScreenUpdating = x
Application.EnableEvents = x
End Sub
Sub HideHints()
'removes data validation input comments which appear as mouse over tips
ToggleStuff False
Dim vRng As Range, c As Range
Dim ws As Worksheet
Set ws = Sheet1
Set vRng = ws.Range("A1:D500")

vRng.Validation.Delete
Set ws = Nothing: Set vRng = Nothing: Set c = Nothing
ToggleStuff True
End Sub

Sub ShowHints()
'replaces the validation input comments which appear as mouse over tips
ToggleStuff False
On Error Resume Next
Dim vRng As Range, c As Range
Dim ws As Worksheet
Set ws = Sheet1
Set vRng = ws.Range("A2:A500")
With vRng.Validation
       .Delete 'deletes the old validation, to prevent errors when replacing it
       .Add (xlValidateInputOnly) 'this is the type of validation
       .InputTitle = "Bonus"
       .InputMessage = "If you leave this blank, all files in " _
         & "the folder will be copied to the destination.  Save time with Bulk Backups!"
End With

Set vRng = Range("B2:B500")
With vRng.Validation
        .Delete
        .Add (xlValidateInputOnly)
        .InputMessage = "Double Click to Browse for a folder."
        .InputTitle = "Note:"
End With
Set vRng = Range("C2:C500")
With vRng.Validation
        .Delete
        .Add (xlValidateInputOnly)
```

```
        .InputMessage = "If the directory does not exist one will be created." & _
            "Double Click to Browse for a folder"
        .InputTitle = "Note:"
End With
Set vRng = Range("D2:D500")
With vRng.Validation
        .Delete
        .Add (xlValidateInputOnly)
        .InputMessage = "If you select YES, the file will ONLY be moved, and not copied." & _
                    "No or left blank will default to COPY"
        .InputTitle = "Move or Copy:"
End With
Set ws = Nothing: Set vRng = Nothing: Set c = Nothing
ToggleStuff True
End Sub

Sub PrintCopy()
'Prints the occupied range of the worksheet
Dim pRng As Range, ws As Worksheet
Set ws = Sheet1
Set pRng = ws.Range(ws.Cells(1, 1), Cells(500, 1).End(xlUp)).Resize(, 7)
ws.PageSetup.PrintArea = pRng.Address 'define the print area as the occupied range
'set some default printer settings..
 With ws.PageSetup
      .CenterHorizontally = False
      .CenterVertically = True
      .FitToPagesTall = 1
      .FitToPagesWide = 3
      .Orientation = xlLandscape
 End With
     ws.PrintOut 'print the ws
Set pRng = Nothing: Set ws = Nothing

End Sub

Sub Saveme()
'save the file now
Application.DisplayAlerts = False
ThisWorkbook.Save
Application.DisplayAlerts = True
End Sub

Sub CopyRoutine()
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
'''Logic method: Cycle through and evaluate listed files,'''''
'''and based on contents of cells, determine options presented'
'Declare all variables
Dim fRng As Range, c As Range, ws As Worksheet
Dim objFSO As FileSystemObject, objFolder As Folder, PathExists As Boolean
Dim objFile As File, strSourceFolder As String, strDestFolder As String
Dim x, Counter As Integer, FoldCount As Integer, Overwrite As String, strFileName As String
Dim strMove As String, strRename As String, OverwriteWarning As Boolean
Dim strF2 As String, strMid As String, strExt As String

ToggleStuff False 'turn off events and screenupdating

Set ws = Sheet1

OverwriteWarning = False 'Boolean flag, get's set True after warning and execution
Range("G2:G500") = "" 'clear the old comments
Counter = 0 'start the file counter
FoldCount = 0 'start the folder counter
With ws 'designate all actions subsequent to be related to the Sheet1

Set fRng = .Range(.Cells(2, 2), .Cells(500, 2).End(xlUp)) 'range to look

For Each c In fRng 'for each occupied cell in above range(column B)
Backformore: ' a location to return to after doing the called folder only option
 If c.Offset(, 5) = "" Then  'if the comments in G are empty, move on
    strFileName = c.Offset(, -1) 'set the file name, column A
    strSourceFolder = c 'set the folder name, columnB
    strDestFolder = c.Offset(, 1) 'set the destination name

        If OverwriteWarning = False Then 'check boolean, if False, show message
           On Error Resume Next
              'determines if path exists
              If Err = 0 Then
              'if 1st path exists, show message as warning, then turn it off
              Overwrite = MsgBox("The folders may contain duplicate files," & vbNewLine & _
                                 "Do you wish to overwrite existing files with same name?", _
                                 vbYesNo, "Alert!")
              OverwriteWarning = True 'sets the message to off TRUE means it was acknowledged
              If Overwrite <> vbYes Then GoTo CancelOut 'if cancelled, exit sub below
                Else: MkDir (strDestFolder) 'will make destination folder if it does not exist
              End If

        End If 'end check of Boolean flag for file warning
```

```
    On Error GoTo ErrHandler
    If c.Offset(, -1) = "" Then GoTo FolderOnly 'no name so copy all in folder
'Main sequence
Set objFSO = New FileSystemObject  'set a new object in memory
Set objFolder = objFSO.GetFolder(strSourceFolder) 'get the folder


If Not objFolder.Files.Count > 0 Then GoTo NoFiles  'if there are no files..whoops!
 Set objFile = objFSO.GetFile(strSourceFolder & "\" & strFileName) 'get the designated file

        If Not UCase(objFile.Name) = UCase(strFileName) Then GoTo Skip 'if the file names don't match, skip.
           strF2 = Left(strFileName, Len(strFileName) - 4) 'file name witout extension
           strExt = Right(strFileName, 4) 'file extension
           strMid = "" 'empty string for possible renaming choices
             If c.Offset(, 3) <> "" Then 'If col E is not blank then rename is possible
                 If c.Offset(, 3) <> "Custom" Then 'if it is not custom, figure out which one
                    Select Case c.Offset(, 3)  'select which option and file strMid
                       Case "Date Modified"
                         strMid = Format(objFile.DateLastModified, "_mmm_dd_yy") '
                       Case "Date Created"
                         strMid = Format(objFile.DateCreated, "_mmm_dd_yy")
                       Case "Todays Year And Month"
                         strMid = Format(Now(), "mmm_yy")
                       Case "Todays Full Date"
                        strMid = Format(Now(), "mmm_dd_yy")
                       Case "Time Stamp"
                        strMid = Format(Now(), "hh_mm")
                       Case "Full Name"
                         strF2 = c.Offset(, 4)

                    End Select
                   strFileName = strF2 & strMid & strExt 'replace the file name with renamed option
                  Else ' Custom
                    strFileName = strF2 & "_" & c.Offset(, 4) & strExt 'col F is custom suffix
                End If
           End If

  If c.Offset(, 3) = "Yes" Then 'is Yes checked for move only?
    objFile.Move strFileName & "\" & strDestFolder 'then move, not copy
  Else
    objFile.Copy strDestFolder & "\" & strFileName 'else copy
  End If
  c.Offset(, 5) = "Copied - " & Now()
  Counter = Counter + 1
```

```
Skip:   'means the file was not found..

   End If
   Next c
   GoTo Done

FolderOnly:


 If Not Err = 0 Then MkDir (strDestFolder) 'will make destination folder if it does not exist

    strSourceFolder = c
    strDestFolder = c.Offset(, 1)
   Set objFSO = New FileSystemObject
   Set objFolder = objFSO.GetFolder(strSourceFolder)
    For Each objFile In objFolder.Files
       objFile.Copy strDestFolder & "\" & strFileName
    Next objFile
    c.Offset(, 5) = "Copied Folder - " & Now()
    FoldCount = FoldCount + 1
   GoTo Backformore
NoFiles:
MsgBox "There Are no files or documents in : " & vbNewLine & vbNewLine & _
 strSourceFolder & vbNewLine & vbNewLine & "Please verify the path!", , "Alert: No Files Found!"

Set objFile = Nothing: Set objFSO = Nothing: Set objFolder = Nothing
GoTo Done 'skip error message and go to DONE

End With
ErrHandler:
MsgBox "Run Time Error :  " & Err.Number & vbCrLf & vbCrLf _
& Err.Number & "  : " & Err.Description & " " & Err.HelpContext

c.Offset(, 5) = "Error:" & Err.Description

Err.Clear
GoTo Backformore 'go back to for loop if needed

GoTo Done:

Done:
If Counter > 0 Or FoldCount > 0 Then
MsgBox FoldCount & " Folders & " & Counter & " Files Copied!", vbOKOnly, "Congatulations!"
End If
```

```
Exit Sub 'exit here to avoid below message

CancelOut:
MsgBox "Operation Cancelled"

ToggleStuff True
Set objFile = Nothing: Set objFSO = Nothing: Set objFolder = Nothing
End Sub
Sub ShowAbout()
About.Show 'Show the About form.
End Sub
Sub ClearData()
With Sheets(1).Range("A2:F500")
   .ClearContents
   .Interior.ColorIndex = 0
End With
End Sub
```

Externe *.ini Datei öffnen in Array einlesen und einzelne Zeilen ändern

Dieser Code wurde entwickelt um Vorgaben aus einer INI Datei in einem Array zu halten, damit Daten in einem Arbeitsblatt zu vergleichen, und eventuell bei Änderungen das Array zu ändern und anschliessend die INI-Datei neu zu schreiben

```
Sub Write_New_Lines_in_extern_File()
'Hilfsvariable für Anzahl Datensätze
Dim Text1 As String
'Variablen für den Array nötig
Dim TxtLines As Long, i As Long
Dim TextArr As Variant
'Schliessen einer geöffneten Datei
Close #1
'1. Öffnen der Datei
'Den Namen und Pfad bitte anpassen
Open "c:\demo.ini" For Input As #1
```

```
'Die anzahl ist nötig um die Grösse des Arrays zu deklarieren
'Zähler auf 0 setzen
TxtLines = 0
Do While Not EOF(1) ' Schleife bis Dateiende.
      Input #1, Text1 ' Hilfsvariable zum einlesen verwenden
'     Zähler hochzählen
     TxtLines = TxtLines + 1
Loop
'Schliessen der Datei weil Dateiende erreicht wurde
Close #1
'Erneutes Öffnen um zum Dateianfang zu kommen
Open "c:\demo.init" For Input As #1 ' Datei zum Einlesen öffnen.
'Array neu auf die Anzahl der Linien initialisieren
ReDim TextArr(TxtLines)
'Einlesen der Dateien in das Array
For i = 1 To TxtLines
      Input #1, TextArr(i)
Next i
Close #1
'Im Prinzip kannst du jetzt mit
TextArr(5) = "Mein Neuer Text " & TextArr(5)
'in der Zeile 5 den Text an dieser Zeile ändern
'oder mit
TextArr(6) = "Mein Neuer Text "
'in der Zeile 6 einen anderen Text einsetzen
'Zum zurückschreiben musst du den ganzen Array
'wieder in die Datei zurückschreiben.
Open "C:\Demo.ini" For Output As #1
For i = 1 To TxtLines
      Write #1, TextArr(i)
Next i
Close #1
End Sub
```

Externe ASCII (*.txt, *.dat) Datei öffnen und in ein Arbeitsblatt einfügen

Dieser Code wurde entwickelt, um eine Textdatei oder CSV die nicht richtig konvertiert wird beim öffnen, schnell einzulesen und in ein aktuelles Arbeitsblatt zu schreiben, ohne eine neue Arbeitsmappe zu erstellen

```
Sub Read_Extern_File()
'Hilfsvariable für Anzahl Datensätze
Dim Text1 As String
'Variablen für den Array nötig
Dim TxtLines As Long, i As Long
Dim TextArr As Variant
Dim ReadFile As String
'Dialog öffnen auf Basis von *.txt, *.log oder *.dat Files
ReadFile = Application.GetOpenFilename("DAT Files (*.txt; *.log; *.dat),")
'Schliessen einer geöffneten Datei
Close #1
'1. Öffnen der Datei
'Den Namen und Pfad bitte anpassen
Open ReadFile For Input As #1
'Die anzahl ist nötig um die Grösse des Arrays zu deklarieren
'Zähler auf 0 setzen
TxtLines = 0
Do While Not EOF(1) ' Schleife bis Dateiende.
        Input #1, Text1 ' Hilfsvariable zum einlesen verwenden
'       Zähler hochzählen
       TxtLines = TxtLines + 1
Loop
'Schliessen der Datei weil Dateiende erreicht wurde
Close #1
'Erneutes Öffnen um zum Dateianfang zu kommen
Open ReadFile For Input As #1 ' Datei zum Einlesen öffnen.
'Array neu auf die Anzahl der Linien initialisieren
ReDim TextArr(TxtLines)
'Einlesen der Dateien in das Array
```

```
For i = 1 To TxtLines
    Input #1, TextArr(i)
Next i
Close #1
'Daten in aktuelles Sheet schreiben
For i = 1 To TxtLines
     Cells(i, 1) = TextArr(i)
Next i
End Sub
```

Externe CSV mit mehr als 256 Spalten und mehr als 65536 Zeilen in EXCEL einfügen

Dieser Code wurde für das Export-File aus einer Datenbank entwickelt, welche Daten als CSV zur Verfügung stellt allerdings mit 912 Spalten und über 120000 Datensätzen.

Die Aufgabe bestand darin, die Spalten 289 bis 534 in ein EXCEL - Sheet einzufügen

```
Sub Read_Externe_Datei_mit_Semikolon_in_Array()
'Variable für die Statusbar
Dim OldStatus As Variant
'Hilfsvariable für Anzahl Datensätze
Dim Text1 As String
'Variablen für den Array nötig
Dim TxtLines As Long, i As Long
Dim TextArr As Variant
'Variablen für den Umwandlungsprozess des TextArrays
Dim CC As Integer, Cr As Long
Dim ArrRun As Long, FoundPlace As Integer, ReplCounter As Integer
Dim FindChr As String, ArrReplace As String, TempTextArr As String, ReadFile as String
'Schliessen einer geöffneten Datei
Close #1
'1. Öffnen der Datei
ReadFile = Application.GetOpenFilename("DAT Files (*.csv),")
Open ReadFile For Input As #1
'Die anzahl ist nötig um die Grösse des Arrays zu deklarieren
```

```
'Zähler auf 0 setzen
TxtLines = 0
Do While Not EOF(1) ' Schleife bis Dateiende.
        Input #1, Text1 ' Hilfsvariable zum einlesen verwenden
        'Zähler hochzählen
       TxtLines = TxtLines + 1
Loop
'Schliessen der Datei weil Dateiende erreicht wurde
Close #1
'Erneutes Öffnen um zum Dateianfang zu kommen
Open ReadFile For Input As #1 ' Datei zum Einlesen öffnen.
'Array neu initialisieren
ReDim TextArr(TxtLines)
'Einlesen der Dateien in das Array
For i = 1 To TxtLines
    Input #1, TextArr(i)
Next i
Close #1
'Umwandeln der ersten Semikolons in allen Datensätzen in Leerstellen
'Dieser ganze Textbereich wird anschliessend abgetrennt und verworfen
'Danach werden die jeweiligen Werte zwischen den Semikolons in die Spalten
'untereinander geschrieben
'------
'Variablen für Cell-Adressierung
CC = 1
Cr = 1
'Aufforderung wieviele Spalten zurückgesetzt werden sollen
ReplCounter = InputBox("Wieviele Spalten von links( Semikolons ) sollen entfernt werden?", "Spaltenreduktion", 150)
'Variable für das zu ersetzende Zeichen
'Hier eventuell anpassen
FindChr = ";"
'Status der Anzeige aufnehmen
OldStatus = Application.StatusBar
'Bildschirmaktualisierung ausschalten
```

```
Application.ScreenUpdating = False
For ArrRun = 1 To TxtLines
    'Statusbar für Benutzerinformation verwenden
    Application.DisplayStatusBar = True
    Application.StatusBar = "Datensatz " & ArrRun & " von " & TxtLines & " wird bearbeitet"
    'Die ersten gewählten Semikolons ersetzen
    On Error Resume Next
    For i = 1 To ReplCounter
'   Suchen des ersten Vorkommens
        FoundPlace = InStr(1, TextArr(ArrRun), FindChr)
       'Aus der Schleife aussteigen wenn weniger Semikolons gefunden wurden
        If FoundPlace = 0 Then
                'Hier muss der Ausstieg sein wenn weniger als x Semikolon gefunden
'               werden. Dann muss die Arrrun Schleife neu starten
                Exit For
        End If
       'Ersetzen des Zeichens
       ArrReplace = Application.WorksheetFunction.Replace(TextArr(ArrRun), FoundPlace, 1, "")
       'Zurückschreiben des Wertes in den Array
       TextArr(ArrRun) = ArrReplace
    Next i
    'Kürzen des Arrays bis zum ersten Semikolon
     If InStr(1, TextArr(ArrRun), FindChr) <> 0 Then
        TextArr(ArrRun) = Right(TextArr(ArrRun), Len(TextArr(ArrRun)) - InStr(1, TextArr(ArrRun), FindChr))
    End If
     'Temporären String für den Datensatz schreiben
     'dies ist nötig wegen der Schleife
    TempTextArr = TextArr(ArrRun)
    'Schreiben des Restes in die Spalten und Zeilen
    'Wiederholungen basieren auf der Länge des restlichen Textes
    'weil die Anzahl der Semikolons unbekannt ist
    For i = 1 To Len(TextArr(ArrRun))
       'Semikolon im Array neu suchen
       FoundPlace = InStr(1, TempTextArr, FindChr)
```

```
            'Aus der Schleife aussteigen wenn weniger Semikolons gefunden wurden
            'und Werte in die Zellen schreiben
             'dabei werden die horizontalen Datensätze in vertikale konvertiert
            If FoundPlace = 0 Then
                  Cells(Cr, CC) = TempTextArr
                  Exit For
            Else
                 Cells(Cr, CC) = Left(TempTextArr, FoundPlace - 1)
            End If
            'Zeilenzähler hochsetzen
            Cr = Cr + 1
            'String des temporären Array-Datensatzes neu schreiben mit dem Rest
            TempTextArr = Right(TempTextArr, Len(TempTextArr) - InStr(1, TempTextArr, FindChr))
        Next i
        'Nächste Spalte
        CC = CC + 1
        'Zeile wieder auf 1 setzen
        Cr = 1
    Next ArrRun
    Application.StatusBar = OldStatus
    Application.ScreenUpdating = True
    End Sub
```

Nach Dateiexport den Exportdatei-Ordner öffnen

Angenommen man hat den Exportordner / Exportdateiname vor dem Export einer Tabelle (zB als CSV) zuvor abgefragt oder ihn fix hinterlegt als Parameter und danach die Datei exportiert.

Abschließend möchte man noch den Ordner mit der Exportdatei automatisch öffnen.

Angenommen die gesamte Exportdatei mit Pfad ist in der Variablen strDateiname enthalten (zB C:\Eigene Dateien\Test.csv) - zum Öffnen des Ordners muss die Datei weg, sonst versucht Excel die Datei zu öffnen. Man möchte also nur den Pfad.

Falls man den nicht fix wo hinterlegt hat als Parameter, muss einfach die Exportdatei vom gesamtem Exportpfad entfernt werden.

Wenn die Datei offen ist, kann man mit ActiveWorkbook.Path den Pfad ohnedies auslesen - falls nicht, schneidet man einfach die Datei weg.

Das geht mit folgendem Code

```
'  strDateiname ist zur Zeit noch zB C:\Eigene Dateien\Test.csv


For T = 1 To Len(strDateiname)

    If Right(strDateiname, 1) <> "\" Then
       strDateiname = Left(strDateiname, Len(strDateiname) - 1)
     Else
        Exit For
    End If

Next T

'  strDateiname ist nun nur noch C:\Eigene Dateien\

MsgBox "Datei wurde exportiert nach" & vbCrLf & strDateiname
Shell "explorer.exe " & strDateiname, vbNormalFocus
```

--- ---

Datei auswählen - Name + Pfad ausgeben

Mein Code aus der Projektmanagement-Software

```
Sub Bild_Import()
' Auswahl der Datei
Dim DATEI
Dim DATEINAME As String             ' zB 1.jpg
Dim DATEINAMEOHNEENDUNG As String   ' zB 1
Dim DATEIPFAD As String             ' zB C:\Eigene Dateien\
Dim DATEIPFADUNDNAME As String      ' zB C:\Eigene Dateien\1.jpg

Dim MITTE_HORIZONTAL As Integer ' Mitte links/rechts des Tabellenblattes
Dim MITTE_VERTIKAL As Integer ' gewünschter Mittelpunkt vertikal auf Tabellenblattes
```

```
MITTE_HORIZONTAL = 200
MITTE_VERTIKAL = 300

' Defaultverzeichnis einstellen
'    ChDrive ("C:\Programme\") ' also nur C
'    ChDir ("C:\Programme\") ' also gesamten Pfad als Default

' Auswahl der Datei
   DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)

' Auslesen Dateivariablen
   DATEIPFADUNDNAME = DATEI
   DATEIPFAD = Left(DATEIPFADUNDNAME, InStrRev(DATEIPFADUNDNAME, "\"))
   DATEINAME = Right(DATEIPFADUNDNAME, Len(DATEIPFADUNDNAME) - InStrRev(DATEIPFADUNDNAME, "\"))
   DATEINAMEOHNEENDUNG = Left(DATEINAME, InStrRev(DATEINAME, ".") - 1)
   'MsgBox DATEIPFADUNDNAME & vbCrLf & DATEIPFAD & vbCrLf & DATEINAME & vbCrLf & DATEINAMEOHNEENDUNG

' Einfügen des Bildes

   On Error GoTo KEINEDATEIAUSGEWÄHLT

   Range("A1").Select
   ActiveSheet.Pictures.Insert(DATEIPFADUNDNAME).Select
   Selection.ShapeRange.LockAspectRatio = msoTrue
   'Selection.ShapeRange.IncrementLeft 7.5 ' verrücken links-rechts
   'Selection.ShapeRange.IncrementTop 5.25 ' verrücken oben-unten
   If Selection.Width > Selection.Height Then ' wenn Querformat
      Selection.Width = 200
     Else ' wenn Hchformat
      Selection.Height = 100
   End If
   Selection.ShapeRange.Left = MITTE_HORIZONTAL - Selection.Width / 2 ' Position links
   Selection.ShapeRange.Top = MITTE_VERTIKAL - Selection.Height / 2 ' Position oben

   Range("A1").Select

Exit Sub

KEINEDATEIAUSGEWÄHLT:
   MsgBox "You have selected either no or a non-supported file"

End Sub
```

Dateiauswahl / Ordner-Auswahl mittels Abfrage-/Auswahlfenster

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Sortieren&action=edit&section=T-2Dialog zur Verzeichnisauswahl

```
Public Type BROWSEINFO
   hOwner As Long
   pidlRoot As Long
   pszDisplayName As String
   lpszTitle As String
   ulFlags As Long
   lpfn As Long
   lParam As Long
   iImage As Long
End Type

Declare Function SHGetPathFromIDList Lib "shell32.dll" _
   Alias "SHGetPathFromIDListA" (ByVal pidl As Long, _
   ByVal pszPath As String) As Long

Declare Function SHBrowseForFolder Lib "shell32.dll" _
   Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Function GetDirectory(Optional msg) As String
   Dim bInfo As BROWSEINFO
   Dim Path As String
   Dim r As Long, x As Long, pos As Integer
   bInfo.pidlRoot = 0&
   If IsMissing(msg) Then
      bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
   Else
      bInfo.lpszTitle = msg
   End If
   bInfo.ulFlags = &H1
   x = SHBrowseForFolder(bInfo)
   Path = Space$(512)
   r = SHGetPathFromIDList(ByVal x, ByVal Path)
   If r Then
      pos = InStr(Path, Chr$(0))
      GetDirectory = Left(Path, pos - 1)
   Else
      GetDirectory = ""
   End If
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Sortieren&action=edit&section=T-3

Version 1 - Ordnerpfad festlegen, in dem anschließend die Datei ausgewählt werden soll

```
Sub DateiAuswahl()
    Dim WB As Workbook
    Dim TB As Worksheet
    Dim i%
    Dim dName
    Dim dFilter$
    dFilter = "Excel-Dateien(*.xls), *.xls"
    ChDrive "d"
    ChDir "d:\MeineDatenl"
    dName = Application.GetOpenFilename(dFilter)
    If dName = False Then Exit Sub
    Set WB = Workbooks.Open(dName)
    Set TB = WB.Worksheets(1)
    For i = 1 To 20
        TB.Cells(i, 5) = "Spalte E - Zeile " & i
    Next i
End Sub
```

Version 2 für Auswahl einer Datei oder eines Ordners

Um diese Einschränkung zu umgehen, kann auf das API GetOpenFileName aus der CommonDialog32-Bibliothek zurückgegriffen werden, sofern diese Datei **comdlg32.dll** auf dem Rechner vorhanden und registriert ist.
Dieses API stellt ein Datei-speichern Dialogsfenster zur Verfügung, das über Parameter flexibel angepasst werden kann: z.B. mit eigenen Formattypen.
Dieser auf den ersten Blick sehr einfache API-Aufruf verbirgt seine Möglichkeiten hinter der Parameter-Struktur OPENFILENAME: Diese Struktur beinhaltet nicht nur Initialisierungsinformationen für das Dialogfenster, sondern liefert auch Informationen über den ausgewählten Dateinamen, Dateipfad und Verzeichnis zurück.

```
Public Declare Function GetOpenFileName Lib "comdlg32.dll" _
  Alias "GetOpenFileNameA" (pOpenfilename As OPENFILENAME) As Long
```

Dieser API-Aufruf liefert jedoch keine Informationen zurück, wenn der Benutzer den Dialog abbricht oder wenn ein Fehler auftritt. Um diese Fehler abzufangen, kann das API CommDlgExtendedError verwendet werden:

```
Public Type OPENFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
```

```
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type
```

Das folgende Beispiel öffnet ein Dialogfenster zum Importieren einer VBA-Prozedur (Userforms,Module oder Klassenmodule) mit dem angegebenen Titel. Über die festgelegten Filter kann die Dateiauswahl auf die angegebenen Dateiendungen eingeschränkt werden.

Wichtig

Diese Funktion liefert als Rückgabewert nur den ausgewählten Dateinamen inkl. Pfad, öffnet aber selbst noch nicht die Datei. Dieses muss der jeweiligen Umgebung, in der das API verwendet wird, angepasst werden:
Soll ein Modul in die VBA-IDE importiert werden, wird die .Import-Methode benötigt; soll ein Word-Dokument geöffnet werden, wird die .Open-Methode benötigt.

```
Function fkt_FileOpen() As String
'used in call setup
Dim sFilters, sType As String
Dim pos As Integer
'used after call
Dim buff As String
Dim sLname As String
Dim sSname As String
sFilters = "VB-Dateien (*.frm;*.bas;*.cls)" & vbNullChar & _
          "*.frm;*.bas;*.cls" & vbNullChar & _
          "Formulardateien(*.frm)" & vbNullChar & "*.frm" & vbNullChar & _
          "Basic-Dateien (*.bas)" & vbNullChar & "*.bas" & vbNullChar & _
          "Klassendateien (*.cls)" & vbNullChar & "*.cls" & vbNullChar
With OFName
  'Setzt die Größe der OPENFILENAME Struktur
  .lStructSize = Len(OFName)
  'Der Window Handle ist bei VBA fast immer &O0
  .hwndOwner = &O0
  ' Formattyp-Filter setzen
  .lpstrFilter = sFilters
  .nFilterIndex = 1
  ' Buffer für Dateinamen erzeugen
  .lpstrFile = sModul & Space$(1024) & vbNullChar & vbNullChar
  ' Maximale Anzahl der Dateinamen-Zeichen
```

```
  .nMaxFile = Len(.lpstrFile)
  ' Buffer für Titel erzeugen
  .lpstrFileTitle = Space$(254)
  ' Maximale Anzahl der Titel-Zeichen
  .nMaxFileTitle = 255
  ' Anfangsverzeichnis vorgeben
  .lpstrInitialDir = "c:\temp"
  ' Titel des Dialogfester festlegen
  .lpstrTitle = "Modul exportieren"
  ' Flags zum Festlegen eines bestimmten Verhaltens,
  ' OFN_LONGNAMES = lange Dateinamen verwenden
  .flags = OFN_LONGNAMES
End With
' API aufrufen und evtl. Fehler abfangen
If GetOpenFileName(OFName) Then
  fkt_FileOpen = Left(OFName.lpstrFile, InStr(1, OFName.lpstrFile, Chr(0)) - 1)
ElseIf intError = 0 Then
  ' Abbruch durch Benutzer oder Fehler
End If
End Function
```

Der Aufruf dieser Funktion sieht folgendermaßen aus:

```
Sub DateiDialog()
Dim sPfad As String
sPfad = fkt_FileOpen
MsgBox "Ausgewählte Datei: " & sPfad, vbInformation, "Dateiauswahl"
End Sub
```

COMDLG32.ocx: Open/Save-Dialog
(Autor: V o l k e r, Antwort nach 12 h 41 Min)

Hallo vielleicht hilft dir diese API-Implementation

```
Private Declare Function GetOpenFileName Lib "COMDLG32" Alias "GetOpenFileNameA" (file As OPENFILENAME) As Long
Private Declare Function CommDlgExtendedError Lib "COMDLG32" () As Long
Private Declare Function lstrlen Lib "kernel32" Alias "lstrlenA" (ByVal lpString As String) As Long
Private Declare Function ChooseColor Lib "COMDLG32.DLL" Alias "ChooseColorA" (COLOR As TCHOOSECOLOR) As Long
```

```
Private Declare Function GetSysColor Lib "user32" (ByVal nIndex As Long) As Long

Private Type OPENFILENAME
lStructSize As Long ' Filled with UDT size
hWndOwner As Long ' Tied to Owner
hInstance As Long ' Ignored (used only by templates)
lpstrFilter As String ' Tied to Filter
lpstrCustomFilter As String ' Ignored (exercise for reader)
nMaxCustFilter As Long ' Ignored (exercise for reader)
nFilterIndex As Long ' Tied to FilterIndex
lpstrFile As String ' Tied to FileName
nMaxFile As Long ' Handled internally
lpstrFileTitle As String ' Tied to FileTitle
nMaxFileTitle As Long ' Handled internally
lpstrInitialDir As String ' Tied to InitDir
lpstrTitle As String ' Tied to DlgTitle
flags As Long ' Tied to Flags
nFileOffset As Integer ' Ignored (exercise for reader)
nFileExtension As Integer ' Ignored (exercise for reader)
lpstrDefExt As String ' Tied to DefaultExt
lCustData As Long ' Ignored (needed for hooks)
lpfnHook As Long ' Ignored (good luck with hooks)
lpTemplateName As Long ' Ignored (good luck with templates)
End Type

Private Enum EOpenFile
OFN_READONLY = &H1
OFN_HIDEREADONLY = &H4
OFN_ENABLEHOOK = &H20
OFN_ENABLETEMPLATE = &H40
OFN_ALLOWMULTISELECT = &H200
OFN_FILEMUSTEXIST = &H1000
'OFN_OVERWRITEPROMPT = &H2
'OFN_NOCHANGEDIR = &H8
'OFN_SHOWHELP = &H10
'OFN_ENABLETEMPLATEHANDLE = &H80
'OFN_NOVALIDATE = &H100
'OFN_EXTENSIONDIFFERENT = &H400
'OFN_PATHMUSTEXIST = &H800
'OFN_CREATEPROMPT = &H2000
'OFN_SHAREAWARE = &H4000
'OFN_NOREADONLYRETURN = &H8000
'OFN_NOTESTFILECREATE = &H10000
'OFN_NONETWORKBUTTON = &H20000
```

```
'OFN_NOLONGNAMES = &H40000
'OFN_EXPLORER = &H80000
'OFN_NODEREFERENCELINKS = &H100000
'OFN_LONGNAMES = &H200000
End Enum

Private Enum EChooseColor
CC_RGBInit = &H1
CC_FullOpen = &H2
CC_PreventFullOpen = &H4
CC_ENABLEHOOK = &H10
CC_ENABLETEMPLATE = &H20
'CC_ColorShowHelp = &H8
'CC_EnableTemplateHandle = &H40
' Win95 only
CC_SolidColor = &H80
CC_AnyColor = &H100
' End Win95 only
End Enum


Private Type TCHOOSECOLOR
lStructSize As Long
hWndOwner As Long
hInstance As Long
rgbResult As Long
lpCustColors As Long
flags As Long
lCustData As Long
lpfnHook As Long
lpTemplateName As Long
End Type

Private Const MAX_PATH = 260

Private m_lApiReturn As Long
Private m_lExtendedError As Long

' Array of custom colors lasts for life of app
Private alCustom(0 To 15) As Long
Private m_bNotFirst As Boolean
Option Explicit

Public Function VBGetOpenFileName(Filename$, Optional FileTitle$, Optional FileMustExist As Boolean = True, _
```

```
Optional MultiSelect As Boolean = False, Optional ReadOnly As Boolean = False, _
Optional HideReadOnly As Boolean = False, Optional Filter$ = "All (*.*)| *.*", _
Optional FilterIndex& = 1, Optional InitDir$, Optional DlgTitle$, _
Optional DefaultExt$, Optional Owner& = -1, Optional flags& = 0) As Boolean

Dim opfile As OPENFILENAME, i&, afFlags&, lMax&, ch$, s$

m_lApiReturn = 0: m_lExtendedError = 0

With opfile
.lStructSize = Len(opfile)
' Add in specific flags and strip out non-VB flags
.flags = (-FileMustExist * OFN_FILEMUSTEXIST) Or (-MultiSelect * OFN_ALLOWMULTISELECT) Or _
(-ReadOnly * OFN_READONLY) Or (-HideReadOnly * OFN_HIDEREADONLY) Or _
(flags And CLng(Not (OFN_ENABLEHOOK Or OFN_ENABLETEMPLATE)))

If Owner <> -1 Then .hWndOwner = Owner 'Owner can take handle of owning window
.lpstrInitialDir = InitDir 'InitDir can take initial directory string
.lpstrDefExt = DefaultExt 'DefaultExt can take default extension
.lpstrTitle = DlgTitle 'DlgTitle can take dialog box title
'To make Windows-style filter, replace | and : with nulls
For i = 1 To Len(Filter)
ch = Mid$(Filter, i, 1)
If ch = "|" Or ch = ":" Then s = s & vbNullChar Else s = s & ch
Next i
s = s & vbNullChar & vbNullChar 'Put double null at end
.lpstrFilter = s: .nFilterIndex = FilterIndex
lMax = MAX_PATH 'Pad file and file title buffers to maximum path
If (.flags And OFN_ALLOWMULTISELECT) = OFN_ALLOWMULTISELECT Then lMax = 8192
s = Filename & String$(lMax - Len(Filename), 0)
.lpstrFile = s
.nMaxFile = lMax
s = FileTitle & String$(lMax - Len(FileTitle), 0)
.lpstrFileTitle = s: .nMaxFileTitle = lMax
m_lApiReturn = GetOpenFileName(opfile) 'All other fields set to zero
Select Case m_lApiReturn
Case 1 'Success
VBGetOpenFileName = True
If (.flags And OFN_ALLOWMULTISELECT) = OFN_ALLOWMULTISELECT Then
FileTitle = ""
lMax = InStr(.lpstrFile, Chr$(0) & Chr$(0))
If (lMax = 0) Then
Filename = StrZToStr(.lpstrFile)
Else
```

```
Filename = Left$(.lpstrFile, lMax - 1)
End If
Else
Filename = StrZToStr(.lpstrFile)
FileTitle = StrZToStr(.lpstrFileTitle)
End If
flags = .flags: FilterIndex = .nFilterIndex 'Return the filter Index
Filter = FilterLookup(.lpstrFilter, FilterIndex) 'Look up the filter the user selected and return that
If (.flags And OFN_READONLY) Then ReadOnly = True
Case 0 'Cancelled
VBGetOpenFileName = False
Filename = "": FileTitle = "": flags = 0
FilterIndex = -1: Filter = ""
Case Else 'Extended error
m_lExtendedError = CommDlgExtendedError()
VBGetOpenFileName = False
Filename = "": FileTitle = "": flags = 0
FilterIndex = -1: Filter = ""
End Select
End With: End Function

Private Function StrZToStr(s$) As String
StrZToStr = Left$(s, lstrlen(s))
End Function

Private Function FilterLookup(ByVal sFilters$, ByVal iCur&) As String
Dim iStart&, iEnd&, s$

iStart = 1
If sFilters = "" Then Exit Function
Do 'Cut out both parts marked by null character
iEnd = InStr(iStart, sFilters, vbNullChar)
If iEnd = 0 Then Exit Function
iEnd = InStr(iEnd + 1, sFilters, vbNullChar)
If iEnd Then
s = Mid$(sFilters, iStart, iEnd - iStart)
Else
s = Mid$(sFilters, iStart)
End If
iStart = iEnd + 1
If iCur = 1 Then
FilterLookup = s
Exit Function
End If
```

```
iCur = iCur - 1
Loop While iCur
End Function

Function VBChooseColor(COLOR&, Optional AnyColor As Boolean = True, Optional FullOpen As Boolean = False, _
Optional DisableFullOpen As Boolean = False, Optional Owner& = -1, Optional flags&) As Boolean
Dim chclr As TCHOOSECOLOR, afMask&

chclr.lStructSize = Len(chclr)
If Owner <> -1 Then chclr.hWndOwner = Owner
chclr.rgbResult = COLOR 'Assign color (default uninitialized value of zero is good default)
afMask = CLng(Not (CC_ENABLEHOOK Or CC_ENABLETEMPLATE)) 'Mask out unwanted bits
'Pass in flags
chclr.flags = afMask And (CC_RGBInit Or IIf(AnyColor, CC_AnyColor, CC_SolidColor) Or _
(-FullOpen * CC_FullOpen) Or (-DisableFullOpen * CC_PreventFullOpen))

If m_bNotFirst = False Then InitColors 'If first time, initialize to white
chclr.lpCustColors = VarPtr(alCustom(0))
m_lApiReturn = ChooseColor(chclr) 'All other fields zero
Select Case m_lApiReturn
Case 1 'Success
VBChooseColor = True
COLOR = chclr.rgbResult
Case 0 'Cancelled
VBChooseColor = False
COLOR = -1
Case Else 'Extended error
m_lExtendedError = CommDlgExtendedError()
VBChooseColor = False: COLOR = -1
End Select
End Function

Private Sub InitColors()
Dim i%
' Initialize with first 16 system interface colors
For i = 0 To 15: alCustom(i) = GetSysColor(i): Next i
m_bNotFirst = True
End Sub
```

Wahrscheinlich sehr ähnlich ist folgendes:

Hallo Gast,

du musst dir die entsprechenden Funktionen aus der Windows API einbinden. Kopier folgenden Code in ein Modul, anschließend kannst du die Funktionen DateiOeffnen bzw. DateiSpeichern aufrufen.

Code:

```
Option Compare Database
Option Explicit

Declare Function SHBrowseForFolder Lib "shell32.dll" Alias _
   "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias _
   "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath$) As Long

Declare Function GetOpenFileName Lib "comdlg32.dll" Alias _
   "GetOpenFileNameA" (pOpenfilename As OPENFILENAME) As Long

Declare Function GetSaveFileName Lib "comdlg32.dll" Alias _
   "GetSaveFileNameA" (pSavefilename As SAVEFILENAME) As Long

Private Type BROWSEINFO
   hOwner As Long
   pidlRoot As Long
   pszDisplayName As String
   lpszTitle As String
   ulFlags As Long
   lpfn As Long
   lParam As Long
   iTmage As Long
End Type

Private Type OPENFILENAME
   lStructSize As Long
   hwndOwner As Long
   hInstance As Long
   lpstrFilter As String
   lpstrCustomFilter As String
   nMaxCustFilter As Long
   nFilterIndex As Long
   lpstrFile As String
   nMaxFile As Long
   lpstrFileTitle As String
   nMaxFileTitle As Long
   lpstrInitialDir As String
```

```
    lpstrTitle As String
    Flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As Long
End Type

Private Type SAVEFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    Flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As Long
End Type

Public Const OFN_FILEMUSTEXIST = &H1000
Public Const OFN_READONLY = &H1
Public Const OFN_HIDEREADONLY = &H4
Public Const BIF_RETURNONLYFSDIRS = &H1

Public Function VerzeichnisWählen(Title$, XHwnd As Long) As String
    Dim x As Long, BInfo As BROWSEINFO, dwIList As Long
    Dim szPath$, wPos%

    With BInfo
```

```
        .hOwner = XHwnd
        .lpszTitle = Title
        .ulFlags = BIF_RETURNONLYFSDIRS
    End With
    dwIList = SHBrowseForFolder(BInfo)
    szPath = Space$(512)
    x = SHGetPathFromIDList(ByVal dwIList, ByVal szPath)
    If x Then
        wPos = InStr(szPath, Chr(0))
        VerzeichnisWählen = Left$(szPath, wPos - 1)
      Else
        VerzeichnisWählen = " "
    End If
End Function

Public Function DateiOeffnen(Optional Titel, Optional Filter, _
                        Optional DefExtension, Optional Aktdir) As String
    Dim strDateiName$, strDlgTitel$, strFilter$
    Dim strDefExtension$, strAktDir$, strNull$
    Dim OpenDlg As OPENFILENAME

    strNull = Chr$(0)
    strDateiName = String$(512, 0)
    If IsMissing(Titel) Then
        strDlgTitel = "Datei öffnen" & strNull
      Else
        strDlgTitel = Titel & strNull
    End If
    If IsMissing(Filter) Then
        strFilter = "Alle Dateien" & strNull & "*.*" & strNull & strNull
      Else
        strFilter = Filter
    End If
    If IsMissing(DefExtension) Then
        strDefExtension = strNull
      Else
        strDefExtension = DefExtension & strNull
    End If
    If IsMissing(Aktdir) Then
        strAktDir = CurDir$ & strNull
      Else
        strAktDir = Aktdir & strNull
    End If
```

```
    With OpenDlg
        .lStructSize = Len(OpenDlg)
        .hwndOwner = Screen.ActiveForm.Hwnd
        .lpstrFilter = strFilter
        .nFilterIndex = 1
        .lpstrFile = strDateiName
        .nMaxFile = Len(strDateiName)
        .lpstrInitialDir = strAktDir
        .lpstrTitle = strDlgTitel
        .Flags = OFN_FILEMUSTEXIST Or OFN_READONLY
        .lpstrDefExt = strDefExtension
        If GetOpenFileName(OpenDlg) <> 0 Then
           DateiOeffnen = Left$(.lpstrFile, InStr(.lpstrFile, strNull) - 1)
         Else
           DateiOeffnen = " "
        End If
    End With
End Function

Public Function DateiSpeichern(Optional Titel, Optional Filter, _
                               Optional DefExtension, Optional Aktdir) _
                              As String
    Dim strDateiName$, strDlgTitel$, strFilter$
    Dim strDefExtension$, strAktDir$, strNull$
    Dim SaveDlg As SAVEFILENAME

    strNull = Chr$(0)
    strDateiName = String$(512, 0)
    If IsMissing(Titel) Then
        strDlgTitel = "Datei öffnen" & strNull
      Else
        strDlgTitel = Titel & strNull
    End If
    If IsMissing(Filter) Then
        strFilter = "Alle Dateien" & strNull & "*.*" & strNull & strNull
      Else
        strFilter = Filter
    End If
    If IsMissing(DefExtension) Then
        strDefExtension = strNull
      Else
        strDefExtension = DefExtension & strNull
    End If
```

```
    If IsMissing(Aktdir) Then
       strAktDir = CurDir$ & strNull
     Else
       strAktDir = Aktdir & strNull
    End If
    With SaveDlg
       .lStructSize = Len(SaveDlg)
       .hwndOwner = Screen.ActiveForm.Hwnd
       .lpstrFilter = strFilter
       .nFilterIndex = 1
       .lpstrFile = strDateiName
       .nMaxFile = Len(strDateiName)
       .lpstrInitialDir = strAktDir
       .lpstrTitle = strDlgTitel
       .Flags = OFN_HIDEREADONLY
       .lpstrDefExt = strDefExtension
       If GetSaveFileName(SaveDlg) <> 0 Then
          DateiSpeichern = Left$(.lpstrFile, InStr(.lpstrFile, strNull) - 1)
        Else
          DateiSpeichern = " "
       End If
    End With
End Function
```

Das ganze Für Outlook:

Hallo,
für den Import von Kontakten in Text-Fmt muß ich einen "DateiAuswahl Dialog" aufrufen ... zB wie ich es von EXCEL gewohnt bin:
If fmt = "TXT" Then Datei_to_Open = Application.GetOpenFilename _
("Dateien (*.txt;*.csv;*.LDIF), *.txt", , " Datei auswählen")
Nur leider mag Outlook/VBA diesen Befehl nicht!

Kann mir hier jemand helfen?

Hallo,

nach etwas suchen habe ich eine Lösung gefunden, die ich auf meine Anforderungen angepaßt habe.
Für alle, die es wissen wollen hier der Code:

```
Attribute VB_Name = "win_selectFolder_Datei"
Public s As String

Public Type BROWSEINFO
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

'32-bit API-Deklarationen
Declare Function SHGetPathFromIDList Lib "shell32.dll" _
    Alias "SHGetPathFromIDListA" _
    (ByVal pidl As Long, ByVal pszPath As String) As Long

Declare Function SHBrowseForFolder Lib "shell32.dll" _
  Alias "SHBrowseForFolderA" _
  (lpBrowseInfo As BROWSEINFO) As Long


Function FSselect(OP As String, titel As String) As String
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Dim bInfo As BROWSEINFO              '    gW 2006.10-07
  Dim strPath As String
  Dim r As Long, x As Long, pos As Integer

 ' --- welche pidl wird gebraucht ?? ------------
    bInfo.pidlRoot = 0&
      ' Der Ausgangsordner ist der Desktop
      ' 0&   ab Root           1&   Internet Explorer
      ' 2&   Programme         3&   Systemsteuerung
      ' 4&   Drucker & Fax     5&   bei Eigene Dateien
      ' 6&   Favoriten         7&   Autostart
      ' 8&   Recent           9&   Sendto
      ' 10&   Papierkorb       11&   Startmenü
      ' 12&   --              13&   eigene Musik
      ' 14&   eigene Videos     15&   --
      ' 16&   Desktop          17&   Arbeitsplatz
      ' 18&   Netzwerkumgebung  19&   Desktop
```

```
    ' 21&   Vorlagen         22&   Startmenu

   FSselect = ""
   bInfo.lpszTitle = titel      ' Dialogtitel

   Select Case OP
     Case "DIR"
       bInfo.ulFlags = &H1      ' Rückgabe des Unterverzeichnisses
     Case "FILE"
       bInfo.ulFlags = &H4000   '   der ausgewählten Datei
     Case Else
       Exit Function
   End Select

   x = SHBrowseForFolder(bInfo)     ' Dialog anzeigen
   strPath = Space$(512)           ' Ergebnis gliedern

  ' Auswahl einlesen
   r = SHGetPathFromIDList(ByVal x, ByVal strPath)
   If r Then
      pos = InStr(strPath, Chr$(0))
      FSselect = Left(strPath, pos - 1)
   End If
End Function
```

Dieser Code geht zurück auf verschiedene Vorschläge im Netz, abgewandelt für alternative Auswahl auf Verzeichnis OP="DIR oder Datei OP="FILE" im Aufruf.

Für meinen Aufruf im Outlook / VBA um eine bestimmte Datei weiter zu verarbeiten lautet es dann:

Code:

```
pLDIF = FSselect("FILE", titel & vbLf & _
   "--> Verzeichnis und LDIF-Datei auswählen")
```

Auf bald

Möchte man nur ein Verzeichnis abfragen und keine Datei:

Version 1:

Häufig benötigt man nur ein bestimmtes Verzeichnis, das dann weiterverwendet werden soll. Leider wird dazu kein integriertes Dialogfenster angeboten, so dass man evtl. über die Dateiauswahl gehen muss. Dieses hat aber den großen Nachteil, dass aus dem vollständigen Dateinamen (inkl. Pfad) erst noch der Dateipfad ermittelt werden muss.
Schneller und bequemer geht es mit zwei APIs, mit denen direkt ein beliebiges Verzeichnis ausgewählt werden kann.
Standardmäßig beginnt die Verzeichnisauswahl auf oberster Ebene, aber über die Angabe eines Startverzeichnisses im Funktionsaufruf, kann die Auswahl direkt in einem Verzeichnis beginnen.
Die Funktion GetFolderInternal() erwartet als Aufrufparameter einen Beschritungstext und ein Standardverzeichnis, als Rückgabewert wird der Name des ausgewählten Verzeichnisses zurückgeliefert.
Dieser Funktion lässt sich erst ab Word2000 das Anfangsverzeichnis mitgeben, da die verwendete Callback-Funktion unter Word97 nicht unterstützt wird.

```
Public Function GetFolderInternal(ByVal Caption As String, _
    ByVal Default As String) As String
```

Um beim nächsten Aufruf dieses Dialogfensters wieder im letzten Verzeichnis zu beginnen, genügt es für das Standardverzeichnis das zuletzt ausgewählte Verzeichnis anzugeben:

```
  Option Explicit
   Private Type BROWSEINFO
      hWndOwner As Long
      pidlRoot As Long
      pszDisplayName As String
      lpszTitle As String
      ulFlags As Long
      lpFn As Long
      lParam As String
      iImage As Long
   End Type
   Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
    Alias "SHBrowseForFolderA" (ByRef lpbi As BROWSEINFO) As Long
   Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
    Alias "SHGetPathFromIDListA" (ByVal pidl As Long, _
    ByVal pszPath As String) As Long
   Private Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal pv As Long)
   Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, _
    ByVal lParam As Long) As Long
   Private Const WM_USER As Long = &H400
   Private Const BIF_RETURNONLYFSDIRS As Long = 1
   Private Const BFFM_INITIALIZED As Long = 1
   Private Const BFFM_SETSELECTION As Long = (WM_USER + 102)
   Private Const MAX_PATH As Long = 260
```

```
Public Function GetFolderInternal(ByVal Caption As String, _
    ByVal Default As String) As String
      Dim BI As BROWSEINFO
      Dim ListIdx As Long
```

```
    Dim Path As String
    With BI
      .lpszTitle = Caption
      .ulFlags = BIF_RETURNONLYFSDIRS
      .lpFn = MakeFktnPtr(AddressOf BrowseCallbackProc)
      .lParam = Default
    End With
    Path = String$(MAX_PATH + 1, vbNullChar)
    ListIdx = SHBrowseForFolder(BI)
    If SHGetPathFromIDList(ListIdx, Path) Then
      GetFolderInternal = Left$(Path, InStr(Path, vbNullChar) - 1)
    End If
    CoTaskMemFree ListIdx
End Function
```

```
Private Function BrowseCallbackProc(ByVal hWnd As Long, ByVal Msg As Long, _
   ByVal lParam As Long, ByVal lpData As Long) As Long
    On Error Resume Next
    If Msg = BFFM_INITIALIZED Then
      SendMessage hWnd, BFFM_SETSELECTION, 1&, lpData
    End If
  End Function
```

```
Private Function MakeFktnPtr(ByVal FktnPtr As Long) As Long
    MakeFktnPtr = FktnPtr
End Function
```

Version 2:

Mit dieser Funktion stellen Sie einen Auswahldialog zur Verfügung, mit dessen Hilfe der Anwender ein gewünschtes Verzeichnis auswählen kann. Der Verzeichnispfad wird als String an die Funktion zurückgegeben.
Die Besonderheit bei diesem Beispiel ist, dass man hier auch ein Startverzeichnis angeben kann.
Achtung:
Der folgenden Quellcode arbeitet nicht mit Access 97! Den Quellcode für Access 97 finden Sie ebenfalls in unserem Tipp-Archiv.
Wie wird es gemacht:
Erstellen Sie ein neues Modul und fügen Sie nachfolgenden Code ein:

```
Option Compare Database
Option Explicit

Private Type BROWSEINFO
   hOwner         As Long
```

```
    pidlRoot        As Long
    pszDisplayName  As String
    lpszTitle       As String
    ulFlags         As Long
    lpfn            As Long
    lParam          As Long
    iImage          As Long
End Type

Private Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias _
          "SHGetPathFromIDListA" (ByVal pidl As Long, _
          ByVal pszPath As String) As Long

Private Declare Function SHBrowseForFolder Lib "shell32.dll" Alias _
          "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) _
          As Long

Private Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" _
          (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, _
          lParam As Any) As Long

Private Const BIF_RETURNONLYFSDIRS = &H1
Private Const BFFM_SETSELECTION = &H466
Private Const BFFM_INITIALIZED = 1

Global StartDir As String

Public Function VerzeichnisSuchen(szDialogTitle As String, _
             StartVerzeichnis As String) As String

  Dim X         As Long
  Dim bi        As BROWSEINFO
  Dim dwIList   As Long
  Dim szPath    As String
  Dim wPos      As Integer

  StartDir = StartVerzeichnis

  With bi
       .hOwner = hWndAccessApp
       .lpszTitle = szDialogTitle
       .ulFlags = BIF_RETURNONLYFSDIRS
       .lpfn = DummyFunc(AddressOf BrowseCallbackProc)
   End With
```

```
    dwIList = SHBrowseForFolder(bi)
    szPath = Space$(512)
    X = SHGetPathFromIDList(ByVal dwIList, ByVal szPath)

    If X Then
        wPos = InStr(szPath, Chr(0))
        VerzeichnisSuchen = Left$(szPath, wPos - 1)
    Else
        VerzeichnisSuchen = ""
    End If
End Function
Public Function BrowseCallbackProc(ByVal hWnd As Long, ByVal uMsg As Long, _
            ByVal lParam As Long, ByVal lpData As Long) As Long

    Dim pathstring  As String
    Dim retval      As Long

    Select Case uMsg
        Case BFFM_INITIALIZED
            pathstring = StartDir
            retval = SendMessage(hWnd, BFFM_SETSELECTION, _
                    ByVal CLng(1), ByVal pathstring)
    End Select

    BrowseCallbackProc = 0

End Function
Public Function DummyFunc(ByVal param As Long) As Long

    DummyFunc = param

End Function
```

Um den Dialog aufzurufen und das Ergebnis einem Feld innerhalb des Formulars zurückzugeben erstellen Sie bitte eine Schaltfläche mit dem Namen **Verzeichnisauswahl**. Der im Beispiel verwendete Namen für das Feld, in welches der Verzeichnispfad zurückgeschrieben wird, lautet **Verzeichnis**. Beide Namen (Schaltfläche & Feld) können Sie natürlich anders benennen.

```
Private Sub Verzeichnisauswahl_Click()

    Dim strVerzeichnisName As String

    If IsNull(Me!Verzeichnis) Then
        Me!Verzeichnis = ""
    End If
```

```
    strVerzeichnisName = VerzeichnisSuchen _
       ("Wählen Sie bitte das Verzeichnis aus!", Me!Verzeichnis)

    If ((Not IsNull(strVerzeichnisName)) And (strVerzeichnisName <> "")) Then
       Me!Verzeichnis = strVerzeichnisName
    End If

End Sub
```

Dateien+Ordner kopieren, verschieben, löschen, umbenennen

2017 – Datei Kopieren

```
Sub Kopieren_File()
Dim myFSO As Object
Dim qFolder As String, tFolder As String
Dim qFile As String
Dim i As Long
Set myFSO = CreateObject("Scripting.FileSystemObject")
qFile = "License_24.dll" ' Quell-Dateiname setzen
qFolder = ActiveWorkbook.Path & "\"
tFolder = "C:\Test\Test\"
MsgBox qFolder & qFile
MsgBox tFolder & qFile
myFSO.CopyFile qFolder & qFile, tFolder & qFile, True
End Sub
```

2017 Datei verschieben aus aktuellem Verzeichnis der Exceldatei

Variante 1

```
Sub DATEIVERSCHIEBEN()

Dim strSource As String
Dim strTarget As String
```

```
strSource = ActiveWorkbook.Path & "\License_24.dll"
strTarget = "C:\test\test\License_24.dll"

' Copy source file
Name strSource As strTarget

End Sub
```

Variante 2

```
    CreateObject("Scripting.FileSystemObject").MoveFile VONPFAD & DATEINAME, NACHPFAD & DATEINAME
```

a.) Move Copy a file using API

```
'
' -------------------------------------------------------------------------------

 ' ----------------------------
' Constants & API Declaration
' ----------------------------

Private Declare Function CopyFile Lib "kernel32" Alias "CopyFileA" (ByVal lpExistingFileName As String, ByVal lpNewFileName As String, ByVal bFailIfExists As
Long) As Long
Private Declare Function MoveFile Lib "kernel32" Alias "MoveFileA" (ByVal lpExistingFileName As String, ByVal lpNewFileName As String) As Long


' ----------
' Functions
' ----------

Sub Copy_File(Source As String, Target As String)
    'Copy File

    Dim l As Long

    l = CopyFile(Trim$(Source), Trim$(Target), False)
    If l Then
        ' File copied!
    Else
        ' Error. File not moved!
    End If
```

```
End Sub

' ================================================== '

Sub Move_File(Source As String, Target As String)
    Dim l As Long

    l = MoveFile(Trim$(Source), Trim$(Target))

    If l Then
        ' File moved!
    Else
        ' Error. File not moved!
    End If
End Sub
```

---

b.) Meine Version Unitreu, des Tools das zu archivierende Ornder kopiert

Hier die Theorie

Mit dieser Vorlage können Projektverzeichnisse in einem frei wählbaren Laufwerk ausgewählt werden und in ein anderes Laufwerk kopiert werden, um von dort automatisch in das NTCS-Archiv eingespielt zu werden.

Dabei muss folgende Hierarchielogik eingehalten werden.

- Auf der obersten Ebene sind die Klientenordner. Diese beginnen mit einem Lexorkürzel und können nach einem Leerzeichen noch weitere Zeichen enthalten wie etwa einen Klientennamen. zB Z:\ABCD
- In der nächsten Ebene sind die NTCS-Archive des Klienten wie etwa APA13 oder Dauerakt. Auf dieser Ebene sollten sich keine einzelnen Dateien befinden, da diese nicht archiviert werden. zB Z:\ABCD\APA13
- In der nächsten Ebene befinden sich die einzelnen Kategorien. zB Z:\ABCD\APA13\a-a Abschluss
- Innerhalb der nächsten Ebene können sowohl Einzeldateien sein, als auch Unterordner mit dem Namen pbc für die Klientendateien. Die Kategorie a-e kann noch weitere Unterordner enthalten, welche Zusatzkategorien in NTCS darstellen; zB:

    Z:\ABCD\APA13\a-a Abschluss\123.txt
    Z:\ABCD\APA13\a-a Abschluss\pbc\123.txt
    Z:\ABCD\APA13\\a-e Arbeitspapiere\at-Latente Steuern\123.txt

Der eigentliche Transfer der Daten eines Projekts erfolgt in der Tabelle TRANSFER mit dem dortigen Button ORDNER FÜR ARCHIVIERUNG WÄHLEN. Es öffnet sich ein Fenster das auf das hinterlegte Quelllaufwerk abzielt und dort ein Navigieren in den gewünschten Klientenordner erlaubt. Ob man dann den gewünschten Projektordner einmal anklickt und dann unten auf "ORDNER AUSWÄHLEN FÜR ARCHIV" klickt - oder man mit Doppelklick auf den gewünschten Projektordner diesen noch öffnen und erst dann auf "ORDNER AUSWÄHLEN FÜR ARCHIV" klickt, ist egal. In letzterem Fall ist es wichtig, dass man dann nicht noch zusätzlich einen Unterordner im Projektordner anklickt, weil sonst irrtümlich dieser ausgewählt wird.
Es erfolgt jedoch vor dem Datentransfer ohnedies eine Überprüfung, dass man sich in der Ordnerhierarchie nicht zu hoch oben (zB einen ganzen KLientenordner mit allen Projekten) oder zu tief (einen Projekt-Kategorie-Unterordner) befindet.

Nach der Auswahl des Projektordners überprüft das Tool, ob das Projekt bereits einmal archiviert wurde (es enthält dann am Ende das Datum der Archivierung im Projektordndername) und würde einen entsprechenden Hinweis geben.

Danach sucht das Tool ausgehend vom Lexorkürzel den korrekten NTCS-Kunden und seine Kundennummer. Wird das Tool nicht fündig, kann man die Daten gleich nacherfassen, indem man das Lexorkürzel manuell in NTCS sucht und hier im Tool nacherfasst. Die Daten werden vom Tool automatisch abgefragt und in der Tabelle NTCS_KUNDEN automatisch unten in der nächsten leeren Zeile eingetragen. (Die Sortierung der Kunden im Tabellenblatt NTCS_KUNDEN ist übrigens völlig irrelevant.)

Anschließend kopiert das Tool das betreffende Projekt mit allen Unterordnern auf das hinterlegte Ziellaufwerk. Dabei wird der Klientenordner angelegt mit der Bezeichnung "Klientennummer Klientenname" - und darunter wird das Projekt 1:1 abgespeichert. Hat man ein Projekt bereits einmal gespeichert und möchte es - etwa weil man noch Dateien vergessen hat, die man zwischenzeitlich im Projektordner dazugespeichert hat - erneut in das Ziellaufwerk speichern, so ist das möglich. Beim erneuten Transfer ins Zielverzeichnis werden alle Dateien noch einmal komplett neu übernommen - jedoch KEINE Dateien im Zielverzeichnis aktiv gelöscht. Daher: wurden nach dem ersten Transfer Dateien im Quellverzeichnis verschoben, empfiehlt es sich den Projektordner im Zielverzeichnis manuell zu löschen und erst dann den Transfer zu wiederholen.

Nach dem erfolgreichen Transfer (Kopieren / nicht verschieben!) des Projektordners, wird der ursprüngliche Projektordner umbenannt, in dem an ihn das Archivierungsdatum angehängt wird - aus APA13 wird dann z.B. APA13 31.12.2014.

Ist die Protokollierung aktiviert, wird der neu transferierte Projektordner in der Tabelle TRANSFER unten hinzugefügt. Zusätzlich wird in der Zeile 2 auch immer der letzte Transfer angezeigt.

```
' ------------------------------------------
' ---- HILFSFUNKTION  /  DEFINITIONEN ----
' ------------------------------------------


Private Declare Function SHFileOperation _
   Lib "shell32.dll" Alias "SHFileOperationA" ( _
   lpFileOp As Any _
   ) As Long
```

```
Private Type SHFILEOPSTRUCT
    hwnd As Long
    wFunc As Long
    pFrom As String
    pTo As String
    fFlags As Integer
    fAnyOperationsAborted As Long
    hNameMappings As Long
    lpszProgressTitle As String
End Type

Private Const FO_DELETE = &H3
Private Const FO_MOVE = &H1
Private Const FO_RENAME = &H4
Private Const FO_COPY = &H2&
Private Const FOF_RENAMEONCOLLISION = &H8
Private Const FOF_NOCONFIRMMKDIR = &H200
Private Const FOF_NOCONFIRMATION = &H10
Private Const FOF_MULTIDESTFILES = &H1
Private Const FOF_ALLOWUNDO = &H40
Public Function CopyMove( _
   ByVal strSourceFolder As String, _
   ByVal strDestinationFolder As String, _
   Optional blnMove As Boolean) As Boolean

   Dim udtXCopy As SHFILEOPSTRUCT

   ' Zweimal Nullchar anhängen, damit das Ende
   ' erkannt wird

   strSourceFolder = strSourceFolder & _
   vbNullChar & vbNullChar

   strDestinationFolder = strDestinationFolder & _
   vbNullChar & vbNullChar

   With udtXCopy

     If blnMove Then
      ' Verschieben
      .wFunc = FO_MOVE
```

```
    Else
     ' Kopieren
     .wFunc = FO_COPY
    End If

    ' Quelle und Ziel
    .pFrom = strSourceFolder
    .pTo = strDestinationFolder

    .fFlags = FOF_MULTIDESTFILES _
    Or FOF_NOCONFIRMATION _
    Or FOF_NOCONFIRMMKDIR _
    Or FOF_ALLOWUNDO

  End With

  ' Ausführen und Ergebnis zurückliefern
  If Dir(strSourceFolder, vbDirectory) = "" Then Exit Function
  If SHFileOperation(udtXCopy) = 0 Then CopyMove = True


End Function


' ------------------------------------------
' ---- ENDE HILFSFUNKTION  /  DEFINITIONEN ----
' ------------------------------------------

Public Sub Ordnerauswahl()

' Mit dieser Prozedur kann man einen Ordner auswählen und er wird 1:1 kopiert und das Original wird umbenannt

    Dim strOrdner As String ' Variable für Auswahl des Projektes
    Dim LEXOR As String     ' Lexorkürzel im Projektpfad
    Dim PROJEKT As String   ' Projektbezeichnung zB APA13
    Dim KL_NUMMER As Long   ' Klientennummer
    Dim KL_NAME As String   ' Klientenname
    Dim NTCS_Z As Long      ' Zeile in der Tabelle NTSC_KUNDEN mit dem betreffenden Kunden

    Dim strSuche            ' Für Suche des Lexorkürzels in Liste mit NTCS-Kunden)
    Dim rngFound As Range   ' hier wurde es gefunden (Zelladresse)
```

```
Dim strPath As String    ' Quellpfad für Datentransfer
Dim strDest As String    ' Ziepfad für Datentransfer
Dim SCHONARCHIVIERT      ' Merkt sich, ob ein Projekt schon mal archiviert worden ist

' Abfrage des gewünschten Projekt-Ordners
With Application.FileDialog(msoFileDialogFolderPicker)
    .InitialFileName = Tabelle1.Range("K4")
    .Title = "Welches Projekt soll archiviert werden ?"
    .ButtonName = "Ordner auswählen für Archiv"
    .InitialView = msoFileDialogViewList
    If .Show = -1 Then
        strOrdner = .SelectedItems(1)
        If Right(strOrdner, 1) <> "\" Then strOrdner = strOrdner & "\"
      Else
        strOrdner = ""
    End If
End With

' Wenn kein Ordner gewählt wurde
If strOrdner = "" Then
    MsgBox ("Es wurde kein Ordner gewählt - Code bricht ab")
    End
End If

' Prüfen der ausgewählten Ebene - optimaler Gesamtpfad lautet "Laufwerk:\Klient\Projekt\" und hat also 3 \
If Tabelle1.Range("C8") = True Then ' Prüfen wenn aktiviert
    If UBound(Split(strOrdner, "\")) = 1 Then
        MsgBox "Sie haben keinen Ordner gewählt." & vbCrLf & vbCrLf & _
        "Bitte gehen Sie beim gewünschten Klient hinein und wählen ein Projekt."
        End
    End If
    If UBound(Split(strOrdner, "\")) = 2 Then
        MsgBox "Sie haben einen ganzen Klientenordner direkt auf der Hauptebene gewählt." & vbCrLf & vbCrLf & _
        "Bitte gehen Sie beim gewünschten Klient hinein und wählen ein Projekt."
        End
    End If
    If UBound(Split(strOrdner, "\")) > 3 Then
        MsgBox "Sie haben nur einen Unterordner in einem Projekt gewählt." & vbCrLf & vbCrLf & _
        "Es können aber nur ganze Projekte kopiert werden." & vbCrLf & vbCrLf & _
        "Bitte gehen Sie beim gewünschten Klient hinein und wählen ein ganzes Projekt."
        End
    End If
```

```
    End If

    ' Ermitteln des Projekts
    PROJEKT = Right(strOrdner, Len(strOrdner) - InStr(strOrdner, "\")) ' wegschneiden alles vor dem ersten \
    PROJEKT = Right(strOrdner, Len(PROJEKT) - InStr(PROJEKT, "\")) ' wegschneiden alles vor dem zweiten \
    PROJEKT = Replace(PROJEKT, "\", "") ' entfernen des abschließenden \

    On Error GoTo WEITER
    If CDbl(Right(PROJEKT, 4)) > 2013 Then

        If MsgBox("Dieses Projekt wurde offensichtlich bereits archiviert." & vbCrLf & vbCrLf & _
        "Möchten Sie es erneut archivieren ?", vbYesNoCancel, "Bereits archiviert") <> vbYes Then
            MsgBox "Es wird nichts erneut archiviert."
            End
        End If

        MsgBox "Bitte entfernen Sie im Projektordnername " & PROJEKT & " die abschließende Endung mit dem
Archivierungsdatum und wiederholen den Vorgang."
        End

    End If

WEITER:
    ' Auslesen Klienten-Lexorkürzel aus Projektpfad
    ' Auf der Hauptebene befinden sich die Klientenordner - diese beginnen mit dem Lexorkürzel - eventuell gefolgt von
einem Leerzeichen und einem Namen
    ' ausgehend vom Lexorkürzel - also den ersten Zeichen bis zum ersten Leerzeichen - matchen wir die Klientennummer
dazu (Tabelle NTCS_KUNDEN)
    LEXOR = Right(strOrdner, Len(strOrdner) - InStr(strOrdner, "\")) ' wegschneiden alles vor dem ersten \
    LEXOR = Left(LEXOR, InStr(LEXOR, "\") - 1) ' wegschneiden alles ab dem zweiten \
    If InStr(LEXOR, " ") > 0 Then ' falls im verbleibenden Projekt-Ordnername ein Leerzeichen vorkommt
        LEXOR = Left(LEXOR, InStr(LEXOR, " ") - 1) ' wegschneiden alles ab dem ersten Leerzeichen
    End If

    ' Suchen der Kundennummer ausgehend vom Lexorkürzel
    Application.ScreenUpdating = False

    Tabelle2.Activate

    strSuche = LEXOR    ' Übergabe des Lexorkürzels in Suchvariable
    Range("B1").Activate ' in die Suchspalte hineinspringen - sonst kommt Fehlermeldung
```

```
    Set rngFound = Columns("B").Find(What:=strSuche, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) ' rows würde
in Zeile suchen

    ' Wenn Lexorkürzel nicht gefunden wurde
    If rngFound Is Nothing Then
        If MsgBox("Das Lexorkürzel " & LEXOR & " wurde leider nicht gefunden." & vbCrLf & vbCrLf & _
        "- Möchten Sie nun die korrekte 6-stellige Klientennummer eingeben (JA) " & vbCrLf & vbCrLf & _
        "- oder den Vorgang Abbrechen (Nein/Abbruch)?", vbYesNoCancel, "Klient nicht gefunden") <> vbYes Then
            MsgBox "Der Datentransfer bricht jetzt ab. " & vbCrLf & vbCrLf & _
            "Sie können das Lexorkürzel jederzeit in der Tabelle NTCS_KUNDEN nachtragen und den Vorgang wiederholen."
            Application.ScreenUpdating = True
            Exit Sub ' nix gefunden
        End If
        ' User kann nun die Klientennummer erfassen
        KL_NUMMER = InputBox("Bitte geben Sie jetzt die korrekte 6-stellige Klientennummer aus NTCS ein: ", "Neuen Klient
anlegen", 2)
        If CDbl(KL_NUMMER) = 0 Or CDbl(KL_NUMMER) > 999999 Then
            MsgBox "Sie haben keine 6-stellige Klientennummer, sondern " & KL_NUMMER & " eingegeben. " & vbCrLf & vbCrLf
& "Bitte starten Sie erneut."
            Application.ScreenUpdating = True
            Exit Sub ' nix gefunden
        End If
        KL_NAME = InputBox("Bitte geben Sie auch einen Klientennamen ein, (muss nicht genau sein und dient nur zur
Infozwecken) :", "Neuen Klient anlegen")
        ' Übernahme des neuen Lexorkürzels / Klienten
        If CDbl(KL_NUMMER) > 0 Then
            NTCS_Z = LETZTEZELLE(Tabelle2.Name).Row + 1 ' Berechnen der nächsten leeren Zeile
            Tabelle2.Cells(NTCS_Z, 2) = LEXOR
            Tabelle2.Cells(NTCS_Z, 1) = KL_NUMMER
            Tabelle2.Cells(NTCS_Z, 3) = KL_NAME
        End If

      Else ' wenn Lexorkürzel gefunden wurde

        NTCS_Z = rngFound.Row
        KL_NAME = Tabelle2.Cells(NTCS_Z, 3)
        KL_NUMMER = Tabelle2.Cells(NTCS_Z, 1)

    End If

    On Error Resume Next
```

```
' ----------------------
' --- DATEN-TRANSFER ---
' ----------------------

' Ermitteln des Zielordners
' Aus Quellordner QUELL-LW:\Lexorkürzel\Projekt\  wird
' Zielordner ZIEL-LW:\Klientennummer Kundenname\Projekt\

strPath = Left(strOrdner, Len(strOrdner) - 1) ' wegschneiden des abschließenden \
If Tabelle1.Range("C12") = True Then ' Wenn Klientenname eingefügt werden soll in Zielordner
    KL_NAME = Left(KL_NAME, Tabelle1.Range("K12"))
    strDest = Tabelle1.Range("K6") & KL_NUMMER & " " & KL_NAME & "\" & PROJEKT
   Else
    strDest = Tabelle1.Range("K6") & KL_NUMMER & "\" & PROJEKT
End If


' Ganzes Verzeichnis in neu angelegtes Verzeichnis kopieren

If CopyMove(strPath, strDest) Then
   ' MsgBox "Verzeichnis" & vbCrLf & _
      strPath & vbCrLf & "nach " & vbCrLf & strDest & vbCrLf & _
      "kopiert"
Else
    If Tabelle1.Range("C6") = True Then
        MsgBox "Fehler beim Kopieren von " & vbCrLf & strPath & vbCrLf & "nach " & vbCrLf & strDest
    End If
    Application.ScreenUpdating = True
    End
End If

' --- Protokoll ---

Z = LETZTEZELLE(Tabelle4.Name).Row + 1

' Datum
Tabelle4.Cells(2, 2) = Now
If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 2) = Now
' Klientennummer
Tabelle4.Cells(2, 3) = KL_NUMMER
If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 3) = KL_NUMMER
```

```
    ' Lexor
    Tabelle4.Cells(2, 4) = LEXOR
    If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 4) = LEXOR
    ' Klientenname
    Tabelle4.Cells(2, 5) = KL_NAME
    If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 5) = KL_NAME
    ' Projekt
    Tabelle4.Cells(2, 6) = PROJEKT
    If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 6) = PROJEKT
    ' Mitarbeiter
    Tabelle4.Cells(2, 7) = Application.UserName
    If Tabelle1.Range("C4") = True Then Tabelle4.Cells(Z, 7) = Application.UserName



    ' Umbenennen des kopierten Projektordners
    If Tabelle1.Range("C10") = True Then Name strPath As strPath & " " & Left(Now, 10)

    Tabelle4.Activate

    Application.ScreenUpdating = True ' Wichtig: setze es auch nach allen END-Abbrüchen !

End Sub
```

Dateiverschieben 1

Hallo wie verschiebe ich den die Datei
C:\V1\Datei.Xls in das Verzeichnis c:\V2 ?
Mit Excel VBA ?

Sub test()
CreateObject("Scripting.FileSystemObject").MoveFile "c:\V1\test.xls", "c:\V2\test.xls"
End Sub

Wahrscheinlich kann man hier, wie auch bei der Lösung 2 direkt hier drunter, durch Verwenden eines abweichenden zweiten Dateinamen die Datei zugleich umbenennen

Dateiverschieben 2 (inkl. Umbenennen)

```
name "c:\filename.csv" as "c:\test\filename.csv
```

Name ist eigentlich die klassische VBA-Funktion zum umbenennen - aber da sie zugleich auch verschieben kann, wird sie hier angeführt. Hier die original VBA-Hilfe zu NAME:

Name-Anweisung

Benennt eine Datei, ein Verzeichnis oder einen Ordner um.

Syntax

Name *AlterPfadname* **As** *NeuerPfadname*

Die Syntax der **Name**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
AlterPfadname	Erforderlich. Zeichenfolgenausdruck, der einen existierenden Dateinamen und seinen Pfad angibt. In dem Wert kann ein Verzeichnis oder Ordner sowie ein Laufwerk enthalten sein.
NeuerPfadname	Erforderlich. Zeichenfolgenausdruck, der einen neuen Dateinamen und seinen Pfad angibt. In dem Wert kann ein Verzeichnis oder Ordner sowie ein Laufwerk enthalten sein. Die in *NeuerPfadname* angebene Datei darf noch nicht existieren.

Bemerkungen

Sowohl der Pfad in *NeuerPfadname* als auch der Pfad in *AlterPfadname* müssen sich auf demselben Laufwerk befinden. Wenn der Pfad in *NeuerPfadname* existiert und vom Pfad in *AlterPfadname* abweicht, verschiebt die **Name**-Anweisung die Datei in das neue Verzeichnis oder in den neuen Ordner und benennt sie ggf. um. Wenn sich die Pfadangaben in *NeuerPfadname* und *AlterPfadname* unterscheiden und der Dateiname gleich ist, verschiebt **Name** die Datei an die neue Position, ohne den Dateinamen zu ändern. Mit **Name** können sie eine Datei von einem Verzeichnis oder Ordner in ein anderes verschieben. Ein Verzeichnis oder einen Ordner können sie jedoch nicht verschieben.

Sie erhalten eine Fehlermeldung, wenn Sie **Name** auf einer geöffneten Datei ausführen. Die Datei muß geschlossen werden, bevor sie umbenannt werden kann. Das Argument **Name** darf die Zeichen ***** (Platzhalter für mehrere Zeichen) und **?** (Platzhalter für ein einzelnes Zeichen) nicht enthalten.

Dateiverschieben 3

```
Public Sub Dateien_verschieben()
    Dim strQuelle As String
    Dim strZiel As String
    Dim objFSO As Object
```

```
    strQuelle = "C:\Temp\*.xls"
    If Dir(strQuelle) = "" Then MsgBox "Nix da!": Exit Sub
    strZiel = "C:\Temp\Freitag\"
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    objFSO.MoveFile strQuelle, strZiel
    Set objFSO = Nothing
End Sub
```

Datei löschen

```
Sub DelFile()
If Len(Dir("c:\windows\test.txt")) > 0 Then
   Kill "c:\windows\test.txt"
   MsgBox "Test.txt has been annihilated"
Else
   MsgBox "Test.Txt never existed"
End If
End Sub
```

Ron de Bruin (last update 18 Aug-2007)
Go back to the Excel tips page

On this page you can find example code to copy, move and delete files and folders.
There are three sections on this page :

1) Copy and Move files and folders
2) Delete files and folders
3) Special Folders
4) VBS script to clear the Temp folder

Copy and Move files and folders

Below are a few examples to copy and move files and folders.

For one file you can use the VBA Name and FileCopy function

and for entire folders or a lot of files use the other macro example's

```
Sub Copy_One_File()
    FileCopy "C:\Users\Ron\SourceFolder\Test.xls", "C:\Users\Ron\DestFolder\Test.xls"
End Sub

Sub Move_Rename_One_File()
'You can change the path and file name
    Name "C:\Users\Ron\SourceFolder\Test.xls" As "C:\Users\Ron\DestFolder\TestNew.xls"
End Sub
```

Filesystemobject example code's

```
Sub Copy_Folder()
'This example copy all files and subfolders from FromPath to ToPath.
'Note: If ToPath already exist it will overwrite existing files in this folder
'if ToPath not exist it will be made for you.
    Dim FSO As Object
    Dim FromPath As String
    Dim ToPath As String

    FromPath = "C:\Users\Ron\Data"  '<< Change
    ToPath = "C:\Users\Ron\Test"    '<< Change

    'If you want to create a backup of your folder every time you run this macro
    'you can create a unique folder with a Date/Time stamp.
    'ToPath = "C:\Users\Ron\" & Format(Now, "yyyy-mm-dd h-mm-ss")

    If Right(FromPath, 1) = "\" Then
        FromPath = Left(FromPath, Len(FromPath) - 1)
    End If

    If Right(ToPath, 1) = "\" Then
        ToPath = Left(ToPath, Len(ToPath) - 1)
    End If

    Set FSO = CreateObject("scripting.filesystemobject")

    If FSO.FolderExists(FromPath) = False Then
        MsgBox FromPath & " doesn't exist"
        Exit Sub
    End If

    FSO.CopyFolder Source:=FromPath, Destination:=ToPath
    MsgBox "You can find the files and subfolders from " & FromPath & " in " & ToPath
```

```
End Sub


Sub Move_Rename_Folder()
'This example move the folder from FromPath to ToPath.
    Dim FSO As Object
    Dim FromPath As String
    Dim ToPath As String

    FromPath = "C:\Users\Ron\Data"  '<< Change
    ToPath = "C:\Users\Ron\Test"    '<< Change
' Hinweis: Das Verzeichnis DATA wird in das Verzeichnis TEST als Unterverzeichnis verschoben
    'Note: It is not possible to use a folder that exist in ToPath

    If Right(FromPath, 1) = "\" Then
        FromPath = Left(FromPath, Len(FromPath) - 1)
    End If

    If Right(ToPath, 1) = "\" Then
        ToPath = Left(ToPath, Len(ToPath) - 1)
    End If

    Set FSO = CreateObject("scripting.filesystemobject")

    If FSO.FolderExists(FromPath) = False Then
        MsgBox FromPath & " doesn't exist"
        Exit Sub
    End If

    If FSO.FolderExists(ToPath) = True Then
        MsgBox ToPath & " exist, not possible to move to a existing folder"
        Exit Sub
    End If

    FSO.MoveFolder Source:=FromPath, Destination:=ToPath
    MsgBox "The folder is moved from " & FromPath & " to " & ToPath

End Sub


Sub Copy_Files_Dates()
'This example copy all files between certain dates from FromPath to ToPath.
'You can also use this to copy the files from the last ? days
```

```
'If Fdate >= Date - 30 Then
'Note: If the files in ToPath already exist it will overwrite
'existing files in this folder
    Dim FSO As Object
    Dim FromPath As String
    Dim ToPath As String
    Dim Fdate As Date
    Dim FileInFromFolder As Object

    FromPath = "C:\Users\Ron\Data"  '<< Change
    ToPath = "C:\Users\Ron\Test"    '<< Change

    If Right(FromPath, 1) <> "\" Then
        FromPath = FromPath & "\"
    End If

    If Right(ToPath, 1) <> "\" Then
        ToPath = ToPath & "\"
    End If

    Set FSO = CreateObject("scripting.filesystemobject")

    If FSO.FolderExists(FromPath) = False Then
        MsgBox FromPath & " doesn't exist"
        Exit Sub
    End If

    If FSO.FolderExists(ToPath) = False Then
        MsgBox ToPath & " doesn't exist"
        Exit Sub
    End If

    For Each FileInFromFolder In FSO.getfolder(FromPath).Files
        Fdate = Int(FileInFromFolder.DateLastModified)
        'Copy files from 1-Oct-2006 to 1-Nov-2006
        If Fdate >= DateSerial(2006, 10, 1) And Fdate <= DateSerial(2006, 11, 1) Then
            FileInFromFolder.Copy ToPath
        End If
    Next FileInFromFolder

    MsgBox "You can find the files from " & FromPath & " in " & ToPath

End Sub
```

```
Sub Copy_Certain_Files_In_Folder()
'This example copy all Excel files from FromPath to ToPath.
'Note: If the files in ToPath already exist it will overwrite
'existing files in this folder
    Dim FSO As Object
    Dim FromPath As String
    Dim ToPath As String
    Dim FileExt As String

    FromPath = "C:\Users\Ron\Data"  '<< Change
    ToPath = "C:\Users\Ron\Test"    '<< Change

    FileExt = "*.xl*"  '<< Change
    'You can use *.* for all files or *.doc for word files

    If Right(FromPath, 1) <> "\" Then
        FromPath = FromPath & "\"
    End If

    Set FSO = CreateObject("scripting.filesystemobject")

    If FSO.FolderExists(FromPath) = False Then
        MsgBox FromPath & " doesn't exist"
        Exit Sub
    End If

    If FSO.FolderExists(ToPath) = False Then
        MsgBox ToPath & " doesn't exist"
        Exit Sub
    End If

    FSO.CopyFile Source:=FromPath & FileExt, Destination:=ToPath
    MsgBox "You can find the files from " & FromPath & " in " & ToPath

End Sub


Sub Move_Certain_Files_To_New_Folder()
'This example move all Excel files from FromPath to ToPath.
'Note: It will create the folder ToPath for you with a date-time stamp
    Dim FSO As Object
    Dim FromPath As String
    Dim ToPath As String
```

```
    Dim FileExt As String
    Dim FNames As String

    FromPath = "C:\Users\Ron\Data"  '<< Change
    ToPath = "C:\Users\Ron\" & Format(Now, "yyyy-mm-dd h-mm-ss") _
          & " Excel Files" & "\"    '<< Change only the destination folder

    FileExt = "*.xl*"   '<< Change
    'You can use *.* for all files or *.doc for word files

    If Right(FromPath, 1) <> "\" Then
       FromPath = FromPath & "\"
    End If

    FNames = Dir(FromPath & FileExt)
    If Len(FNames) = 0 Then
       MsgBox "No files in " & FromPath
       Exit Sub
    End If

    Set FSO = CreateObject("scripting.filesystemobject")

    FSO.CreateFolder (ToPath)

    FSO.MoveFile Source:=FromPath & FileExt, Destination:=ToPath
    MsgBox "You can find the files from " & FromPath & " in " & ToPath

End Sub
```

Delete files and folders

Important !
Read this page from Chip Pearson first
http://www.cpearson.com/excel/Recycle.htm

From Chip's site :
You need to remember, though, that Kill permanently deletes the file.
There is no way to "undo" the delete. The file is not sent to the Windows Recycle Bin
(Same for the macro's that use the filesystemobject)

```
Sub DeleteExample1()
```

```
'You can use this to delete all the files in the folder Test
    On Error Resume Next
    Kill "C:\Users\Ron\Test\*.*"
    On Error GoTo 0
End Sub

Sub DeleteExample2()
'You can use this to delete all xl? files in the folder Test
    On Error Resume Next
    Kill "C:\Users\Ron\Test\*.xl*"
    On Error GoTo 0
End Sub

Sub DeleteExample3()
'You can use this to delete one xls file in the folder Test
    On Error Resume Next
    Kill "C:\Users\Ron\Test\ron.xls"
    On Error GoTo 0
End Sub

Sub DeleteExample4()
'You can use this to delete the whole folder
'Note: RmDir delete only a empty folder
    On Error Resume Next
    Kill "C:\Users\Ron\Test\*.*"    ' delete all files in the folder
    RmDir "C:\Users\Ron\Test\"  ' delete folder
    On Error GoTo 0
End Sub

Sub Delete_Whole_Folder()
'Delete whole folder without removing the files first like in DeleteExample4
    Dim FSO As Object
    Dim MyPath As String

    Set FSO = CreateObject("scripting.filesystemobject")

    MyPath = "C:\Users\Ron\Test"  '<< Change

    If Right(MyPath, 1) = "\" Then
        MyPath = Left(MyPath, Len(MyPath) - 1)
    End If

    If FSO.FolderExists(MyPath) = False Then
        MsgBox MyPath & " doesn't exist"
```

```
        Exit Sub
    End If

    FSO.deletefolder MyPath

End Sub


Sub Clear_All_Files_And_SubFolders_In_Folder()
'Delete all files and subfolders
'Be sure that no file is open in the folder
    Dim FSO As Object
    Dim MyPath As String

    Set FSO = CreateObject("scripting.filesystemobject")

    MyPath = "C:\Users\Ron\Test"  '<< Change

    If Right(MyPath, 1) = "\" Then
        MyPath = Left(MyPath, Len(MyPath) - 1)
    End If

    If FSO.FolderExists(MyPath) = False Then
        MsgBox MyPath & " doesn't exist"
        Exit Sub
    End If

    On Error Resume Next
    'Delete files
    FSO.deletefile MyPath & "\*.*", True
    'Delete subfolders
    FSO.deletefolder MyPath & "\*.*", True
    On Error GoTo 0

End Sub
```

SpecialFolders

How do I get the path of a special folder and open the folder ?

```
Sub GetSpecialFolder()
'Special folders are : AllUsersDesktop, AllUsersStartMenu
'AllUsersPrograms, AllUsersStartup, Desktop, Favorites
'Fonts, MyDocuments, NetHood, PrintHood, Programs, Recent
'SendTo, StartMenu, Startup, Templates
```

```
'Get Favorites folder and open it
    Dim WshShell As Object
    Dim SpecialPath As String

    Set WshShell = CreateObject("WScript.Shell")
    SpecialPath = WshShell.SpecialFolders("Favorites")
    MsgBox SpecialPath
    'Open folder in Explorer
    Shell "explorer.exe " & SpecialPath, vbNormalFocus
End Sub


Sub VBA_GetSpecialFolder_functions()
'Here are a few VBA path functions
    MsgBox Application.Path
    MsgBox Application.DefaultFilePath
    MsgBox Application.TemplatesPath
    MsgBox Application.StartupPath
    MsgBox Application.UserLibraryPath
    MsgBox Application.LibraryPath
End Sub
```

Temp folder

Without code you can do this to open the temp folder

Start>Run
Enter %temp%
OK

Or use one of the two code examples

```
Sub GetTempFolder_1()
    MsgBox Environ("Temp")
    'Open folder in Explorer
    Shell "explorer.exe " & Environ("Temp"), vbNormalFocus
End Sub

Sub GetTempFolder_2()
    Dim FSO As Object, TmpFolder As Object
```

```
    Set FSO = CreateObject("scripting.filesystemobject")
    Set TmpFolder = FSO.GetSpecialFolder(2)
    MsgBox TmpFolder
    'Open folder in Explorer
    Shell "explorer.exe " & TmpFolder, vbNormalFocus
End Sub
```

0 = The Windows folder contains files installed by the Windows operating sys
1 = The System folder contains libraries, fonts, and device drivers

VBS script to clear the Temp folder

It is smart to delete all files and folders in your temp folder at least once a week to avoid problems.
Important: Do this always after you reboot your system.

Manual you can use this to open the folder and then delete all files and folders in the Temp folder.

Start>Run
Enter %temp%
OK

But it is easier to use a vbs file on your desktop to do this.
A vbs file is a text file with script in it with a vbs extension.

You simply double click on the file then on your desktop and it will do all the work for you.

MVP Michael Harris posted a great script in the newsgroup
http://groups.google.com/groups?threadm=%23bXVsIHnAHA.920%40tkmsftngp02

1) Open Notepad
2) Copy/Paste the script in Notepad
3) Save the file as DeleteTempFiles.txt
4) Change the extension from txt to vbs

Or download the vbs file in a zip file
DeleteTempFiles.zip

File Manipulation from VBA

Following are some of the functions that you can frequently use to copy, move and rename your file from VBA. You can either copy the following code directly in a module or alternatively download the attached file. For renaming a file there are two methods given for this...Try this out!

```
Option Explicit


Function blnFileExists(strFile As String) As Boolean
    Dim objFileSystem       As Object

    Set objFileSystem = CreateObject("Scripting.FileSystemObject")
    If objFileSystem.FileExists(strFile) Then
        blnFileExists = True
    Else
        blnFileExists = False
    End If

    Set objFileSystem = Nothing
End Function


Sub CopyFile(strFileToCopy As String, strTargetFolder As String)
    Dim objFileSystem       As Object
    Dim strFileName         As String

    strFileName = Mid(strFileToCopy, InStrRev(strFileToCopy, "\") + 1, 9999) ' "test.xls" ' change to match the strFileName name
    strTargetFolder = strTargetFolder & "\"
```

```
    Set objFileSystem = CreateObject("Scripting.FileSystemObject")

    If Not blnFileExists(strFileToCopy) Then
        MsgBox strFileName & " does not exist!", vbExclamation, "Source File Missing"
    ElseIf Not objFileSystem.FileExists(strTargetFolder & strFileName) Then
        objFileSystem.CopyFile (strFileToCopy), strTargetFolder, True
    Else
        MsgBox strTargetFolder & "\" & strFileName & " already exists!", vbExclamation, "Destination File Exists"
    End If

    Set objFileSystem = Nothing
End Sub

Sub MoveFile(strFileToMove As String, strTargetFolder As String)
    Dim objFileSystem           As Object
    Dim strFileName             As String

    strFileName = Mid(strFileToMove, InStrRev(strFileToMove, "\") + 1, 9999)
    strTargetFolder = strTargetFolder & "\"

    Set objFileSystem = CreateObject("Scripting.FileSystemObject")

    If Not blnFileExists(strFileToMove) Then
        MsgBox strFileName & " does not exist!", vbExclamation, "Source File Missing"
    ElseIf Not objFileSystem.FileExists(strTargetFolder & strFileName) Then
        objFileSystem.CopyFile (strFileToMove), strTargetFolder
    Else
```

```
        MsgBox strTargetFolder & "\" & strFileName & " already exists!", vbExclamation, "Destination File Exists"
    End If

    Set objFileSystem = Nothing
End Sub

Sub RenameFile1(strCompleteFilePath As String, strNewFileName As String)
    Dim strContainingFolder As String
    strContainingFolder = Left(strCompleteFilePath, InStrRev(strCompleteFilePath, "\"))
    Name strCompleteFilePath As strContainingFolder & strNewFileName
End Sub

Sub RenameFile2(strCompleteFilePath As String, strNewFileName As String)
    Dim strContainingFolder         As String
    Dim objFileSystem               As Object

    strContainingFolder = Left(strCompleteFilePath, InStrRev(strCompleteFilePath, "\"))
    Set objFileSystem = CreateObject("Scripting.FileSystemObject")
    objFileSystem.MoveFile strCompleteFilePath, strContainingFolder & strNewFileName

    Set objFileSystem = Nothing
End Sub
```

Datei umbenennen

```
Name "C:\test\mappe1.csv" As "C:\test\mappe1.txt"
```

Datei umbenennen (ev. existierende Datei löschen)

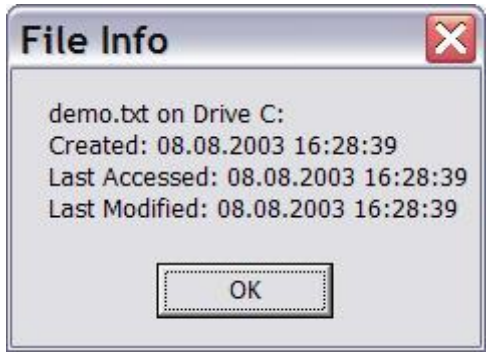
The instruction to rename a file is Name *oldname* As *newname*. If the path for the new filename is different from the path of the old filename, the file will be moved. You must delete the existing file first, using Kill *filename*. For example:

```
Sub MoveThatFile()
Dim strSource As String
Dim strTarget As String

strSource = "C:My DocumentsMyFile.pdf"
strTarget = "F:NetworkfolderNewFile.pdf"

' Test if target file exists
If Not Dir(strTarget) = "" Then
' Yes, so delete it
Kill strTarget
End If

' Move source file
Name strSource As strTarget
End Sub
```

Dateieigenschaften anzeigen+ändern

Hier sehen Sie ein Beispiel wie man Datei-Informationen abfragen kann,.. z.B. die Unversehrtheit bestimmter Dateien zu prüfen ;-))

```
Sub Give_FileInfo()
   Dim myFSO As Object, myFile As Object, strInfo As String
   Dim chkFile As String
   Set myFSO = CreateObject("Scripting.FileSystemObject")
   'chkFile kann als Parameter übergeben werden oder
   'über eine Inputbox abgefragt werden
   chkFile = "C:\Demo.txt"
   Set myFile = myFSO.GetFile(chkFile)
   strInfo = myFile.Name & " on Drive " & UCase(myFile.Drive) & vbCrLf
   strInfo = strInfo & "Created: " & myFile.DateCreated & vbCrLf
   strInfo = strInfo & "Last Accessed: " & myFile.DateLastAccessed & vbCrLf
   strInfo = strInfo & "Last Modified: " & myFile.DateLastModified
   MsgBox strInfo, 0, "File Info"
End Sub
```



http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Dateieigenschaften&action=edit&section=T-1

Über Dateieigenschaften

Über VBA-Prozeduren können Dateieigenschaften gelesen und geschrieben werden. Voraussetzung hierfür ist, dass das jeweilige Dokument geöffnet ist.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Dateieigenschaften&action=edit&section=T-2

Programmierbeispiele

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Dateieigenschaften&action=edit&section=T-3

Dateieigenschaften lesen

- Prozedur: ReadDocumentProperties
- Art: Sub
- Modul: Standardmodul
- Zweck: Dateieigenschaften in eine Tabelle einlesen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Datenbereich leeren
    - Fehlerroutine starten
    - Rahmen um die BuiltInDocumentProperties bilden
    - Schleife über alle Elemente bilden

- Den Namen der Eigenschaft eintragen
- Den Wert der Eigenschaft eintragen
- Den Typ der Eigenschaft eintragen
- Wenn ein Fehler aufgetreten ist...
- Den Fehlerwert eintragen
- Fehler-Objekt zurücksetzen
- Rahmen um die CustomDocumentProperties bilden
- Schleife über alle Elemente bilden
- Den Namen der Eigenschaft eintragen
- Den Wert der Eigenschaft eintragen
- Den Typ der Eigenschaft eintragen
- Wenn ein Fehler aufgetreten ist...
- Den Fehlerwert eintragen
- Fehler-Objekt zurücksetzen

- Code:

```
Sub ReadDocumentProperties()
   Dim iRow As Integer
   Range("A4:F35").ClearContents
   On Error Resume Next
   With ActiveWorkbook.BuiltinDocumentProperties
      For iRow = 1 To .Count
         Cells(iRow + 3, 1).Value = .Item(iRow).Name
         Cells(iRow + 3, 2).Value = .Item(iRow).Value
         Cells(iRow + 3, 3).Value = .Item(iRow).Type
         If Err.Number <> 0 Then
            Cells(iRow + 3, 2).Value = CVErr(xlErrNA)
            Err.Clear
```

```
         End If
      Next iRow
   End With
   With ActiveWorkbook.CustomDocumentProperties
      For iRow = 1 To .Count
         Cells(iRow + 3, 5).Value = .Item(iRow).Name
         Cells(iRow + 3, 6).Value = .Item(iRow).Value
         Cells(iRow + 3, 7).Value = .Item(iRow).Type
         If Err.Number <> 0 Then
            Cells(iRow + 3, 6).Value = CVErr(xlErrNA)
            Err.Clear
         End If
      Next iRow
   End With
   On Error GoTo 0
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Dateieigenschaften&action=edit&section=T-4

Dateieigenschaften schreiben

- Prozedur: WriteDocumentProperties
- Art: Sub
- Modul: Standardmodul
- Zweck: Dateieigenschaften in eine Datei schreiben
- Ablaufbeschreibung:
  - Variablendeklaration
  - Aktives Blatt an eine Objekt-Variable übergeben
  - Wenn die Zelle A4 leer ist...
  - Warnton
  - Warnmeldung
  - Prozedur verlassen
  - Neue Arbeitsmappe anlegen

- Rahmen um die BuiltInDocumentProperties bilden
- Eine Schleife um den Datenbereich bilden
- Wenn die Zelle in Spalte A der aktuellen Zeile leer ist, Prozedur verlassen
- Wenn sich in Spalte B der aktuellen Zeile kein Fehlerwert befindet...
- Wert für die Dateieigenschaft gem. Spalte A der aktuellen Zeile festlegen
- Rahmen um die CustomDocumentProperties bilden
- Eine Schleife um den Datenbereich bilden
- Eine benutzerdefinierte Eigenschaft hinzufügen
- Vollzugsmeldung anzeigen

- Code:

```
Sub WriteDocumentProperties()
   Dim wks As Worksheet
   Dim iRow As Integer
   Set wks = ActiveSheet
   If IsEmpty(Range("A4")) Then
      Beep
      MsgBox "Sie müssen zuerst die Eigenschaften einlesen!"
      Exit Sub
   End If
   Workbooks.Add
   With ActiveWorkbook.BuiltinDocumentProperties
      For iRow = 4 To 35
         If IsEmpty(wks.Cells(iRow, 1)) Then Exit For
         If IsError(wks.Cells(iRow, 2)) = False Then
            .Item(wks.Cells(iRow, 1).Value) = wks.Cells(iRow, 2).Value
         End If
      Next iRow
   End With
   With ActiveWorkbook.CustomDocumentProperties
      For iRow = 4 To 4
         .Add Name:=wks.Cells(iRow, 5).Value, LinkToContent:=False, _
            Type:=msoPropertyTypeDate, Value:=wks.Cells(iRow, 6).Value
      Next iRow
   End With
```

```
   MsgBox "Die editierbaren Dateieigenschaften wurden auf diese neue" & vbLf & _
      "Arbeitsmappe übertragen, bitte prüfen."
End Sub
```

Datei (Excel Arbeitsmappe) öffnen

1.Version ohne Fehlerabfangen

```
DATEI = "C:\Hallo.xls"
Workbooks.Open Filename:=DATEI
```

2.Version mit Fehlerhinweis:

```
Sub Open_HN_BMD_KG()
   On Error GoTo FEHLER
   Workbooks.Open Filename:="F:\LeXor Transfer\HN-BMD-KG.xls"
   Exit Sub

FEHLER:
   MsgBox "Die Datei 'F:\LeXor Transfer\HN-BMD-KG.xls' wurde leider nicht gefunden."
End Sub
```

Datei (Word, PDF...) öffnen

Code der unter 32 Bit und 64 Bit gleichermaßen läuft:

```
Sub OEFFNE_DATEI(Dateipfad As String)


If Dir(Dateipfad) = "" Then
       MsgBox "Die Datei gibt's leider nicht"
     Else
       ActiveWorkbook.FollowHyperlink Dateipfad
End If



End Sub

Sub TESTE()

OEFFNE_DATEI ("C:\13.pdf")
```

End Sub

Nur für 32 bit (Blau ist die Lösung, die auch auf 64 bit läuft)

Beide Lösungen benötigen:

```
' Funktionen für die Funktion gbOpenFile
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal
lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
```

1.) WORD

```
Sub OEFFNE_ANHANG_KLEIN()

Dim DATEIPFAD As String
DATEIPFAD = "M:\Workshop Wiki für Betreuer\Projektumsetzung Bilanzierungscheckliste\Vorlagen von Georg\Anhang_kleine_GmbH_UGB_11.2.2008.doc"

    If ShellExecute(0, "open", DATEIPFAD & vbNullChar, vbNullString, vbNullString, SW_SHOWMAXIMIZED) > 32 Then
       Else
        If ShellExecute(0, "open", DATEIPFAD & vbNullChar, vbNullString, vbNullString, SW_SHOWMAXIMIZED) > 32 Then
            '
          Else
            MsgBox "DIE DATEI KONNTE LEIDER NICHT GEFUNDEN WERDEN"
        End If
    End If
End Sub
```

2.) PDF

```
Public Function gbOpenFile(rsPath As String) As Boolean
' Für PDF-Handbuch öffnen
    If ShellExecute(0, "open", ActiveWorkbook.Path & "\Handbuch\" & rsPath & vbNullChar, vbNullString, vbNullString, SW_SHOWMAXIMIZED) > 32 Then
        gbOpenFile = True
       Else
        If ShellExecute(0, "open", ActiveWorkbook.Path & "\" & rsPath & vbNullChar, vbNullString, vbNullString, SW_SHOWMAXIMIZED) > 32 Then
```

```
        gbOpenFile = True
      Else
        MsgBox "DAS HANDBUCH KONNTE LEIDER NICHT GEÖFFNET WERDEN"
      End If
   End If
End Function
```

Ich hatte diesen Code bei der Simplefibu im Einsatz, aber sobald man den Code unter 64-bit Excel öffnete kam ein Compilierungsfehler, weil er mit der folgenden Zeile nicht zurecht kam

```
' Funktionen für die Funktion gbOpenFile
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
```

Änderte man diese Zeile jedoch um auf Folgendes – dann klappte es. (Wichtig: die #-Zeichen erlauben ein If, else, End if auch außerhalb von einer Sub !)

```
#If VBA7 Then
    Private Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
        (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal
nShowCmd As Long) As Long
#Else
    Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
        (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal
nShowCmd As Long) As Long
#End If
```

Jemand schrieb, dass die korrekte Unterscheidung zwischen 32 bit und 64 nicht nur mit VBA7 erfolgen sollte, sondern besser so

```
#If Win64 Then
' Win64=true, Win32=true, Win16= false
#ElseIf Win32 Then
' Win32=true, Win16=false
#Else
' Win16=true
#End If
```

Und noch jemand schrieb, dass es besser so geht:

Corrected typo from the book "Microsoft Excel 2010 Power Programming with VBA".

```
#If vba7 and win64 then
  declare ptrsafe function ....
#Else
```

```
  declare function ....
#End If
```

Bei mir klappte aber gleich die erste Lösung in Simplefibu unter Exc el 2013 64 bit

Eine gesamte Auflistung zu API-Aufruf-Lösungen unter 32 / 64-Bitsystemen gibt's im Kapitel ALLGEMEINES – Unterkapitel 32/64-Bit API-Aufrufe

VERSION 3

```
'  Title: Lauches an application based on file extension
'


' ------------------------------------------------------------------------

' ---------------------------
' Constants & API Declarations
' ---------------------------

Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal
lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
Private Declare Function GetDesktopWindow Lib "user32" Alias "GetDesktopWindow" () As Long

' ----------
' Function
' ----------

Function StartDoc (DocName As String) as long
   Dim Scr_hDC as long

   Scr_hDC = GetDesktopWindow()

   ' Change "Open" to "Explore" to bring up file explorer
   StartDoc = ShellExecute(Scr_hDC, "Open", DocName, "", "C:\", 1)
End Function
```

Dateiinformationen auslesen

```
'  Title: Gets the file description information
```

```
'
' ---------------------------------------------------------------------------

'-------------------------------------------------------------
' FUNCTION: GetFileDescription
'
' Gets the file description information.
'
' IN: [strFilename] - name of file to get description of.
'
' Returns: Description (vbNullString if not found)
'-------------------------------------------------------------
'
Function GetFileDescription(ByVal sFile As String) As String
    Dim lVerSize As Long, lTemp As Long, lRet As Long
    Dim bInfo() As Byte
    Dim lpBuffer As Long
    Dim sDesc As String
    Dim sKEY As String
    Const sEXE As String = "\FileDescription"

    GetFileDescription = vbNullString

    '
    'Get the size of the file version info, allocate a buffer for it, and get the
    'version info.  Next, we query the Fixed file info portion, where the internal
    'file version used by the Windows VerInstallFile API is kept.  We then copy
    'the info into a string.
    '
    lVerSize = GetFileVersionInfoSize(sFile, lTemp)
    ReDim bInfo(lVerSize)
    If lVerSize > 0 Then
        lRet = GetFileVersionInfo(sFile, lTemp, lVerSize, VarPtr(bInfo(0)))
        If lRet <> 0 Then
            sKEY = GetNLSKey(bInfo)
            lRet = VerQueryValue(VarPtr(bInfo(0)), sKEY & sEXE, lpBuffer, lVerSize)
            If lRet <> 0 Then
                sDesc = Space$(lVerSize)
                lstrcpyn sDesc, lpBuffer, lVerSize
                GetFileDescription = sDesc
            End If
        End If
```

```
    End If
End Function
Private Function GetNLSKey(byteVerData() As Byte) As String
    Const strTRANSLATION$ = "\VarFileInfo\Translation"
    Const strSTRINGFILEINFO$ = "\StringFileInfo\"
    Const strDEFAULTNLSKEY$ = "040904E4"
    Const LOCALE_IDEFAULTLANGUAGE& = &H9&
    Const LOCALE_IDEFAULTCODEPAGE& = &HB&

    Static strLANGCP As String

    Dim lpBufPtr As Long
    Dim strNLSKey As String
    Dim fGotNLSKey As Integer
    Dim intOffset As Integer
    Dim lVerSize As Long
    Dim ltmp As Long
    Dim lBufLen As Long
    Dim lLCID As Long
    Dim strTmp As String

    On Error GoTo GNLSKCleanup

    If VerQueryValue(VarPtr(byteVerData(0)), strTRANSLATION, lpBufPtr, lVerSize) <> 0 Then ' (Pass byteVerData array via
reference to first element)
        If Len(strLANGCP) = 0 Then
            lLCID = GetUserDefaultLCID()
            If lLCID > 0 Then
                strTmp = Space$(8)

                GetLocaleInfoA lLCID, LOCALE_IDEFAULTCODEPAGE, strTmp, 8
                strLANGCP = StripTerminator(strTmp)
                While Len(strLANGCP) < 4
                    strLANGCP = gsZERO & strLANGCP
                Wend

                GetLocaleInfoA lLCID, LOCALE_IDEFAULTLANGUAGE, strTmp, 8
                strLANGCP = StripTerminator(strTmp) & strLANGCP
                While Len(strLANGCP) < 8
                    strLANGCP = gsZERO & strLANGCP
                Wend
            End If
```

```
        End If

        If VerQueryValue(VarPtr(byteVerData(0)), strLANGCP, ltmp, lBufLen) <> 0 Then
            strNLSKey = strLANGCP
        Else
            For intOffset = 0 To lVerSize - 1 Step 4
                CopyMemory ltmp, ByVal lpBufPtr + intOffset, 4
                strTmp = Hex$(ltmp)
                While Len(strTmp) < 8
                    strTmp = gsZERO & strTmp
                Wend

                strNLSKey = strSTRINGFILEINFO & Right$(strTmp, 4) & Left$(strTmp, 4)

                If VerQueryValue(VarPtr(byteVerData(0)), strNLSKey, ltmp, lBufLen) <> 0 Then
                    fGotNLSKey = True
                    Exit For
                End If
            Next

            If Not fGotNLSKey Then
                strNLSKey = strSTRINGFILEINFO & strDEFAULTNLSKEY
                If VerQueryValue(VarPtr(byteVerData(0)), strNLSKey, ltmp, lBufLen) <> 0 Then
                    fGotNLSKey = True
                End If
            End If
        End If
    End If

GNLSKCleanup:
    If fGotNLSKey Then
        GetNLSKey = strNLSKey
    End If
End Function
```

Dateiname der aktuellen Datei ohne Dateiendung

Manchmal möchte man den Dateinamen einer Datei prüfen unabhängig von der Dateiendung.

```
MsgBox Left(ActiveWorkbook.Name, InStr(ActiveWorkbook.Name, ".") - 1)
```

Datei-pfad /-Name /-Gesamtpfad der aktuellen Datei

Es wird bei VBA unterschieden zwischen ActiveWorkbook und ThisWorkbook. Letzteres ist immer die Arbeitsmappe, wo sich aktuell der VBA-Code befindet - ActiveWorkbook ist jedoch dann wichtig, wenn man mit einer unsichtbaren Vorlage - wie etwa den Powertools - aktive Arbeitsmappen bearbeiten möchte.

```
Dim DATEINAME
Dim DATEIPFAD
Dim DATEIGESAMT

DATEIGESAMT = ActiveWorkbook.FullName
DATEINAME = ActiveWorkbook.Name
DATEIPFAD = ActiveWorkbook.Path
```

Geht es um eine Datei, die nicht offen ist und man hat nur irgendwo einen gesamten Dateipfad - zB C:\Eigene Dateien\Test.csv - dann kann man dennoch den Pfad und den Dateiname extrahieren

```
Sub TEST()
Dim DATEIGESAMT
Dim DATEIPFAD
Dim DATEINAME

DATEIGESAMT = "C:\Eigene Dateien\Test.csv"

DATEIPFAD = Left(DATEIGESAMT, InStrRev(DATEIGESAMT, "\"))
DATEINAME = Right(DATEIGESAMT, Len(DATEIGESAMT) - InStrRev(DATEIGESAMT, "\"))

End Sub
```

Dateipfad der aktuellen Datei für Datenimport im selben Verzeichnis festlegen

Möchte man den aktuellen Speicherpfad als aktuellen Pfad einstellen (zB um für einen Datenimport gleich im selben Verzeichnis zu sein wie die aktuelle Importvorlage):

```
ChDrive Left(ActiveWorkbook.FullName, 1)
ChDir Left(ActiveWorkbook.FullName, InStrRev(ActiveWorkbook.FullName, "\"))
```

Datei soll sich selbst neu öffnen

Ich hatte mal den Fall, dass ich von Excel aus ACCESS fernsteuere und da wird ein Pointer (irgendein Fernsteuerverweis) von der Exceldatei auf Access gelegt, welcher nicht per VBA entfernt werden kann, aber solange er besteht kann ACCESS nicht erneut angesteuert werden. Darum musste zuerst jeweils die Exceldatei geschlossen werden - wodurch der Pointer gelöscht wurde - und erst beim nächsten Öffnen der Vorlage klappt die Fernsteuerung von ACCESS wieder.

Weil dies nervte wollte ich das Schließen und Neuöffnen der Excel-Vorlage automatisieren:

dies geht nicht direkt - sondern nur mit Hilfe einer Hilfs-Datei:

1.) Die Vorlage selbst wird per VBA geschlossen und ruft ganz zuletzt noch die Hilfs-Datei auf
2.) Die Hilfsdatei selbst hat im Workbook.Open-Bereich den Befehl, dass sie sich selbst gleich wieder schließt, aber kurz davor soll sie zuletzt noch die ursprüngliche, sie aufrufende Datei erneut wieder aufrufen.

Fein ist es Application.Screenupdating für diesen Vorgangzu deaktivieren, dann sieht man das Schließen der usprünglichen Vorlage, das Öffnen der Rückaufruf-Datei, deren Schließen und das neue Öffnen der ursprünlichen Datei nicht.

Ich bekam dann leider immer wieder nicht debuggbare Anwendungs- und Objektdefinierter Fehler und beim Beheben stieß ich auf folgende zwei Fehlerquellmöglichkeiten

1.) Zu schneller erneuter Aufruf der Hauptvorlage, die noch gar nicht korrekt geschlossen ist

Wichtig: In der Regel wird man den Aufruf (das Öffnen) der Hilfsdatei vor dem Workbook.Close-Befehl der Hauptdatei machen (was Sinn macht, sonst muss man die Hilfsdtei im Workbook_BeforeClose-Teil öffnen - was aber dann zu mühsam dort zu exklusivisieren ist, damit nicht bei einem normalen Schließen der Datei durch den User die Hilfsdatei aufgerufen wird). Öffnet man aber die Hilfsdatei vor dem Workbook.Close-Befehl, wird noch in der Hauptvorlage der Workbook_BeforeClose-Teil abgearbeitet (zB Löschen von einer Symbolleiste etc.) während eventuell die Hilfsdatei schon offen ist und bereits versucht die Hauptvorlage erneut zu öffnen, obwohl sie ja gerade noch offen ist … - das führte bei mir immer wieder zu Anwendungs- und Objektfehlern (die gar kein Debuggen-Fenster anboten).

Abhilfe: einfach in der Hilfsdatei vor deren Rück-Aufruf der Hauptvorlage ein bis zwei Sekunden Pause einbauen. Und darauf achten, dass nach dem Aufruf der Hilfsdatei in der Hauptvorlage nicht mehr allzuviel Code kommt - ein automatisches Sichern der Hauptvorlage muss auf jeden Fall noch vor dem Aufruf der Hilfsdatei erfolgen.

2.) Doch auch als der obige, erste Punkt korrekt behoben war, gab es immer wieder noch die Fehler. Ich bemerkte, dass sie allerdings nur auftraten, wenn ich mit dem Makroaufrufen über die Symbolleiste arbeitete und nicht mit den Makros, die über meine Bild-Schaltflächen direkt in den Tabellen aufgerufen wurden. Selbst wenn ich dasselbe Marko einmal über ein Symbolleistensymbol aufrief und einmal mittels JPG-Bild, dem dasselbe Makro zugeordnet war, so klappt es beim Bild-Makro sehr gut - beim Symbolleisten-Aufruf kam die Fehlermeldung.

Abhilfe: leider nur, dass ich auf die Symbolleiste verzichte

Datei umbenennen

siehe weiter oben DATEIEN+ORDNER kopieren, verschieben, löschen, umbenennen

Datei verschieben

siehe weiter oben DATEIEN+ORDNER kopieren, verschieben, löschen, umbenennen

Dateien die in einer Tabelle aufgelistet sind, aus einem Ordner in einen anderen kopieren

Dateien die in einer Tabelle aufgelistet sind, aus einem Ordner in einen anderen kopieren

Tabelle1

	A	B	C
1	**Dateinamen**	**Kopier Ordner**	**Ziel Ordner**
2	text1.txt	C:\Test1\	C:\Test2\
3	text2.txt		

```
Option Explicit

Sub Copy_Files_based_on_Excel_Sheet()
'(c) Ramses
On Error GoTo myErrorHandler
Dim i As Long, Cr As Long, Cc As Integer
Dim wks As Worksheet, Qe As Integer
Dim strCFolder As String, strTFolder As String
Dim myFs As Object
'Erstellen des FileSystemObjectes
Set myFs = CreateObject("Scripting.FileSystemObject")
'Tabelle wo die Filenamen stehen
Set wks = Worksheets("Tabelle1")
'Spalte in der die Filenamen stehen
Cc = 1
'Letzten Eintrag festlegen
Cr = Cells(65536, Cc).End(xlUp).Row
'Ordner in dem die Dateien liegen
'Achtung: Mit Backslash am Schluss
strCFolder = "C:\test1\"
If Not myFs.folderexists(strCFolder) Then
```

```
    Qe = MsgBox("Der Ordner aus dem die Dateien kopiert werden sollen existiert nicht.", vbCritical + vbOKOnly, "Abbruch")
    Exit Sub
End If
'Alternativ wenn der Pfad in einer Zelle steht. Hier in B1
'strCFolder = wks.Cells(2,2)
'Ordner in den kopiert werden soll
strTFolder = "C:\test2\"
'Alternativ wenn der Pfad in einer Zelle steht. Hier in B1
'strTFolder = wks.Cells(2,3)
If Not myFs.folderexists(strCFolder) Then
    Qe = MsgBox("Der Ordner in den die Dateien kopiert werden sollen existiert nicht.", vbCritical + vbOKOnly, "Abbruch")
    Exit Sub
End If
'Kopierschleife starten
'1 wenn die Filenamen in Zeile 1 beginnen,
'sonst ab welcher Zeile die Filenamen beginnen
For i = 1 To Cr
    myFs.CopyFile strCFolder & wks.Cells(i, Cc).Text, strTFolder & wks.Cells(i, Cc).Text
Next i
'Fehlerbehandlung
myErrorExit:
Exit Sub

myErrorHandler:
MsgBox (Err.Number & ": " & Err.Description)
Resume myErrorExit
End Sub
```

Dateien in einem Verzeichnis auslesen

```
Option Explicit

Private Type FILETIME
dwLowDateTime As Long
dwHighDateTime As Long
End Type
Private Const MAX_PATH = 260
Private Const FILE_ATTRIBUTE_DIRECTORY = &H10

Private Type WIN32_FIND_DATA
dwFileAttributes As Long
ftCreationTime As FILETIME
ftLastAccessTime As FILETIME
ftLastWriteTime As FILETIME
nFileSizeHigh As Long
nFileSizeLow As Long
```

```
dwReserved0 As Long
dwReserved1 As Long
cFileName As String * MAX_PATH
cAlternate As String * 14
End Type
Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As
Long) As Long
Private Declare Function FindFirstFile Lib "kernel32" Alias
"FindFirstFileA" (ByVal lpFileName As String, lpFindFileData As
WIN32_FIND_DATA) As Long
Private Declare Function FindNextFile Lib "kernel32" Alias
"FindNextFileA" _
(ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long
Private nCount&

'Main code =================

Private Sub GetDirectoryListing(ByVal Root$)
' Function to calculate bytes used in Root$ and all subdirectories of Root$.
' Root$ should be entered in the form c:\Dir

Dim FData As WIN32_FIND_DATA
Dim fHand&
Dim sPath$
Dim StillOK&
Dim ByteTotal&
Dim nPos%
Dim DirName$, FileName$

sPath$ = Root$ + "\*.*"

fHand& = FindFirstFile(sPath$, FData)

If fHand& <= 0 Then
Exit Sub
End If

ByteTotal& = 0
Do
If (FData.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY) =
FILE_ATTRIBUTE_DIRECTORY Then
nPos% = InStr(FData.cFileName, Chr$(0))
DirName$ = Left$(FData.cFileName, nPos% - 1)
If DirName$ <> "." And DirName$ <> ".." Then
GetDirectoryListing Root$ + "\" + DirName$
End If
Else
nCount& = nCount& + 1
nPos% = InStr(FData.cFileName, Chr$(0))
FileName$ = Left$(FData.cFileName, nPos% - 1)
Cells(nCount&, 1).Value = Root$ + "\" + FileName$
'
>>>>>>>>>>>>>>>>>>>&
```

```
gt;>
' If you don't want the path just use:
' Cells(nCount&,1).value = FileName$
End If

StillOK& = FindNextFile(fHand&, FData)
Loop Until StillOK = 0

fHand& = FindClose(fHand&)

End Sub

Public Sub GetFileList()
Dim Path$

nCount& = 0
Path$ = InputBox("Enter the root for the file listing (e.g. 'c:\dir' or
c:")
If Len(Path$) = 0 Then Exit Sub
GetDirectoryListing Path$

End Sub
```

Dateien in Ordner und Unterordner suchen

1.) DIR

Der DIR-Befehl ist gänzlich ungeeignet, da er nicht zwischen Datei und Ordner unterscheiden kann.

Darum gabs auch folgende Frage zu seinem Code

ich habe in VBA ein Such funktion geschrieben, die nach einer bestimmten Datei in einem Ordner sucht.
Ich habe das mit Dir gelöst. Jetzt soll die Funktion auch die Unterordner durchsuchen. Geht das. (Nein)

```
Public Function findfiles(sFileName) As String
'--Funktion gibt ein zusammengestelltes String-Kürzel der gefundenen Dateierweiterungen zurück
'--z.B.: " msg ppt doc"

    '--In diesem Pad wird gesucht
    Const sPath = "W:\Proj\CBD\001_TEAMDRIVE\Tribo_Systems\100_Reports"


    findfiles = ""

    If sFileName = "" Then Exit Function

    If Dir(sPath & "\" & sFileName & ".msg") <> "" Then
```

```
        findfiles = findfiles & " msg"
    End If

    If Dir(sPath & "\" & sFileName & ".doc") <> "" Then
        findfiles = findfiles & " doc"
    End If

    If Dir(sPath & "\" & sFileName & ".pdf") <> "" Then
        findfiles = findfiles & " pdf"
    End If

    If Dir(sPath & "\" & sFileName & ".ppt") <> "" Then
        findfiles = findfiles & " ppt"
    End If

    If Dir(sPath & "\" & sFileName & ".pptx") <> "" Then
        findfiles = findfiles & " pptx"
    End If

    If Dir(sPath & "\" & sFileName & ".docx") <> "" Then
        findfiles = findfiles & " docx"
    End If


End Function
```

2.) FileSystemObject

Dazu wäre eine Lösung per **FileSystemObject** passender.

```
Sub aaa()
  MsgBox FindFiles("mappe16")
End Sub

Public Function FindFiles(sFileName) As String
 '--Funktion gibt ein zusammengestelltes String-Kürzel der gefundenen Dateierweiterungen zurück
 '--z.B.: " msg ppt doc"
  Dim FSO As Object, oFolder As Object, oDictF As Object

     '--In diesem Pad wird gesucht
  Const sPath = "W:\Proj\CBD\001_TEAMDRIVE\Tribo_Systems\100_Reports"

  If sFileName = "" Then Exit Function
  Set FSO = CreateObject("Scripting.FileSystemObject")
  Set oFolder = FSO.GetFolder(sPath)
  Set oDictF = CreateObject("Scripting.dictionary")

  prcFiles oFolder, oDictF, sFileName
  prcSubFolders oFolder, oDictF, sFileName
```

```
  If oDictF.Count Then
    FindFiles = Join(oDictF.keys, " ")
  Else
    FindFiles = sFileName & " nicht vorhanden."
  End If

 End Function

Sub prcFiles(oFolder, oDictF, sFileName)
  Dim oFile As Object
  For Each oFile In oFolder.Files
    With oFile
      If InStr(.Name, ".") Then
      If LCase(Left(.Name, InStrRev(.Name, ".") - 1)) = LCase(sFileName) Then
        oDictF(Right(.Name, Len(.Name) - InStrRev(.Name, "."))) = 0
      End If
      End If
    End With
  Next
End Sub

Sub prcSubFolders(oFolder, oDictF, sFileName)
  Dim oSubFolder As Object
  For Each oSubFolder In oFolder.SubFolders
    prcFiles oSubFolder, oDictF, sFileName
    prcSubFolders oSubFolder, oDictF, sFileName
  Next
End Sub
```

3.) Filesearch

Demo 1

```
'*************************************************************
'* DEMO (Word+Excel) zum Öffnen, Bearbeiten, Schließen aller    *
'* Dateien eines bestimmten Dateityps eines bestimmten Ordners. *
'* Erstellt am : 10.01.2007 von NoNet - www.excelei.de          *
'*************************************************************

Sub AlleDateienAbarbeiten()
    Dim Pfad, Dateien, AktuelleDatei
    Dateien = "Dat*.xls" ' hier den Dateityp anpassen !
    'Dateien = "*.txt" ' Für Excel: *.txt Dateien durchsuchen
    Pfad = "C:\Temp" 'Hier das Verzeichnis anpassen, das durchsucht werden soll
    With Application.FileSearch
        .LookIn = Pfad
        .SearchSubFolders = True
```

```
        .Filename = Dateien
        If .Execute() > 0 Then
            For i = 1 To .FoundFiles.Count
                AktuelleDatei = .FoundFiles(i)
                'MsgBox AktuelleDatei 'Dateinamen der Datei ausgeben
                'BearbeitenWord AktuelleDatei 'Datei mit WORD öffnen und Bearbeiten
                BearbeitenExcel AktuelleDatei 'Datei mit Excel öffnen und Bearbeiten
            Next i
        End If
    End With
End Sub

Sub BearbeitenExcel(Dateiname)
    Workbooks.Open Dateiname
    MsgBox Dateiname
    'Hier weitere Bearbeitungsschritte
    Workbooks(Split(Dateiname, "\")(UBound(Split(Dateiname, "\")))).Close False 'Datei schließ _
en
End Sub
```

Demo 2

Bestimmte Unterordner durchsuchen

von: Luu

Geschrieben am: 10.12.2009 13:12:59

Hallo zusammen!

Ich habe folgendes Problem:

Auf einem Server werden Ordner mit Projektnamen angelegt, von denen ich keine Ahnung habe wie sie heissen werden. Allerdings wird in diesen Projektordnern immer die gleiche Struktur sein.

Laufwerk Y

---Projekte

------Projekt A

---------Ordner 1
---------Ordner 2
------------neu
------Projekt B
---------Ordner 1
---------Ordner 2
...

Nun kann es noch hinzukommen, dass wie oben gezeigt, in einem der Unterordner noch ein beliebiger vorhanden ist.

Zur Zeit suche ich mit Filesearch nach einem bestimmten Ausdruck. Da aber ca. 140k Dateien durchsucht werden, dauert das "ewig". Gibt es eine Möglichkeit, dass VBA zwar in jedes Projekt reinschaut, dann aber nur immer in Ordner 2 und Unterordner die sich in diesem befinden?

Mit etwas stöbern habe ich in der VBA Hilfe unter "SearchFolders-Auflistung" etwas gefunden, wo bestimmte Ordner durchsucht werden können. Gearbeitet wird mit "Searchscope". Dabei kann ich allerdings nur den Arbeitsplatz angeben und nicht schon da einen bestimmten Ordner. Weiss jemand wie ich das ändern könnte, dann hätte ich da einen Ansatz.

Sollten noch Verständnisfragen sein, bitte melden. Bin für alles dankbar.

Gruß
Luu

Betrifft: AW: Bestimmte Unterordner durchsuchen
von: fcs
Geschrieben am: 10.12.2009 16:44:13

Hallo Luu,

hier eine Mischung aus Dir und FileSearch. Allerdings nicht an einer Mamut-Dateistruktur getestet.

Gruß

Franz

```
'Erstellt unter Excel 2003
Sub FindFiles_Projekte_Ordner2()
  Dim objFS_Ordner2 As FileSearch
  Dim obj_P As Variant, obj_Ordner2 As Variant, arrFilesFound() As String
  Dim varProjektFolder As Variant, FileCount As Long
  On Error GoTo Fehler
  'Verzeichnis mit Projektordnern wählen
  With Application.FileDialog(msoFileDialogFolderPicker)
    .Title = "Bitte Verzeichnis mit ProjektOrdnern auswählen"
    If .Show = -1 Then
      varProjektFolder = .SelectedItems(1)
      'Ordner der Projekte finden
      obj_P = Dir(varProjektFolder & "\*", vbDirectory)
      Do Until obj_P = ""
          'Prüfen ob gefundenes Element einen Ordner2 enthält
          Select Case VBA.GetAttr(obj_P & Application.PathSeparator & "Ordner2")
            Case vbDirectory, vbDirectory + vbReadOnly, vbDirectory + vbReadOnly + vbArchive, _
              vbDirectory + vbArchive
              'Ordner2 durchsuchen
              Set objFS_Ordner2 = Application.FileSearch
              With objFS_Ordner2
                .NewSearch
                .LookIn = varProjektFolder & Application.PathSeparator _
                      & obj_P & Application.PathSeparator & "Ordner2"
                .SearchSubFolders = True
                'Exceldateien finden
                .Filename = "*.xl*"

                If .Execute > 0 Then
                  'Dateien in Array schrieben
                  For Each obj_Ordner2 In .FoundFiles
                    FileCount = FileCount + 1
                    ReDim Preserve arrFilesFound(1 To FileCount)
                    arrFilesFound(FileCount) = obj_Ordner2
                  Next
                End If
              End With
            Case Else
              'do nothing
          End Select
Resume01:
        'nächsten ProjektOrdner finden
        obj_P = VBA.Dir
    Loop
```

```
    End If
  End With
  If FileCount > 0 Then
    'gefundenen Dateien weiterverarbeiten
    'Dateiliste in neue Tabelle ausgeben
    Worksheets.Add
    For FileCount = 1 To FileCount
'       MsgBox "Datei " & FileCount & "  : " & arrFilesFound(FileCount)
      Cells(FileCount + 1, 1) = arrFilesFound(FileCount)
    Next
  Else
    MsgBox "No Files founds"
  End If
Fehler:
  With Err
    Select Case .Number
      Case 0
      Case 53 'Datei wurde nicht gefunden
        'MsgBox "Fehler-Nr. " & .Number & vbLf & .Description
        Resume Resume01
      Case 76 'Pfad nicht gefunden
        'MsgBox "Fehler-Nr. " & .Number & vbLf & .Description
        Resume Resume01
      Case Else
      MsgBox "Fehler-Nr. " & .Number & vbLf & .Description
    End Select
  End With
End Sub
```

Demo 3

Sehr gut klappt dieses: liest von wählbarem Verzeichnis alle Unterordner aus

```
'#######################################################
'#                                     #
'#      Diese Makros stammen von Bert Körn        #
'#         E-Mail: bert@bert-koern.de          #
'#       Homepage: http://www.bert-koern.de       #
'#                                     #
'#######################################################

' Muß erwähnt sein: Der API-Aufruf stammt nicht von mir.
' Die Quelle ist mir nicht mehr bekannt.

Public Type BROWSEINFO
   hOwner As Long
   pidlRoot As Long
```

```
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Sub Verzeichnisse_auflisten()
Dim Pfad1, Name1, Anzahl, X, X0, X1, X2, Verz, Anzverz, Größe
Dim TB1, TB2 As Worksheet
Dim msg As String
Set TB1 = ThisWorkbook.Worksheets(1)
Set TB2 = ThisWorkbook.Worksheets(2)
start = Now
TB1.[a:D] = ""
TB2.[a:D] = ""
'überflüssige Tabellenblätter löschen
If ThisWorkbook.Worksheets.Count > 2 Then
    Application.DisplayAlerts = False
    For X = 3 To ThisWorkbook.Worksheets.Count
    ThisWorkbook.Worksheets(3).Delete
    Next X
    Application.DisplayAlerts = True
End If

' Pfad abfragen
    msg = "Wählen Sie bitte einen Ordner aus:"
    Pfad1 = getdirectory(msg)
    If Pfad1 = "" Then Exit Sub
    Name1 = Dir(Pfad1, vbDirectory) ' Ersten Eintrag abrufen.
    TB1.[a2] = Pfad1
    Anzahl = 2
    TB1.[a1] = "Pfad"
    TB1.[b1] = "UnterVerz."
    TB1.[c1] = "Anz. Dateien"
    TB1.[d1] = "Datgröße in Verz."
    X0 = 2
    X1 = 2
    Do While TB1.Cells(Rows.Count, 1).End(xlUp).Row <> TB1.Cells(Rows.Count, 2).End(xlUp).Row
       For X2 = X0 To X1
```

```
        Pfad1 = TB1.Cells(X2, 1)  ' Pfad setzen.
        If Right(Pfad1, 1) <> "\" Then Pfad1 = Pfad1 & "\"
        Name1 = Dir(Pfad1, vbDirectory) ' Ersten Eintrag abrufen.
        Verz = 0
        Do While Name1 <> "" ' Schleife beginnen.
        ' Aktuelles und übergeordnetes Verzeichnis ignorieren.
        If Name1 <> "." And Name1 <> ".." Then
        ' Mit bit-weisem Vergleich sicherstellen, daß Name1 ein
        ' Verzeichnis ist.
        If (GetAttr(Pfad1 & Name1) And vbDirectory) = vbDirectory Then
        Anzahl = Anzahl + 1
        TB1.Cells(Anzahl, 1) = Pfad1 & Name1 & "\"
        Verz = Verz + 1
        'Eintrag nur anzeigen, wenn es sich um ein Verzeichnis handelt.
        End If
        End If
        Name1 = Dir ' Nächsten Eintrag abrufen.
        Loop
        TB1.Cells(X2, 2) = Verz
        Next X2
        X0 = X1 + 1
        X1 = X2
    Loop

'Dateien aus den Verzeichnissen auslesen

    Anzverz = TB1.Cells(Rows.Count, 1).End(xlUp).Row
    i = 1
    ii = 0
    For Verz = 2 To Anzverz
    Anzahl = 0
    Größe = 0
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFolder(TB1.Cells(Verz, 1))
    Set fc = f.Files

    For Each f1 In fc
    If i = 65536 Then
    ii = ii + 1
    ThisWorkbook.Worksheets.Add.Move After:=ThisWorkbook.Worksheets(ThisWorkbook.Worksheets.Count)
    ThisWorkbook.Worksheets(ii + 2).Name = "Dateien " & ii + 1
    Set TB2 = ThisWorkbook.Worksheets(ii + 2)
    i = 1
```

```
    End If
    i = i + 1
    Anzahl = Anzahl + 1
    TB2.Cells(i, 1) = f1.Name
    TB2.Cells(i, 2) = f & "\" & f1.Name
    'Hyperlink auf die Datei einfügen
    TB2.Hyperlinks.Add Anchor:=TB2.Cells(i, 2), Address:= _
       f & "\" & f1.Name
    TB2.Cells(i, 3) = FileLen(f1)
    TB2.Cells(i, 4) = FileDateTime(f1)
    Größe = Größe + FileLen(f1)
    Next
    TB1.Cells(Verz, 3) = Anzahl
    TB1.Cells(Verz, 4) = Größe / 1024 / 1024
    Next Verz
    'MsgBox (ii * 65536) + i

    ende = Now
    MsgBox "Anzahl der Verzeichnisse: " & Verz & Chr(13) & _
       "Anzahl der Dateien: " & (ii * 65536) + i & Chr(13) & _
       Chr(13) & "Dauer: " & Format(ende - start, "nn:ss")
End Sub

' Muß erwähnt sein: Diese Funktion stammt nicht von mir.
' Die Quelle ist mir nicht mehr bekannt.

Function getdirectory(Optional msg) As String
    Dim bInfo As BROWSEINFO
    Dim Path As String
    Dim r As Long, X As Long, pos As Integer
'   Ausgangsordner = Desktop
    bInfo.pidlRoot = 0&
'   Dialogtitel
    If IsMissing(msg) Then
       bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
    Else
       bInfo.lpszTitle = msg
    End If
'   Rückgabe des Unterverzeichnisses
    bInfo.ulFlags = &H1
'   Dialog anzeigen
    X = SHBrowseForFolder(bInfo)
'   Ergebnis gliedern
    Path = Space$(512)
```

```
    r = SHGetPathFromIDList(ByVal X, ByVal Path)
    If r Then
        pos = InStr(Path, Chr$(0))
        getdirectory = Left(Path, pos - 1)
    Else
        getdirectory = ""
    End If
End Function
```

DEMO 4

Unterordner suchen - Name nur teilweise bekannt

Hoffe, ich erhalte hier die Hilfe, die ich brauche

Arbeite in einem Excel-Tabellenblatt und will dieses per Macro nun in diversen Ordnern ablegen.
In den Zellen R8 bis max R22 befinden sich entweder 6-stellige Ids oder "".
Gibt es eine Id, so muß geprüft werden, ob es in "C:\works\" in bis zu 3 Unterebenen einen Ordner gibt (es sollte dann auch nur einen geben), der diese 6-stellige ID beinhaltet (z.B. "xxx123456zzz").
Gibt es ihn, so soll diese Datei in dessen Unterordner "LA" abgelegt werden.
Gibt es diesen Unterordner nicht, so muß er vorher angelegt werden.
Dasselbe gilt für alle weiteren vorkommenden IDs innerhalb R8:R22!
Jede ID, die keinen Ordner hat, ist zum Abschluß in einer Auflistung als MSBox darzustellen.

du willst also dieselbe Datei in x verschiede Ordner ablegen?
Dann probier das mal:

Code:

```
Option Explicit

Const strSuchPfad As String = "C:\works\"
Const intEbenen As Integer = 3
Dim arrV() As String
Dim o As Long
Dim i As Integer

Sub SucheOrdnerInDreiEbenen()
Dim shQ As Worksheet
Dim strDatei As String, strText As String
Dim j As Long, k As Long
Dim bolVorhanden As Boolean
Dim fs As Object
    Set fs = CreateObject("Scripting.FileSystemObject")
```

```
    Set shQ = ActiveSheet ' ggf. anpassen
    ReDim arrV(0) As String
    arrV(0) = strSuchPfad
    Call OrdnerListe(strSuchPfad)
    ReDim arrD(1)
    For k = 8 To 22
        If Trim(shQ.Range("R" & k)) <> "" Then
            For j = 0 To UBound(arrV)
'               If Split(arrV(j), "\")(UBound(Split(arrV(j), "\")) - 1) Like "*" & shQ.Range("R" & k) & "*" Then
                If fs.GetBaseName(arrV(j)) Like "*" & shQ.Range("R" & k) & "*" Then
                    If Dir(arrV(j) & "LA", vbDirectory) = "" Then MkDir (arrV(j) & "LA")
                    ThisWorkbook.SaveCopyAs arrV(j) & "LA\" & ThisWorkbook.Name
                    bolVorhanden = True
                    Exit For
                End If
            Next j
            If Not bolVorhanden Then strText = IIf(strText = "", "", strText & vbLf) & shQ.Range("R" & k)
            bolVorhanden = False
        End If
    Next k
    If strText <> "" Then
        MsgBox "Folgende ID's nicht gefunden:" & vbLf & strText, vbInformation, "Info"
    End If
    ReDim arrV(0) As String: o = 0: i = 0
End Sub

Private Sub OrdnerListe(strPfad As String)
    Dim strOrdner As String
    strOrdner = Dir(strPfad & "*.", vbDirectory)
    Do Until strOrdner = ""
        If strOrdner <> "." And strOrdner <> ".." Then
            If GetAttr(strPfad & strOrdner) And vbDirectory Then
                ReDim Preserve arrV(UBound(arrV) + 1) As String
                arrV(UBound(arrV)) = strPfad & strOrdner & "\"
            End If
        End If
        strOrdner = Dir()
    Loop
    If o < UBound(arrV) And UBound(Split(Replace(arrV(UBound(arrV)), strSuchPfad, ""), "\")) < intEbenen Then
        For o = o + 1 To UBound(arrV)
            OrdnerListe (arrV(o))
        Next o
    End If
End Sub
```

Vielen Dank, Klaus

Testlauf war perfekt.

DEMO 5

Auslesen Ordnerstruktur (Ordner, Unterordner und Dateinamen)

Hallo zsammen,

ich bin auf der Suche nach einem Makro...
Ich gebe einen Pfad an und das Makro spuckt mir alle ORdner, unterordner und enhtaltenen Dateinamen mit Endung aus.

also ungefährt so

c:\progs
c:\progs\games
c:\progs\games\spiel1.exe
c:\progs\games\spiel2.exe
c:\progs\games\spiel1.doc
c:\progs\office
c:\progs\office\word\....

Hoffe ihr könnt mir helfen... hab ein anderen foren gesuht, aber da werden entweder nur die direkten ordner ausgegeben oder die dateinamen aber nix in kombi :/

Hi, so geht's:

Code:

```
Sub ordner()
Dim pfad$, i%, anz%
pfad = "C:\test\" '### oder über InputBox einlesen mit \ am Ende
With Application.FileSearch
    .LookIn = pfad
    .NewSearch
    .SearchSubFolders = True
    .Filename = "*.*"
    .Execute
    anz = .FoundFiles.Count
    For i = 2 To anz
        Sheets(1).Cells(i, 1).Value = .FoundFiles(i)
    Next i
End With
End Sub
```

Gruß Bernd

Wie könnte man den code erweitern, wenn man nach xls-dateien sucht die mit ETB beginnen? gibts da eine möglichkeit?

so:

Code:

```
Sub ordner()
    Dim pfad$, i%, anz%
    pfad = "C:\test\" '### oder über InputBox einlesen mit \ am Ende
    With Application.FileSearch
        .NewSearch
        .LookIn = pfad
        .SearchSubFolders = True
        .Filename = "ETB*.*"
        .FileType = msoFileTypeExcelWorkbooks
        .Execute
        anz = .FoundFiles.Count
        For i = 2 To anz
            Sheets(1).Cells(i, 1).Value = .FoundFiles(i)
        Next i
    End With
End Sub
```

DEMO 6

Makro soll auch Unterordner durchsuchen

Hi Leute  ,

ich nutze in einem Makro folgende Ordnerabfrage:

Code:

```

Dim mySHFolderItem As Object, myObjShell As Object
    Dim myDefaultPath As Variant
    Dim myObjFolder As Object
    myDefaultPath = ""
    Set myObjShell = CreateObject("Shell.Application")
    Set myObjFolder = myObjShell.BrowseForFolder(0&, "Ordner auswählen...", 0&, myDefaultPath)
    If myObjFolder Is Nothing Then Exit Sub
    Set mySHFolderItem = myObjFolder.Self
    strPath = mySHFolderItem.Path & ("\")
    Application.ScreenUpdating = False
    strFile = Dir(strPath & "*.xls")
```

klappt auch super, bis auf die Tatsache, dass die Unterordner vom angegebenen Ordner nicht mitdurchsucht werden !

Hi,

mit DIR kannst Du nicht so einfach 'rekursiv' Unterordner durchsuchen. Nimm besser Application.FileSearch, das FSO oder gleich die Win-API.

Siehe vielleicht:
http://www.office-loesung.de/ftopic199191_0_0_asc.php
http://www.office-loesung.de/ftopic148247_0_0_asc.php
http://www.online-excel.de/fom/fo_read.php?f=3&bzh=0&h=120&ao=1#a123x

cu, Bernd

Lösung 1 von Bernd

versuche mal sowas. Ist M.E. zwar nicht die schnellste Möglichkeit, aber eine der Einfachsten. Ich bin mir nicht mehr sicher, ob Excel 2000 Application.FileSearch schon kennt?

Falls nicht, melde Dich nochmals, dann baue ich Dir das eben um.

HTH, Bernd
--

Code:

```
Option Explicit

Sub ListAllXLSFiles()
   Dim strPath As String, i As Long

   strPath = GetFolder("", "Ordner auswählen...")
   If strPath = "" Then Exit Sub
   If Right(strPath, 1) <> "\" Then strPath = strPath & "\"
   With Application.FileSearch
      .LookIn = strPath
      .SearchSubFolders = True
      .NewSearch
      .Filename = "*.xls"
      .FileType = msoFileTypeExcelWorkbooks
      If .Execute > 0 Then
         For i = 1 To .FoundFiles.Count
            Debug.Print .FoundFiles(i)
         Next
      End If
   End With
End Sub

Function GetFolder(Optional ByVal varDefDir As Variant = "", Optional ByVal strTitle As String = "")
    Dim objShell As Object, objFolder As Object

    GetFolder = ""
    Set objShell = CreateObject("Shell.Application")
    Set objFolder = objShell.BrowseForFolder(0, strTitle, 0, varDefDir)
    If Not objFolder Is Nothing Then GetFolder = objFolder.Self.Path
    Set objFolder = Nothing
    Set objShell = Nothing
End Function
```

Lösung 2 von Bernd

```
Option Explicit

Sub daten_uebernehmen()
   Dim Counter As Long
   Dim Addition As Long
   Dim avg As Long
   Dim i As Integer
   Dim j As Integer
   Dim strFile As String
   Dim strPath As String
   Dim loZeileZielmappe As Long
   Dim inSpalte As Integer
   Dim loZeileQuellmappe As Long
   Dim ZielDatumZeile As Long
   Dim Datum As String
   Dim ZielDatumSpalte As Long
   Dim loZaehler As Long
   Dim myDefaultPath As Variant
   Dim intCounter As Integer  ' by bst

   myDefaultPath = ""
   strPath = GetFolder(myDefaultPath, "Ordner auswählen...")
   If strPath = "" Then Exit Sub
   If Right(strPath, 1) <> "\" Then strPath = strPath & "\"

   Application.ScreenUpdating = False

   loZeileZielmappe = 6
   loZaehler = 6
   ZielDatumZeile = 5
   ZielDatumSpalte = 1
   Counter = 0
   i = 6

   With Application.FileSearch
      .LookIn = strPath
      .SearchSubFolders = True
      .NewSearch
      .Filename = "*.xls"
      .FileType = msoFileTypeExcelWorkbooks
      If .Execute > 0 Then
         For i = 1 To .FoundFiles.Count
            SplitPath .FoundFiles(i), strPath, strFile
            ' Debug.Print .FoundFiles(i), strPath, strFile
            If strFile <> ThisWorkbook.Name Then
               For inSpalte = 2 To 7
                  loZeileZielmappe = loZaehler
                  For loZeileQuellmappe = 8 To 32
                     Cells(loZeileZielmappe, inSpalte).Formula = "='" & strPath & "[" & strFile & "]" & "Tabelle1" & "'!" & Chr(inSpalte + 64) &
loZeileQuellmappe
                     loZeileZielmappe = loZeileZielmappe + 1
                  Next loZeileQuellmappe
               Next inSpalte
               Cells(ZielDatumZeile, ZielDatumSpalte).Formula = "='" & strPath & "[" & strFile & "]" & "tabelle1" & "'!A33"
            End If

            For intCounter = 1 To 25
               Cells(i, 8) = (Cells(i, 2) + Cells(i, 3) + Cells(i, 4) + Cells(i, 5) + Cells(i, 6)) / 5
               i = i + 1
            Next
```

```
            i = i + 2

            loZaehler = loZaehler + 27
            ZielDatumZeile = ZielDatumZeile + 27
            loZeileZielmappe = loZaehler
            strFile = Dir()
            Counter = Counter + 125
         Next
      End If
   End With

   Range("B6:G" & loZeileZielmappe).Copy
   Range("B6:G" & loZeileZielmappe).PasteSpecial Paste:=xlPasteValues
   Application.CutCopyMode = False
   Application.ScreenUpdating = True
   Addition = Range("L8")
   Range("L7") = Counter
   Range("K13") = Counter / 5
End Sub

Private Function GetFolder(Optional ByVal varDefDir As Variant = "", Optional ByVal strTitle As String = "")
    Dim objShell As Object, objFolder As Object

    GetFolder = ""
    Set objShell = CreateObject("Shell.Application")
    Set objFolder = objShell.BrowseForFolder(0&, strTitle, 0&, varDefDir)
    If Not objFolder Is Nothing Then GetFolder = objFolder.Self.Path
    Set objFolder = Nothing
    Set objShell = Nothing
End Function

Private Function SplitPath(ByVal strFullName As String, _
   ByRef strPath As String, ByRef strName As String) As Boolean

   Dim intPos As Integer

   intPos = InStrRev(strFullName, "\")
   If intPos > 0 Then
      strPath = Left(strFullName, intPos)
      strName = Mid(strFullName, intPos + 1)
   Else
      strPath = ""
      strName = strFullName
   End If
   SplitPath = intPos > 0
End Function
```

Dateisuche unter Excel2007 (Filesearch-Ersatz)

CODE 1

Hallo,

nachdem es dieses Objekt in XL2007 nicht mehr gibt und in anderen Versionen nicht richtig funktioniert (Sortieren in XL2000 geht z.B. nicht), hier eine andere Möglichkeit.

In einem Standardmodul:

Code:

```
Option Explicit

Public Enum SORT BY
    Sort by None
    Sort by Name
    Sort by Path
    Sort by Size
    Sort by Last Access
    Sort by Last Modyfy
    Sort by Date Create
End Enum

Public Enum SORT ORDER
    Sort Order Ascending
    Sort Order Descending
End Enum

Public Type FILEINFO
    strFilename As String
    strPath As String
    lngSize As Long
    dmtLastAccess As Date
    dmtLastModify As Date
    dmtDateCreate As Date
End Type

Public Sub Test()
    Dim objFileSearch As clsFileSearch
    Dim lngIndex As Long

    Set objFileSearch = New clsFileSearch
    With objFileSearch
        .CaseSenstiv = True
        .Extension = "*.xls"
        .FolderPath = "D:\"
        .SearchLike = "Test*"
        .SubFolders = True
        If .Execute(Sort by Size, Sort Order Descending) > 0 Then
            For lngIndex = 1 To .FileCount
                With .Files(lngIndex)
                    Debug.Print .strFilename, .lngSize
                End With
            Next
        End If
    End With
    Set objFileSearch = Nothing
End Sub
```

In einem Klassenmodul mit dem Namen **clsFileSearch**:

Code:

```
Option Explicit

Private Declare Function FindFirstFile Lib "kernel32.dll" Alias "FindFirstFileA" (
    ByVal lpFileName As String,
    ByRef lpFindFileData As WIN32 FIND DATA) As Long
```

```
Private Declare Function FindNextFile Lib "kernel32.dll" Alias "FindNextFileA" (
    ByVal hFindFile As Long,
    ByRef lpFindFileData As WIN32 FIND DATA) As Long
Private Declare Function FindClose Lib "kernel32.dll" (
    ByVal hFindFile As Long) As Long
Private Declare Function FileTimeToLocalFileTime Lib "kernel32.dll" (
    ByRef lpFileTime As FILETIME,
    ByRef lpLocalFileTime As FILETIME) As Long
Private Declare Function FileTimeToSystemTime Lib "kernel32.dll" (
    ByRef lpFileTime As FILETIME,
    ByRef lpSystemTime As SYSTEMTIME) As Long

Private Enum FILE ATTRIBUTE
    FILE ATTRIBUTE READONLY = &H1
    FILE ATTRIBUTE HIDDEN = &H2
    FILE ATTRIBUTE SYSTEM = &H4
    FILE ATTRIBUTE DIRECTORY = &H10
    FILE ATTRIBUTE ARCHIVE = &H20
    FILE ATTRIBUTE NORMAL = &H80
    FILE ATTRIBUTE TEMPORARY = &H100
End Enum

Private Const MAX PATH = 260&
Private Const INVALID HANDLE VALUE = -1&

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

Private Type WIN32 FIND DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX PATH
    cAlternate As String * 14
End Type

Private mlngFileCount As Long
Private mudtFiles() As FILEINFO
Private mstrFolderPath As String
Private mstrExtension As String
Private mstrSearchLike As String
```

```
Private mblnSubFolders As Boolean
Private mblnCaseSenstiv As Boolean

Friend Property Get Files(lngIndex As Long) As FILEINFO
    Files = mudtFiles(lngIndex)
End Property

Friend Property Get FileCount() As Long
    FileCount = mlngFileCount
End Property

Friend Property Let FolderPath(strFolderPath As String)
    mstrFolderPath = strFolderPath
End Property

Friend Property Let Extension(strExtension As String)
    mstrExtension = strExtension
End Property

Friend Property Let SearchLike(strSearchLike As String)
    mstrSearchLike = strSearchLike
End Property

Friend Property Let SubFolders(blnSubFolders As Boolean)
    mblnSubFolders = blnSubFolders
End Property

Friend Property Let CaseSenstiv(blnCaseSenstiv As Boolean)
    mblnCaseSenstiv = blnCaseSenstiv
End Property

Friend Function Execute(Optional enmSortBy As SORT BY = Sort by None,
    Optional enmSortOrder As SORT ORDER = Sort Order Ascending) As Long
    Call FindFiles(mstrFolderPath)
    If mlngFileCount > 1 And enmSortBy <> Sort by None Then
        Call prcSort(1, mlngFileCount, enmSortBy, enmSortOrder)
    Execute = mlngFileCount
End Function

Private Sub FindFiles(ByVal strFolderPath As String)
    Dim WFD As WIN32 FIND DATA, lngSearch As Long, strDirName As String
    On Error GoTo ErrorHandling
    If Right$(strFolderPath, 1) <> "\" Then strFolderPath = strFolderPath & "\"
    lngSearch = FindFirstFile(strFolderPath & "*.*", WFD)
    If lngSearch <> INVALID HANDLE VALUE Then
        Call GetFilesInFolder(strFolderPath)
        If mblnSubFolders Then
            Do
                If (WFD.dwFileAttributes And FILE ATTRIBUTE DIRECTORY) Then
                    strDirName = Left$(WFD.cFileName, InStr(WFD.cFileName, Chr$(0)) - 1)
                    If (strDirName <> ".") And (strDirName <> "..") Then
                        Call FindFiles(strFolderPath & strDirName)
                End If
            Loop While FindNextFile(lngSearch, WFD)
        End If
        FindClose lngSearch
    End If
    Exit Sub
ErrorHandling:
```

```
    MsgBox "Fehler " & CStr(Err.Number) & vbLf & vbLf &
        Err.Description, vbCritical, "Fehler"
End Sub

Private Sub GetFilesInFolder(ByVal strFolderPath As String)
    Dim WFD As WIN32_FIND_DATA, lngSearch As Long, strFilename As String
    Dim udtFiletime As FILETIME, udtSystemtime As SYSTEMTIME
    On Error GoTo ErrorHandling
    If Right$(strFolderPath, 1) <> "\" Then strFolderPath = strFolderPath & "\"
    lngSearch = FindFirstFile(strFolderPath & mstrExtension, WFD)
    If lngSearch <> INVALID_HANDLE_VALUE Then
        Do
            If (WFD.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY) <> FILE_ATTRIBUTE_DIRECTORY Then
                strFilename = Left$(WFD.cFileName, InStr(WFD.cFileName, Chr$(0)) - 1)
                If IIf(mblnCaseSenstiv, strFilename, LCase$(strFilename)) Like
                    IIf(mblnCaseSenstiv, mstrSearchLike, LCase$(mstrSearchLike)) Then
                    mlngFileCount = mlngFileCount + 1
                    ReDim Preserve mudtFiles(1 To mlngFileCount)
                    With mudtFiles(mlngFileCount)
                        .strPath = strFolderPath & strFilename
                        .strFilename = strFilename
                        .lngSize = WFD.nFileSizeLow
                        FileTimeToLocalFileTime WFD.ftCreationTime, udtFiletime
                        FileTimeToSystemTime udtFiletime, udtSystemtime
                        .dmtDateCreate = CDate(DateSerial(udtSystemtime.wYear, udtSystemtime.wMonth, udtSystemtime.wDay) + _
                            TimeSerial(udtSystemtime.wHour, udtSystemtime.wMinute, udtSystemtime.wSecond))
                        FileTimeToLocalFileTime WFD.ftLastAccessTime, udtFiletime
                        FileTimeToSystemTime udtFiletime, udtSystemtime
                        .dmtLastAccess = CDate(DateSerial(udtSystemtime.wYear, udtSystemtime.wMonth, udtSystemtime.wDay) +
                            TimeSerial(udtSystemtime.wHour, udtSystemtime.wMinute, udtSystemtime.wSecond))
                        FileTimeToLocalFileTime WFD.ftLastWriteTime, udtFiletime
                        FileTimeToSystemTime udtFiletime, udtSystemtime
                        .dmtLastModify = CDate(DateSerial(udtSystemtime.wYear, udtSystemtime.wMonth, udtSystemtime.wDay) +
                            TimeSerial(udtSystemtime.wHour, udtSystemtime.wMinute, udtSystemtime.wSecond))
                    End With
                End If
            End If
        Loop While FindNextFile(lngSearch, WFD)
        FindClose lngSearch
    End If
    Exit Sub
ErrorHandling:
    MsgBox "Fehler " & CStr(Err.Number) & vbLf & vbLf &
        Err.Description, vbCritical, "Fehler"
End Sub

Private Sub prcSort(lngLBorder As Long, lngUBorder As Long, enmSortBy As SORT BY, enmSortOrder As SORT ORDER)
    Dim lngIndex1 As Long, lngIndex2 As Long
    Dim udtBuffer As FILEINFO, vntTemp As Variant

    lngIndex1 = lngLBorder
    lngIndex2 = lngUBorder
    Select Case enmSortBy
      Case Sort by Name: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).strFileName
      Case Sort by Path: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).strPath
      Case Sort by Size: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).lngSize
      Case Sort by Last Access: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).dmtLastAccess
      Case Sort by Last Modyfy: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).dmtLastModify
      Case Sort by Date Create: vntTemp = mudtFiles((lngLBorder + lngUBorder) \ 2).dmtDateCreate
```

```
    End Select
    Do
        Select Case enmSortBy
          Case Sort by Name
            If enmSortOrder = Sort Order Ascending Then
                Do While mudtFiles(lngIndex1).strFileName < vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp < mudtFiles(lngIndex2).strFileName
                    lngIndex2 = lngIndex2 - 1
                Loop
              Else
                Do While mudtFiles(lngIndex1).strFileName > vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp > mudtFiles(lngIndex2).strFileName
                    lngIndex2 = lngIndex2 - 1
                Loop
            End If
          Case Sort by Path
            If enmSortOrder = Sort Order Ascending Then
                Do While mudtFiles(lngIndex1).strPath < vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp < mudtFiles(lngIndex2).strPath
                    lngIndex2 = lngIndex2 - 1
                Loop
              Else
                Do While mudtFiles(lngIndex1).strPath > vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp > mudtFiles(lngIndex2).strPath
                    lngIndex2 = lngIndex2 - 1
                Loop
            End If
          Case Sort by Size
            If enmSortOrder = Sort Order Ascending Then
                Do While mudtFiles(lngIndex1).lngSize < vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp < mudtFiles(lngIndex2).lngSize
                    lngIndex2 = lngIndex2 - 1
                Loop
              Else
                Do While mudtFiles(lngIndex1).lngSize > vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp > mudtFiles(lngIndex2).lngSize
                    lngIndex2 = lngIndex2 - 1
                Loop
            End If
          Case Sort by Last Access
            If enmSortOrder = Sort Order Ascending Then
                Do While mudtFiles(lngIndex1).dmtLastAccess < vntTemp
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While vntTemp < mudtFiles(lngIndex2).dmtLastAccess
                    lngIndex2 = lngIndex2 - 1
                Loop
```

```
            Else
              Do While mudtFiles(lngIndex1).dmtLastAccess > vntTemp
                  lngIndex1 = lngIndex1 + 1
              Loop
              Do While vntTemp > mudtFiles(lngIndex2).dmtLastAccess
                  lngIndex2 = lngIndex2 - 1
              Loop
          End If
        Case Sort by Last Modyfy
          If enmSortOrder = Sort Order Ascending Then
              Do While mudtFiles(lngIndex1).dmtLastModify < vntTemp
                  lngIndex1 = lngIndex1 + 1
              Loop
              Do While vntTemp < mudtFiles(lngIndex2).dmtLastModify
                  lngIndex2 = lngIndex2 - 1
              Loop
            Else
              Do While mudtFiles(lngIndex1).dmtLastModify > vntTemp
                  lngIndex1 = lngIndex1 + 1
              Loop
              Do While vntTemp > mudtFiles(lngIndex2).dmtLastModify
                  lngIndex2 = lngIndex2 - 1
              Loop
          End If
        Case Sort_by_Date_Create
          If enmSortOrder = Sort Order Ascending Then
              Do While mudtFiles(lngIndex1).dmtDateCreate < vntTemp
                  lngIndex1 = lngIndex1 + 1
              Loop
              Do While vntTemp < mudtFiles(lngIndex2).dmtDateCreate
                  lngIndex2 = lngIndex2 - 1
              Loop
            Else
              Do While mudtFiles(lngIndex1).dmtDateCreate > vntTemp
                  lngIndex1 = lngIndex1 + 1
              Loop
              Do While vntTemp > mudtFiles(lngIndex2).dmtDateCreate
                  lngIndex2 = lngIndex2 - 1
              Loop
          End If
       End Select
       If lngIndex1 <= lngIndex2 Then
           udtBuffer = mudtFiles(lngIndex1)
           mudtFiles(lngIndex1) = mudtFiles(lngIndex2)
           mudtFiles(lngIndex2) = udtBuffer
           lngIndex1 = lngIndex1 + 1
           lngIndex2 = lngIndex2 - 1
       End If
    Loop Until lngIndex1 > lngIndex2
    If lngLBorder < lngIndex2 Then Call prcSort(lngLBorder, lngIndex2, enmSortBy, enmSortOrder)
    If lngIndex1 < lngUBorder Then Call prcSort(lngIndex1, lngUBorder, enmSortBy, enmSortOrder)
End Sub
```

CODE 2

Hi,

Ich hab mal wieder nen kleinen Workshop anzubieten:
Folgendes Mako startet mit einem Pfadauswahldialog, durchsucht dann alle unterordner nach einer angegebenen Datei (z.B. "*.xlsx" oder "*.xls*" oder auch "test.xlsx"). Aus jeder gefundenen Datei wird eine bestimmte Zelle ausgelesen (welche Zelle das ist, ist hardCoded).

Anschließend werden die Werte dieser Zelle zu einer "Empfängerliste" zusammengebaut und anschließend eine E-Mail mit den eingetragenen Empfängern erzeugt.

Was lässt sich da verbessern?

[*] Einige Stellen im Code sind "quick & dirty" programmiert.
[*] Es müsste noch eine Art Validation eingebaut werden, sodass nur solche Werte zur Email-Liste hinzugefügt werden, die auch wirklich dem Format einer korrekten Mail-Adresse entsprechen.
[*] Es fehlt sowas, wie ein Exception-Handling =)

Das Ganze ist also nicht perfekt, aber vielleicht nützt es irgendwem. Würd mich über ne positive Bewertung freuen.

Hier ist der Code:

Code:

```
' Etwas dirty: Größe der folgenden Arrays ist fixed.
' Besser wäre: sollte sich dynamisch an die Anzahl dergefundenen Dateien anpassen.
Private FILES(300) As String
Private MAILS(300) As String
Private size As Integer


' Main-Methode: Durchsucht einen Ordner und alle seine Unterordner und generiert eine E-Mail
Public Sub generateMail()
    Call findMails
    Dim OutApp As Object, Mail As Object
    Dim Nachricht
    Set OutApp = CreateObject("Outlook.Application")
    Set Nachricht = OutApp.CreateItem(0)
    With Nachricht
            .To = generateSendToString 'Adresse
            .Subject = InputBox("Wie soll die Betreffzeile der Mail lauten?") 'Betreffzeile
            '.Body = "Dies ist eine TestMail, die dazu dient, mein Makro zu testen" 'Sendetext
            '.Send
            .Display
            'SendKeys "%s", True
    End With
End Sub

Private Function readFile(ByVal file As String) As String
    Application.Workbooks.Open (file)
    Sheets("Tabelle1").Activate 'Hier kann man das auszuwählende Sheet ändern
    readFile = Cells(1, 1) 'Hier kann man die auszulesende Celle ändern
    ActiveWorkbook.Close
End Function
```

```
Private Function generateSendToString() As String
    Dim sendString
    Dim AnzahlMails As Integer
    AnzahlMails = 0
    sendString = ""
    For i = 0 To UBound(MAILS)
        If MAILS(i) <> "" Then
        AnzahlMails = AnzahlMails + 1
            If i = 0 Then
                sendString = MAILS(i)
            Else
                sendString = sendString & "; " & MAILS(i)
            End If
        End If
    Next i
    MsgBox "es wurden " & AnzahlMails & " Mail-Adressen gefunden."
    generateSendToString = sendString
    'Abfrage, ob der SendString gespeichert werden soll
    If sendString <> "" Then
        Dim t As Integer
        t = MsgBox("Möchten sie die Liste der E-Mail-Adressen speichern?", vbOKCancel)
        If t = 1 Then
            Sheets("Mails").Activate
            Cells(1, 1) = sendString
        End If
    End If
End Function


Private Function findMails()
    Dim filename
    filename = InputBox("Welche Dateien sollen gesucht werden?")
    If filename <> "" Then
        Filesearch filename
        Call readfiles
    Else
        MsgBox ("Fehler! Es wurde kein Dateiname angegeben")
    End If
End Function

'Alle gefundenen Dateien auslesen
Private Function readfiles()
    Sheets("Mails").Activate
    Dim Mail As String
    Mail = ""

    For i = 0 To UBound(FILES)
        If FILES(i) <> "" Then
            MAILS(i) = readFile(FILES(i))
        End If
    Next i
End Function

' Einen Ordner auswählen
Private Function getOrdner() As String
    Dim strOrdner As String
    strOrdner = ""
    With Application.FileDialog(msoFileDialogFolderPicker)
        .InitialFileName = "C:\"
```

```
        .Title = "Ordnerauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewList
        If .Show = -1 Then
            strOrdner = .SelectedItems(1)
            If Right(strOrdner, 1) <> "\" Then
                strOrdner = strOrdner & "\"
            End If
            getOrdner = strOrdner
        Else
            MsgBox "Es wurde kein Ordner ausgewaehlt!"
            getOrdner = ""
            Exit Function
        End If
    End With
End Function

'Vorbereiten der Dateisuche
Private Function Filesearch(ByVal name As String)
   Dim strDir As String
   Dim objFSO As Object
   Dim objDir As Object

   Set objFSO = CreateObject("scripting.filesystemobject")
   strDir = getOrdner()
   Set objDir = objFSO.GetFolder(strDir)
   size = -1
   getInfo objDir, name
   Set objDir = Nothing
   Set objFSO = Nothing
End Function

'Alle Dateien finden
Private Function getInfo(ByVal pCurrentDir As Object, ByVal strName As String)
   Dim aItem As Variant
   For Each aItem In pCurrentDir.FILES
      If aItem.name Like strName Then
         size = size + 1
         FILES(size) = aItem.Path
      End If
   Next
   For Each aItem In pCurrentDir.SubFolders
      getInfo aItem, strName
   Next
End Function
```

CODE 3

Hallo zusammen,

mal wieder steh ich auf dem Schlauch  und würde mich freuen, wenn mir jemand helfen könnte. 

Ich lese mit dem folgende Script ein Verzeichnis aus. jede gefundene Datei erzeugt eine Excelzeile.
Soweit Sogut
Frage : Wie bekomme ich es hin, dass mir auch die Unterverzeichnisse mit ausgelesen werden ??

Könnt ihr mir helfen ??

HTML-Code:

```
Sub Dateien_ermitteln()
   Dim i As Long, intPos As Integer
   Dim Dateidatum As Date
   Dim KW As Integer
'------------------------------
Dim objFile As Object
Dim objFSO As Object
Set objFSO = CreateObject("scripting.filesystemobject")
Dim DatNam
Dim BB
'------------------------------
'KW = Range("AA1")
   strsuchpfad = VerzeichnisErmitteln(s)
   If strsuchpfad = False Then Exit Sub
   If strsuchpfad = 0 Then Exit Sub
'-------------
DatNam = InputBox("Dateinamensteileingeben :" & Chr(13) & "z.B. Eingabe_*.xls")
Set objFSO = CreateObject("scripting.filesystemobject")
For Each objFile In objFSO.getfolder(strsuchpfad).Files
If objFile.Name Like DatNam Then
```

```
    Cells(65000, 1).End(xlUp).Offset(1, 0).Select
    ActiveCell.Offset(0, 0) = strsuchpfad
    ActiveCell.Offset(0, 1) = objFile.Name
    ActiveCell.Offset(0, 2) = objFile.path 'strsuchpfad & "\" & objFile.Name
    ActiveCell.Offset(0, 3) = objFile.datecreated
End If
Next
Set objFSO = Nothing
'-----------------------------------------------


End Sub
```

Hatte mal was ähnlichen mit FileSearch, ABER der geht unter 2007 nicht mehr

LÖSUNG 1

Hallo DeBabba,

mit dem FileSystemObject benötigt man dazu eine Prozedur, die sich für jedes Unterverzeichnis rekursiv selbst aufruft. Ungetestet:

Code:

```
Sub Dateien_ermitteln()

Dim objFSO As Object
Dim objFolder As Object
Dim strDatNam As String

Set objFSO = CreateObject("Scripting.FileSystemObject")

strsuchpfad = VerzeichnisErmitteln(s)
If strsuchpfad = False Then Exit Sub
```

```
If strsuchpfad = 0 Then Exit Sub

strDatNam = InputBox("Dateinamensteileingeben :" & Chr(13) & "z.B. Eingabe_*.xls")
Set objFolder = objFSO.GetFolder(strsuchpfad)

Rekursiv objFolder, strDatNam

Set rng = Nothing
Set objFile = Nothing
Set objFolder = Nothing
Set objSubFolder = Nothing
Set objFSO = Nothing

End Sub

Sub Rekursiv(objFolder As Object, strDatNam As String)

Dim objFile As Object
Dim objSubFolder As Object
Dim rng As Range

For Each objFile In objFolder.Files
  If objFile.Name Like strDatNam Then
    Set rng = Cells(Rows.Count, 1).End(xlUp).Offset(1, 0)
    rng = strsuchpfad
    rng.Offset(0, 1) = objFile.Name
    rng.Offset(0, 2) = objFile.Path 'strsuchpfad & "\" & objFile.Name
    rng.Offset(0, 3) = objFile.datecreated
  End If
Next 'objFile

For Each objSubFolder In objFolder.SubFolders
  Set objFolder = objSubFolder
  Rekursiv objFolder, strDatNam
Next 'objSubFolder

End Sub
```

Lösung 2

Hallo DeBabba,

probiers mal so.

```
' ********************************************************************
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

Option Explicit

Sub DateienErmitteln()
  Dim objFiles() As Object, lngRet As Long, lngIndex As Long, lngRow As Long
  Dim strPath As String, strFile As String

  strPath = fncBrowseForFolder

  If strPath <> "" Then
    strFile = InputBox("Dateinamensteileingeben :" & Chr(13) & "z.B. Eingabe_*.xls")
    If strFile <> "" Then
      lngRet = FileSearchINFO(objFiles, strPath, strFile, True)
      If lngRet > 0 Then
        lngRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
        For lngIndex = 0 To lngRet - 1
          Cells(lngRow + lngIndex, 1) = strPath
          Cells(lngRow + lngIndex, 2) = objFiles(lngIndex).Name
          Cells(lngRow + lngIndex, 3) = objFiles(lngIndex).ParentFolder.Path
          Cells(lngRow + lngIndex, 4) = objFiles(lngIndex).DateCreated
        Next
      End If
    End If
  End If
End Sub

Private Function FileSearchINFO(ByRef Files() As Object, ByVal InitialPath As String, Optional ByVal FileName As String = "*", _
```

```
  Optional ByVal SubFolders As Boolean = False) As Long

'# PARAMETERINFO:
'# Files: Datenfeld zur Ausgabe der Suchergebnisse
'# InitialPath: String der das zu durchsuchende Verzeichnis angibt
'# FileName: String der den gesuchten Dateityp oder Dateinamen enthält (Optional, Standard="*.*" findet alle Dateien)
'# Beispiele: "*.txt" - Findet alle Textdateien
'# "*name*" - Findet alle Dateien mit "name" im Dateinamen
'# "*.avi;*.mpg" - Findet .avi und .mpg Dateien (Dateitypen mit ; trennen)
'# SubFolders: Boolean gibt an, ob Unterordner durchsucht werden sollen (Optional, Standard=False)


Dim fobjFSO As Object, ffsoFolder As Object, ffsoSubFolder As Object, ffsoFile As Object
Dim intC As Integer, varFiles As Variant

Set fobjFSO = CreateObject("Scripting.FileSystemObject")

Set ffsoFolder = fobjFSO.GetFolder(InitialPath)

On Error GoTo ErrExit

If InStr(1, FileName, ";") > 0 Then
  varFiles = Split(FileName, ";")
Else
  Redim varFiles(0)
  varFiles(0) = FileName
End If
For Each ffsoFile In ffsoFolder.Files
  If Not ffsoFile Is Nothing Then
    For intC = 0 To UBound(varFiles)
      If LCase(fobjFSO.GetFileName(ffsoFile)) Like LCase(varFiles(intC)) Then
        If IsArray(Files) Then
          Redim Preserve Files(UBound(Files) + 1)
        Else
          Redim Files(0)
```

```
          End If
          Set Files(UBound(Files)) = ffsoFile
          Exit For
        End If
      Next
    End If
  Next

  If SubFolders Then
    For Each ffsoSubFolder In ffsoFolder.SubFolders
      FileSearchINFO Files, ffsoSubFolder, FileName, SubFolders
    Next
  End If

  If IsArray(Files) Then FileSearchINFO = UBound(Files) + 1
  ErrExit:
  Set fobjFSO = Nothing
  Set ffsoFolder = Nothing
End Function

Private Function fncBrowseForFolder(Optional ByVal defaultPath = "") As String
  Dim objFlderItem As Object, objShell As Object, objFlder As Object

  Set objShell = CreateObject("Shell.Application")
  Set objFlder = objShell.BrowseForFolder(0&, "Ordner auswählen...", 0&, defaultPath)

  If objFlder Is Nothing Then GoTo ErrExit

  Set objFlderItem = objFlder.Self
  fncBrowseForFolder = objFlderItem.Path

  ErrExit:

  Set objShell = Nothing
  Set objFlder = Nothing
```

```
  Set objFlderItem = Nothing
End Function
```

CODE 4

HAllo

Da Excel12 Application.FileSearch nicht mehr kennt, hier ein kleiner Code um sich Ordner und Unterordner in einem definierten Startverzeichniss listen zu lassen.

Der Code arbeitet rekursiv und hat als Basis das FileSystemObject.

```
' **********************************************************************
' Modul: Tabelle1 Typ: Element der Mappe(Sheet, Workbook, ...)
' **********************************************************************

Option Explicit
Dim zaehler As Long

Public Sub Aufruf()
Dim objShell As Object
Dim objFolder As Object
Dim objItem As Object
Set objShell = CreateObject("Shell.Application")
With objShell
    Set objFolder = .BrowseForFolder(0&, "Was soll ich machen?", 0)
End With
If Not objFolder Is Nothing Then
    zaehler = 1
    Set objItem = objFolder.Self
    Cells(zaehler, 1) = objItem.Path
    Else: Exit Sub
End If
Schreiben objItem.Path, True 'True wenn die Unterordner auch wieder geschrieben werden sollen.
'Sonst False oder weglassen. Entspricht SearchSubfolders
End Sub


Public Sub Schreiben(Suchordner, Optional sbfolds As Boolean = False)
Dim fso As Object
Dim ordner
Dim Unterordner
Set fso = CreateObject("Scripting.FileSystemObject")
Set ordner = fso.getfolder(Suchordner)
On Error Resume Next
Select Case sbfolds
    Case True
        For Each Unterordner In ordner.subfolders
            zaehler = zaehler + 1
```

```
            Cells(zaehler, 1) = Unterordner.Path
            Schreiben Unterordner, True
        Next
    Case False
        For Each Unterordner In ordner.subfolders
            zaehler = zaehler + 1
            Cells(zaehler, 1) = Unterordner.Path
        Next
End Select
Set fso = Nothing
Set ordner = Nothing
End Sub
```

Dateien rekursiv suchen

Lösung 1

Hallo Funktionsfreunde,

auf Anregung von Nepumuk beginne ich mal ganz einfach.
Wer macht Erweiterungs/Verbesserungsvorschläge?

HG
Rolf

```
'Verzeichnisse + Unterverzeichnisse ermitteln
Function allfolders(foldspec)
    Static x()
    Static i As Long
    Dim fs, f, f1, fc
    Dim n As Integer
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFolder(foldspec)
    Set fc = f.SubFolders
    n = fc.Count
    For Each f1 In fc
        If n > 0 Then i = i + 1
        ReDim Preserve x(i)
        x(i) = foldspec & f1.Name
```

```
        allfolders (x(i) & "\") 'rekursiver Funktionsaufruf!!!
    Next
    allfolders = x
End Function
```

Lösung 2

Auch Hallo,

ein paar Alternativen.

a) via Shell oder besser einem 'ShellAndWait' geht ganz einfach ein:

dir /b /s /A:D dirspec

Ich ganz persönlich mag' sowas schon, warum immer alles kodieren, wenn's sowas eh' schon gibt :-)

Bei Bedarf dieses in eine Datei umleiten oder via gclip.exe - von den UNXUTILS, etwas das auf jedem Rechner installiert sein müßte, es sei denn man hat CYGWIN - in die Zwischenablage kopieren. Z.B. so:

```
Option Explicit

Sub ShellIt()
   Shell ("cmd.exe /c dir /b /s /A:D d:\daten\excel | c:\uti\unxutil\gclip")
End Sub
```

b) mit DIR(), auch sowas geht...

```
Option Explicit

Dim zeile As Integer

Sub test()
   Worksheets(1).Activate
   Cells.Clear
   zeile = 1
   Tree "D:\daten\excel", "*.xls", False
End Sub

Sub Tree(actdir As String, filename As String, showfiles As Boolean)
   Dim fname
   Dim i As Integer, j As Integer
   Dim subdirs() As String

   Call ShowDir(actdir, filename, showfiles)
   i = 0
```

```
   fname = Dir(actdir & "\*.*", vbDirectory)
   While fname <> ""
      If fname <> "." And fname <> ".." And (GetAttr(actdir & "\" & fname) And vbDirectory) = vbDirectory Then
         i = i + 1
         ReDim Preserve subdirs(i)
         subdirs(i) = actdir & "\" & fname
      End If
      fname = Dir
   Wend
   For j = 1 To i
      Call Tree(subdirs(j), filename, showfiles)
   Next
   ReDim subdirs(0)
End Sub

Private Sub ShowDir(actdir As String, filename As String, showfiles As Boolean)
   Dim fname

   If showfiles Then
      fname = Dir(actdir & "\" & filename)
      While fname <> ""
         Cells(zeile, 1).Value = actdir & "\" & fname
         zeile = zeile + 1
         fname = Dir
      Wend
   Else
      Cells(zeile, 1).Value = actdir
      zeile = zeile + 1
   End If
End Sub
```

c) mit API, das lasse ich Nepumuk übrig ;-), der kann das besser.

cu, Bernd

Lösung 3

Nochmals Hallo,

dann halt auch noch das Teil mit API.

Prinzipiell aus dem API-Guide von http://www.mentalis.org/index2.shtml kopiert und lediglich etwas angepaßt.

cu, Bernd

--

```
Option Explicit

Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" ( _
   ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
```

```
Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" ( _
   ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Declare Function FindClose Lib "kernel32" ( _
   ByVal hFindFile As Long) As Long

Private Declare Function GetFileAttributes Lib "kernel32" Alias "GetFileAttributesA" ( _
   ByVal lpFileName As String) As Long

Const MAX_PATH = 260
Const MAXDWORD = &HFFFF
Const INVALID_HANDLE_VALUE = -1
Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_DIRECTORY = &H10
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H80
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4
Const FILE_ATTRIBUTE_TEMPORARY = &H100

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type

Dim zeile As Long

Sub TestIt()
   zeile = 1
   Cells.ClearContents
   Call Tree("d:\daten\excel")
End Sub

Sub Tree(StartDir As String)
   Dim DirName As String      ' SubDirectory Name
   Dim nDir As Integer        ' Number of directories in this path
   Dim hSearch As Long        ' Search Handle
   Dim WFD As WIN32_FIND_DATA ' WIN32_FIND_DATA
   Dim Cont As Integer        ' Result from FindNextFile
```

```
   If Right(StartDir, 1) <> "\" Then StartDir = StartDir & "\"

   ' Search for subdirectories.
   nDir = 0
   Cont = True
   hSearch = FindFirstFile(StartDir & "*.*", WFD)
   If hSearch <> INVALID_HANDLE_VALUE Then
      Do While Cont
         DirName = StripNulls(WFD.cFileName)
         ' Ignore . and ..
         If (DirName <> ".") And (DirName <> "..") Then
            DirName = StartDir & DirName
            ' Check for directory with bitwise comparison.
            If GetFileAttributes(DirName) And FILE_ATTRIBUTE_DIRECTORY Then
               Cells(zeile, 1).Value = DirName
               zeile = zeile + 1
               Call Tree(DirName)
            End If
        End If
        Cont = FindNextFile(hSearch, WFD) 'Get next subdirectory.
      Loop
      Cont = FindClose(hSearch)
   End If
End Sub

Function StripNulls(OriginalStr As String) As String
   If (InStr(OriginalStr, Chr(0)) > 0) Then
      OriginalStr = Left(OriginalStr, InStr(OriginalStr, Chr(0)) - 1)
   End If
   StripNulls = OriginalStr
End Function
```

Lösung 4

Hallo Bernd,

falls Nepumuk beabsichtigt, die kürzeste Lösung
zu prämieren, ist deine DOS-Anzeige wohl kaum zu schlagen -
wie bringst du die in ein Blatt?
lG Rolf

Morgen Rolf,

Um sowas ins Excel zu bekommen gibt's mehrere Möglichkeiten, u.a.:

- über die Zwischenablage
- über eine Textdatei
- in einer Batchdatei über eine Environmentvariable und dem Start einer neuen Excel-Instanz

Ich benutze sowas hin und wieder wirklich, meistens aber nur in 2 Situationen:

1. Rein interaktiv über die Zwischenablage, wenn ich zu faul bin das in VBA auszukodieren oder es mir ganz einfach nicht notwendig erscheint. Wie oben, allerdings mit cygwin's putclip, welches wie das gclip von UNXUTIL die Standardeingabe in die Windows Zwischenablage kopiert, also sowas:

irgendeinkommandozeilenkommando | putclip

Nett ist z.B. dieses, welches sämtliche URL's der IE Favoriten findet (ich habe meinen Favoriten-Ordner umgebogen):

d:\daten\favoriten>grep -r -m1 "^URL=" * | putclip

Oder dieses - NUR unter 4NT - welches alle heute oder gestern geänderten XLS-Dateien findet:

d:\daten\excel>dir /sb /[d-1,1] *.xls | putclip

2. Über eine Textdatei, die ich dann zumeist auch gleich in ein CSV-Format bringe.
Also prinzipiell sowas (google mal einfach nach shellandwait):

ShellAndWait "irgendwas > c:\temp\ausgabe.txt"
Workbooks.OpenText "c:\temp\ausgabe.txt", ...

Übrigens, just for fun, mit 4NT geht natürlich auch 'Nepumuks Hausaufgabe' in einer einzigen Kommandozeile ;-)
Hier nur für A-E und NICHT rekursiv, ich will dieses Jahr noch fertig werden...

d:> for %i in (A B C D E) do if %@READY[%i:] eq 1 (cdd %i:\ && dir /f *.txt) | putclip

cu, Bernd

Lösung 5

Hallo Nepumuk,

Bernds Ideen sind schon ganz pfiffig -
nur in diesem Fall ein wenig am Thema vorbei.

Ich habe zunächst die realisierungtechnisch m.E. einfachste Variante gewählt:
mit Filesearch, da braucht's nix rekursives. Ist natürlich nicht sehr schnell.

LG
Rolf

```
Option Explicit

Sub start_dateisuche()
    Dim fs As Object, dc As Object, d As Object
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set dc = fs.Drives
    For Each d In dc
        Debug.Print dateisuche((d), "user32.dll")
    Next
End Sub

Function dateisuche(drv$, fil$) As String
    Dim fso As Object, fs As Object
    Dim i As Integer

    Set fs = CreateObject("Scripting.FileSystemObject")
    Set fso = Application.FileSearch

    With fso
        .NewSearch
        .LookIn = drv$
        .filename = fil$
        .SearchSubFolders = True
        If .Execute() > 0 Then
            For i = 1 To .FoundFiles.Count
                If UCase(fil$) = UCase(fs.GetFileName(.FoundFiles(i))) Then
                    dateisuche = .FoundFiles(i)
                    Exit For
                End If
            Next
        Else
            dateisuche = "Laufwerk " & drv$ & " nix"
        End If
    End With

End Function
```

Lösung 6 (wichtig: optimiert durch Lösung 7 und 9)

Hallo Rolf,

wenn du mit der Variante ein Laufwerk mit 160GB durchsuchst, welches auch noch ziemlich voll ist, dann startest du die Suche am besten kurz bevor du ins Bett gehst. Es könnte dann am nächsten Morgen fast fertig sein. Du musst mit Suchzeiten von 5-6 Stunden rechnen. Das Filesystemobject ist zwar nicht die schnellste Methode, aber damit wird es nur 3 bis 4 Minuten dauern. Mal ein Beispiel:

```
Option Explicit

Public Sub test()
    MsgBox fncSearch("*.xlb")
End Sub

Private Function fncSearch(strSearch As String) As String
    Dim objFSO As Object, objDrive As Object
    Dim strReturn As String
    On Error GoTo Err_Exit
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    For Each objDrive In objFSO.Drives
        If objDrive.IsReady Then Call prcSearchFolders(objFSO, _
            objDrive.Path & "\", strSearch, strReturn)
        If strReturn <> "" Then Exit For
    Next
    If strReturn <> "" Then
        fncSearch = strReturn
    Else
        fncSearch = "Nix gefunden!"
    End If
Err_Exit:
End Function

Private Sub prcSearchFolders(ByRef objFSO As Object, ByVal strPath As String, _
    ByVal strSearch As String, ByRef strReturn As String)
    Dim objFolder As Object
    On Error GoTo Err_Exit
    Call prcSearchFiles(objFSO, strPath, strSearch, strReturn)
    For Each objFolder In objFSO.GetFolder(strPath).SubFolders
        If strReturn <> "" Then Exit For
        Call prcSearchFiles(objFSO, objFolder.Path & "\", strSearch, strReturn)
        If strReturn <> "" Then Exit For
        Call prcSearchFolders(objFSO, objFolder.Path & "\", strSearch, strReturn)
```

```
    Next
Err_Exit:
End Sub

Private Sub prcSearchFiles(ByRef objFSO As Object, ByVal strPath As String, _
    ByVal strSearch As String, ByRef strReturn As String)
    Dim objFile As Object
    On Error GoTo Err_Exit
    For Each objFile In objFSO.GetFolder(strPath).Files
        If objFile.Name Like strSearch Then
            strReturn = objFile.Path
            Exit For
        End If
    Next
Err_Exit:
End Sub
```

Gruß
Nepumuk

Lösung 7

Hi Rolf,

1. Da ist ein Fehler im Makro "prcSearchFolders". Der Code durchsucht alle Dateien zweimal. So ist er richtig:

```
Private Sub prcSearchFolders(ByRef objFSO As Object, ByVal strPath As String, _
    ByVal strSearch As String, ByRef strReturn As String)
    Dim objFolder As Object
    On Error GoTo Err_Exit
    Call prcSearchFiles(objFSO, strPath, strSearch, strReturn)
    For Each objFolder In objFSO.GetFolder(strPath).SubFolders
        If strReturn <> "" Then Exit For
        Call prcSearchFolders(objFSO, objFolder.Path & "\", strSearch, strReturn)
    Next
Err_Exit:
End Sub
```

2. Du benutzt Windows98 oder so was ähnliches. Ab Windows2000 / NT / XP steht dir NTFS (**N**ew **T**echnology **F**ile **S**ystem) zur Verfügung. Damit geht es wesentlich schneller. Der folgende Code durchsucht bei mir 17 Laufwerke mit 48,3 GB = 6058 Ordner mit 105.670 Dateien in 60,22 Sekunden. Dabei findet er die Datei "USER32.DLL" 5 mal. Du musst einen Verweis auf Microsoft Scripting Runtime (scrrun.dll) setzten.

```
Option Explicit

Private Type FileSet
    datDateCreated As Date 'dd.mm.yyyy HH:MM:SS
    datDateLastAccessed As Date
    datDateLastModified As Date
    sngSize As Single 'Einheit Byte
    strName As String
    strPath As String
End Type

Private objFileSystemObject As FileSystemObject, objFile As File
Private objFolder As Folder, objDrive As Drive
Private typFiles() As FileSet
Private bolCaseSensitiv As Boolean, bolSearchSubFolders As Boolean, bolSortOrder As Boolean
Private strfilename As String, strTempName2 As String, strTempExtension2 As String
Private bytSortKey As Byte
Private sngFileCounter As Single, sngFolderCounter As Single
Private lngFileCount As Long

Public Sub prcFileSearch()
    Dim dblTimer As Double
    Dim strPath As String
    Dim lngIndex As Long

    dblTimer = Timer 'Bei Bedarf

    Rem SortKey: 1=DateCreated / 2=DateLastAccessed / 3=DateLastModified / 4=Size / 5=Name / 6=Path
    bytSortKey = 4
    Rem SortOrder: True=Aufsteigend / False=Absteigend
    bolSortOrder = False

    Rem CaseSensitiv: True / False
    bolCaseSensitiv = False
    Rem SearchSubFolders: True / False
    bolSearchSubFolders = True
```

```
    Rem Abschließender Schrägstrich erforderlich !!! Keine Angabe = alle Laufwerke
    strPath = "" '"C:\" '
    strfilename = "user32.dll" 'Platzhalterzeichen --> Like - Operator

    lngFileCount = 0
    sngFileCounter = 0 'Bei Bedarf
    sngFolderCounter = 0 'Bei Bedarf
    strTempName2 = ""
    strTempExtension2 = ""
    Erase typFiles

    Set objFileSystemObject = New Scripting.FileSystemObject

    If bolCaseSensitiv Then
        If InStr(1, strfilename, ".") <> 0 Then
            strTempName2 = Left$(strfilename, Len(strfilename) - _
                InStr(1, StrReverse(strfilename), "."))
            strTempExtension2 = LCase$(Right$(strfilename, _
                InStr(1, StrReverse(strfilename), ".") - 1))
        Else
            strTempName2 = strfilename
        End If
    Else
        strTempName2 = LCase$(strfilename)
    End If

    If strPath <> "" Then
        Call prcFindFiles(strPath)
    Else
        For Each objDrive In objFileSystemObject.Drives
            If objDrive.IsReady Then prcFindFiles (objDrive.Path & "\")
        Next
    End If

    MsgBox "Durchsucht" & vbLf & "Odner:      " & CStr(sngFolderCounter) & vbLf & _
        "Dateien:    " & CStr(sngFileCounter) _
        & vbLf & vbLf & "Gefunden" & vbLf & "Dateien:    " & CStr(lngFileCount) _
        & vbLf & "Sekunden: " & Format(CStr(Timer - dblTimer), "0.00") & _
        String(30, " "), 64, "Information"

'    If lngFileCount > 0 Then Call prcSort(1, lngFileCount)'bei Bedarf
```

```
    For lngIndex = 1 To lngFileCount
        Debug.Print typFiles(lngIndex).strPath
    Next

    Set objFileSystemObject = Nothing

End Sub

Private Sub prcFindFiles(ByVal strPath As String)
    Call prcGetFilesInFolder(strPath)
    sngFolderCounter = sngFolderCounter + 1 'Bei Bedarf
    If bolSearchSubFolders Then
        On Error GoTo Errorexit
        For Each objFolder In objFileSystemObject.GetFolder(strPath).SubFolders
            Call prcFindFiles(objFolder.Path & "\")
        Next
    End If
Errorexit:
End Sub

Private Sub prcGetFilesInFolder(ByVal strPath As String)
    Dim strTempName1 As String, strTempExtension1 As String
    On Error Resume Next
    For Each objFile In objFileSystemObject.GetFolder(strPath).Files
        If Err.Number = 0 Then
            On Error GoTo 0
            If bolCaseSensitiv Then
                If InStr(1, strfilename, ".") <> 0 Then
                    If InStr(1, objFile.Name, ".") <> 0 Then
                        strTempName1 = Left$(objFile.Name, Len(objFile.Name) - _
                            InStr(1, StrReverse(objFile.Name), "."))
                        strTempExtension1 = LCase$(Right$(objFile.Name, _
                            InStr(1, StrReverse(objFile.Name), ".") - 1))
                    Else
                        strTempName1 = objFile.Name
                    End If
                Else
                    If InStr(1, objFile.Name, ".") <> 0 Then
                        strTempName1 = Left$(objFile.Name, Len(objFile.Name) - _
                            InStr(1, StrReverse(objFile.Name), "."))
                    Else
```

```
                        strTempName1 = objFile.Name
                    End If
                End If
            Else
                strTempName1 = LCase$(objFile.Name)
            End If
            If strTempName1 Like strTempName2 And strTempExtension1 Like strTempExtension2 Then
                lngFileCount = lngFileCount + 1
                Redim Preserve typFiles(1 To lngFileCount)
                With typFiles(lngFileCount)
                    .datDateCreated = objFile.DateCreated
                    .datDateLastAccessed = objFile.DateLastAccessed
                    .datDateLastModified = objFile.DateLastModified
                    .sngSize = objFile.Size
                    .strName = objFile.Name
                    .strPath = objFile.Path
                End With
            End If
            sngFileCounter = sngFileCounter + 1 'Bei Bedarf
            On Error Resume Next
        Else
            Err.Clear
        End If
    Next
End Sub

Private Sub prcSort(ByVal lngLBorder As Long, ByVal lngUBorder As Long)
    Dim lngIndex1 As Long, lngIndex2 As Long
    Dim datTemp As Date, datTempBuffer As Date
    Dim sngTemp As Single, sngTempBuffer As Single
    Dim strTemp As String, strTempBuffer As String
    lngIndex1 = lngLBorder
    lngIndex2 = lngUBorder
    Select Case bytSortKey
        Case 1: datTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).datDateCreated
        Case 2: datTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).datDateLastAccessed
        Case 3: datTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).datDateLastModified
        Case 4: sngTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).sngSize
        Case 5: strTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).strName
        Case 6: strTempBuffer = typFiles(Fix((lngLBorder + lngUBorder) / 2)).strPath
        Case Else: Exit Sub
    End Select
```

```
Do
    Select Case bytSortKey
        Case 1
            If bolSortOrder Then
                Do While typFiles(lngIndex1).datDateCreated < datTempBuffer
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While datTempBuffer < typFiles(lngIndex2).datDateCreated
                    lngIndex2 = lngIndex2 - 1
                Loop
            Else
                Do While typFiles(lngIndex1).datDateCreated > datTempBuffer
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While datTempBuffer > typFiles(lngIndex2).datDateCreated
                    lngIndex2 = lngIndex2 - 1
                Loop
            End If
        Case 2
            If bolSortOrder Then
                Do While typFiles(lngIndex1).datDateLastAccessed < datTempBuffer
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While datTempBuffer < typFiles(lngIndex2).datDateLastAccessed
                    lngIndex2 = lngIndex2 - 1
                Loop
            Else
                Do While typFiles(lngIndex1).datDateLastAccessed > datTempBuffer
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While datTempBuffer > typFiles(lngIndex2).datDateLastAccessed
                    lngIndex2 = lngIndex2 - 1
                Loop
            End If
        Case 3
            If bolSortOrder Then
                Do While typFiles(lngIndex1).datDateLastModified < datTempBuffer
                    lngIndex1 = lngIndex1 + 1
                Loop
                Do While datTempBuffer < typFiles(lngIndex2).datDateLastModified
                    lngIndex2 = lngIndex2 - 1
                Loop
```

```
    Else
        Do While typFiles(lngIndex1).datDateLastModified > datTempBuffer
            lngIndex1 = lngIndex1 + 1
        Loop
        Do While datTempBuffer > typFiles(lngIndex2).datDateLastModified
            lngIndex2 = lngIndex2 - 1
        Loop
    End If
Case 4
    If bolSortOrder Then
        Do While typFiles(lngIndex1).sngSize < sngTempBuffer
            lngIndex1 = lngIndex1 + 1
        Loop
        Do While sngTempBuffer < typFiles(lngIndex2).sngSize
            lngIndex2 = lngIndex2 - 1
        Loop
    Else
        Do While typFiles(lngIndex1).sngSize > sngTempBuffer
            lngIndex1 = lngIndex1 + 1
        Loop
        Do While sngTempBuffer > typFiles(lngIndex2).sngSize
            lngIndex2 = lngIndex2 - 1
        Loop
    End If
Case 5
    If bolSortOrder Then
        Do While typFiles(lngIndex1).strName < strTempBuffer
            lngIndex1 = lngIndex1 + 1
        Loop
        Do While strTempBuffer < typFiles(lngIndex2).strName
            lngIndex2 = lngIndex2 - 1
        Loop
    Else
        Do While typFiles(lngIndex1).strName > strTempBuffer
            lngIndex1 = lngIndex1 + 1
        Loop
        Do While strTempBuffer > typFiles(lngIndex2).strName
            lngIndex2 = lngIndex2 - 1
        Loop
    End If
Case 6
    If bolSortOrder Then
```

```
            Do While typFiles(lngIndex1).strPath < strTempBuffer
                lngIndex1 = lngIndex1 + 1
            Loop
            Do While strTempBuffer < typFiles(lngIndex2).strPath
                lngIndex2 = lngIndex2 - 1
            Loop
        Else
            Do While typFiles(lngIndex1).strPath > strTempBuffer
                lngIndex1 = lngIndex1 + 1
            Loop
            Do While strTempBuffer > typFiles(lngIndex2).strPath
                lngIndex2 = lngIndex2 - 1
            Loop
        End If
End Select
If lngIndex1 <= lngIndex2 Then

    datTemp = typFiles(lngIndex1).datDateCreated
    typFiles(lngIndex1).datDateCreated = typFiles(lngIndex2).datDateCreated
    typFiles(lngIndex2).datDateCreated = datTemp

    datTemp = typFiles(lngIndex1).datDateLastAccessed
    typFiles(lngIndex1).datDateLastAccessed = typFiles(lngIndex2).datDateLastAccessed
    typFiles(lngIndex2).datDateLastAccessed = datTemp

    datTemp = typFiles(lngIndex1).datDateLastModified
    typFiles(lngIndex1).datDateLastModified = typFiles(lngIndex2).datDateLastModified
    typFiles(lngIndex2).datDateLastModified = datTemp

    sngTemp = typFiles(lngIndex1).sngSize
    typFiles(lngIndex1).sngSize = typFiles(lngIndex2).sngSize
    typFiles(lngIndex2).sngSize = sngTemp

    strTemp = typFiles(lngIndex1).strName
    typFiles(lngIndex1).strName = typFiles(lngIndex2).strName
    typFiles(lngIndex2).strName = strTemp

    strTemp = typFiles(lngIndex1).strPath
    typFiles(lngIndex1).strPath = typFiles(lngIndex2).strPath
    typFiles(lngIndex2).strPath = strTemp

    lngIndex1 = lngIndex1 + 1
```

```
            lngIndex2 = lngIndex2 - 1
        End If
    Loop Until lngIndex1 > lngIndex2
    If lngLBorder < lngIndex2 Then Call prcSort(lngLBorder, lngIndex2)
    If lngIndex1 < lngUBorder Then Call prcSort(lngIndex1, lngUBorder)
End Sub
```

Gruß
Nepumuk
Lösung 8

(Bei den Testläufen der Lösungen mit FilesystemObjekt wurden manchmal Dateien auf dem Laufwerk C nicht gefunden - auf D schon, daher schreibt Nepomuk hier :) - (Anmerkung: er fand auch die Lösung für das Problem des FilysystemObjekts - siehe Lösung 9)

Mein Fazit: Das Filesystemobjekt ist zum suchen von Dateien ungeeignet. Ob ein User mit Filesearch oder API arbeitet, bleibt seinem persönlichen Geschmack überlassen. Werden mehrere Suchen nacheinander gestartet, ziehe ich aber API vor.

Ach so, der API - Code:

```
Option Explicit

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" ( _
    ByVal lpFileName As String, _
    lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" ( _
    ByVal hFindFile As Long, _
    lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function FindClose Lib "kernel32" ( _
    ByVal hFindFile As Long) As Long

Private Enum FILE_ATTRIBUTE
    FILE_ATTRIBUTE_READONLY = &H1
    FILE_ATTRIBUTE_HIDDEN = &H2
    FILE_ATTRIBUTE_SYSTEM = &H4
    FILE_ATTRIBUTE_DIRECTORY = &H10
    FILE_ATTRIBUTE_ARCHIVE = &H20
    FILE_ATTRIBUTE_NORMAL = &H80
```

```
    FILE_ATTRIBUTE_TEMPORARY = &H100
End Enum

Private Const MAX_PATH = 260
Private Const INVALID_HANDLE_VALUE = -1

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type

Private strFile As String
Private strFolder As String
Private bolfound As Boolean

Public Sub suchen()
    Dim myFileSystemObject As Object, myDrive As Object, t
    t = Timer
    Set myFileSystemObject = CreateObject("Scripting.FileSystemObject")
    For Each myDrive In myFileSystemObject.Drives
        If myDrive.IsReady Then
            FindFiles myDrive.DriveLetter & ":\", "user32.dll"
            If bolfound Then
                Debug.Print strFolder & strFile
                bolfound = False
            Else
                Debug.Print myDrive.DriveLetter & " Nix gefunden!"
            End If
        End If
```

```
    Next
    Set myFileSystemObject = Nothing
    Debug.Print Timer - t
End Sub

Private Sub FindFiles(ByVal strFolderPath As String, ByVal strSearch As String)
    Dim WFD As WIN32_FIND_DATA
    Dim lngSearch As Long
    Dim strDirName As String
    lngSearch = FindFirstFile(strFolderPath & "*.*", WFD)
    If lngSearch <> INVALID_HANDLE_VALUE Then
        GetFilesInFolder strFolderPath, strSearch
        Do
            If bolfound Then Exit Do
            If (WFD.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY) Then
                strDirName = Left$(WFD.cFileName, InStr(WFD.cFileName, Chr(0)) - 1)
                If (strDirName <> ".") And (strDirName <> "..") Then FindFiles strFolderPath & strDirName & "\",
strSearch
            End If
        Loop While FindNextFile(lngSearch, WFD)
        FindClose lngSearch
    End If
End Sub

Private Sub GetFilesInFolder(ByVal strFolderPath As String, ByVal strSearch As String)
    Dim WFD As WIN32_FIND_DATA
    Dim lngSearch As Long
    Dim strFileName As String
    lngSearch = FindFirstFile(strFolderPath & strSearch, WFD)
    If lngSearch <> INVALID_HANDLE_VALUE Then
        Do
            If (WFD.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY) <> FILE_ATTRIBUTE_DIRECTORY Then
                strFileName = Left$(WFD.cFileName, InStr(WFD.cFileName, Chr(0)) - 1)
                bolfound = True
                strFile = strFileName
                strFolder = strFolderPath
                Exit Do
            End If
        Loop While FindNextFile(lngSearch, WFD)
        FindClose lngSearch
    End If
End Sub
```

Gruß
Nepumuk

Guten Morgen Nepumuk,

da haben wir die Wissenschaft von der Dateisuche
doch glatt ein Stückchen nach vorn gebracht.

Dein Fazit teile ich (fast) uneingeschränkt, wobei
das "fast" sich lediglich auf unsere Verfahrenspräferenz bezieht.

Der FileSearch-Code ist für den "VBA-Normalsterblichen" sicher
übersichtlicher und leichter nachvollziehbar,
der API-Code dafür performanter.

Vielleicht magst du in deiner Zusammenfassung beide,
entsprechend kommentiert, anbieten.

LG
Rolf

Lösung 9 (Ergänzung)

Fehler gefunden
Von: Nepumuk Am: 17.07.2005 20:56:16

Hi Rolf,

das Makro steigt komplett aus der Schleife aus, wenn auf eine Datei nicht zugegriffen werden kann. So sollte die Datei gefunden werden:

```
Private Sub prcSearchFiles(ByRef objFSO As Object, ByVal strPath As String, _
    ByVal strSearch As String, ByRef strReturn As String)
    Dim objFile As Object
    On Error GoTo Err_Exit
    For Each objFile In objFSO.GetFolder(strPath).Files
        If objFile.Name Like strSearch Then
```

```
            strReturn = objFile.Path
            Exit For
        End If
Err_Exit:
    Next
End Sub
```

Dateien einer Ordnerliste in einer Exceltabelle in einer Tabelle zusammenstellen

Problem: Wie kann ich mir eine Liste aller Dateien erstellen lassen, deren Verzeichnisse in Spalte A einer Tabelle stehen?

```
Sub DateiListe()
   Dim wks As Worksheet
   Dim iRow As Integer, iCounter As Integer, iRowT As Integer
   Application.ScreenUpdating = False
   Set wks = ActiveSheet
   Workbooks.Add 1
   iRow = 1
   Do Until IsEmpty(wks.Cells(iRow, 1))
      Cells(1, iRow).Value = wks.Cells(iRow, 1).Value
      iRowT = 1
      With Application.FileSearch
         .NewSearch
         .LookIn = wks.Cells(iRow, 1).Value
         .Execute
         For iCounter = 1 To .FoundFiles.Count
            iRowT = iRowT + 1
            Cells(iRowT, iRow).Value = _
               .FoundFiles(iCounter)
         Next iCounter
      End With
      iRow = iRow + 1
   Loop
   Rows(1).Font.Bold = True
   Columns.AutoFit
   Application.ScreenUpdating = True
End Sub
```

Daten aus auswählbarer Excelarbeitsmappe importieren

```
' Version von FANGENE-Reportingtool

Sub IMPORTIERE()

' Das ist die Importtaste in der Tabelle IMP mit der die Quelldateien ausgelesen werden

   Dim IMP_DATEI As String ' Quelldateiname inkl Pfad
   Dim IMP_DATEIPFAD As String ' Quelldateipfad
   Dim IMP_DATEINAME As String ' Quelldateiname
   Dim I As Long
   Dim T As Integer
   Dim WERT
   Dim ZIELTABELLE As String ' Name der Zieltabelle
   Dim BEREICH As String ' der zu kopierende Zellbereich der Quelldatei
   Dim SPALTE As Integer
   Dim ZEILE As Integer
   Dim ERSTEZEILE As Integer
   Dim ZEILENDIFFERENZ As Integer ' wieviel höher die Zieltabellenzeilen sind als die Quelltabellenzeilen
   Dim AKTUELLEDATEI


    AKTUELLEDATEI = ActiveWorkbook.Name

' Auswahl der zu importierenden Datei

    With Application.FileDialog(msoFileDialogFilePicker)
       .InitialFileName = ThisWorkbook.Path & "\"
       .Title = "Dateiauswahl"
       .ButtonName = "Auswahl..."
       .InitialView = msoFileDialogViewDetails
       If .Show = -1 Then
          IMP_DATEI = .SelectedItems(1)
       Else
          MsgBox "Es wurde keine Datei ausgewaehlt!"
          Exit Sub
       End If
    End With
```

```
    IMP_DATEIPFAD = Left(IMP_DATEI, InStrRev(IMP_DATEI, "\"))
    ' MsgBox IMP_DATEIPFAD
    IMP_DATEINAME = Right(IMP_DATEI, Len(IMP_DATEI) - InStrRev(IMP_DATEI, "\"))
    ' MsgBox IMP_DATEINAME


' Leeren der Importtabelle

   Sheets("IMP").Range("A1:IV65000").ClearContents

'Öffnen und Kopieren der Daten aus der zu importierenden Datei

 Application.ScreenUpdating = False  ' Bild nicht aktualisieren

    Workbooks.Open IMP_DATEI
    Workbooks(AKTUELLEDATEI).Sheets("IMP").Range("E2:AZ2000").Value =
Workbooks(IMP_DATEINAME).Sheets("Tageserfassung").Range("E99:AZ2097").Value
    Workbooks(IMP_DATEINAME).Close

Application.ScreenUpdating = True

Exit Sub

DATEI_NICHT_GEFUNDEN:

MsgBox "Die Datei '" & IMP_DATEI & "' konnte nicht im Verzeichnis '" & IMP_DATEIPFAD & "' gefunden werden." & vbCrLf & vbCrLf & _
"Stellen Sie sicher, dass die Datei im angegebenen Verzeichnis existiert."
End Sub



---------------------------------------------------------------

VARIANTE 2

Sub IMPORTIERE()

' Das ist die Importtaste in der Tabelle IMPORTLISTE mit der alle Quelldateien ausgelesen und
' in die betreffenden Zieltabellen eingefügt werden

   Dim DATEI As String ' Quelldateiname
   Dim PFAD As String ' Quelldateipfad
   Dim I As Long
```

```
   Dim T As Integer
   Dim D As Integer
   Dim WERT
   Dim ZIELTABELLE As String ' Name der Zieltabelle
   Dim BEREICH As String ' der zu kopierende Zellbereich der Quelldatei
   Dim SPALTE As Integer
   Dim ZEILE As Integer
   Dim ERSTEZEILE As Integer
   Dim ZEILENDIFFERENZ As Integer ' wieviel höher die Zieltabellenzeilen sind als die Quelltabellenzeilen
   Dim AKTUELLEDATEI

   ' Quellpfad um \ erweitern bei Bedarf
   If Right(Sheets("Importliste").Range("F7"), 1) <> "\" Then Sheets("Importliste").Range("F7") = Sheets("Importliste").Range("F7") & "\"

'   ChDrive (Left(Sheets("Importliste").Range("F7"), 1))
'   ChDir (Sheets("Importliste").Range("F7"))

AKTUELLEDATEI = ActiveWorkbook.Name

On Error GoTo DATEI_NICHT_GEFUNDEN

' Schleife durch alle Dateien in der Tabelle IMPORTLISTE


For D = 10 To LETZTEZELLE(Worksheets("Importliste")).Row

   PFAD = Sheets("Importliste").Range("F7")
   DATEI = Sheets("Importliste").Cells(D, 5)
   ZIELTABELLE = Sheets("Importliste").Cells(D, 6)
   BEREICH = "A" & Sheets("Importliste").Cells(D, 6) & ":IU60000"
   ERSTEZEILE = Sheets("Importliste").Cells(D, 7)

   Sheets(ZIELTABELLE).Range("A1:IV65000").ClearContents


 Application.ScreenUpdating = False  ' Bild nicht aktualisieren

    Workbooks.Open PFAD & DATEI
    ZEILENDIFFERENZ = ERSTEZEILE - 1
    For ZEILE = ERSTEZEILE To LETZTEZELLE(Workbooks(DATEI).Worksheets(1)).Row
       For SPALTE = 1 To LETZTEZELLE(Workbooks(DATEI).Worksheets(1)).Column

          Workbooks(AKTUELLEDATEI).Sheets(ZIELTABELLE).Cells(ZEILE - ZEILENDIFFERENZ, SPALTE) = Workbooks(DATEI).Worksheets(1).Cells(ZEILE,
SPALTE)
```

```
        Next SPALTE
    Next ZEILE
    Workbooks(DATEI).Close

Application.ScreenUpdating = True

Next D

Exit Sub

DATEI_NICHT_GEFUNDEN:

MsgBox "Die Datei '" & DATEI & "' konnte nicht im Verzeichnis '" & PFAD & "' gefunden werden." & vbCrLf & vbCrLf & _
"Stellen Sie sicher, dass die Datei im angegebenen Verzeichnis existiert oder ändern Sie die Einstellungen hier in der Tabelle 'Importliste'."
End Sub
```

Daten aus anderer Arbeitsmappe abrufen

```
MsgBox Workbooks("Countrates_Overview.xls").Sheets("Sheet2").Range("A1")
```

Daten aus externen Mappen lesen ohne diese zu öffnen

Variante 1

```
Sub KONSOLIDIERUNG()
' Zum Auslesen und Konsolidieren von Daten der zwei gleich aufgebauten Tabellen UVA in zwei Dateien

Dim PFAD1 ' Pfad der ersten Datei - zB D:\
Dim PFAD2 ' Pfad der zweiten Datei
Dim DATEI1 As String ' Dateiname der ersten Datei: zB BH1.xls
Dim DATEI2 As String ' Dateiname 2
Dim ADDY1 As String ' Gesamter Name der ersten Datei
Dim ADDY2 As String ' Gesamter Name der weiten Datei

Dim TABELLENNAME ' Name der gleichnamigen Tabelle in beiden Dateien

Dim ZEILE As Integer ' Für schleife durch alle Zeilen
Dim SPALTE As Integer ' Für Schleife durch alle Spalten
```

```
Dim ZELLE ' Adresse der auszulesenden Zelle

PFAD1 = Range("C5")
PFAD2 = Range("I5")
DATEI1 = Range("C4")
DATEI2 = Range("I4")
TABELLENNAME = "UVA"

ADDY1 = "'" & PFAD1 & "[" & DATEI1 & "]" & TABELLENNAME & "'!"
ADDY2 = "'" & PFAD2 & "[" & DATEI2 & "]" & TABELLENNAME & "'!"

SPALTE = 9

For ZEILE = 87 To 127

   ZELLE = Cells(ZEILE, SPALTE).Address(ReferenceStyle:=R1C1)

   MsgBox ExecuteExcel4Macro(ADDY1 & ZELLE)


Next ZEILE
```

Variante 2

```
Public Function GetDataClosedWB(SourcePath As String, _
                                SourceFile As String, _
                                sourceSheet As String, _
                                SourceRange As String, _
                                TargetRange As Range) As Boolean

'Holt einen Bereich aus einer _geschlossenen_ Arbeitsmappe

   Dim strQuelle       As String
   Dim Zeilen          As Long
   Dim Spalten         As Byte

   On Error GoTo InvalidInput

   strQuelle = "'" & SourcePath & "[" & SourceFile & "]" & sourceSheet & "'!" & _
            Range(SourceRange).Cells(1, 1).Address(0, 0)
```

```
  Zeilen = Range(SourceRange).Rows.Count
  Spalten = Range(SourceRange).Columns.Count

  With TargetRange.Cells(1, 1).Resize(Zeilen, Spalten)
    .Formula = "=IF(" & strQuelle & "="""","""","" & strQuelle & ")"
    .Value = .Value
  End With

  GetDataClosedWB = True
  Exit Function

InvalidInput:
  MsgBox "Die Quelldatei oder der Quellbereich ist ungültig!", _
       vbExclamation, "Datenimport aus Quelldatei"
  GetDataClosedWB = False
End Function



Public Sub HoleDaten()
Dim Pfad           As String
Dim Dateiname      As String
Dim Blatt          As String

  Pfad = "C:\" ' Quelldateipfad
  Dateiname = "1.xls" ' Quelldatei
  Blatt = "Tabelle1" ' Quellarbeitsblatt (mit Sheets(1) kann man auch das erste Blatt nehmen)

  If GetDataClosedWB(Pfad, Dateiname, Blatt, "A1:IV9999", _
                Worksheets("Tabelle3").Range("A1")) Then 'A1-IV999 der zu kopierende Bereich - Tabelle3=die Zieltabelle, A1=Zielzelle für einfügen (ab dieser
Zelle)
    MsgBox "Daten importiert"
  End If
End Sub
```

Variante 3

```
Sub Read_All_Datas_from_defined_Workbooks_without_Opening()
'by Ramses
'Liest alle Daten aus geschlossenen Arbeitsblättern
'aus einem bestimmten Bereich ein.
'Alle eingelesenen Daten werden untereinander aufgelistet.
'Die Daten werden in Dateien mit dem Datei-Teilbegriff "Report"
'gesucht und eingelesen
Dim i As Long, totFiles As Long
Dim ColCounter As Integer, rowCounter As Long
Dim n As Integer, k As Integer
Dim gefFile As String, TeilName As String
Dim Suchpfad As String, Suchbegriff As String, Dateiform As String
Dim tmpPfad As String, tmpName As String, tmpFile As String
Dim curWB As Workbook, tarwks As Worksheet, datWKS As String
Dim oldStatus As Variant
Dim myR1 As String, myR2 As String, myR3 As String
Suchpfad = InputBox("Geben Sie den Ordner an, der durchsucht werden soll.", "Pfad definieren", "D:") 'Application.DefaultFilePath)
If Suchpfad = "" Then Exit Sub
Dateiform = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.xls")
If Dateiform = "" Then Exit Sub
Application.ScreenUpdating = True 'zur definitiven Ausführung auf False setzen
oldStatus = Application.StatusBar
'ZählVariablen setzen
rowCounter = 1
ColCounter = 2
'Variablen für aktive Mappe setzen
Set curWB = Workbooks(ThisWorkbook.name)
Set tarwks = curWB.Worksheets("Tabelle1")
'zu kopierende Bereiche definieren
'Variablen für den DateiNamen der entsprechenden Tabelle ersezten
TeilName = "Report"
'Tabellenname in der Mappe mit dem Teilstring "TeilName"
datWKS = "Summary"
'zu lesende Bereich definieren
myR1 = datWKS & "'!R3C2"
myR2 = datWKS & "'!R17C4"
'Datumsformat in Spalte D zuweisen
Columns(4).NumberFormat = "m/d/yyyy"
'Dateisuche starten
With Application.FileSearch
   .LookIn = Suchpfad
```

```
    .SearchSubFolders = False
    .FileName = Dateiform
    'Wenn gefunden,..
    'Schleifenauswertung beginnen
    If .Execute() > 0 Then
        totFiles = .FoundFiles.count
        Application.StatusBar = "Total " & totFiles & " gefunden"
        For i = 1 To .FoundFiles.count
            gefFile = .FoundFiles(i)
            'Namen und String zusammensetzen
            tmpName = Right(gefFile, Len(gefFile) - InStrRev(gefFile, "\", -1))
            tmpPfad = Left(gefFile, Len(gefFile) - Len(tmpName))
            tmpFile = "'" & tmpPfad & "[" & tmpName & "]"
            'Die Formel für das Excel4-Macro muss im R1C1 - Format erstellt werden
            'Auch die Rechteckklammern müssen eingebaut werden
            'Hochkomma's nicht vergessen !!
            ''D:\[Muster.xls]Summary'!R3C2
            If UCase(Left(Right(gefFile, Len(gefFile) - 3), Len(TeilName))) = UCase(TeilName) Then
                'In Tabelle eintragen
                tarwks.Cells(rowCounter, 1) = Application.ExecuteExcel4Macro(tmpFile & myR1)
                tarwks.Cells(rowCounter, 2) = Application.ExecuteExcel4Macro(tmpFile & myR2)
                tarwks.Cells(rowCounter, 2).NumberFormat = "0.00%"
                'Zwei neue Schleifen um die einzelnen zellen in
                'den Zieldateien auszulesen
                'Datenbereich von B23 : F39 einlesen
                For k = 23 To 39
                    For n = ColCounter To 6
                        myR3 = datWKS & "'!R" & k & "C" & n & ":R" & k & "C" & n
                        tarwks.Cells(rowCounter, n + 1) = Application.ExecuteExcel4Macro(tmpFile & myR3)
                    Next n
                    rowCounter = rowCounter + 1
                Next k
                rowCounter = rowCounter + 1
            End If
        Next i
    End If
End With
Application.StatusBar = oldStatus
Application.ScreenUpdating = True
End Sub
```

Variante 4

Information

The basic examples in the workbook that you can download use ADO to copy data from a closed workbook or workbooks without opening the workbook or workbooks. This can be very fast to merge data from many workbooks but when you open workbooks with code you have much more control and more options.

See the pages below if you want to open the files with code and merge the data

Merge data from all workbooks in a folder(1)
http://www.rondebruin.nl/copy3.htm

Merge data from all workbooks in a folder(2)
http://www.rondebruin.nl/fso.htm

RDBMerge Add-in (very easy)
http://www.rondebruin.nl/merge.htm

Important info about the ADO examples

1) The code in the workbook is working in Excel 2000-2007.

2) In a Database you cannot mix data types, a column must be all numbers or all text. If there are different data types in the column ADO will copy only the Data type that have the majority.

3) If you want to copy only one cell from each workbook then use A3:A3 and not A3 in the code.

How do I use ADO

Click here to Download a example workbook with 7 code examples and also a data file named test.xls.
With this two workbooks you can test the code (copy both files in the same folder).
Note: Read the info on the worksheet and also the commented lines in the macro's.

All macros in the Ado Tester.xls file call a macro named GetData that do almost all the work.

There are six arguments in this macro

Public Sub GetData(SourceFile As Variant, SourceSheet As String, _
SourceRange As String, TargetRange As Range, Header As Boolean, UseHeaderRow As Boolean)

If we look at the first test macro that copy "A1:C5" from "Sheet1" in the test.xls workbook we see:

```
GetData ThisWorkbook.Path & "\test.xls", "Sheet1", _
        "A1:C5", Sheets("Sheet1").Range("A1"), True, True
```

1) SourceFile : Path/Name of the source file
   ThisWorkbook.Path & "\test.xls"

2) SourceSheet : Name of the sheet in the SourceFile
   "Sheet1"

3) SourceRange : Range in the SourceSheet
   "A1:C5"

Note: You can also use a named range if you want like "MyRange".
leave the SourceSheet argument empty "" if you want to copy from a workbook level name.

4) TargetRange: Destination Sheet/Range
   Sheets("Sheet1").Range("A1")

5) Header: Does the range have a header row?
True

6) UseHeaderRow :Do you want to copy the header row ?
True

I hope that it is easy to use the code examples in the Ado Tester.xls file.
Let me know if you have suggestions or problems with the code.
Remember that I am a simple Excel user and no ADO expert.

More information

Ole P. Erlandsen's Web Site
http://www.erlandsendata.no/english/index.php?t=envbadac

John Walkenbach
http://www.j-walk.com/ss/excel/tips/tip82.htm

Create a summary worksheet from different workbooks (with formulas)
http://www.rondebruin.nl/summary2.htm

Copy a range from closed workbook (Local, Network and on the internet)
http://www.rondebruin.nl/copy7.htm

Variante 5

Copy a range from closed workbook (Local, Network and on the internet)

Ron de Bruin (last update 1 May 2006)
Go to the Excel tips page

You can find three examples here to get a range out of a closed workbook.
Note: All examples use the **GetRange** macro.

1: file on the internet (this file exist on my site)
2: file in network folder
3: File in local folder

Other useful pages are :
http://www.rondebruin.nl/ado.htm
http://www.rondebruin.nl/summary2.htm
http://www.j-walk.com/ss/excel/tips/tip82.htm

Note: there are five arguments in the macro's
1: Path
2: File name
3: Source sheet name
4: Source range
5: Destination sheet/range

Note: There is no error check on this moment for the path and file and sheet name.

```
Sub File_On_Website()
    Application.ScreenUpdating = False
    On Error Resume Next

    'Call the macro GetRange
    GetRange "http://www.rondebruin.nl/files", "test1.xls", "Sheet1", "A1:B100", _
            Sheets("Sheet1").Range("A1")

    On Error GoTo 0
    Application.ScreenUpdating = True
End Sub
```

```
Sub File_In_Network_Folder()
    Application.ScreenUpdating = False
    On Error Resume Next

    'Call the macro GetRange
    GetRange "\\Jdb\shareddocs", "test2.xls", "Sheet1", "A1:B100", _
```

```
        Sheets("Sheet1").Range("A1")

    On Error GoTo 0
    Application.ScreenUpdating = True
End Sub


Sub File_In_Local_Folder()
    Application.ScreenUpdating = False
    On Error Resume Next

    'Call the macro GetRange
    GetRange "C:\Data", "test3.xls", "Sheet1", "A1:B100", _
        Sheets("Sheet1").Range("A1")

    On Error GoTo 0
    Application.ScreenUpdating = True
End Sub
```

Main macro

```
Sub GetRange(FilePath As String, FileName As String, SheetName As String, _
        SourceRange As String, DestRange As Range)

    Dim Start

    'Go to the destination range
    Application.Goto DestRange

    'Resize the DestRange to the same size as the SourceRange
    Set DestRange = DestRange.Resize(Range(SourceRange).Rows.Count, _
                    Range(SourceRange).Columns.Count)

    'Add formula links to the closed file
    With DestRange
        .FormulaArray = "='" & FilePath & "/[" & FileName & "]" & SheetName _
                & "'!" & SourceRange

        'Wait
        Start = Timer
        Do While Timer < Start + 2
            DoEvents
        Loop
```

```
      'Make values from the formulas
      .Copy
      .PasteSpecial xlPasteValues
      .Cells(1).Select
      Application.CutCopyMode = False
   End With
End Sub
```

VARIANTE 6

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-3

Auslesen eines Wertes aus geschlossener Arbeitsmappe

```
Function xl4Value(strParam As String) As Variant
    xl4Value = ExecuteExcel4Macro(strParam)
End Function

Sub CallValue()
   Dim strSource As String
   strSource = _
      "'" & _
      Range("A2").Text & _
      "\" & Range("B2").Text & _
      "]" & Range("C2").Text & _
      "'!" & Range("D2").Text
   MsgBox "Zellwert Zelle A1: " & xl4Value(strSource)
End Sub


oder:
Sub Zelle_auslesen()
   Dim Adresse As String, Zeile As Integer, Spalte As Integer, Zellbezug As String
   Pfad = "D:\neue Dokumente\"
   Datei = "Urlaub 2009.xls"
   Register = "Kalender"
   Zeile = 14: Spalte = 20 ' entspricht T14
   Zellbezug = Cells(Zeile, Spalte).Address(ReferenceStyle:=xlR1C1)

   Adresse = "'" & Pfad & "" & Datei & "]" & Register & "'!" & Zellbezug

   Ergebnis = ExecuteExcel4Macro(Adresse)
   MsgBox ("Wert der Zelle T14: " & Ergebnis)

End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-4

Auslesen des ANZAHL2-Wertes aus geschlossener Arbeitsmappe

```
Function xl4CountA(strParam As String) As Variant
    xl4CountA = _
      ExecuteExcel4Macro("CountA(" & strParam & ")")
End Function

Sub CallCountA()
   Dim strSource As String
   strSource = _
      "'" & _
      Range("A3").Text & _
      "\" & Range("B3").Text & _
      "]" & Range("C3").Text & _
      "'!" & Range("D3").Text
   MsgBox "ANZAHL2 in A1:A100: " & xl4CountA(strSource)
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-5

Auslesen einer Summe aus geschlossener Arbeitsmappe

```
Function xl4Sum(strParam As String) As Variant
    xl4Sum = _
      ExecuteExcel4Macro("Sum(" & strParam & ")")
End Function

Sub CallSum()
   Dim strSource As String
   strSource = _
      "'" & _
      Range("A4").Text & _
      "\" & Range("B4").Text & _
      "]" & Range("C4").Text & _
      "'!" & Range("D4").Text
   MsgBox "SUMME in A1:B100: " & xl4Sum(strSource)
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-6

Auslesen eines SVERWEIS-Wertes aus geschlossener Arbeitsmappe

```
Function xl4VLookup(strParam As String) As Variant
    xl4VLookup = ExecuteExcel4Macro _
      ("VLookup(""" & Range("E5").Text & _
      """, " & strParam & ", " & _
```

```
      Range("F5").Text & ", " & _
      Range("G5").Text & ")")
End Function

Sub CallVLookup()
   Dim strSource As String
   strSource = _
      "'" & _
      Range("A5").Text & _
      "\" & Range("B5").Text & _
      "]" & Range("C5").Text & _
      "'!" & Range("D5").Text
   MsgBox "SVERWEIS in A1:B100: " & _
      xl4VLookup(strSource)
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-7

Auslesen einer Tabelle aus geschlossener und Einlesen in neue Arbeitsmappe

```
Sub ReadTable()
   Dim wks As Worksheet
   Dim intRow As Integer, intCol As Integer
   Dim strSource As String
   Application.ScreenUpdating = False
   Set wks = ActiveSheet
   Workbooks.Add
   For intRow = 1 To 20
      For intCol = 1 To 2
         strSource = _
            "'" & _
            wks.Range("A3").Text & _
            "\" & wks.Range("B2").Text & _
            "]" & wks.Range("C2").Text & _
            "'!R" & intRow & "C" & intCol
         Cells(intRow, intCol).Value = _
            xl4Value(strSource)
      Next intCol
   Next intRow
   Application.ScreenUpdating = True
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_XL4-Makros_in_VBA_verwenden&action=edit&section=T-8

Datum einer Datei auslesen

1. Möglichkeit:

Msgbox FileDateTime(ActiveWorkbook.Fullname)

Leider nennen die verschiedenen Handbücher, dass diese Funktion das ERSTELLUNGS oder das ÄNDERUNGSDATUM anzeigt - daher welches NUN ?

Um auf Nummer sicher zu gehen nimmt die 2. Möglichkeit

2. Möglichkeit:

```
Sub DATEI_DATUM()
    Dim fso
    Dim Datei_Pfad_und_Name
    Dim Datei

    Datei_Pfad_und_Name = ActiveWorkbook.FullName
    Set fso = CreateObject("Scripting.filesystemobject")
    Set Datei = fso.getfile(Datei_Pfad_und_Name)

    MsgBox Datei.DateCreated
    MsgBox Datei.DateLastAccessed
    MsgBox Datei.DateLastModified
End Sub
```

Wichtig: Möchte man das Datum einer Datei mit dem heutigen Datum vergleichen, muss man die Datumsvariable DATE umwandeln in einen String:

```
    If Left(DATEI_OBJEKT.DateLastModified, 10) <> Str$(Date) Then 'Wenn das Datum der Datei nicht der heutige Tag ist
```

Filesystem auslesen (NTFS - FAT32)

```
Sub Give_Filesystem()
   Dim myFSO As Object, myDrv As Object, strFS As String
   Set myFSO = CreateObject("Scripting.FileSystemObject")
   'Laufwerkbuchstabe angeben, oder über InputBox abfragen
   Set myDrv = myFSO.GetDrive("e:")
   strFS = myDrv.FileSystem
   MsgBox strFS
End Sub
```

Kopie der aktuellen Arbeitsmappe speichern

siehe ARBEITSMAPPEN unter gleichnamigem Eintrag

Laufwerke anzeigen

Um mit dem File-System-Object z.B. alle Laufwerke ihres Rechners anzuzeigen, können Sie folgenden Code verwenden

```
Sub Show_all_Drives()
'(c) ramses
Dim myFSO As Object, myDrv As Object, drvCount, drvStr As String, vName As String, drvTyp As String
Set myFSO = CreateObject("Scripting.FileSystemObject")
Set drvCount = myFSO.Drives
'Fehlerbehandlung einschalten für Laufwerke welche nicht erkannt werden können oder
'vor dem User versteckt wurden
On Error Resume Next
'String definieren
drvStr = ""
'Jedes erkannte Laufwerk identifizieren
For Each myDrv In drvCount
                    'Laufwerkstyp erkennen
    Select Case myDrv.DriveType
       Case 0: drvTyp = "Unknown"
       Case 1: drvTyp = "Removable"
       Case 2: drvTyp = "Fixed"
       Case 3: drvTyp = "Network"
       Case 4: drvTyp = "CD-ROM"
       Case 5: drvTyp = "RAM Disk"
    End Select
                    'String erstellen
    drvStr = drvStr & drvTyp & ": " & myDrv.DriveLetter & " - "
                '3 = Netzlaufwerk
        'hier wird der Freigabename je nach Typ abgerufen
    If myDrv.DriveType = 3 Then
       vName = myDrv.ShareName
    Else
       vName = myDrv.VolumeName
    End If
    drvStr = drvStr & vName & vbCrLf
Next
'Ausgabe der Information
MsgBox drvStr
End Sub
```

Der Code sollte selbsterklärend sein

Listenfeld in Userform mit allen Dateien eines Ordners zur Auswahl

Man erzeugt zuerst eine Userform1 und darin ein Listenfeld Listbox1

```
Sub Fill_Listbox_with_Filenames()
Dim i As Long, totFiles As Long
Dim gefFile As String, dname As String
Dim Suchpfad As String, suchbegriff As String, DateiForm As String
Dim oldStatus As Variant
Suchpfad = InputBox("Geben Sie den Ordner an, der durchsucht werden soll.", "Pfad definieren", Application.DefaultFilePath)
If Suchpfad = "" Then Exit Sub
DateiForm = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.xls")
If DateiForm = "" Then Exit Sub
'Bildschirmaktualisier                                                                    abschalten
Application.ScreenUpdating = True
'Text                der          Statusbar          und          alten          Status          aufnehmen
oldStatus = Application.StatusBar
'Start                                        der                                        Suchroutine
With Application.FileSearch
   .LookIn = Suchpfad
   .SearchSubFolders = True
   .Filename = DateiForm
   If .Execute() > 0 Then
      totFiles = .FoundFiles.Count
                                    'Ausgabe                    in                    Statusbar
      Application.StatusBar = "Total " & totFiles & " gefunden"
      For i = 1 To .FoundFiles.Count
         gefFile = .FoundFiles(i)
         'In Listbox eintragen          mit          der          AddItem          Methode
         Userform1.ListBox1.AddItem (gefFile)
      Next i
   End If
End With
'Status          der          Statusbar          wieder          herstellen
Application.StatusBar = oldStatus
'Bildschirmaktualisierung                              wieder                              einschalten
Application.ScreenUpdating = True
Userform1.Show

End Sub
```

Listenfeld füllen mit Dateinamen mit genauer Übereinstimmung oder teilweiser Übereinstimmung im Dateinamen

Als Suchbegriff sind die bekannten Platzhalter wie "*" und "?" zugelassen.
Sie können in der Inputbox also auch Werte wie "Mu*datei.xls" oder "Ma?er.xls" oder "*2002.xls" eingeben

```
Sub Find_Files_with_Textfragment()
Dim i As Long
Dim gefFile As String, dname As String
Dim Suchpfad As String, Suchbegriff As String, DateiForm As String
Dim oldStatus As Variant, myMatch As Boolean, msgTxt As String, Qe As Variant
Suchpfad = InputBox("Geben Sie den Ordner an, der durchsucht werden soll:", "Pfad definieren", Application.DefaultFilePath)
If Suchpfad = "" Then Exit Sub
DateiForm = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.xls")
If DateiForm = "" Then Exit Sub
Suchbegriff = InputBox("Geben Sie den Text an der im                                    Dateinamen gesucht werden soll", "Textfragment", "")
If Suchbegriff = "" Then Exit Sub
'Text                    der                    MsgBox                    zusammensetzen
msgTxt = "Soll auf exakte Übereinstimmung mit dem Dateinamen gesucht werden ? "
msgTxt = msgTxt & vbCrLf & "Bei ""Nein"" werden als Ergebnis auch Dateien angezeigt,"
msgTxt = msgTxt & vbCrLf & "bei denen nur ein Teil des Namens mit:"" " & Suchbegriff & " "" übereinstimmt !"
Qe = MsgBox(msgTxt, vbQuestion + vbYesNo, "Suchroutine")
'WAHR                    oder                    FALSCH                    der                    Variable                    zuweisen
If Qe = vbOK Then
   myMatch = True
Else
   myMatch = False
End If
'Bildschirmaktualisier abschalten
Application.ScreenUpdating = True
'Text der Statusbar und alten Status aufnehmen
oldStatus = Application.StatusBar
'Start der Suchroutine
With Application.FileSearch
   .NewSearch
   .LookIn = Suchpfad
   .TextOrProperty = Suchbegriff
   ' = True wenn der Suchbegriff GENAU übereinstimmen soll
   ' = False wenn nur ein Teil des Dateinamens übereinstimmen soll
   .MatchTextExactly = myMatch
   .FileType = DateiForm
   If .Execute() > 0 Then
      totFiles = .FoundFiles.Count
      'Ausgabe in Statusbar
      Application.StatusBar = "Total " & totFiles & " gefunden"
      For i = 1 To .FoundFiles.Count
         gefFile = .FoundFiles(i)
```

```
            'In Listbox eintragen mit der AddItem Methode
            Me.ListBox1.AddItem (gefFile)
        Next i
    End If
End With
```

Optional: Auswahl einschränken nach Änderungsdatum:

Benutzen Sie diesen Parameter um ihre Suche wie im Beispiel "Listbox füllen mit Dateinamen mit genauer Übereinstimmung oder teilweiser Übereinstimmung im Dateinamen" weiter einzuschränken. Blenden Sie doch eine Userform in der Form ein:

und überlassen Sie dem User die Auswahl. Die Auswahl des Users können Sie an eine Public-Variable (in der Userform zu definieren !!) übergeben und in die File-Search-Methode einbinden

```
Option Explicit
Public SearchArea As String
```

```
Private Sub btnOk_Click()
'Mit der Routine spart man sich das programmieren
'jedes einzelnen Buttons :-)
Dim myC As Control, SearchValue As Integer
For Each myC In Me.Controls
   'Alle OptionButtons abfragen
   If Left(myC.Name, 6) = "optBtn" Then
      'Die Nummer des Optionsbuttons auslesen
      'ACHTUNG:
      'Erstellen Sie die OptionButtons in der richtigen
      'Reihenfolge oder bennenen Sie sie entsprechend um
      'Die Optionbuttons in diesem Beispiel haben die Namen
      'gemäss MS-Nomenklatur
      'optBtnGrpTimeStamp1, optBtnGrpTimeStamp2 usw
                        'und          gehören          zur          Option          Gruppe          "TimeStamp"
                         'Abfragen          ob          der          Button          aktiviert          ist
                             If          myC          =          True          Then
                                 SearchValue          =          Right(myC.Name,          1)
                                  'Ausstieg          aus          der          Schleife
                                             Exit          For
                                             End          If
      Exit For
   End If
Next
'Wert auslesen und den String an "SearchArea" übergeben
Select Case SearchValue
   Case 1
      SearchArea = "msoLastModifiedAnyTime"
   Case 2
      SearchArea = "msoLastModifiedToday"
   Case 3
      SearchArea = "msoLastModifiedYesterday"
   Case 4
      SearchArea = "msoLastModifiedThisWeek"
   Case 5
      SearchArea = "msoLastModifiedLastWeek"
   Case 6
      SearchArea = "msoLastModifiedThisMonth"
   Case 7
      SearchArea = "msoLastModifiedLastMonth"
End Select
```

```
End Sub
```
Das sieht dann in etwa so aus, in Verbindung mit Beispiel von oben :
```
With Application.FileSearch
   .NewSearch
   .LookIn = Suchpfad
   .TextOrProperty = Suchbegriff
   .MatchTextExactly = myMatch
   .FileType = DateiForm
                                                                                    '----------------------
                                  .LastModified                    =                SearchArea
                                                                                    '----------------------
   If .Execute() > 0 Then
      totFiles = .FoundFiles.Count
      'Ausgabe in Statusbar
      Application.StatusBar = "Total " & totFiles & " gefunden"
      For i = 1 To .FoundFiles.Count
         gefFile = .FoundFiles(i)
          'In Listbox eintragen mit der AddItem Methode
         Me.ListBox1.AddItem (gefFile)
      Next i
   End If
End With
```

Name der aktuellen Arbeitsmappe

```
msgbox Right(ActiveWorkbook.FullName, Len(ActiveWorkbook.FullName) - InStrRev(ActiveWorkbook.FullName, "\"))
```

Ordner erzeugen, falls nicht vorhanden

Lösung 1:

```
Private Sub Workbook_Open()
If Dir("C:\Temp", vbDirectory) = "" Then
  MkDir ("C:\Temp")
  MsgBox "Ordner ''Temp'' wurde angelegt!"
Else
  MsgBox "Ordner ''Temp'' ist vorhanden!"
```

```
End If
End Sub
```

Lösung 2:

Problem: Es soll geprüft werden, ob ein Verzeichnis mit Unterverzeichnissen bereits besteht. Wenn ja, soll eine entsprechende Meldung erfolgen, wenn nein sollen sie erstellt werden.

```
StandardModule: Modul1

Sub Pruefen()
   Dim arrOrdner As Variant
   Dim iOrdner As Integer
   Dim sDrive As String, sOrdner As String, sTmp As String
   sOrdner = InputBox("Zu erstellendes Verzeichnis:", , Range("B1").Value)
   If sOrdner = "" Then Exit Sub
   If Right(sOrdner, 1) = "\" Then
      sOrdner = Left(sOrdner, Len(sOrdner) - 1)
   End If
   arrOrdner = fncFolders(sOrdner)
   For iOrdner = UBound(arrOrdner) To 1 Step -1
      If fncIfFolderExists(CStr(arrOrdner(iOrdner))) Then
         MsgBox "Ordner " & arrOrdner(iOrdner) & " ist bereits vorhanden!"
      Else
         MkDir arrOrdner(iOrdner)
      End If
   Next iOrdner
End Sub

Private Function fncFolders(sFolder As String) As Variant
   Dim arr() As String
   Dim iCounter As Integer, iFolder As Integer
   ReDim arr(1 To 1)
   arr(1) = sFolder
   iFolder = 1
   For iCounter = Len(sFolder) To 4 Step -1
      If Mid(sFolder, iCounter, 1) = "\" Or iCounter = 1 Then
         iFolder = iFolder + 1
         ReDim Preserve arr(1 To iFolder)
         arr(iFolder) = Left(sFolder, iCounter - 1)
      End If
   Next iCounter
   fncFolders = arr
End Function

Private Function fncIfFolderExists(sFolder As String) As Boolean
```

```
    Dim sOld As String
    sOld = CurDir
    On Error Resume Next
    ChDrive Left(sFolder, 1)
    ChDir sFolder
    If Err = 0 Then fncIfFolderExists = True
    On Error GoTo 0
    ChDrive Left(sOld, 1)
    ChDir sOld
End Function
```

Lösung 3

```
'  Title: Creates the specified directory path (including subdirs if necessary)
'


' ----------------------------------------------------------------------------


'-------------------------------------------------------------
' FUNCTION: MakePathAux
'
' Creates the specified directory path.
'
' No user interaction occurs if an error is encountered.
' If user interaction is desired, use the related
'   MakePathAux() function.
'
' IN: [strDirName] - name of the dir path to make
'
' Returns: True if successful, False if error.
'-------------------------------------------------------------
'
Function MakePathAux(ByVal strDirName As String) As Boolean
    Dim strPath As String
    Dim intOffset As Integer
    Dim intAnchor As Integer
    Dim strOldPath As String

    On Error Resume Next

    '
    'Add trailing backslash
```

```
    '
    If Right$(strDirName, 1) <> gstrSEP_DIR Then
        strDirName = strDirName & gstrSEP_DIR
    End If

    strOldPath = CurDir$
    MakePathAux = False
    intAnchor = 0

    '
    'Loop and make each subdir of the path separately.
    '
    intOffset = InStr(intAnchor + 1, strDirName, gstrSEP_DIR)
    intAnchor = intOffset 'Start with at least one backslash, i.e. "C:\FirstDir"
    Do
        intOffset = InStr(intAnchor + 1, strDirName, gstrSEP_DIR)
        intAnchor = intOffset

        If intAnchor > 0 Then
            strPath = Left$(strDirName, intOffset - 1)
            ' Determine if this directory already exists
            Err = 0
            ChDir strPath
            If Err Then
                ' We must create this directory
                Err = 0
#If LOGGING Then
                NewAction gstrKEY_CREATEDIR, """" & strPath & """"
#End If
                MkDir strPath
#If LOGGING Then
                If Err Then
                    LogError ResolveResString(resMAKEDIR) & " " & strPath
                    AbortAction
                    GoTo Done
                Else
                    CommitAction
                End If
#End If
            End If
        End If
    Loop Until intAnchor = 0
```

```
    MakePathAux = True
Done:
    ChDir strOldPath

    Err = 0
End Function
```

Ordner / Pfad auswählen lassen

--- NEUE LÖSUNG 2019 ohne 32bit-Bezüge

V1

```
Public Sub Ordner_Auswahl()
    Dim strVerzeichnis As String
    Dim strOrdner As String
    strOrdner = ThisWorkbook.Path & "\" ' oder "C:\Temp\" - wichtig immer ist \ am Ende
    With Application.FileDialog(msoFileDialogFolderPicker)
        .InitialFileName = strOrdner
        .Title = "Ordnerauswahl"
        .ButtonName = "Auswahl..."
        .InitialView = msoFileDialogViewList
          If .Show = -1 Then
            strVerzeichnis = .SelectedItems(1)
            If Right(strVerzeichnis, 1) <> "\" Then strVerzeichnis = strVerzeichnis & "\"
          Else
            MsgBox "Es wurde kein Ordner ausgewaehlt!"
            Exit Sub
          End If
      End With
    MsgBox strVerzeichnis
End Sub
```

V2

```
Sub ttt()
  Dim sPfad As String
```

```
  With Application.FileDialog(msoFileDialogFolderPicker)
    .Title = "Test"
    If .Show = -1 Then
      sPfad = .SelectedItems(1)
      MsgBox sPfad
    End If
  End With
End Sub
```

oder mit Initialfolder

```
Sub ttt()
  Dim sPfad As String
  With Application.FileDialog(msoFileDialogFolderPicker)
    .Title = "Test"
    .InitialFileName = ThisWorkbook.Path & "\" ' oder "E:\temp\"  - wichtig immer ist \ am Ende
    If .Show = -1 Then
      sPfad = .SelectedItems(1)
      MsgBox sPfad
    End If
  End With
End Sub
```

V3

```
Sub ordnerauswahl()
   Dim AppShell As Object
   Dim BrowseDir As Variant
   Dim Pfad As String

   Set AppShell = CreateObject("Shell.Application")
   Set BrowseDir = AppShell.BrowseForFolder(0, "Ordner auswählen", &H1000, 17)
   On Error Resume Next
   Pfad = BrowseDir.items().Item().Path
   If Pfad = "" Then Exit Sub
   MsgBox Pfad
   On Error Goto 0
End Sub
```

V4

```
Public Function VerzeichnisAuswaehlen()
   Dim objFiledialog As FileDialog
```

```
    Set objFiledialog = _
      Application.FileDialog(msoFileDialogFolderPicker)
    With objFiledialog
       .AllowMultiSelect = False
       If .Show = True Then
          VerzeichnisAuswaehlen = .SelectedItems(1)
       End If
    End With
    Set objFiledialog = Nothing
End Function
```

V5

```
Public Sub Beispiel()
    Dim strPath As String
    With Application.FileDialog(msoFileDialogFolderPicker)
       .AllowMultiSelect = False
       .InitialFileName = "C:\Windows\System32\"
       .Title = "Ordner auswählen"
       If .Show Then strPath = .SelectedItems(1)
    End With
    If strPath <> vbNullString Then
       MsgBox strPath
    End If
End Sub
```

----ALTE LÖSUNGEN ----------

die eigentliche SUB ist dann die letzte, blaue Prozedur – der Rest ist nur Hilfe (der Code ist entnommen von ARBEITSMAPPEN / "Alle Arbeitsmappen in einem Ordner zusammenfassen in neue gemeinsame Arbeitsmappe" – dort ist dann noch mehr code in der hier blauen Sub inkl Alle Mappen in diesem Ordner durchlaufen und öffnen und Daten kopieren)

```
' ********************************************************************
' Modul: Modul1 Typ: Allgemeines Modul
' ********************************************************************

'BrowseForFolder mit Extra-Funktionen
'VB -Versionen: VB5 , VB6
'Betriebssystem: Win9x , WinNT, Win2000, WinME, WinXP
'Autor: Marco Wünschmann Homepage: ohne
'Datum: 23.08.2004

'Option Explicit

' == Dialog-Einstellungen ================================

' String, der vor dem aktuell ausgewählen Verzeichnis angezeigt wird,
' falls der ShowCurrentPath-Paramter True ist.
Private Const DIALOG_CURRENT_SELECTION_TEXT As String = "Auswahl: "


' == API-Deklarationen ===================================

Private Type BROWSEINFO
  hOwner As Long
  pidlRoot As Long
  pszDisplayName As String
  lpszTitle As String
  ulFlags As Long
  lpfnCallback As Long
  lParam As Long
  iImage As Long
End Type

Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
```

```
End Type

Private Type Size
  cx As Long
  cy As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
  Alias "SHBrowseForFolderA" ( _
  lpBrowseInfo As BROWSEINFO) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
  Alias "SHGetPathFromIDListA" ( _
  ByVal lPIDL As Long, _
  ByVal pszPath As String) As Long

Private Declare Sub CoTaskMemFree Lib "ole32.dll" ( _
  ByVal pv As Long)

Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" ( _
  ByVal hwnd As Long, _
  ByVal wMsg As Long, _
  ByVal wParam As Long, _
  lParam As Any) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
  Alias "RtlMoveMemory" ( _
  pDest As Any, _
  pSource As Any, _
  ByVal dwLength As Long)

Private Declare Function ILCreateFromPath Lib "shell32" _
  Alias "#157" ( _
  ByVal sPath As String) As Long

Private Declare Function LocalAlloc Lib "kernel32" ( _
  ByVal uFlags As Long, _
  ByVal uBytes As Long) As Long

Private Declare Function LocalFree Lib "kernel32" ( _
  ByVal hmem As Long) As Long

Private Declare Function lstrcpyA Lib "kernel32" ( _
```

```
  lpString1 As Any, _
  lpString2 As Any) As Long

Private Declare Function lstrlenA Lib "kernel32" ( _
  lpString As Any) As Long

Private Declare Function FindWindowEx Lib "user32.dll" _
  Alias "FindWindowExA" ( _
  ByVal hWnd1 As Long, _
  ByVal hWnd2 As Long, _
  ByVal lpsz1 As String, _
  ByVal lpsz2 As String) As Long

Private Declare Function GetWindowDC Lib "user32.dll" ( _
  ByVal hwnd As Long) As Long

Private Declare Function GetWindowRect Lib "user32.dll" ( _
  ByVal hwnd As Long, _
  ByRef lpRect As RECT) As Long

Private Declare Function GetTextExtentPoint Lib "gdi32.dll" _
  Alias "GetTextExtentPointA" ( _
  ByVal hDC As Long, _
  ByVal lpszString As String, _
  ByVal cbString As Long, _
  ByRef lpSize As Size) As Long

Private Declare Function PathCompactPath Lib "shlwapi.dll" _
  Alias "PathCompactPathA" ( _
  ByVal hDC As Long, _
  ByVal pszPath As String, _
  ByVal dx As Long) As Long

Private Const MAX_PATH = 260

Private Const WM_USER = &H400

Private Const BFFM_INITIALIZED = 1
Private Const BFFM_SELCHANGED As Long = 2
Private Const BFFM_SETSTATUSTEXTA As Long = (WM_USER + 100)
Private Const BFFM_SETSTATUSTEXTW As Long = (WM_USER + 104)
Private Const BFFM_ENABLEOK As Long = (WM_USER + 101)
Private Const BFFM_SETSELECTIONA As Long = (WM_USER + 102)
Private Const BFFM_SETSELECTIONW As Long = (WM_USER + 103)
```

```
Private Const BIF_NEWDIALOGSTYLE As Long = &H40
Private Const BIF_RETURNONLYFSDIRS As Long = &H1
Private Const BIF_BROWSEINCLUDEFILES As Long = &H4000
Private Const BIF_STATUSTEXT As Long = &H4

Private Const LMEM_FIXED = &H0
Private Const LMEM_ZEROINIT = &H40
Private Const LPTR = (LMEM_FIXED Or LMEM_ZEROINIT)

' Zeigt den BrowseForFolder-Dialog an.
Public Function BrowseForFolder(DialogText As String, _
  DefaultPath As String, _
  OwnerhWnd As Long, _
  Optional ShowCurrentPath As Boolean = True, _
  Optional RootPath As Variant, _
  Optional NewDialogStyle As Boolean = False, _
  Optional IncludeFiles As Boolean = False) As String

' Parameter:
' o DialogText Dialogtext, der oben im Dialog angezeigt wird.
' o DefaultPath Standardmäßig ausgewähltes Verzeichnis.
' o OwnerhWnd hWnd des übergeordneten Fensters (in den meisten
' Fällen Me.hWnd).
' o ShowCurrentPath Legt fest, ob die aktuelle Verzeichnisauswahl
' angezeigt werden soll. Verfügbar ab
' Internet Explorer 4.0 (-> PathCompactPath).
' o RootPath Root-Verzeichnis. Wird es angegeben, werden nur die
' Ordner unterhalb dieses Verzeichnisses angezeigt.
' o NewDialogStyle Legt fest, ob der Dialog in der neuen Darstellung
' angezeigt werden soll (Dialog kann vergrößert/
' verkleinert werden, es ist eine Schaltfläche zum
' Anlegen eines neuen Ordners vorhanden, es können
' Dateioperationen wie löschen etc. ausgeführt
' werden, ...). Ist dieser Parameter True, hat der
' Parameter ShowCurrentPath keine Wirkung. Verfügbar
' unter WinME und Betriebsystemen ab Win2000.
' o IncludeFiles Legt fest, ob auch Dateien im Dialog angezeigt und
' ausgewählt werden können.
' Verfügbar ab Win98 und Internet Explorer 4.0 (bei
' frühreren Windowsversionen muss IE4 inkl. der
' Integrated Shell installiert sein).

Dim biBrowseInfo As BROWSEINFO
```

```
Dim lPIDL As Long
Dim sBuffer As String
Dim lBufferPointer As Long

With biBrowseInfo
  ' Handle des übergeordneten Fensters
  .hOwner = OwnerhWnd

  ' PIDL des Rootordners zuweisen
  If Not IsMissing(RootPath) Then .pidlRoot = PathToPIDL(RootPath)

  ' Dialogtext zuweisen
  If ShowCurrentPath And DialogText = "$" Then DialogText = "" ' Wird intern nicht zugelassen
  .lpszTitle = DialogText

  ' Stringbuffer für aktuell selektierten Pfad zuweisen
  If ShowCurrentPath Then .pszDisplayName = sBuffer

  ' Dialogeinstellungen zuweisen
  .ulFlags = BIF_RETURNONLYFSDIRS + _
    IIf(ShowCurrentPath, BIF_STATUSTEXT, 0) + _
    IIf(NewDialogStyle, BIF_NEWDIALOGSTYLE, 0) + _
    IIf(IncludeFiles, BIF_BROWSEINCLUDEFILES, 0)

  ' Callbackfunktion-Adresse zuweisen
  .lpfnCallback = FARPROC(AddressOf CallbackString)

  ' PIDL des vorselektierten Ordnerpfades zuweisen (wird im
  ' lpData-Parameter an die Callback-Funktion weitergeleitet)
  .lParam = PathToPIDL(DefaultPath)
End With

' BrowseForFolder-Dialog anzeigen
lPIDL = SHBrowseForFolder(biBrowseInfo)

If lPIDL Then
  ' Stringspeicher reservieren
  sBuffer = Space$(MAX_PATH)

  ' Selektierten Pfad aus der zurückgegebenen PIDL ermitteln
  SHGetPathFromIDList lPIDL, sBuffer

  ' Nullterminierungszeichen des Strings entfernen
  sBuffer = Left$(sBuffer, InStr(sBuffer, vbNullChar) - 1)
```

```
  ' Selektierten Pfad zurückgeben
  BrowseForFolder = sBuffer

  ' Reservierten Task-Speicher wieder freigeben
  Call CoTaskMemFree(lPIDL)
End If

' Stringspeicher wieder freigeben
If ShowCurrentPath Then Call LocalFree(lBufferPointer)
End Function


Private Function CallbackString(ByVal hwnd As Long, ByVal uMsg As Long, _
  ByVal lParam As Long, ByVal lpData As Long) As Long

' Callback-Funktion des BrowseForFolder-Dialogs. Wird bei
' eintretenden Ereignissen des Dialogs aufgerufen.

Dim sBuffer As String
Dim lStaticWnd As Long
Dim lStaticDC As Long
Dim sPath As String
Dim rctStatic As RECT
Dim szTextSize As Size

' Meldungen herausfiltern
Select Case uMsg
  Case BFFM_INITIALIZED
    ' Dialog wurde initialisiert

    ' Standardmäßig markierten Pfad (dessen PIDL wurde in lpData
    ' übergeben) im Dialog selektieren
    Call SendMessage(hwnd, BFFM_SETSELECTIONA, False, ByVal lpData)
  Case BFFM_SELCHANGED
    ' Selektion hat sich geändert

    ' Stringspeicher reservieren
    sBuffer = Space$(MAX_PATH)

    ' Aktuell selektierten Pfad ermitteln und anzeigen, wenn möglich
    If SHGetPathFromIDList(lParam, sBuffer) Then
      ' Temporäre Zeichenfolge an das Anzeigelabel senden, um
      ' dessen Handle anhand dieser Zeichenfolge ermitteln zu können
```

```
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal "$"

      ' Handle und DeviceContext des Anzeigelabels ermitteln
      lStaticWnd = FindWindowEx(hwnd, ByVal 0&, ByVal "Static", ByVal "$")
      lStaticDC = GetWindowDC(lStaticWnd)

      ' Abmessungen des Anzeigelabels ermitteln
      GetWindowRect lStaticWnd, rctStatic

      ' Textabmessungen der Zeichenfolge "Auswahl: " im Anzeigelabel
      ' ermitteln
      GetTextExtentPoint lStaticDC, ByVal DIALOG_CURRENT_SELECTION_TEXT, _
        ByVal Len(DIALOG_CURRENT_SELECTION_TEXT), szTextSize

      ' Anzuzeigenden Pfad auf die Abmessungen des Anzeigelabels
      ' kürzen; falls dies nicht möglich ist, gesamten Pfad anzeigen
      sPath = sBuffer
      If PathCompactPath(ByVal lStaticDC, sPath, ByVal (rctStatic.Right - _
        rctStatic.Left - szTextSize.cx + 80)) = 0 Then sPath = sBuffer

      ' Nullterminierung entfernen
      sPath = Left$(sPath, InStr(1, sPath, vbNullChar) - 1)

      ' Pfad im Dialog anzeigen
      Call SendMessage(hwnd, BFFM_SETSTATUSTEXTA, 0&, _
        ByVal DIALOG_CURRENT_SELECTION_TEXT & sPath)
    Else
      ' Pfadanzeige leeren
      SendMessage hwnd, BFFM_SETSTATUSTEXTA, 0&, ByVal ""
    End If
End Select
End Function


Private Function FARPROC(FunctionPointer As Long) As Long
' Funktion wird benötigt, um Funktions-Adresse ermitteln
' zu können, dessen Adresse mit AddressOf übergeben und
' anschließend wieder zurückgegeben wird.

FARPROC = FunctionPointer
End Function


' Gibt die lPIDL zum übergebenen Pfad zurück.
```

```
Private Function PathToPIDL(ByVal sPath As String) As Long
Dim lRet As Long

lRet = ILCreateFromPath(sPath)
If lRet = 0 Then
  sPath = StrConv(sPath, VbStrConv.vbUnicode)
  lRet = ILCreateFromPath(sPath)
End If

PathToPIDL = lRet
End Function



Public Sub SearchFileAndCopySheet()
Dim objFS As FileSearch
Dim objFO As Object
Dim objWb As Workbook, objNew As Workbook
Dim strPath As String
Dim intIndex As Integer


strPath = BrowseForFolder("Quellverzeichnis auswählen", ThisWorkbook.Path, 0, , , True, False)

msgbox strPath

End Sub
```

LÖSUNG ANGEPASST für "dvo-Importe=>BMD-Import"-Konverter:

- es wird zuerst ein Datentyp und zwei Funktionen festgelegt
- und dann eine neue Funktion angelegt

- und zuletzt folgt die eigentliche Prozedur zum Speichern (die einen bereits hinterlegten Speicherpfad vorschlägt und ihn abändern lässt)

```
Public Type BROWSEINFO
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    Ipfn As Long
    IParam As Long
    iImage As Long
End Type
Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Function OrdnerAuswahl() As String
    Dim bInfo As BROWSEINFO
    Dim strPath As String
    Dim r As Long
    Dim X As Long
    Dim pos As Integer

    ' Der Ausgangsordner ist der Desktop :
    bInfo.pidlRoot = 0& ' 5& wären die Eigenen Dateien
    ' Dialogtitel
    bInfo.lpszTitle = "Wohin soll die Datei gespeichert werden ?"
    ' Rückgabe des Unterverzeichnisses
    bInfo.ulFlags = &H1
    ' Dialog anzeigen
    X = SHBrowseForFolder(bInfo)
    ' Ergebnis gliedern
    strPath = Space$(512)
    ' Ausgewähltes Verzeichnis einlesen
    r = SHGetPathFromIDList(ByVal X, ByVal strPath)
    If r Then
    pos = InStr(strPath, Chr$(0))
    OrdnerAuswahl = Left(strPath, pos - 1)
    Else
    OrdnerAuswahl = ""
    End If
End Function
```

```
Sub SPEICHERN_PERSONENKONTEN()

Dim Z As Integer ' Zeilenvariable
Dim LZ As Integer ' Letzte Zeile mit Daten
Dim S As Integer ' Spaltenvariable
Dim DATEINAME
Dim PFAD

Sheets("PK").Activate

' Auslesen des Speicherpfades
PFAD = Sheets("Stamm").Range("K6")

' Speicherpfad bestätigen oder auswählen lassen
If MsgBox("Ist der Speicherpfad " & vbCrLf & vbCrLf & PFAD & vbCrLf & vbCrLf & "(in der Tabelle 'Stamm' hinterlegt)" & vbCrLf & vbCrLf & "korrekt ?",
vbYesNo, "SPEICHERPFAD") = vbNo Then
   PFAD = OrdnerAuswahl
   If PFAD = "" Then
      MsgBox "Es wurde kein Speicherverzeichnis ausgewählt"
      Exit Sub
   End If
End If

' Festlegen dieses Speicherpfades
   On Error GoTo FEHLER
   ChDrive (PFAD)
   ChDir (PFAD)

' zur ersten Datenzeile springen
   LZ = LETZTEZELLE("PK").Row

   DATEINAME = PFAD & "/PK.txt"

   Open DATEINAME For Output As #1
   Range("A2").Select
   For Z = 2 To LZ
      For S = 1 To 8
         ZELLWERT = ZELLWERT & ActiveCell.Value & vbTab
         ActiveCell.Offset(0, 1).Select
      Next S
      Print #1, ZELLWERT
      ZELLWERT = ""
      Range("A" & Z).Select
   Next Z
```

```
    Close #1

Cells(2, 1).Select


    MsgBox "Die Personenkonten wurden als Datei 'PK.txt' nach " & PFAD & "  gespeichert." & vbCrLf & vbCrLf & _
       "Sie können die Konten in BMD mit  'pr08i'  und dem Paramter  'SIMPLEFI'  importieren."

    Exit Sub

FEHLER:
MsgBox "Es ist folgender Fehler aufgetreten :" & vbCrLf & vbCrLf & "' Error " & Err.Number & " - " & Err.Description & " '" & vbCrLf & vbCrLf & _
"Bitte überprüfen Sie, ob das ausgewählte Verzeichnis zum Speichern auch wirklich existiert."

End Sub
```

LÖSUNG ORIGINAL

1.) Zuerst werden ein Typ und zwei Funktionen aus der Bibliothek SHELL32.DLL deklariert, diese Anweisungen müssen am oberen Modulrand stehen

```
Public Type BROWSEINFO
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    Ipfn As Long
    IParam As Long
    iImage As Long
End Type
Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long
```

2.) Schreibe eine Funktion, die mithilfe der Bibliotheksfunktionen ein Dialogfeld mit allen Laufwerken und Ordnern der aktuellen Windows-Betriebssystemumgebung produziert:

```
Function OrdnerAuswahl() As String
    Dim bInfo As BROWSEINFO
    Dim strPath As String
    Dim r As Long
    Dim X As Long
```

```
    Dim pos As Integer

    ' Der Ausgangsordner ist der Desktop :
    bInfo.pidlRoot = 0& ' 5& wären die Eigenen Dateien
    ' Dialogtitel
    bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
    ' Rückgabe des Unterverzeichnisses
    bInfo.ulFlags = &H1
    ' Dialog anzeigen
    X = SHBrowseForFolder(bInfo)
    ' Ergebnis gliedern
    strPath = Space$(512)
    ' Ausgewähltes Verzeichnis einlesen
    r = SHGetPathFromIDList(ByVal X, ByVal strPath)
    If r Then
    pos = InStr(strPath, Chr$(0))
    OrdnerAuswahl = Left(strPath, pos - 1)
    Else
    OrdnerAuswahl = ""
    End If
End Function
```

Den obigen Parameter blnfo.pidlRoot = 0& kann man variieren, um einen anderen Ausgangsordner zu bestimmen.

Nachfolgende Prozedur erlaubt nun eine Ordnerauswahl

```
Sub WAEHLE_ORDNER()
Dim neuOrdner

neuOrdner = OrdnerAuswahl
If neuOrdner = "" Then
    Exit Sub
  Else
    ChDir neuOrdner
End If
MsgBox neuOrdner
End Sub
```

Ordner unsichtbar machen und wieder sichtbar machen

SetAttr pathname, attributes

z.B.

SetAttr "C:\Backup", vbHidden ' Den Ordner verbergen

SetAttr "C:\Backup", vbNormal ' Den Ordner sichtbar machen

Attributes:
vbNormal 0 Normal (Voreinstellung).
vbReadOnly 1 Schreibgeschützt.
vbHidden 2 Versteckt.

Ordner / Verzeichnisse auslesen

- Makro: Unterverzeichnis
Auslesen des eingegebenen Verzeichnisses einschließlich der Unterverzeichnisse,
Auflistung in einer neu eingefügten Tabelle
in Zeile 1 stehen die Verzeichnisse, ab Zeile 2 werden die Dateien mit Hyperlink aufgelistet;
maximal 256 Verzeichnisse

- Makro: Dateiname_Hyperlink
Auslesen des eingegebenen Verzeichnisses einschließlich der Unterverzeichnisse,
Auflistung in einer neu eingefügten Tabelle
Auflisten des Dateinamens mit Hyperlink, Dateigröße und Dateidatum in separaten Spalten

- Makro: Verzeichnisse_3
Auslesen von .XLS Dateine aus 3 im Code festgelegten Verzeichnissen;
bitte Ordner im Code anpassen

Auflistung in einer neu eingefügten Tabelle ohne Hyperlink, je Verzeichnis eine Spalte

erst ab Version **XP** (Abtrennen des Dateinamens)

- Makro: Dateiname_Pfad
Auslesen des eingegebenen Verzeichnisses einschließlich der Unterverzeichnisse,
Auflistung in einer neu eingefügten Tabelle

Auflisten des Dateinamens einschließlich Pfad ohne Hyperlink, Dateigröße und Dateidatum

' Bitte beachten das Modul Verzeichnisbaum gehört zu diesem Makro

```
Sub Dateiname_Hyperlink()
'************************************************
'* 07.10.04, 31.07.05; 02.08.05                 *
'* erstellt von Ramses Rainer                   *
'* Anpassungen von Hajo                         *
'* http://home.media-n.de/ziplies/              *
'************************************************
    Dim StDateiname As String
    Dim Dateiform As String
    Dim LoI As Long, TotFiles As Long
    Dim Suchpfad As String
    Dim OldStatus As Variant
    Suchpfad = GetAOrdner                    ' Verzeichnis auswählen
    If Suchpfad = "" Then Exit Sub
    Dateiform = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.xls")
    If Dateiform = "" Then Exit Sub
    Application.ScreenUpdating = False       ' Bildschirmaktulalisierung aus
    OldStatus = Application.StatusBar        ' Inhalt Statusleiste merken
'   neue Tabelle anlegen hinter der letzten Tabelle, Ergänzung Hajo
    Sheets.Add After:=Worksheets(Worksheets.Count)
'   ******
    With Application.FileSearch
       .LookIn = Suchpfad                    ' Suchverzeichnis
       .SearchSubFolders = True              ' suchen auch in Unterverzeichnissen
       .Filename = Dateiform                 ' Dateityp
       If .Execute() > 0 Then                ' Dateien im Verzeichnis vorhanden
'      Sortiert nach letzter Änderung absteigend
'       If .Execute(msoSortByLastModified, msoSortOrderDescending) > 0 Then
          TotFiles = .FoundFiles.Count       ' Anzahl der gefundenen Dateien
          Application.StatusBar = "Total " & TotFiles & " gefunden"
          For LoI = 1 To .FoundFiles.Count   ' Schleife über alle gefundenen Dateien
             Application.StatusBar = "Datei: " & LoI & " von " & TotFiles
''            Dateinamen abtrennen für Versionen vor XP
'             Dim LoK as Long
'             For LoK = Len(.FoundFiles(LoK)) To 1 Step -1
'                If Mid(.FoundFiles(LoK), LoK, 1) = "\" Then
'                   StDateiname = Mid(.FoundFiles(LoK), LoK + 1, Len(.FoundFiles(LoK)) - LoK + 2)
'                   Exit For
'                End If
'             Next LoK
'             Dateiname abtrennen ab XP
              StDateiname = Mid(.FoundFiles(LoI), InStrRev(.FoundFiles(LoI), "\") + 1)
'             ergänzt Hyperlink, Dateigröße und Dateidatum
              ActiveSheet.Hyperlinks.Add Anchor:=Cells(LoI, 1), _
```

```
            Address:=.FoundFiles(LoI), TextToDisplay:=StDateiname      ' Hyperlink
            Cells(LoI, 2) = FileLen(.FoundFiles(LoI))                  ' Dateigröße
            Cells(LoI, 3) = FileDateTime(.FoundFiles(LoI))             ' Dateidatum
'           ******
         Next LoI
      End If
   End With
   Columns("A:C").Columns.AutoFit          ' optimale Breit für Spalte A:C
   Application.StatusBar = OldStatus       ' Statuszeile zurücksetzen
   Application.ScreenUpdating = True       ' Bildschirmaktualisierung ein
End Sub


'  Bitte beachten das Modul Verzeichnisbaum gehört zu diesem Makro

Sub Dateiname_Pfad()
'************************************************
'* 07.10.04, 31.07.05; 02.08.05                 *
'* erstellt von Ramses Rainer                   *
'* Anpassungen von Hajo                         *
'* http://home.media-n.de/ziplies/              *
'************************************************
   Dim Dateiform As String
   Dim LoI As Long, TotFiles As Long
   Dim Suchpfad As String
   Dim OldStatus As Variant
   Suchpfad = GetAOrdner                   ' Verzeichnis auswählen
   If Suchpfad = "" Then Exit Sub
   Dateiform = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.xls")
   If Dateiform = "" Then Exit Sub
   Application.ScreenUpdating = False      ' Bildschirmaktulalisierung aus
   OldStatus = Application.StatusBar       ' Inhalt Statusleiste merken
'  neue Tabelle anlegen hinter der letzten Tabelle, Ergänzung Hajo
   Sheets.Add After:=Worksheets(Worksheets.Count)
'  ******
   With Application.FileSearch
      .LookIn = Suchpfad                   ' Suchverzeichnis
      .SearchSubFolders = True             ' suchen auch in Unterverzeichnissen
      .Filename = Dateiform                ' Dateityp
      If .Execute() > 0 Then               ' Dateien im Verzeichnis vorhanden
'     Sortiert nach letzter Änderung absteigend
'      If .Execute(msoSortByLastModified, msoSortOrderDescending) > 0 Then
         TotFiles = .FoundFiles.Count      ' Anzahl der gefundenen Dateien
         Application.StatusBar = "Total " & TotFiles & " gefunden"
```

```
        For LoI = 1 To .FoundFiles.Count    ' Schleife über alle gefundenen Dateien
           Application.StatusBar = "Datei: " & LoI & " von " & TotFiles
'          Dateiname in Zelle schreiben
           Cells(LoI, 1) = .FoundFiles(LoI)                ' Dateiname mit Pfad
'          ergänzt  Dateigröße und Dateidatum
           Cells(LoI, 2) = FileLen(.FoundFiles(LoI))       ' Dateigröße
           Cells(LoI, 3) = FileDateTime(.FoundFiles(LoI))  ' Dateidatum
'          *****
        Next LoI
      End If
   End With
   Columns("A:C").Columns.AutoFit          ' optimale Breit für Spalte A:C
   Application.StatusBar = OldStatus       ' Statuszeile zurücksetzen
   Application.ScreenUpdating = True       ' Bildschirmaktualisierung ein
End Sub


'  Bitte beachten das Modul Verzeichnisbaum gehört zu diesem Makro

Sub Unterverzeichnis()
'************************************************
'* 07.10.04, 31.07.05; 02.08.05                 *
'* erstellt von Ramses Rainer                   *
'* Anpassungen von Hajo                         *
'* http://home.media-n.de/ziplies/              *
'************************************************
   Dim Dateiform As String
   Dim J As Integer
   Dim Bereich As Range
   Dim Dateiname As String
   Dim I As Long, TotFiles As Long
   Dim Suchpfad As String
   Dim OldStatus As Variant
   Dim L As Integer
   J = 1
   Suchpfad = GetAOrdner                   ' Verzeichnis auswählen
   If Suchpfad = "" Then Exit Sub
   Dateiform = InputBox("Geben Sie den Dateityp an der gesucht werden soll", "Dateierweiterung", "*.*")
   If Dateiform = "" Then Exit Sub
   Application.ScreenUpdating = False      ' Bildschirmaktulalisierung aus
   OldStatus = Application.StatusBar       ' Inhalt Statusleiste merken
'  neue Tabelle anlegen hinter der letzten Tabelle, Ergänzung Hajo
   Sheets.Add After:=Worksheets(Worksheets.Count)
'  ******
```

```
    With Application.FileSearch
        .LookIn = Suchpfad                  ' Suchverzeichnis
        .SearchSubFolders = True              ' suchen auch in Unterverzeichnissen
        .Filename = Dateiform                ' Dateityp
        If .Execute() > 0 Then               ' Dateien im Verzeichnis vorhanden
'       Sortiert nach letzter Änderung absteigend
'        If .Execute(msoSortByLastModified, msoSortOrderDescending) > 0 Then
            TotFiles = .FoundFiles.Count        ' Anzahl der gefundenen Dateien
            Application.StatusBar = "Total " & TotFiles & " gefunden"
            For I = 1 To .FoundFiles.Count    ' Schleife über alle gefundenen Dateien
'               ergänzt für Unterverzeichnisse
'               festellen aller Unterverzeichnisse und in Zeile 1 schreiben
'               feststellen des Verzeichnisses
                For L = Len(.FoundFiles(I)) To 1 Step -1
                    If Mid(.FoundFiles(I), L, 1) = "\" Then Exit For
                Next L
                Set Bereich = ActiveSheet.Range("A1:IV256").Find(Mid(.FoundFiles(I), 1, L), lookat:=xlWhole)
                If Bereich Is Nothing Then
                    Cells(1, J) = Mid(.FoundFiles(I), 1, L)
                    J = J + 1
                    If J > 256 Then MsgBox "Es sind mehr als 256 Unterverzeichnisse": GoTo Ende
                End If
'               *****
            Next I
'           Dateien je Verzeichnis auslesen
            For I = 1 To Cells(1, Columns.Count).End(xlToLeft).Column
                Dateiname = Dir(Cells(1, I) & Dateiform)
                Do While Dateiname <> ""
'                   Veränderung Hyperlink
                    ActiveSheet.Hyperlinks.Add Anchor:=Cells(Cells(Rows.Count, I).End(xlUp).Row + 1, I), _
                        Address:=Cells(1, I) & Dateiname, TextToDisplay:=Dateiname
'                   *****
                    Dateiname = Dir
                Loop
            Next I
        End If
    End With
Ende:
    Columns("A:IV").Columns.AutoFit          ' optimale Breit für Spalte A:IV
    Application.StatusBar = OldStatus        ' Statuszeile zurücksetzen
    Application.ScreenUpdating = True        ' Bildschirmaktualisierung ein
End Sub
```

```
'   Bitte beachten das Modul Verzeichnisbaum gehört zu diesem Makro

Sub Verzeichnisse_3()
'************************************************
'* 07.10.04, 31.07.05; 02.08.05                 *
'* erstellt von Ramses Rainer                   *
'* Anpassungen von Hajo                         *
'* http://home.media-n.de/ziplies/              *
'************************************************
    Dim LoI As Long, LoJ As Long                 ' Schleifenvariable
    Dim ByI As Byte                              ' Schleife Verzeichnisse 0 bis 255 max.
    Dim OldStatus As Variant                     ' alter Zustand Statusbar
    Dim StVerzeichnis(2) As String               ' Array für 3 Verzeichnisse (0; 1; 2)
'   Verzeichnisse die ausgelesen werden sollen, bitte anpassen
    StVerzeichnis(0) = "L:\Eigene Dateien\Hajo"
    StVerzeichnis(1) = "L:\Eigene Dateien\Kornelia"
    StVerzeichnis(2) = "L:\Eigene Dateien\Klockmann"
    Application.ScreenUpdating = False           ' Bildschirmaktulalisierung aus
    OldStatus = Application.StatusBar            ' Inhalt Statusleiste merken
'   neue Tabelle anlegen hinter der letzten Tabelle, Ergänzung Hajo
    Sheets.Add After:=Worksheets(Worksheets.Count)
'   ******
    For ByI = 0 To 2                             ' Schleife über alle vorgegebenen Verzeichnisse
       Dim StDateien() As String                 ' Array hier definieren, damit es zu Beginn der Schleife geleert wird
       LoJ = 0                                   ' Schleifenvariable auf Null setzen
       With Application.FileSearch
          .LookIn = StVerzeichnis(ByI)           ' Suchpfad
          .SearchSubFolders = False              ' suchen nicht in Unterverzeichnissen
          .Filename = "*.xls"                    ' Dateityp nach dem gesucht wird
          If .Execute() > 0 Then                 ' Dateien im Verzeichnis vorhanden
'         Sortiert nach letzter Änderung absteigend
'         If .Execute(msoSortByLastModified, msoSortOrderDescending) > 0 Then
             For LoI = 1 To .FoundFiles.Count    ' Schleife über alle gefundenen Dateien
                Application.StatusBar = "Datei: " & LoI & " von " & .FoundFiles.Count
                LoJ = LoJ + 1
                ReDim Preserve StDateien(1 To LoJ)
''               Dateinamen abtrennen für Versionen vor XP
'                 Dim LoK as Long
'                For LoK = Len(.FoundFiles(LoJ)) To 1 Step -1
'                   If Mid(.FoundFiles(LoJ), LoK, 1) = "\" Then
'                      StDateiname = Mid(.FoundFiles(LoJ), LoK + 1, Len(.FoundFiles(LoJ)) - LoK + 2)
'                      Exit For
'                   End If
'                Next LoK
```

```
'                Dateiname abtrennen ab Version XP und auf Array schreiben
                 StDateien(LoI) = Mid(.FoundFiles(LoI), InStrRev(.FoundFiles(LoI), "\") + 1)
              Next LoI
'             Array in Tabelle schreiben
              Cells(1, ByI + 1) = StVerzeichnis(ByI)
              If UBound(StDateien()) = 1 Then
                 Range(Cells(2, ByI + 1), Cells(2, ByI + 1)) = _
                    WorksheetFunction.Transpose(StDateien())
              Else
                 Range(Cells(2, ByI + 1), Cells(UBound(StDateien()), ByI + 1)) = _
                    WorksheetFunction.Transpose(StDateien())
              End If
           End If
        End With
    Next ByI
    Columns("A:C").Columns.AutoFit               ' optimale Breit für Spalte A:C
    Application.StatusBar = OldStatus            ' Statuszeile zurücksetzen
    Application.ScreenUpdating = True            ' Bildschirmaktualisierung ein
End Sub


' MODUL VERZEICHNISBAUM

Private Type InfoT
    hwnd As Long
    Root As Long
    DisplayName As Long
    Title As Long
    Flags As Long
    FName As Long
    lParam As Long
    Image As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32" (lpbi As InfoT) As Long
Private Declare Function CoTaskMemFree Lib "ole32" (ByVal hMem As Long) As Long
Private Declare Function lstrcat Lib "kernel32" Alias "lstrcatA" (ByVal lpStr1 As String, ByVal lpStr2 As String) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" (ByVal pList As Long, ByVal lpBuffer As String) As Long
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassname As String, ByVal lpWindowName As String) As Long

Function GetAOrdner() As String
    Dim xl As InfoT, IDList As Long, RVal As Long, FolderName As String
    With xl
       .hwnd = FindWindow("xlmain", vbNullString)
```

```
'       .hwnd = FindWindow("", "Auswahl")  ' Userform Auswahl
        .Title = lstrcat("Bitte wählen Sie ein Verzeichnis", "")
        .Flags = 1
    End With
    IDList = SHBrowseForFolder(xl)
    If IDList <> 0 Then
        FolderName = Space(256)
        RVal = SHGetPathFromIDList(IDList, FolderName)
        CoTaskMemFree (IDList)
        FolderName = Trim(FolderName)
        FolderName = Left(FolderName, Len(FolderName) - 1)
    End If
    GetAOrdner = FolderName
End Function
```

Ordner im Explorer öffnen

```
target = "D:\temp"
    Shell "explorer.exe " & target, vbNormalFocus
```

TABELLENBLATT AUTOMATISCH BEIM AKTIVIEREN AUF BESTIMMTE SPALTEN ZOOMEN

```
Private Sub Worksheet_Activate()

    Dim Adresse

    Adresse = ActiveCell.Address

    ActiveWindow.Zoom = 100 ' wichtig: denn wenn die Zoomeinstellung nicht zu ändern ist (weil schon früher korrekt geändert),
                            ' wird auch nicht auf die Spalten gescrollt
    Columns("C:Q").Select
    ActiveWindow.Zoom = True
    ActiveWindow.Zoom = ActiveWindow.Zoom

    Range(Adresse).Select

End Sub
```

Prüfen ob es eine Datei / Verzeichnis wirklich gibt

Simple Variante

Am einfachsten prüft man das Vorhandensein einer Datei mit diesem Befehl

If Dir("C:\Ordner\Datei.xyz")<>"" then MsgBox "Datei existiert !"

Prüfen ob Datei oder Ordner existiert

One function to check both file and folder if they exist...

```
Public Function FileFolderExists(strFullPath As String) As Boolean
On Error GoTo EarlyExit
If Not Dir(strFullPath, vbDirectory) = vbNullString Then FileFolderExists = True
EarlyExit:
On Error GoTo 0
End Function
```

Prüfen ob Ordner existiert

```
    If Dir(PFADAKTUELL & "\Backup", vbDirectory) = "" Then
        MkDir (PFADAKTUELL & "\Backup")
    End If
```

Variante 4 klappt auch bei Verzeichnissen

VARIANTE 0 für DATEIEN

```
'  Title: Determines whether the specified file exists
'


' ----------------------------------------------------------------------------

'-------------------------------------------------------
' FUNCTION: FileExists
' Determines whether the specified file exists
'
```

```
' IN: [strPathName] - file to check for
'
' Returns: True if file exists, False otherwise
'-------------------------------------------------------------
'
Function FileExists(ByVal strPathName As String) As Integer
    Dim intFileNum As Integer

    On Error Resume Next

    ' If the string is quoted, remove the quotes.
    strPathName = strUnQuoteString(strPathName)

    'Remove any trailing directory separator character
    If Right$(strPathName, 1) = gstrSEP_DIR Then
        strPathName = Left$(strPathName, Len(strPathName) - 1)
    End If

    'Attempt to open the file, return value of this function is False
    'if an error occurs on open, True otherwise
    intFileNum = FreeFile
    Open strPathName For Input As intFileNum
    FileExists = IIf(Err = 0, True, False)
    Close intFileNum

    Err = 0
End Function
```

Variante 0 für Ordner

```
'  Title: Determines whether the specified directory name exist
'



' ------------------------------------------------------------------------------

'-------------------------------------------------------------
' FUNCTION: DirExists
'
' Determines whether the specified directory name exists.
' This function is used (for example) to determine whether
' an installation floppy is in the drive by passing in
' something like 'A:\'.
```

```
'
' IN: [strDirName] - name of directory to check for
'
' Returns: True if the directory exists, False otherwise
'-------------------------------------------------------
'
Public Function DirExists(ByVal strDirName As String) As Integer
    Const strWILDCARD$ = "*.*"

    Dim strDummy As String

    On Error Resume Next

    AddDirSep strDirName
    strDummy = Dir$(strDirName & strWILDCARD, vbDirectory)
    DirExists = Not (strDummy = vbNullString)

    Err = 0
End Function
```

Variante 1

```
If Dir("C:\test.txt") = "" Then
       MsgBox "gibts nicht"
     Else
       MsgBox "gibts"
End If
```

Variante 2

```
If Dir("C:\Ordner\Datei.xyz")<>"" then MsgBox "Datei existiert !"
```

Variante 3

Auf die Existenz einer Datei prüfen

Dazu gibt es vielerlei Möglichkeiten, mein Favorit ist aber immer noch das gute alte Dir.
Bei vielen Beispielen wird aber einfach vernachlässigt, dass eine Datei durchaus existieren kann,
aber z.B. der Server nicht zur Verfügung steht (oder die Diskette nicht eingelegt ist oder oder...)

Im nachfolgendem Code wird False(0) zurückgegeben, wenn die Datei existiert, ansonsten eine 1 für nicht da an der angegebenen Adresse und eine 2 für andere Probleme (in der Regel Laufwerksprobleme)

Es lohnt sich, den Code zu studieren.

(Warum len(Dir) - Ganz einfach: Dies ist schneller als andere Konstrukte)

```
Option Explicit

Public Sub TestePfad()
        Dim sPfad As String, retVal As Byte
        sPfad = "C:\Testordner\Test.xls" ' Pfad ändern für Tests
        retVal = DateNichtDa(sPfad)
        If retVal Then
                MsgBox IIf(retVal = 1, "Datei nicht da", "Vermutlich Laufwerkfehler")
        Else
                MsgBox "Ist Da"
        End If
End Sub

Private Function DateNichtDa(DerPfad As String) As Byte
        ' Peter Haserodt
        On Error GoTo PfadError
        DateNichtDa = IIf(Len(Dir(DerPfad)) > 1, 0, 1)
        Exit Function
        PfadError:
        DateNichtDa = 2
End Function
```

Variante 4

PRÜFEN OB ES EIN VERZEICHNIS GIBT

mit "VBA-Hausmitteln" hilft mir diese Funktion, festzustellen, ob ein Verzeichnis oder Datei existiert:
Code:

```
Public Function PathExists(ByVal strPath As String) As Boolean
  On Error Resume Next
  GetAttr strPath
  PathExists = (Err = 0)
End Function

Sub ORDNER

   If PathExists("C:\Temp") = False Then
```

```
    MkDir "C:\Temp"
  End If

End Sub
```

VARIANTE 5

```
Function FileExists(sFilePath As String) As Boolean

  If Trim(sFilePath) = "" Then Exit Function
  If Right(sFilePath, 1) = "\" Then Exit Function
'// -------------------------------------------------------------------
'// Fehlerhandling einschalten, um VB-Meldung abzufangen
'// -------------------------------------------------------------------
  On Error Resume Next
  FileExists = Dir(sFilePath) <> ""
  FileExists = FileExists And Err.Number = 0
'// -------------------------------------------------------------------
'// Fehlerhandling wieder auschalten
'// -------------------------------------------------------------------
  On Error GoTo 0

End Function

Sub Test()

  Const csMYPATH = "C:\Programme\Microsoft Office\Office10\WINWORD.EXE"
  Debug.Print FileExists(csMYPATH)

End Sub
```

Variante 6

```
Function FileExist(Filename As String) As Boolean
On Error GoTo HandleError
FileExist = False
If Len(Filename) > 0 Then FileExist = (Dir(Filename) <> "")
Exit Function
HandleError:
FileExist = False
If (Err = 1005) Then
```

```
MsgBox "Error - printer missing"
Resume Next
Else
If (Err = 68) Or (Err = 76) Then
MsgBox "Unit or Path do not exist: " & Filename, vbExclamation
Resume Next
Else
MsgBox "Unexpected error " & Str(Err) & " : " &
Error(Err), vbCritical
End
End If
End If
End Function
```

Prüfen ob eine Excel-Datei bereits geöffnet ist

Version 1

```
Public Function IsWorkbookOpen(strWB As String) As Boolean
  On Error Resume Next
  IsWorkbookOpen = Not Workbooks(strWB) Is Nothing
End Function

Sub test()
  If IsWorkbookOpen("Mappe2.xls") Then
    MsgBox "OFFEN"
  Else
    MsgBox "Nicht offen"
  End If
End Sub
```

Version 2

```
Public Function MappeOp(strName As String) As Boolean

Dim Mappe As Workbook

MappeOp = False

For Each Mappe In Application.Workbooks
        If Mappe.Name = strName Then

                MappeOp = True
```

```
                Exit Function

        End If
Next Mappe

End Function


Sub MappeOffen()
        If MappeOp("test.xlsm") = True Then
                MsgBox "Die Mappe ist geöffnet"
          Else
                MsgBox "Die Mappe ist geschlossen"
        End If
End Sub
```

Version 3

Schreibschutz Auslesen (Mappenschutz)

Msgbox Activeworkbook.Readonly

Sicherungskopie der aktuellen Arbeitsmappe automatisch erstellen

Siehe Bereich ARBEITSMAPPEN

Speicherpfad der aktuellen Arbeitsmappe und Speichern

Den aktuellen Speicherpfad der aktuellen Datei erfährt man mit

msgbox ThisWorkbook.Path

Speichern kann man mit

```
ActiveWorkbook.SaveAs "C:\Programme\Test.xls"
```

Speicherpfad von User auswählen lassen

```
Public Type BROWSEINFO
hOwner As Long
pidlRoot As Long
pszDisplayName As String
lpszTitle As String
ulFlags As Long
lpfn As Long
lParam As Long
iImage As Long
End Type
'32-bit API-Deklarationen
Declare Function SHGetPathFromIDList Lib "shell32.dll" _
Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As
String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" _
Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long
Sub DirAuswahl()
Dim msg As String
msg = "Wählen Sie bitte einen Ordner aus:"
MsgBox getdirectory(msg)
End Sub
Function getdirectory(Optional msg) As String
Dim bInfo As BROWSEINFO
Dim Path As String
Dim r As Long, x As Long, pos As Integer
' Ausgangsordner = Desktop
bInfo.pidlRoot = 0&
' Dialogtitel
If IsMissing(msg) Then
bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
Else
bInfo.lpszTitle = msg
End If
' Rückgabe des Unterverzeichnisses
bInfo.ulFlags = &h1
' Dialog anzeigen
x = SHBrowseForFolder(bInfo)
' Ergebnis gliedern
Path = Space$(512)
r = SHGetPathFromIDList(ByVal x, ByVal Path)
If r Then
```

```
pos = InStr(Path, Chr$(0))
getdirectory = Left(Path, pos - 1)
Else
getdirectory = ""
End If
End Function
```

Speicherpfad von User auswählen lassen

```
Public Type BROWSEINFO
      hOwner As Long
      pidlRoot As Long
      pszDisplayName As String
      lpszTitle As String
      ulFlags As Long
      lpfn As Long
      lParam As Long
      iImage As Long
End Type
'32-bit API-Deklarationen
Declare Function SHGetPathFromIDList Lib "shell32.dll" _
      Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As
String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" _
   Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long
Sub DirAuswahl()
      Dim msg As String
      msg = "Wählen Sie bitte einen Ordner aus:"
      MsgBox getdirectory(msg)
End Sub
Function getdirectory(Optional msg) As String
      Dim bInfo As BROWSEINFO
      Dim Path As String
      Dim r As Long, x As Long, pos As Integer
'    Ausgangsordner = Desktop
      bInfo.pidlRoot = 0&
'    Dialogtitel
      If IsMissing(msg) Then
            bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
      Else
            bInfo.lpszTitle = msg
      End If
'    Rückgabe des Unterverzeichnisses
      bInfo.ulFlags = &h1
'    Dialog anzeigen
      x = SHBrowseForFolder(bInfo)
'    Ergebnis gliedern
      Path = Space$(512)
      r = SHGetPathFromIDList(ByVal x, ByVal Path)
      If r Then
            pos = InStr(Path, Chr$(0))
            getdirectory = Left(Path, pos - 1)
```

```
        Else
            getdirectory = ""
        End If
    End Function
```

Gleiche Version - aber ohne Einrückung

```
Public Type BROWSEINFO
hOwner As Long
pidlRoot As Long
pszDisplayName As String
lpszTitle As String
ulFlags As Long
lpfn As Long
lParam As Long
iImage As Long
End Type
’32-bit API-Deklarationen
Declare Function SHGetPathFromIDList Lib "shell32.dll" _
Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As
String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" _
Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long
Sub DirAuswahl()
Dim msg As String
msg = "Wählen Sie bitte einen Ordner aus:"
MsgBox getdirectory(msg)
End Sub
Function getdirectory(Optional msg) As String
Dim bInfo As BROWSEINFO
Dim Path As String
Dim r As Long, x As Long, pos As Integer
’ Ausgangsordner = Desktop
bInfo.pidlRoot = 0&
’ Dialogtitel
If IsMissing(msg) Then
bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
Else
bInfo.lpszTitle = msg
End If
’ Rückgabe des Unterverzeichnisses
bInfo.ulFlags = &h1
’ Dialog anzeigen
```

```
x = SHBrowseForFolder(bInfo)
’ Ergebnis gliedern
Path = Space$(512)
r = SHGetPathFromIDList(ByVal x, ByVal Path)
If r Then
pos = InStr(Path, Chr$(0))
getdirectory = Left(Path, pos - 1)
Else
getdirectory = ""
End If
End Function
```

Speicherpfad voreinstellen

Will man, dass bei einem "Speichern-Unter"-Dialog man bereits in einem bestimmten Ordner ist, dann braucht man

ChDrive "D" ' Auf Laufwerk D: wechseln.
und
ChDir "D:\VERZ1" ' Aktuelles Verzeichnis oder aktuellen Ordner auf "VERZ1" ändern.

Excel unterscheidet in 2 Ebenen: aktuelles Laufwerk und aktueller Pfad für das jeweilige Laufwerk.

Die ChDir-Anweisung wechselt das Standardverzeichnis, aber nicht das Standardlaufwerk. Ist das Standardlaufwerk zum Beispiel C:, dann wechselt die folgende Anweisung zwar das Standardverzeichnis auf Laufwerk D:, aber C: bleibt das Standardlaufwerk:

ChDir "D:\TMP"

Speichern unter - Dialogfeld aufrufen

1. Möglichkeit:

```
Sub test()
UsrRsp = Application.Dialogs(xlDialogSaveAs).Show("test.xls")
Select Case UsrRsp
Case -1 'Gesichert
Case 0 'Abgebrochen
End Select
End Sub
```

2. Möglichkeit:

```
Sub Speichern()
test = Application.GetSaveAsFilename([a1])
If test = False Then Exit Sub
ActiveWorkbook.SaveAs test
End Sub
```

3. Möglichkeit: nur das reine Dialogfenster öffnen

```
Application.Dialogs(xlDialogSaveAs).Show
```

4. Möglichkeit: das Dialogfenster öffnen und Dateiname vorschlagen

```
Application.Dialogs(xlDialogSaveAs).Show ("Standardname")
```

5. Möglichkeit: einen möglichen Speicherpfad und Speichername vorschlagen

```
Public Sub Speichern()
Dim strVerzeichnis As String
Dim strDateiname As String
Dim Dateivorschlag as string
strVerzeichnis = "C:\Temp\"
Dateivorschlag="Demodatei"
strDateiname = Application.GetSaveAsFilename(InitialFileName:=strVerzeichnis & dateivorschlag & _
  ".xls", FileFilter:="Microsoft Excel-Arbeitsmappe (*.xls), *.xls")
  Select Case strDateiname
    Case False
      Exit Sub
    Case Else
      ThisWorkbook.SaveAs Filename:=strDateiname
  End Select
End Sub
```

6. Möglichkeit: einen möglichen Speicherpfad und Speichername vorschlagen

```
Private Sub CommandButton1_Click()

Dim strDateiname As String
ChDrive "c:\"
ChDir "\Programme\"

strDateiname = "Demodatei.xls"
Application.Dialogs(xlDialogSaveAs).Show (strDateiname)
```

End Sub

Speichern von einzelnen Tabellenblättern in neuer Datei

```
Sub Exportieren()

' eine variable Anzahl (in unserem BSP max 6) von Tabellen soll in einer eigenen, neuen Excel-Tabelle gespeichert werden

Dim NEUERNAME
Dim EXPORTTABELLEN(6) As String
Dim T As Integer
Dim WIEVIELTETABELLE As Integer
Dim SCHLIESSEN As Integer


    ' Auslesen, ob nach Export die Datei geschlossen werden soll

    If Range("F22") <> "" Then SCHLIESSEN = 1

    ' Auslesen des Dateinamens, unter dem gespeichert werden soll
    NEUERNAME = "Reporting " & Range("F11") & " - " & Date & ".xls"

    Application.ScreenUpdating = False

    ' Auslesen der zu exportierenden Tabellen (maximal 6 - ihre Daten sind in den Zeilen 14-19)
    ' In den Zellen N14-N19 ist ein X enthalten, wenn die betreffende Tabelle exportiert werden soll
    ' In den Zellen E14-E19 sind die Namen der Tabellen
    For T = 0 To 5 ' Schleife durch die 6 Tabellen
       If Range("N" & (14 + T)) <> "" Then
          EXPORTTABELLEN(WIEVIELTETABELLE) = Range("E" & (14 + T))
          WIEVIELTETABELLE = WIEVIELTETABELLE + 1
       End If
    Next T

    Select Case WIEVIELTETABELLE
       Case 0
```

```
            MsgBox "In den Zellen N14 bis N19 sind keine Tabellen für den Export vorgemerkt."
            End
        Case 1
            Sheets(Array(EXPORTTABELLEN(0))).Copy
        Case 2
            Sheets(Array(EXPORTTABELLEN(0), EXPORTTABELLEN(1))).Copy
        Case 3
            Sheets(Array(EXPORTTABELLEN(0), EXPORTTABELLEN(1), EXPORTTABELLEN(2))).Copy
        Case 4
            Sheets(Array(EXPORTTABELLEN(0), EXPORTTABELLEN(1), EXPORTTABELLEN(2), EXPORTTABELLEN(3))).Copy
        Case 5
            Sheets(Array(EXPORTTABELLEN(0), EXPORTTABELLEN(1), EXPORTTABELLEN(2), EXPORTTABELLEN(3), EXPORTTABELLEN(4))).Copy
        Case 6
            Sheets(Array(EXPORTTABELLEN(0), EXPORTTABELLEN(1), EXPORTTABELLEN(2), EXPORTTABELLEN(3), EXPORTTABELLEN(4),
EXPORTTABELLEN(5))).Copy
    End Select

    ' den Speichern-Unterdialog anbieten zum auswählen
    Application.Dialogs(xlDialogSaveAs).Show (NEUERNAME)
    Application.ScreenUpdating = True
    ' wenn gewünscht, anschließend die neu exportierte und gespeicherte Datei wieder schließen
    If SCHLIESSEN = 1 Then ActiveWorkbook.Close ' wenn in Zelle F22 eingestellt ist, dass nach dem Export die neue Datei geschlossen werden soll
    Application.ScreenUpdating = True

End Sub
```

Speichern in Excel 2007

You see a lot of old SaveAs code that does not specify the FileFormat parameter. In Excel versions before Excel 2007, code without this parameter will not cause too many problems because Excel will use the current FileFormat of the existing file -- and the default FileFormat for new files is a normal workbook.

But because there are so many new file formats in Excel 2007, we shouldn't use code like this that does not specify the FileFormat parameter.

In Excel 2007, SaveAs requires you to provide both the FileFormat parameter and the correct file extension.

For example, in Excel 2007, this will **fail** if the ActiveWorkbook is not an **xlsm** file
ActiveWorkbook.SaveAs "C:\ron.xlsm"

This code will always work
ActiveWorkbook.SaveAs "C:\ron.xlsm", fileformat:=52
' 52 = xlOpenXMLWorkbookMacroEnabled = xlsm (with macro's in 2007)

These are the main file formats in Excel 2007:

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

Note: I always use the FileFormat numbers instead of the defined constants in my code so that it will compile OK when I copy the code into an Excel 97-2003 workbook. (For example, Excel 97-2003 won't know what the xlOpenXMLWorkbookMacroEnabled constant is.)

Examples

Below are two basic code examples to copy the ActiveSheet to a new Workbook and save it in a format that matches the file extension of the parent workbook. The second example use GetSaveAsFilename to ask you for a file path/name. (Example 1 you can use in Excel 97-2007 , Example 2 you can use in Excel 2000-2007)

If you run the code in Excel 2007 it will look at the FileFormat of the parent workbook and save the new file in that format. Only if the parent workbook is an xlsm file and if there is no VBA code in the new workbook it will save the new file as xlsx.

If the parent workbook is not an xlsx, xlsm or xls then it will be saved as xlsb.

If you always want to save in a certain format you can replace this part of the macro

```
 Select Case Sourcewb.FileFormat
Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
Case 52:
    If .HasVBProject Then
        FileExtStr = ".xlsm": FileFormatNum = 52
    Else
        FileExtStr = ".xlsx": FileFormatNum = 51
    End If
Case 56: FileExtStr = ".xls": FileFormatNum = 56
Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
End Select
```

With one of the one liners from this list

```
FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56
```

Or maybe you want to save the one sheet workbook to csv, txt or prn.
(you can use this also if you run the code in Excel 97-2003)

```
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
```

Examples

```
Sub Copy ActiveSheet 1()
'Working in Excel 97-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
```

```
Dim Sourcewb As Workbook
Dim Destwb As Workbook
Dim TempFilePath As String
Dim TempFileName As String

With Application
    .ScreenUpdating = False
    .EnableEvents = False
End With

Set Sourcewb = ActiveWorkbook

'Copy the sheet to a new workbook
ActiveSheet.Copy
Set Destwb = ActiveWorkbook

'Determine the Excel version and file extension/format
With Destwb
    If Val(Application.Version) < 12 Then
        'You use Excel 97-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007
        'We exit the sub when your answer is NO in the security dialog that you
        'only see when you copy a sheet from a xlsm file with macro's disabled.
        If Sourcewb.Name = .Name Then
            With Application
                .ScreenUpdating = True
                .EnableEvents = True
            End With
            MsgBox "Your answer is NO in the security dialog"
            Exit Sub
        Else
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
                End If
            Case 56: FileExtStr = ".xls": FileFormatNum = 56
```

```
                Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
                End Select
            End If
        End If
    End With

    '    'Change all cells in the worksheet to values if you want
    '    With Destwb.Sheets(1).UsedRange
    '        .Cells.Copy
    '        .Cells.PasteSpecial xlPasteValues
    '        .Cells(1).Select
    '    End With
    '    Application.CutCopyMode = False

    'Save the new workbook and close it
    TempFilePath = Application.DefaultFilePath & "\"
    TempFileName = "Part of " & Sourcewb.Name & " " & Format(Now, "yyyy-mm-dd hh-mm-ss")

    With Destwb
        .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
        .Close SaveChanges:=False
    End With

    MsgBox "You can find the new file in " & TempFilePath

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub



Sub Copy_ActiveSheet_2()
'Working in Excel 2000-2007
    Dim fname As Variant
    Dim NewWb As Workbook
    Dim FileFormatValue As Long

    'Check the Excel version
    If Val(Application.Version) < 9 Then Exit Sub
    If Val(Application.Version) < 12 Then
```

```
    'Only choice in the "Save as type" dropdown is Excel files(xls)
    'because the Excel version is 2000-2003
    fname = Application.GetSaveAsFilename(InitialFileName:="", _
    filefilter:="Excel Files (*.xls), *.xls", _
    Title:="This example copies the ActiveSheet to a new workbook")

    If fname <> False Then
        'Copy the ActiveSheet to new workbook
        ActiveSheet.Copy
        Set NewWb = ActiveWorkbook

        'We use the 2000-2003 format xlWorkbookNormal here to save as xls
        NewWb.SaveAs fname, FileFormat:=-4143, CreateBackup:=False
        NewWb.Close False
        Set NewWb = Nothing

    End If
Else
    'Give the user the choice to save in 2000-2003 format or in one of the
    'new formats. Use the "Save as type" dropdown to make a choice,Default =
    'Excel Macro Enabled Workbook. You can add or remove formats to/from the list

    fname = Application.GetSaveAsFilename(InitialFileName:="", filefilter:= _
    " Excel Macro Free Workbook (*.xlsx), *.xlsx," & _
    " Excel Macro Enabled Workbook (*.xlsm), *.xlsm," & _
    " Excel 2000-2003 Workbook (*.xls), *.xls," & _
    " Excel Binary Workbook (*.xlsb), *.xlsb", _
    FilterIndex:=2, Title:="This example copies the ActiveSheet to a new workbook")

    'Find the correct FileFormat that match the choice in the "Save as type" list
    If fname <> False Then
        Select Case LCase(Right(fname, Len(fname) - InStrRev(fname, ".", , 1)))
        Case "xls": FileFormatValue = 56
        Case "xlsx": FileFormatValue = 51
        Case "xlsm": FileFormatValue = 52
        Case "xlsb": FileFormatValue = 50
        Case Else: FileFormatValue = 0
        End Select

        'Now we can create/Save the file with the xlFileFormat parameter
        'value that match the file extension
```

```
            If FileFormatValue = 0 Then
                MsgBox "Sorry, unknown file extension"
            Else
                'Copies the ActiveSheet to new workbook
                ActiveSheet.Copy
                Set NewWb = ActiveWorkbook

                'Save the file in the format you choose in the "Save as type" dropdown
                NewWb.SaveAs fname, FileFormat:= _
                             FileFormatValue, CreateBackup:=False
                NewWb.Close False
                Set NewWb = Nothing

            End If
        End If
    End If
End Sub
```

Suchbegriff in allen Mappen eines definierten Verzeichnisses suchen

```
Sub MultiSeek_in_Folder()
'By Ramses
'Durchsucht in einem Verzeichnis alle Mappen nach einem Suchbegriff
Dim Suchpfad As String, findStr As String, Dateiform As String, msgTxt As String
Dim Qe As Integer, myMatch As String, sAddress As String
Dim wks As Worksheet, wb As Workbook
Dim myRng As Range, totFiles As Integer, i As Integer, gefFile As Variant
Dim oldStatus As Variant
'Variablen füllen
Dateiform = "*.xls"
Suchpfad = InputBox("Geben Sie den Ordner an, der durchsucht werden soll:", "Pfad definieren", Application.DefaultFilePath)
If Suchpfad = "" Then Exit Sub
findStr = InputBox("Geben Sie den Text an der gesucht werden soll", "Textteil", "Suchtext")
If findStr = "" Then Exit Sub
msgTxt = "Soll auf exakte Übereinstimmung mit dem Fragment gesucht werden ? "
msgTxt = msgTxt & vbCrLf & "Bei ""Nein"" werden als Ergebnisse angezeigt,"
msgTxt = msgTxt & vbCrLf & "bei denen nur ein Teil des Textes mit:"" " & findStr & " "" übereinstimmt !"
Qe = MsgBox(msgTxt, vbQuestion + vbYesNo, "Suchroutine")
If Qe = vbOK Then
   myMatch = xlWhole
Else
   myMatch = xlPart
```

```
End If
'Bildschirmaktualisierung abschalten
Application.ScreenUpdating = True 'Nicht ausschalten = True
'Text der Statusbar und alten Status aufnehmen
oldStatus = Application.StatusBar
'Start der Suchroutine
With Application.FileSearch
    .NewSearch
    .LookIn = Suchpfad
    .FileName = Dateiform
    If .Execute() > 0 Then
        totFiles = .FoundFiles.count
        'Ausgabe in Statusbar
        Application.StatusBar = "Total " & totFiles & " gefunden"
        For i = 1 To .FoundFiles.count
            gefFile = .FoundFiles(i)
            Set wb = Application.Workbooks.Open(gefFile)
            Application.StatusBar = "Datei " & i & " von " & totFiles & " wird bearbeitet"
            For Each wks In wb.Worksheets
                Set myRng = wks.Cells.Find(What:=findStr, _
                        LookAt:=myMatch, LookIn:=xlFormulas)
                If Not myRng Is Nothing Then
                    sAddress = myRng.Address
                    Do
                        Application.GoTo myRng, True
                        'Für die Automation kann die "If"-Anweisung auskommentiert werden
                        If MsgBox("Weiter suchen", vbYesNo + vbQuestion) = vbNo Then
                            GoTo exitsearch
                        End If
                        '--
                        Set myRng = Cells.FindNext(after:=ActiveCell)
                    Loop
                End If
            Next
            wb.Close False
            Set wb = Nothing
        Next i
    End If
    'Exitfor:
End With
MsgBox prompt:="Keine neue Fundstelle!"
exitsearch:
Application.StatusBar = oldStatus
Application.ScreenUpdating = True
```

```
End Sub
```

Systempfade (Windows...) siehe System

Übergeordneten Ordner der aktuellen Datei

Beispiel: Vom Ordnerpfad der Excelarbeitsmappe, die den VBA-Code enthält (daher ThisWorkbook und nicht ActiveWorkbook) soll der übergeordnete Ordnerpfad herausgefunden werden:

```
PFAD = ThisWorkbook.Path ' ergibt zB: C:\OP_Lexor\Schnittstelle\Unterordner
MsgBox Left(PFAD, InStrRev(PFAD, "\")-1) ' ergibt C:\OP_Lexor\Schnittstelle
```

Überprüfen, ob eine Datei/Ordner existiert bzw offen ist

Variante 1

Nach der Variablendefinition einer SUB-Prozedur folgt dieser Code:

```
Set DATEISUCHE= CreateObject("Scripting.FileSystemObject")
      DATEI="C:\TEST.CSV"
   If DATEISUCHE.FileExists(DATEI) = False Then
      MsgBox "Die folgende Datei konnte nicht gefunden werden" & vbCrLf & vbCrLf & _
      "Stellen Sie sicher, dass die folgende Datei im entsprechenden Ordner existiert: " & DATEI
   End If
```

Variante 2

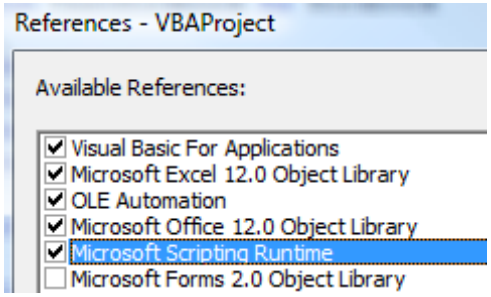
On this page, you will find code to check if a folder, file or Sheet exists and code to test if a workbook is open. In every example, I use a MsgBox to tell you if the folder, file or sheet exists or not. Replace the MsgBox for the code that you want to run when the folder, file or sheet exists or not. If you want to test the code you must copy the code and functions in a standard module of your workbook's project.

See this page if you just started with VBA.
http://www.rondebruin.nl/code.htm

Test if Folder exist

Below are three examples to test if the folder "test" exist
FolderPath = "C:\Users\Ron\test"

The first one uses VBA Dir and the other two use FSO (FileSystemObject).
Note read the information in the second FSO example good because you must set a reference
to the "Microsoft Scripting Runtime" in the VBA editor if you want to use this example.



When you use the Intellisense help you see that there are a lot more nice options when you use FSO.

I use it for example on this page
http://www.rondebruin.nl/folder.htm

```
Sub Test_Folder_Exist_With_Dir()
    Dim FolderPath As String
    Dim TestStr As String

    FolderPath = "C:\Users\Ron\test"
    If Right(FolderPath, 1) <> "\" Then
        FolderPath = FolderPath & "\"
    End If

    TestStr = ""
    On Error Resume Next
```

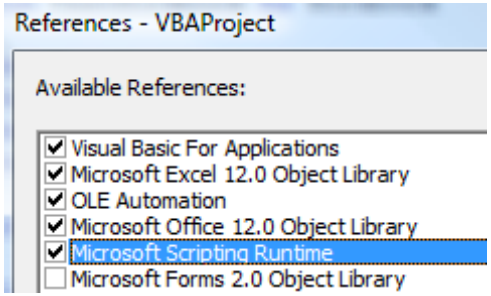
```
    TestStr = Dir(FolderPath)
    On Error GoTo 0
    If TestStr = "" Then
        MsgBox "Folder doesn't exist"
    Else
        MsgBox "Folder exist"
    End If

End Sub


Sub Test_Folder_Exist_FSO_Late_binding()
'No need to set a reference if you use Late binding
    Dim FSO As Object
    Dim FolderPath As String

    Set FSO = CreateObject("scripting.filesystemobject")

    FolderPath = "C:\Users\Ron\test"
    If Right(FolderPath, 1) <> "\" Then
        FolderPath = FolderPath & "\"
    End If

    If FSO.FolderExists(FolderPath) = False Then
        MsgBox "Folder doesn't exist"
    Else
        MsgBox "Folder exist"
    End If

End Sub


Sub Test_Folder_Exist_FSO_Early_binding()
'If you want to use the Intellisense help showing you the properties
'and methods of the objects as you type you can use Early binding.
'Add a reference to "Microsoft Scripting Runtime" in the VBA editor
'(Tools>References)if you want that.

    Dim FSO As Scripting.FileSystemObject
    Dim FolderPath As String

    Set FSO = New Scripting.FileSystemObject

    FolderPath = "C:\Users\Ron\test"
```

```
    If Right(FolderPath, 1) <> "\" Then
        FolderPath = FolderPath & "\"
    End If

    If FSO.FolderExists(FolderPath) = False Then
        MsgBox "Folder doesn't exist"
    Else
        MsgBox "Folder exist"
    End If

End Sub
```

Test if File exist

Below are three examples to test if the file "book1.xlsm" exists
FilePath = "C:\Users\Ron\test\book1.xlsm"

The first one uses VBA Dir and the other two use FSO (FileSystemObject).
Note read the information in the second FSO example good because you must set a reference
to the "Microsoft Scripting Runtime" in the VBA editor if you want to use this example.



When you use the the Intellisense help you see that there are a lot more nice options when you use FSO.

I use it for example on this page
http://www.rondebruin.nl/folder.htm

Extension

Be sure you enter the correct extension in the FilePath.
If you not see the extensions of your files in Windows Explorer you can do this to see them.

1: Open Windows Explorer
2: Win XP : Click on Tools>Folder Options
2: Vista : Click on Organize>Folder and Search Options
3: On the View tab uncheck “Hide extensions for known file types"

```
Sub Test_File_Exist_With_Dir()
    Dim FilePath As String
    Dim TestStr As String

    FilePath = "C:\Users\Ron\test\book1.xlsm"

    TestStr = ""
    On Error Resume Next
    TestStr = Dir(FilePath)
    On Error GoTo 0
    If TestStr = "" Then
        MsgBox "File doesn't exist"
    Else
        MsgBox "File exist"
    End If

End Sub


Sub Test_File_Exist_FSO_Late_binding()
'No need to set a reference if you use Late binding
    Dim FSO As Object
    Dim FilePath As String

    Set FSO = CreateObject("scripting.filesystemobject")

    FilePath = "C:\Users\Ron\test\book1.xlsm"

    If FSO.FileExists(FilePath) = False Then
```

```
    MsgBox "file doesn't exist"
  Else
    MsgBox "File exist"
  End If

End Sub

Sub Test_File_Exist_FSO_Early_binding()
'If you want to use the Intellisense help showing you the properties
'and methods of the objects as you type you can use Early binding.
'Add a reference to "Microsoft Scripting Runtime" in the VBA editor
'(Tools>References)if you want that.

  Dim FSO As Scripting.FileSystemObject
  Dim FilePath As String

  Set FSO = New Scripting.FileSystemObject

  FilePath = "C:\Users\Ron\test"

  If FSO.FolderExists(FilePath) = False Then
    MsgBox "File doesn't exist"
  Else
    MsgBox "File exist"
  End If

End Sub
```

Test if file is open

Below are two examples to test if the file "book1.xlsm" is open.
The second example use a function(UDF), the advantage of a UDF is that all macros in your workbook can call this function and if you copy the UDF in a add-in all your open workbooks can use the UDF. Less code in your macros.

```
Sub Test_If_File_Is_Open_1()
  Dim TestWorkbook As Workbook

  Set TestWorkbook = Nothing
  On Error Resume Next
```

```
    Set TestWorkbook = Workbooks("Book1.xlsm")
    On Error GoTo 0

    If TestWorkbook Is Nothing Then
        MsgBox "The File is not open!"
    Else
        MsgBox "The File is open!"
    End If

End Sub
```

Do not forget to copy the function if you use the example below

```
Sub Test_If_File_Is_Open_2()
    If bIsBookOpen("Book1.xlsm") Then
        MsgBox "The File is open!"
    Else
        MsgBox "The File is not open!"
    End If
End Sub

Function bIsBookOpen(ByRef szBookName As String) As Boolean
' Rob Bovey
    On Error Resume Next
    bIsBookOpen = Not (Application.Workbooks(szBookName) Is Nothing)
End Function
```

Note: if you have more then one Excel instance open and want to test if the workbook is open in one of them then look at the code in this KB article.
http://support.microsoft.com/?kbid=138621

See also this page from Chip Pearson
http://www.cpearson.com/excel/IsFileOpen.aspx

Test if Sheet exists

Below are three examples to test if a sheet named "total" exists.

The third example use a function(UDF), the advantage of a UDF is that all macros in your workbook can call this function and if you copy the UDF in a add-in all your open workbooks can use the UDF. Less code in your macros.

```
Sub Sheet_Test_1()
   Dim sh As Worksheet

   On Error Resume Next
   Set sh = ActiveWorkbook.Sheets("total")
   If Err.Number <> 0 Then
      MsgBox "The sheet doesn't exist"
      Err.Clear
      On Error GoTo 0
   Else
      MsgBox "The sheet exist"
   End If

End Sub

Sub Sheet_Test_2()
   Dim SheetExist As Boolean

   SheetExist = False
   On Error Resume Next
   SheetExist = CBool(Len(ActiveWorkbook.Sheets.Item("total").Name))
   On Error GoTo 0

   If SheetExist = False Then
      MsgBox "The sheet doesn't exist"
   Else
      MsgBox "The sheet exist"
   End If

End Sub

Sub Sheet_Test_3_With_Function()
   If SheetExists("total", ActiveWorkbook) = False Then
      MsgBox "The sheet doesn't exist"
   Else
      MsgBox "The sheet exist"
   End If
```

```
End Sub

Function SheetExists(SName As String, _
                Optional ByVal wb As Workbook) As Boolean
'Chip Pearson
    On Error Resume Next
    If wb Is Nothing Then Set wb = ThisWorkbook
    SheetExists = CBool(Len(wb.Sheets(SName).Name))
End Function
```

Verknüpfung erstellen auf Desktop zu Ordner oder Datei

1. Link zu Ordner auf Desktop

Meine Version

```
Sub Link_von_Ordner_auf_Desktop()

Dim MyWSH As Object
Dim MyTarLink As Object
Dim myTarDeskTop As String

Set MyWSH = CreateObject("WScript.Shell")
myTarDeskTop = MyWSH.SpecialFolders("Desktop")

' Prüfen, ob es die Verknüpfung schon gibt

   Set DATEISUCHE = CreateObject("Scripting.FileSystemObject")
   DATEI = myTarDeskTop & "\" & "Mitarbeiter" & ".lnk"
   If DATEISUCHE.FileExists(DATEI) = False Then
       MsgBox "Die Verknüpfung " & DATEI & " konnte noch nicht gefunden werden." & vbCrLf & vbCrLf & _
          "Sie wird nun automatisch erstellt. "
       ' Definieren des Links
       Set MyTarLink = MyWSH.CreateShortcut(myTarDeskTop & "\" & "Mitarbeiter" & ".lnk")
       ' Dem Link den Ordner für das Ausführen hinzufügen
       With MyTarLink
          .Targetpath = "D:\EIGENE DATEIEN\SONSTIGES\PROJEKTE\ZEITERFASSUNG\Stundenlisten 2016\Mitarbeiter\"
          .Save
       End With
     Else
       MsgBox "Die Verknüpfung " & DATEI & " existierte bereits"
    End If
```

```
Set MyWSH = Nothing
Set MyTarLink = Nothing
End Sub
```

Original Version

```
Sub Create_Link_On_Desktop()
Dim MyWSH As Object
Dim MyTarLink As Object
Dim myTarDeskTop As String
Set MyWSH = CreateObject("WScript.Shell")
myTarDeskTop = MyWSH.SpecialFolders("Desktop")
Set MyTarLink = MyWSH.CreateShortcut(myTarDeskTop & _
   "\" & ThisWorkbook.name & ".lnk")
With MyTarLink
   .Targetpath = ThisWorkbook.FullName
   .Save
End With
Set MyWSH = Nothing
Set MyTarLink = Nothing
End Sub
```

2. Link in einen Unterordner erstellen

```
Sub Create_Link_into_Folder()
Dim MyWSH As Object
Dim MyTarLink As Object
Dim myTarFolder As String
Set MyWSH = CreateObject("WScript.Shell")
myTarFolder = "C:\Ordner\Unterordner\"
Set MyTarLink = MyWSH.CreateShortcut(myTarFolder & "Restore.lnk")
With MyTarLink
   .Targetpath = "N:\Freigabename\Unterordner\Unterordner\Dateil.xls"
   .Save
End With
Set MyWSH = Nothing
Set MyTarLink = Nothing
End Sub
```

3. Link zu Datei

```
Public Sub Test()
    Dim strDesktop As String
    Dim objLink As Object
    Dim strFile As String
    Dim strPath As String
    Dim objWSH As Object
    On Error GoTo Fin
    strPath = "C:\Temp\" ' anpassen
    strFile = "Book1.xls" ' anpassen
    Set objWSH = CreateObject("WScript.Shell")
    strDesktop = objWSH.SpecialFolders("Desktop")
    Set objLink = objWSH.CreateShortcut(strDesktop & "\" & strFile & ".lnk")
    With objLink
        .Targetpath = strPath & strFile
        .Save
    End With
Fin:
    Set objWSH = Nothing
End Sub
```

Verzeichnisgröße auslesen

```
Option Explicit
Private Type FILETIME
dwLowDateTime As Long
dwHighDateTime As Long
End Type
Private Const MAX_PATH = 260
Private Const FILE_ATTRIBUTE_DIRECTORY = &H10

Private Type WIN32_FIND_DATA
dwFileAttributes As Long
ftCreationTime As FILETIME
ftLastAccessTime As FILETIME
ftLastWriteTime As FILETIME
nFileSizeHigh As Long
nFileSizeLow As Long
dwReserved0 As Long
dwReserved1 As Long
```

```
cFileName As String * MAX_PATH
cAlternate As String * 14
End Type

Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As
Long) As Long
Private Declare Function FindFirstFile Lib "kernel32" Alias
"FindFirstFileA" (ByVal lpFileName As String, lpFindFileData As
WIN32_FIND_DATA) As Long
Private Declare Function FindNextFile Lib "kernel32" Alias
"FindNextFileA" (ByVal hFindFile As Long, lpFindFileData As
WIN32_FIND_DATA) As Long

Function GetDirectorySize(ByVal Root$) As Double
' Function to calculate bytes used in Root$ and all subdirectories of Root$.
' Root$ should be entered in the form c:\Dir

Dim FData As WIN32_FIND_DATA
Dim fHand&
Dim sPath$
Dim StillOK&
Dim ByteTotal&
Dim nPos%
Dim DirName$

sPath$ = Root$ + "\*.*"

fHand& = FindFirstFile(sPath$, FData)

If fHand& <= 0 Then
GetDirectorySize = 0
Exit Function
End If

ByteTotal& = 0
Do
If (FData.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY) =
FILE_ATTRIBUTE_DIRECTORY Then
nPos% = InStr(FData.cFileName, Chr$(0))
DirName$ = Left$(FData.cFileName, nPos% - 1)
If DirName$ <> "." And DirName$ <> ".." Then
ByteTotal& = ByteTotal& + GetDirectorySize(Root$ + "\" +
DirName$)
End If
Else
ByteTotal& = ByteTotal& + FData.nFileSizeLow
End If

StillOK& = FindNextFile(fHand&, FData)
Loop Until StillOK = 0

fHand& = FindClose(fHand&)
GetDirectorySize = ByteTotal&
```

```
End Function
```

XML-Datei öffnen und als TXT/CSV-Datei für BMD speichern

```
Sub XML_Konverter()

' Liest XML-Dateien nach Excel ein und speichert sie als Textdatei ab (Tabstopp-getrennt)

Dim DATEINAME_IMPORT
Dim DATEIPFAD_IMPORT
Dim DATEINAME_EXPORT
Dim DATEIPFAD_EXPORT

' Variablen auslesen
   DATEINAME_IMPORT = Sheets("Parameter").Range("F4")
   DATEINAME_EXPORT = Sheets("Parameter").Range("K4")

' Variablen überprüfen, dass nicht leer
   If DATEINAME_IMPORT = "" Then
      MsgBox "Bitte tragen Sie in der Tabelle PARAMETER in Zelle F4 den Dateinamen der zu importierenden XML-Datei ein"
      End
   End If
   If DATEINAME_EXPORT = "" Then
      MsgBox "Bitte tragen Sie in der Tabelle PARAMETER in Zelle K4 den Dateinamen für die Export-Datei ein"
      End
   End If

' Festlegen der Pfade
   DATEIPFAD_IMPORT = ThisWorkbook.Path & "\" & DATEINAME_IMPORT
   DATEIPFAD_EXPORT = ThisWorkbook.Path & "\" & DATEINAME_EXPORT

' Öffnen der XML-Datei
   Workbooks.OpenXML Filename:=DATEIPFAD_IMPORT, LoadOption:=xlXmlLoadImportToList

' Speichern der Datei
   ActiveWorkbook.SaveAs Filename:=DATEIPFAD_EXPORT, FileFormat:=xlText, CreateBackup:=False

' Schließen der Exportdatei, die beim Speichern entsteht
   Workbooks(DATEINAME_EXPORT).Close Savechanges:=False

End Sub
```

VARIANTE 1

```
Sub XML_Dateien_Einlesen()

    Dim Datei$, Pfad$, DateiMatch$
    Dim AnfZelle As Range, Wb As Workbook, Ws As Worksheet, lo As ListObject, mainLO As ListObject, col As ListRow
    On Error Resume Next

    Set Wb = ActiveWorkbook
    Set Ws = Wb.ActiveSheet

    'Range vorher löschen bevor wir importieren
    Ws.Range("A:H").Delete

    Set AnfZelle = Ws.Range("A1") '<== Anfangszelle im aktiven Arbeitsblatt der aktiven Arb.Mappe

    Pfad$ = "D:\_TEST\" '<== Pfad zum Verzeichnis einstellen
    DateiMatch$ = "TEST*.xml" '<== Datei-Matching, um gewünschte Dateien zu filtern

    Datei$ = Dir(Pfad$ & DateiMatch$, vbNormal)

    counter = 1
    Do While Datei$ <> ""
        Wb.XmlImport URL:=Pfad$ & Datei$, ImportMap:=Nothing, Overwrite:=True, Destination:=AnfZelle
        If counter > 1 Then
            Set lo = Ws.ListObjects(2)
            Set mainLO = Ws.ListObjects(1)
            Set col = mainLO.ListRows.Add
            lo.DataBodyRange.Copy col.Range
            intNextOffset = lo.ListRows.Count
            lo.Delete
        Else
            Set lo = Ws.ListObjects(1)
            intNextOffset = lo.ListRows.Count
        End If
        Set AnfZelle = AnfZelle.Offset(intNextOffset, 0)
        Datei$ = Dir()
        counter = counter + 1
    Loop
End Sub
```

VARIANTE 2

```
Sub import()
     Dim strTargetFile As String
     Application.DisplayAlerts = False
     strTargetFile = "C:\Users\thomas\Desktop\xml2\Neuer Ordner\HID1_PROD*.xml"
     Workbooks.OpenXML Filename:=strTargetFile, LoadOption:=xlXmlLoadImportToList
     Application.DisplayAlerts = True

     End Sub
```

ALLE XML-Dateien eines Ordners öffnen in einer gemeinsamen Tabelle

```
Sub ImportXML()
    Const XMLPATH ="C:\Users\thomas\Desktop\xml2\Neuer Ordner"
    Dim f As Object, c As Object
    Set fso = CreateObject("Scripting.Filesystemobject")
    Application.DisplayAlerts = False
    Application.ScreenUpdating = False
    With Sheets(1)
        For Each f In fso.GetFolder(XMLPATH).Files
            If LCase(fso.GetExtensionName(f.Name)) = "xml" Then
                ActiveWorkbook.XmlImport URL:=f.Path, ImportMap:=Nothing, Overwrite:=True, Destination:=.Range("A" &
.UsedRange.SpecialCells(xlCellTypeLastCell).Row + 1)
                ActiveWorkbook.Connections(ActiveWorkbook.Connections.Count).Delete
            End If
        Next
    End With
    Application.DisplayAlerts = True
    Application.ScreenUpdating = True
    MsgBox "Importvorgang beendet!", vbInformation
    Set fso = Nothing
End Sub
```

ALTERNATIVE

```
Sub XML_Dateien_Einlesen()
    'Filesystem Object erstellen
    Set fso = CreateObject("Scripting.FileSystemObject")
    ' Pfad der XML Dateien
    Pfad = "C:\Temp\quelle"
    With ActiveSheet
            'Import-Bereich löschen
```

```
            .UsedRange.Delete
             'Eventuell vorhandene Import-Definitionen löschen
            While ActiveWorkbook.XmlMaps.Count > 0
                ActiveWorkbook.XmlMaps(1).Delete
            Next

            For Each f In fso.GetFolder(Pfad).Files
                    If LCase(extension) = LCase(fso.GetExtensionName(f.Path)) Then
                    'Import der XML-Datei
                    ActiveWorkbook.XmlImport URL:=f.Path, ImportMap:=Nothing, Overwrite:=False,
Destination:=.Cells(Rows.Count, 1).End(xlUp).Offset(1, 0)
                    'Import-Definitionen löschen
                    While ActiveWorkbook.XmlMaps.Count > 0
                        ActiveWorkbook.XmlMaps(1).Delete
                    Next
                End If
            Next
    End With
    Set fso = Nothing
End Sub
```

ALTERNATIVE

Folgendes Script lädt alle XML Dateien ein:

```
Sub ImportXML()
    Const XMLPATH = "C:\Users\kaiuwe28\Downloads\Test_xml"
    Dim f As Object, c As Object
    Set fso = CreateObject("Scripting.Filesystemobject")
    Application.DisplayAlerts = False
    Application.ScreenUpdating = False
    With Sheets("Import")
        For Each f In fso.GetFolder(XMLPATH).Files
            If LCase(fso.GetExtensionName(f.Name)) = "xml" Then
                ActiveWorkbook.XmlImport URL:=f.Path, ImportMap:=Nothing, Overwrite:=True, Destination:=.Range("A" &
.UsedRange.SpecialCells(xlCellTypeLastCell).Row + 1)
                ActiveWorkbook.Connections(ActiveWorkbook.Connections.Count).Delete
            End If
        Next
    End With
```

```
    Set fso = Nothing

    Application.DisplayAlerts = True
    Application.ScreenUpdating = True

    MsgBox "Importvorgang beendet!", vbInformation

End Sub
```

ALLE XML-DATEIEN EINES ORDNERS ÖFFNEN

```
Sub Import
      Dim strPath As String
      Dim strTargetFile As String
      Application.DisplayAlerts = False
      strPath = "C:\Users\thomas\Desktop\xml2\Neuer Ordner\"
      strTargetFile = Dir(strPath & "*.xml")
      Do Until strTargetFile = ""
      Workbooks.OpenXML Filename:=strPath & strTargetFile, LoadOption:=xlXmlLoadImportToList
      Application.DisplayAlerts = True
      strTargetFile = Dir
      Loop
End Sub
```

XML-Datei laden:

```
Option Explicit
Dim xDoc As MSXML2.DOMDocument60
Public Sub LoadDocument()
Set xDoc = New MSXML2.DOMDocument60
xDoc.validateOnParse = True
If xDoc.Load("C:\Users\?\Downloads\123098.xml") Then
' The document loaded successfully.
Debug.Print "Load success ...."
```

```
Else
' The document failed to load.
Debug.Print "Load failed ...."
End If
End Sub
Sub GetKD_ID(myCustomer As String)
Dim oSeqNodes As IXMLDOMNodeList
Dim oSeqNode As IXMLDOMNode
Dim oSeqChild As IXMLDOMNode
Set oSeqNodes = xDoc.SelectNodes("//BrandData/" & myCustomer)
If oSeqNodes.Length = 0 Then
'show some message
Else
For Each oSeqNode In oSeqNodes
For Each oSeqChild In oSeqNode.ChildNodes
Debug.Print oSeqChild.nodeName
Next
Next
End If
End Sub
```

- ZIPPEN - ENTZIPPEN -

Zip Activeworkbook, Folder, File or Files with 7-Zip (VBA)

Ron de Bruin (last update 17-May-2009)
Go back to the Excel tips page
The basic examples on this page use VBA code to zip the ActiveWorkbook, Folder, File or Files with:

http://www.7-zip.org/

For example code to unzip a zip file with 7-Zip visit :
http://www.rondebruin.nl/7zipwithexcelunzip.htm

For examples for WinZip or the default Windows Zip program see the Zip (compress) section here
http://www.rondebruin.nl/tips.htm

Examples

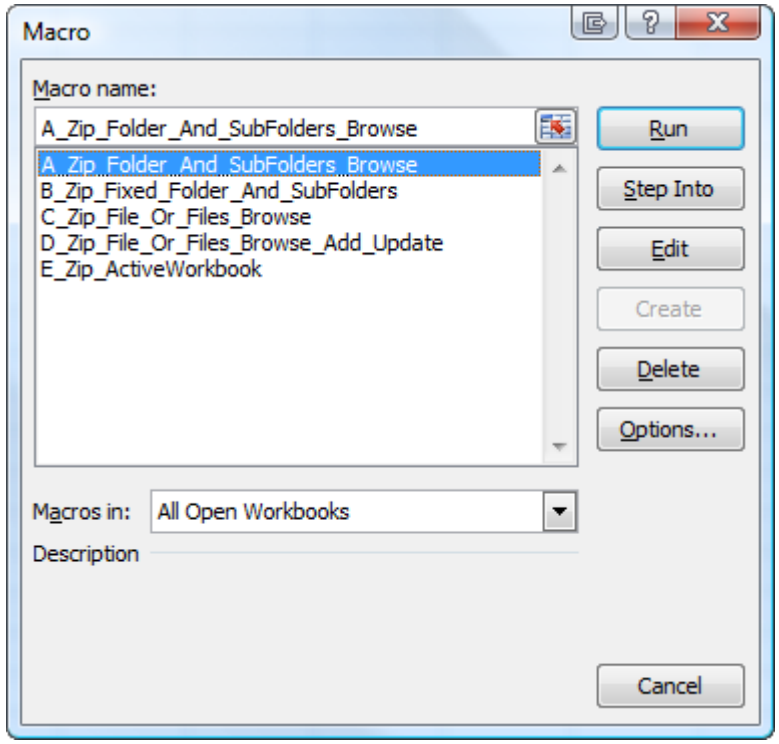
I have add all the code in a txt file on my site so it is easy to copy it in a module of your workbook.

Click on the link below
http://www.rondebruin.nl/files/code7zippage.txt

1. Ctrl a to select all the code
2. Ctrl c to copy
3. Press the Back button in your browser to go back to this page

Open Excel or make Excel the active program
1. Alt-F11 to open the VBA Editor
2. Insert>Module from the Menu bar
3. Ctrl v to Paste the Code
4. Alt-q to go back to Excel

When you use the shortcut Alt F8 now you can see and run the macros.



Note: I use the .zip extension in my examples but you can also use the extension .7z
If you use the extension .7z you will see that the file size is smaller.

Four examples above you can test without changing the code.
Only if you want to test the macro: B_Zip_Fixed_Folder_And_SubFolders
You must change the string of FolderName to the folder you want to zip in this code line
FolderName = "C:\Users\Ron\Desktop\TestFolder"

Important: Read the comments above and in the code good and after you test the code examples
you can try the commented examples in the first macro Zip_Folder_And_SubFolders_Browse

To test it replace :

```
ShellStr = PathZipProgram & "7z.exe a -r" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "*.*" & Chr(34)
```

With one of the strings below :

```
'Zip the txt files in the folder and subfolders, use "*.xl*" for all excel files
ShellStr = PathZipProgram & "7z.exe a -r" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "*.txt" & Chr(34)
'Zip all files in the folder and subfolders with a name that start with Week
ShellStr = PathZipProgram & "7z.exe a -r" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "Week*.*" & Chr(34)
'Zip every file with the name ron.xlsx in the folder and subfolders
ShellStr = PathZipProgram & "7z.exe a -r" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "ron.xlsx" & Chr(34)
'Add -ppassword -mhe of you want to add a password to the zip file(only 7z files)
ShellStr = PathZipProgram & "7z.exe a -r -ppassword -mhe" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "*.*" & Chr(34)
'Add -seml if you want to open a mail with the zip attached
ShellStr = PathZipProgram & "7z.exe a -r -seml" _
        & " " & Chr(34) & NameZipFile & Chr(34) _
        & " " & Chr(34) & FolderName & "*.*" & Chr(34)
```

7-Zip have also a good help file (7-zip.chm) so if you want to know more read this first
You can find it it this folder : C:\Program Files\7-Zip
There is also a forum on the 7Zip site that you can visit.

Unzip a zip file with 7-Zip (VBA)

Ron de Bruin (last update 17-May-2009)
Go back to the Excel tips page
The basic examples on this page use VBA code to unzip a zip file with:

http://www.7-zip.org/

For example code to Zip files with 7-Zip visit :
Zip Activeworkbook, Folder, File or Files with 7-Zip (VBA)

For examples for WinZip or the default Windows Zip program see the Zip (compress) section here
http://www.rondebruin.nl/tips.htm

Examples

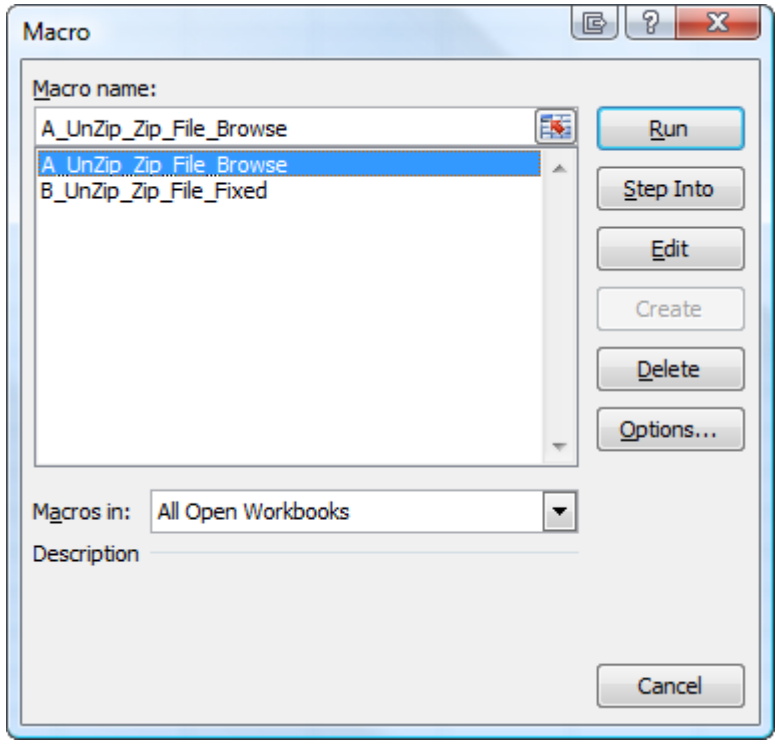
I have add all the code in a txt file on my site so it is easy to copy it in a module of your workbook.

Click on the link below (der CODE IST IM ANSCHLUSS EINGEFÜGT)
http://www.rondebruin.nl/files/code7zippageunzip.txt

1. Ctrl a to select all the code
2. Ctrl c to copy
3. Press the Back button in your browser to go back to this page

Open Excel or make Excel the active program
1. Alt-F11 to open the VBA Editor
2. Insert>Module from the Menu bar
3. Ctrl v to Paste the Code
4. Alt-q to go back to Excel

When you use the shortcut Alt F8 now you can see and run the macros.



Note: I use the .zip extension in my examples but you can also use the extension .7z

The first example above you can test without changing the code.
Only if you want to test the macro: B_UnZip_Zip_File_Fixed
You must change the string NameUnZipFolder to the folder where you want to unzip the files
NameUnZipFolder = "C:\Users\Ron\TestFolder\"

Important: Read the comments above and in the code good for tips.

7-Zip have also a good help file (7-zip.chm) so if you want to know more read this first
You can find it it this folder : C:\Program Files\7-Zip

There is also a forum on the 7-Zip site that you can visit.

Hier der Code aus dem TXT-File:

```
Declare Function OpenProcess Lib "kernel32" _
                        (ByVal dwDesiredAccess As Long, _
                         ByVal bInheritHandle As Long, _
                         ByVal dwProcessId As Long) As Long

Declare Function GetExitCodeProcess Lib "kernel32" _
                             (ByVal hProcess As Long, _
                              lpExitCode As Long) As Long

Public Const PROCESS_QUERY_INFORMATION = &H400
Public Const STILL_ACTIVE = &H103


Public Sub ShellAndWait(ByVal PathName As String, Optional WindowState)
   Dim hProg As Long
   Dim hProcess As Long, ExitCode As Long
   'fill in the missing parameter and execute the program
   If IsMissing(WindowState) Then WindowState = 1
   hProg = Shell(PathName, WindowState)
   'hProg is a "process ID under Win32. To get the process handle:
   hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False, hProg)
   Do
      'populate Exitcode variable
      GetExitCodeProcess hProcess, ExitCode
      DoEvents
   Loop While ExitCode = STILL_ACTIVE
End Sub
```

```
'With this example you browse to the zip or 7z file you want to unzip
'The zip file will be unzipped in a new folder in: Application.DefaultFilePath
'Normal if you have not change it this will be your Documents folder
'The name of the folder that the code create in this folder is the Date/Time
'You can change this folder to this if you want to use a fixed folder:
'NameUnZipFolder = "C:\Users\Ron\TestFolder\"
'Read the comments in the code about the commands/Switches in the ShellStr
'There is no need to change the code before you test it

Sub A_UnZip_Zip_File_Browse()
    Dim PathZipProgram As String, NameUnZipFolder As String
    Dim FileNameZip As Variant, ShellStr As String

    'Path of the Zip program
    PathZipProgram = "C:\program files\7-Zip\"
    If Right(PathZipProgram, 1) <> "\" Then
        PathZipProgram = PathZipProgram & "\"
    End If

    'Check if this is the path where 7z is installed.
    If Dir(PathZipProgram & "7z.exe") = "" Then
        MsgBox "Please find your copy of 7z.exe and try again"
        Exit Sub
    End If

    'Create path and name of the normal folder to unzip the files in
    'In this example we use: Application.DefaultFilePath
    'Normal if you have not change it this will be your Documents folder
    'The name of the folder that the code create in this folder is the Date/Time
    NameUnZipFolder = Application.DefaultFilePath & "\" & Format(Now, "yyyy-mm-dd h-mm-ss")
    'You can also use a fixed path like
```

```
    'NameUnZipFolder = "C:\Users\Ron\TestFolder"

    'Select the zip file (.zip or .7z files)
    FileNameZip = Application.GetOpenFilename(filefilter:="Zip Files, *.zip, 7z Files, *.7z", _
                                        MultiSelect:=False, Title:="Select the file that you want to unzip")

    'Unzip the files/folders from the zip file in the NameUnZipFolder folder
    If FileNameZip = False Then
        'do nothing
    Else
        'There are a few commands/Switches that you can change in the ShellStr
        'We use x command now to keep the folder stucture, replace it with e if you want only the files
        '-aoa Overwrite All existing files without prompt.
        '-aos Skip extracting of existing files.
        '-aou aUto rename extracting file (for example, name.txt will be renamed to name_1.txt).
        '-aot auto rename existing file (for example, name.txt will be renamed to name_1.txt).
        'Use -r if you also want to unzip the subfolders from the zip file
        'You can add -ppassword if you want to unzip a zip file with password (only 7zip files)
        'Change "*.*" to for example "*.txt" if you only want to unzip the txt files
        'Use "*.xl*" for all Excel files: xls, xlsx, xlsm, xlsb
        ShellStr = PathZipProgram & "7z.exe x -aoa -r" _
                & " " & Chr(34) & FileNameZip & Chr(34) _
                & " -o" & Chr(34) & NameUnZipFolder & Chr(34) & " " & "*.*"

        ShellAndWait ShellStr, vbHide
        MsgBox "Look in " & NameUnZipFolder & " for extracted files"

    End If
End Sub
```

```
'With this example you unzip a fixed zip file: FileNameZip = "C:\Users\Ron\Test.zip"
'Note this file must exist, this is the only thing that you must change before you test it
'The zip file will be unzipped in a new folder in: Application.DefaultFilePath
'Normal if you have not change it this will be your Documents folder
'The name of the folder that the code create in this folder is the Date/Time
'You can change this folder to this if you want to use a fixed folder:
'NameUnZipFolder = "C:\Users\Ron\TestFolder\"
'Read the comments in the code about the commands/Switches in the ShellStr

Sub B_UnZip_Zip_File_Fixed()
    Dim PathZipProgram As String, NameUnZipFolder As String
    Dim FileNameZip As Variant, ShellStr As String

    'Path of the Zip program
    PathZipProgram = "C:\program files\7-Zip\"
    If Right(PathZipProgram, 1) <> "\" Then
        PathZipProgram = PathZipProgram & "\"
    End If

    'Check if this is the path where 7z is installed.
    If Dir(PathZipProgram & "7z.exe") = "" Then
        MsgBox "Please find your copy of 7z.exe and try again"
        Exit Sub
    End If

    'Create path and name of the normal folder to unzip the files in
    'In this example we use: Application.DefaultFilePath
    'Normal if you have not change it this will be your Documents folder
    'The name of the folder that the code create in this folder is the Date/Time
    NameUnZipFolder = Application.DefaultFilePath & "\" & Format(Now, "yyyy-mm-dd h-mm-ss")
    'You can also use a fixed path like
    'NameUnZipFolder = "C:\Users\Ron\TestFolder\"
```

```
    'Name of the zip file that you want to unzip (.zip or .7z files)
    FileNameZip = "C:\Users\Ron\Test.zip"

    'There are a few commands/Switches that you can change in the ShellStr
    'We use x command now to keep the folder stucture, replace it with e if you want only the files
    '-aoa Overwrite All existing files without prompt.
    '-aos Skip extracting of existing files.
    '-aou aUto rename extracting file (for example, name.txt will be renamed to name_1.txt).
    '-aot auto rename existing file (for example, name.txt will be renamed to name_1.txt).
    'Use -r if you also want to unzip the subfolders from the zip file
    'You can add -ppassword if you want to unzip a zip file with password (only .7z files)
    'Change "*.*" to for example "*.txt" if you only want to unzip the txt files
    'Use "*.xl*" for all Excel files: xls, xlsx, xlsm, xlsb
    ShellStr = PathZipProgram & "7z.exe x -aoa -r" _
            & " " & Chr(34) & FileNameZip & Chr(34) _
            & " -o" & Chr(34) & NameUnZipFolder & Chr(34) & " " & "*.*"

    ShellAndWait ShellStr, vbHide
    MsgBox "Look in " & NameUnZipFolder & " for extracted files"

End Sub
```

Unzip file or files with the default Windows zip program (VBA)

Ron de Bruin (last update 26-May-2009)
Go back to the Excel tips page
Information

Warning: The code below is not supported by Microsoft
It is not possible to hide the copy dialog when you copy from a zip folder (only working with normal folders).

Also there is no possibility to avoid that someone can cancel the CopyHere operation or that your VBA code will be notified that the operation has been cancelled.

Note: Do not Dim for example FileNameFolder as String in the code examples.
This must be a Variant, if you change this the code will not work.

If you want to zip files see this page on my site.
http://www.rondebruin.nl/windowsxpzip.htm

See also the the Zip (compress) section on my site for examples for 7-zip and WinZip.

Example 1

With this example you can browse to the zip file.
After you select the zip file the macro will create a new folder in your DefaultFilePath and unzip the Zip file in that folder. You can run the code without any changes.

```
Sub Unzip1()
    Dim FSO As Object
    Dim oApp As Object
    Dim Fname As Variant
    Dim FileNameFolder As Variant
    Dim DefPath As String
    Dim strDate As String

    Fname = Application.GetOpenFilename(filefilter:="Zip Files (*.zip), *.zip", _
                                MultiSelect:=False)
    If Fname = False Then
        'Do nothing
    Else
        'Root folder for the new folder.
        'You can also use DefPath = "C:\Users\Ron\test\"
        DefPath = Application.DefaultFilePath
        If Right(DefPath, 1) <> "\" Then
            DefPath = DefPath & "\"
        End If

        'Create the folder name
```

```
        strDate = Format(Now, " dd-mm-yy h-mm-ss")
        FileNameFolder = DefPath & "MyUnzipFolder " & strDate & "\"

        'Make the normal folder in DefPath
        MkDir FileNameFolder

        'Extract the files into the newly created folder
        Set oApp = CreateObject("Shell.Application")

        oApp.Namespace(FileNameFolder).CopyHere oApp.Namespace(Fname).items

        'If you want to extract only one file you can use this:
        'oApp.Namespace(FileNameFolder).CopyHere _
         'oApp.Namespace(Fname).items.Item("test.txt")

        MsgBox "You find the files here: " & FileNameFolder

        On Error Resume Next
        Set FSO = CreateObject("scripting.filesystemobject")
        FSO.deletefolder Environ("Temp") & "\Temporary Directory*", True
    End If
End Sub
```

Example 2

The macro below is almost the same as above. The only difference is that it will only extract txt files from the Zip file. Change this "*.txt" to extract the files you want.
If you want to extract one file from a Zip file see the commented code in the macro above.

```
Sub Unzip2()
    Dim FSO As Object
    Dim oApp As Object
    Dim Fname As Variant
    Dim FileNameFolder As Variant
    Dim DefPath As String
    Dim strDate As String
    Dim fileNameInZip As Variant

    Fname = Application.GetOpenFilename(filefilter:="Zip Files (*.zip), *.zip", _
                                MultiSelect:=False)
```

```
    If Fname = False Then
        'Do nothing
    Else
        'Root folder for the new folder.
        'You can also use DefPath = "C:\Users\Ron\test\"
        DefPath = Application.DefaultFilePath
        If Right(DefPath, 1) <> "\" Then
            DefPath = DefPath & "\"
        End If

        'Create the folder name
        strDate = Format(Now, " dd-mm-yy h-mm-ss")
        FileNameFolder = DefPath & "MyUnzipFolder " & strDate & "\"

        'Make the normal folder in DefPath
        MkDir FileNameFolder

        'Extract the files into the newly created folder
        Set oApp = CreateObject("Shell.Application")

        'Change this "*.txt" to extract the files you want
        For Each fileNameInZip In oApp.Namespace(Fname).items
            If LCase(fileNameInZip) Like LCase("*.txt") Then
                oApp.Namespace(FileNameFolder).CopyHere _
                        oApp.Namespace(Fname).items.Item(CStr(fileNameInZip))
            End If
        Next

        MsgBox "You find the files here: " & FileNameFolder

        On Error Resume Next
        Set FSO = CreateObject("scripting.filesystemobject")
        FSO.deletefolder Environ("Temp") & "\Temporary Directory*", True
    End If
End Sub
```

Example 3

The macros above will create a new folder for you to copy the files in

but this macro unzip the zip file in a fixed folder "C:\Users\Ron\test\"
See the commented code in the macro to delete the files in the folder first if you want.

```
Sub Unzip3()
    Dim FSO As Object
    Dim oApp As Object
    Dim Fname As Variant
    Dim FileNameFolder As Variant
    Dim DefPath As String

    Fname = Application.GetOpenFilename(filefilter:="Zip Files (*.zip), *.zip", _
                                MultiSelect:=False)
    If Fname = False Then
        'Do nothing
    Else
        'Destination folder
        DefPath = "C:\Users\Ron\test\"    '<<< Change path
        If Right(DefPath, 1) <> "\" Then
            DefPath = DefPath & "\"
        End If

        FileNameFolder = DefPath

        '       'Delete all the files in the folder DefPath first if you want
        '       On Error Resume Next
        '       Kill DefPath & "*.*"
        '       On Error GoTo 0

        'Extract the files into the Destination folder
        Set oApp = CreateObject("Shell.Application")
        oApp.Namespace(FileNameFolder).CopyHere oApp.Namespace(Fname).items

        MsgBox "You find the files here: " & FileNameFolder

        On Error Resume Next
        Set FSO = CreateObject("scripting.filesystemobject")
        FSO.deletefolder Environ("Temp") & "\Temporary Directory*", True
    End If
End Sub
```

Example 4

The macro below is almost the same as Example 1. The only difference is that you can select more then one zip file to unzip in the same folder it create.

```
Sub Unzip4()
    Dim FSO As Object
    Dim oApp As Object
    Dim Fname As Variant
    Dim FileNameFolder As Variant
    Dim DefPath As String
    Dim strDate As String
    Dim I As Long
    Dim num As Long

    Fname = Application.GetOpenFilename(filefilter:="Zip Files (*.zip), *.zip", _
                                MultiSelect:=True)
    If IsArray(Fname) = False Then
        'Do nothing
    Else
        'Root folder for the new folder.
        'You can also use DefPath = "C:\Users\Ron\test\"
        DefPath = Application.DefaultFilePath
        If Right(DefPath, 1) <> "\" Then
            DefPath = DefPath & "\"
        End If

        'Create the folder name
        strDate = Format(Now, " dd-mm-yy h-mm-ss")
        FileNameFolder = DefPath & "MyUnzipFolder " & strDate & "\"

        'Make the normal folder in DefPath
        MkDir FileNameFolder

        'Extract the files into the newly created folder
        Set oApp = CreateObject("Shell.Application")

        For I = LBound(Fname) To UBound(Fname)
            num = oApp.Namespace(FileNameFolder).items.Count

            oApp.Namespace(FileNameFolder).CopyHere oApp.Namespace(Fname(I)).items

        Next I
```

```
        MsgBox "You find the files here: " & FileNameFolder

        On Error Resume Next
        Set FSO = CreateObject("scripting.filesystemobject")
        FSO.deletefolder Environ("Temp") & "\Temporary Directory*", True
    End If
End Sub
```

Zip file or files with the default Windows zip program (VBA)

Ron de Bruin (last update 26-May-2009)
Go back to the Excel tips page

Information

Warning: The code below is not supported by Microsoft
It is not possible to hide the copy dialog when you copy to a zip folder (only working with normal folders).
Also there is no possibility to avoid that someone can cancel the CopyHere operation or that your VBA code will be notified that the operation has been cancelled.

Note: Do not Dim for example FileNameZip as String in the code examples.
This must be a Variant, if you change this the code will not work.

If you want to Unzip files see this page on my site.
http://www.rondebruin.nl/windowsxpunzip.htm

See also the the Zip (compress) section on my site for examples for 7-zip and WinZip.

Important

Before we can try one of the macro examples we must copy the following code in a normal module of your workbook. Every macro use the sub NewZip and the first example also use both functions.
See this page how : http://www.rondebruin.nl/code.htm

```
Sub NewZip(sPath)
'Create empty Zip File
'Changed by keepITcool Dec-12-2005
    If Len(Dir(sPath)) > 0 Then Kill sPath
    Open sPath For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
End Sub


Function bIsBookOpen(ByRef szBookName As String) As Boolean
' Rob Bovey
    On Error Resume Next
    bIsBookOpen = Not (Application.Workbooks(szBookName) Is Nothing)
End Function


Function Split97(sStr As Variant, sdelim As String) As Variant
'Tom Ogilvy
    Split97 = Evaluate("{""" & _
                    Application.Substitute(sStr, sdelim, """,""") & """}")
End Function
```

Examples

There are five examples on this page that you can copy in a normal module of your workbook.
Please read the information good above before you start testing the code below.

Browse to the folder you want and select the file or files

```
Sub Zip_File_Or_Files()
    Dim strDate As String, DefPath As String, sFName As String
    Dim oApp As Object, iCtr As Long, I As Integer
    Dim FName, vArr, FileNameZip

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
       DefPath = DefPath & "\"
    End If

    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
```

```
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    'Browse to the file(s), use the Ctrl key to select more files
    FName = Application.GetOpenFilename(filefilter:="Excel Files (*.xl*), *.xl*", _
                MultiSelect:=True, Title:="Select the files you want to zip")
    If IsArray(FName) = False Then
        'do nothing
    Else
        'Create empty Zip File
        NewZip (FileNameZip)
        Set oApp = CreateObject("Shell.Application")
        I = 0
        For iCtr = LBound(FName) To UBound(FName)
            vArr = Split97(FName(iCtr), "\")
            sFName = vArr(UBound(vArr))
            If bIsBookOpen(sFName) Then
                MsgBox "You can't zip a file that is open!" & vbLf & _
                      "Please close it and try again: " & FName(iCtr)
            Else
                'Copy the file to the compressed folder
                I = I + 1
                oApp.Namespace(FileNameZip).CopyHere FName(iCtr)

                'Keep script waiting until Compressing is done
                On Error Resume Next
                Do Until oApp.Namespace(FileNameZip).items.Count = I
                    Application.Wait (Now + TimeValue("0:00:01"))
                Loop
                On Error GoTo 0
            End If
        Next iCtr

        MsgBox "You find the zipfile here: " & FileNameZip
    End If
End Sub
```

Browse to a folder and zip all files in it

```
Sub Zip_All_Files_in_Folder_Browse()
    Dim FileNameZip, FolderName, oFolder
    Dim strDate As String, DefPath As String
    Dim oApp As Object
```

```
    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
       DefPath = DefPath & "\"
    End If

    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    Set oApp = CreateObject("Shell.Application")

    'Browse to the folder
    Set oFolder = oApp.BrowseForFolder(0, "Select folder to Zip", 512)
    If Not oFolder Is Nothing Then
       'Create empty Zip File
       NewZip (FileNameZip)

       FolderName = oFolder.Self.Path
       If Right(FolderName, 1) <> "\" Then
          FolderName = FolderName & "\"
       End If

       'Copy the files to the compressed folder
       oApp.Namespace(FileNameZip).CopyHere oApp.Namespace(FolderName).items

       'Keep script waiting until Compressing is done
       On Error Resume Next
       Do Until oApp.Namespace(FileNameZip).items.Count = _
       oApp.Namespace(FolderName).items.Count
          Application.Wait (Now + TimeValue("0:00:01"))
       Loop
       On Error GoTo 0

       MsgBox "You find the zipfile here: " & FileNameZip

    End If
End Sub
```

Zip all files in the folder that you enter in the code

Note: Before you run the macro below change the folder in this macro line
FolderName = "C:\Users\Ron\test\"

```
Sub Zip_All_Files_in_Folder()
    Dim FileNameZip, FolderName
    Dim strDate As String, DefPath As String
    Dim oApp As Object

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
        DefPath = DefPath & "\"
    End If

    FolderName = "C:\Users\Ron\test\"    '<< Change

    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    'Create empty Zip File
    NewZip (FileNameZip)

    Set oApp = CreateObject("Shell.Application")
    'Copy the files to the compressed folder
    oApp.Namespace(FileNameZip).CopyHere oApp.Namespace(FolderName).items

    'Keep script waiting until Compressing is done
    On Error Resume Next
    Do Until oApp.Namespace(FileNameZip).items.Count = _
       oApp.Namespace(FolderName).items.Count
        Application.Wait (Now + TimeValue("0:00:01"))
    Loop
    On Error GoTo 0

    MsgBox "You find the zipfile here: " & FileNameZip
End Sub
```

Zip the ActiveWorkbook

This sub will make a copy of the Activeworkbook and zip it in "C:\Users\Ron\test\" with a date-time stamp.
Change this folder or use your default path Application.DefaultFilePath

```
Sub Zip_ActiveWorkbook()
    Dim strDate As String, DefPath As String
    Dim FileNameZip, FileNameXls
    Dim oApp As Object
```

```
Dim FileExtStr As String

DefPath = "C:\Users\Ron\test\"    '<< Change
If Right(DefPath, 1) <> "\" Then
   DefPath = DefPath & "\"
End If

'Create date/time string and the temporary xl* and Zip file name
If Val(Application.Version) < 12 Then
   FileExtStr = ".xls"
Else
   Select Case ActiveWorkbook.FileFormat
   Case 51: FileExtStr = ".xlsx"
   Case 52: FileExtStr = ".xlsm"
   Case 56: FileExtStr = ".xls"
   Case 50: FileExtStr = ".xlsb"
   Case Else: FileExtStr = "notknown"
   End Select
   If FileExtStr = "notknown" Then
      MsgBox "Sorry unknown file format"
      Exit Sub
   End If
End If

strDate = Format(Now, " yyyy-mm-dd h-mm-ss")

FileNameZip = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & ".zip"

FileNameXls = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & FileExtStr

If Dir(FileNameZip) = "" And Dir(FileNameXls) = "" Then

   'Make copy of the activeworkbook
   ActiveWorkbook.SaveCopyAs FileNameXls

   'Create empty Zip File
   NewZip (FileNameZip)

   'Copy the file in the compressed folder
   Set oApp = CreateObject("Shell.Application")
   oApp.Namespace(FileNameZip).CopyHere FileNameXls
```

```
        'Keep script waiting until Compressing is done
        On Error Resume Next
        Do Until oApp.Namespace(FileNameZip).items.Count = 1
           Application.Wait (Now + TimeValue("0:00:01"))
        Loop
        On Error GoTo 0
        'Delete the temporary xls file
        Kill FileNameXls

        MsgBox "Your Backup is saved here: " & FileNameZip

    Else
        MsgBox "FileNameZip or/and FileNameXls exist"

    End If
End Sub
```

Zip and mail the ActiveWorkbook

This will only work if you use Outlook as your mail program

This sub will send a newly created workbook (copy of the Activeworkbook).
It save and zip the workbook before mailing it with a date/time stamp.
After the zip file is sent the zip file and the workbook will be deleted from your hard disk.

```
Sub Zip_Mail_ActiveWorkbook()
    Dim strDate As String, DefPath As String, strbody As String
    Dim oApp As Object, OutApp As Object, OutMail As Object
    Dim FileNameZip, FileNameXls
    Dim FileExtStr As String

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
        DefPath = DefPath & "\"
    End If

    'Create date/time string and the temporary xl* and zip file name
    If Val(Application.Version) < 12 Then
        FileExtStr = ".xls"
    Else
        Select Case ActiveWorkbook.FileFormat
```

```
    Case 51: FileExtStr = ".xlsx"
    Case 52: FileExtStr = ".xlsm"
    Case 56: FileExtStr = ".xls"
    Case 50: FileExtStr = ".xlsb"
    Case Else: FileExtStr = "notknown"
    End Select
    If FileExtStr = "notknown" Then
       MsgBox "Sorry unknown file format"
       Exit Sub
    End If
End If

strDate = Format(Now, " yyyy-mm-dd h-mm-ss")

FileNameZip = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & ".zip"

FileNameXls = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & FileExtStr


If Dir(FileNameZip) = "" And Dir(FileNameXls) = "" Then

    'Make copy of the activeworkbook
    ActiveWorkbook.SaveCopyAs FileNameXls

    'Create empty Zip File
    NewZip (FileNameZip)

    'Copy the file in the compressed folder
    Set oApp = CreateObject("Shell.Application")
    oApp.Namespace(FileNameZip).CopyHere FileNameXls

    'Keep script waiting until Compressing is done
    On Error Resume Next
    Do Until oApp.Namespace(FileNameZip).items.Count = 1
       Application.Wait (Now + TimeValue("0:00:01"))
    Loop
    On Error GoTo 0

    'Create the mail
    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(0)
    strbody = "Hi there" & vbNewLine & vbNewLine & _
```

```
            "This is line 1" & vbNewLine & _
            "This is line 2" & vbNewLine & _
            "This is line 3" & vbNewLine & _
            "This is line 4"

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = strbody
        .Attachments.Add FileNameZip
        .Display   'or use .Send
    End With
    On Error GoTo 0

    'Delete the temporary Excel file and Zip file you send
    Kill FileNameZip
    Kill FileNameXls
  Else
    MsgBox "FileNameZip or/and FileNameXls exist"
  End If
End Sub
```

DATENBANKEN DAO / ADO

-→ Grundlagen DAO / ADO

There are several ways to connect to a database in VBA whether it is a relational database (RDBMS) or a flat-file database (like Excel).

Also, where and what type of database application/server it is will pretty much determine which is considered best for the job.

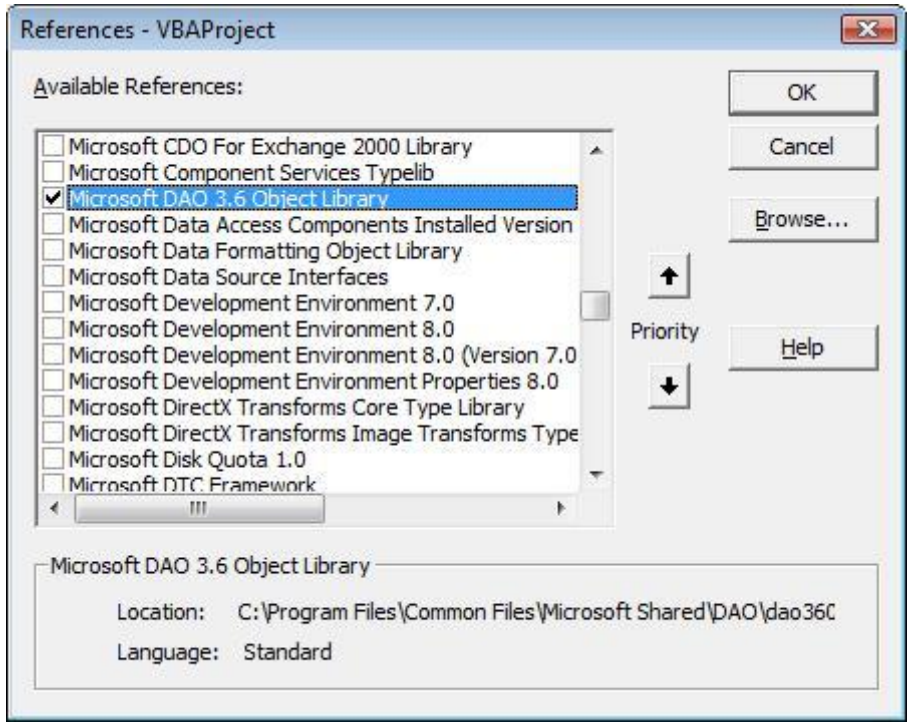
The two I'm going to mention in the article (**DAO** and **ADO**) is considered the more popular techniques deployed but for your reference you may want to investigate the older **RDO** (*Remote Data Object*) which has been really replaced with **DAO**, **OLE-DB** and **ODBC** to help establish which would be best for your solution.

DAO

DAO stands for **D***ata* **A***ccess* **O***bjects* and is one of the technologies to allow communications to external applications (*mainly databases*).

In order to use this feature, users will need to add the **DAO** library to the project.

Choose from the **Tools** menu and select **Reference...**



This library will than allow objects to be created to interrogate a database, tables, fields and return information to populate a spreadsheet. This will also allow users to add, edit, update and delete data to an external file without the need to open the associated application.

An advanced feature of this library will even allow users to create, modify and delete structures of a database whether a table, query, stored procedure or fields.

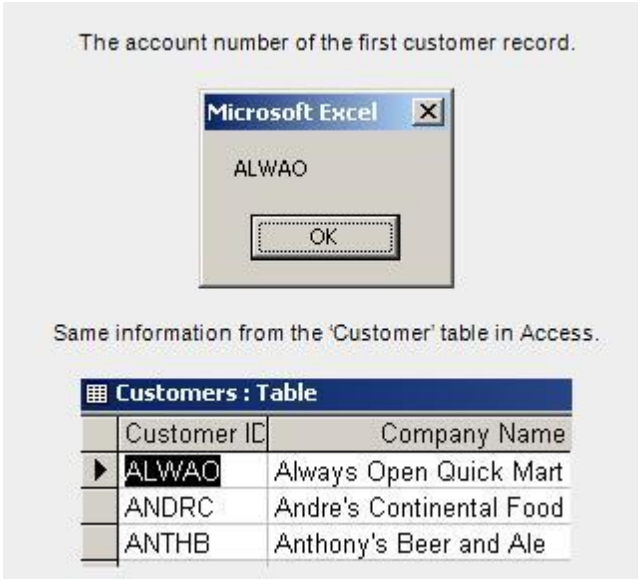
Using the control flow techniques as discussed in this manual, the user can fully control how data should be handled - opening the potential power of VBA.

Note: In order to test this section, users will need an Access database and will need to familiarise themselves with the database. It is not essential to have Microsoft Access loaded as this reference uses the *backdoor* but it will be difficult to check the database without it!

Example - Connecting to a database:

```
Sub ConnectDB()
    Dim db As Database
    Dim rst As Recordset

    Set db = OpenDatabase("C:\db1.mdb")
    Set rst = db.OpenRecordset("Customers")

    'displays the first record and first field
    MsgBox rst.Fields(0)

    'close the objects
    rst.Close
    db.Close

    'destroy the variables
    Set rst = Nothing
    Set db = Nothing
End Sub
```

The above example opens an Access database (db1.mdb) in memory and sets a reference to one of its known tables using the '*OpenRecordset'* method. It then displays the first row and first field of the table:



The property **`Fields`** of the **`RecordSet`** object is a collection (*or array*) that is held in memory and by changing the element number, users can return a different field (column of the table).

```
rst.Fields(1)
```

The above illustration would show the customer's name instead of the ID number.

A good discipline is to close and set an object to **`Nothing`** that releases memory, hence the last four lines of code.

Example - Working with records:

```
'Opens a connection to the Customers table
'and populates a blank worksheet.
Sub PopulateCustomers()
     Dim db As Database
     Dim rst As Recordset
     Dim i As Long

     Set db = OpenDatabase("C:\db1.mdb")
```

```
    Set rst = db.OpenRecordset("Customers")

    'look through each record and populate
    'ID, Name and Country into a worksheet.
    Do Until rst.EOF
       ActiveCell.Offset(i, 0).Value = rst.Fields(0)
       ActiveCell.Offset(i, 1).Value = rst.Fields(1)
       ActiveCell.Offset(i, 2).Value = rst.Fields(8)
       i = i + 1
       rst.MoveNext
    Loop

   'close the objects
   rst.Close
   db.Close

   'destroy the variables
   Set rst = Nothing
   Set db = Nothing
End Sub
```

The above example once again opens the table 'Customers'. Using a conditional loop at which point the property **EOF** (*End Of File*) returns **True** or **False** every time the record changes using the **MoveNext** method, three columns in the worksheet from the starting active cell are populated by three different field indexes.

Even though the above example used **rst.Fields(8)** to determine the ninth column, it may be fair to say that users may not know the position number of the field but instead know its fieldname. In this case, users can refer to the name of the field as a string argument.

```
rst.Fields("Post Code").
```

Note: Be careful to include a command to increment the collection (**MoveNext** method) otherwise this would cause the procedure to loop infinitely or run out of worksheet rows firing an error. **Save your work first before testing the above.**

When interrogating a table in a database, it may be required to test to see if the table actually has records in it before iterating through each record.

Wrap an **If** statement around the loop to test this out:

```
If Not rst.EOF And Not rst.BOF Then

       [code here]...
```

```
End If
```

If this returns **True** then at least one record is present. If both **EOF** and **BOF** are **True**, it means the cursor is positioned at the beginning and at the end of the record set (*which means it's empty*).

The **Not** keyword inverses the returning value which means that in the above example, both must be **False** if this is to run any code in between the statement.

Example - Editing records in a database:

Not only can users populate data from an external database, but also it is possible to change data in an external database.

```
'Opens a connection to the table Customers
'and adds a new record and then updates and closes
Sub AddNewRecord()
    Dim db As Database
    Dim rst As Recordset

    Set db = OpenDatabase("C:\db1.mdb")
    Set rst = db.OpenRecordset("Customers")

    rst.AddNew
    rst.Fields("Customer ID") = "XYZ"
    rst.Fields("Company Name") = "XYZ Foods Ltd"
    rst.Fields("Post Code") = "NW1 8PY"
    rst.Update

    'close the objects
    rst.Close
    db.Close

    'destroy the variables
    Set rst = Nothing
    Set db = Nothing
End Sub
```

The above example once again opens a connection to the '*Customer'* table and then uses two methods to add and update the new record.

The **Add** method triggers the mode to add the record but does not save it to the table until you call the **Update** method.

Note: Be careful to consider the table's structure and database rules that are often implemented such as primary keys and foreign indexes. The above example would fail if the customer id field was a unique primary key and the table already had such a reference.

Further coding would be required to test to see if the record number existed, before adding and updating the record.

To edit a record, users must first locate the record (*if it can be found*) and then use the **`Edit`** method.

```
rst.Edit
rst.Fields("Customer ID") = "XYZ"
rst.Fields("Company Name") = "XYZ Foods Ltd"
rst.Fields("Post Code") = "W12 6RF"
rst.Update
```

Example - Creating a table:

```
'Opens a connection to the table Customers
'and adds a new record and then updates and closes
Sub CreateTable()
    Dim db As Database
    Dim rst As Recordset
    Dim tbl As TableDef

    Set db = OpenDatabase("C:\db1.mdb")
    Set tbl = db.CreateTableDef("Contact Log")

    With tbl
        .Fields.Append .CreateField("Log ID", dbInteger)
        .Fields.Append .CreateField("Date", dbDate)
        .Fields.Append .CreateField("Caller", dbText)
        .Fields.Append .CreateField("Comment", dbText)
        .Fields.Append .CreateField("Completed", dbBoolean)
        db.TableDefs.Append tbl
    End With

    Set rst = db.OpenRecordset("Contact Log")
    rst.AddNew
    rst.Fields("Log ID") = 1
    rst.Fields("Date") = Date
    rst.Fields("Caller") = "Ben Beitler"
    rst.Fields("Comment") = "Arranged VBA training next week."
    rst.Fields("Completed") = True
    rst.Update

    'close the objects
    rst.Close
    db.Close

    'destroy the variables
    Set rst = Nothing
    Set tbl = Nothing
    Set db = Nothing
End Sub
```

The above example will create a new table '*Contact Log'*, create new fields and then bind it to the new table using **`db.TableDefs.Append tbl`**. Next it will add a single record using the correct data to match the data types as defined in the table.

This procedure will only run once and then cause an error if executed again. This is due to the fact this database cannot contain duplicate named tables.

Therefore, users need to add error-handling procedures as well as testing to see if the table exists.

To delete a table along with its records, use `db.TableDefs.Delete "Contact Log"`.

Again an error will be fired if the system cannot locate the table (*misspelling or already deleted*).

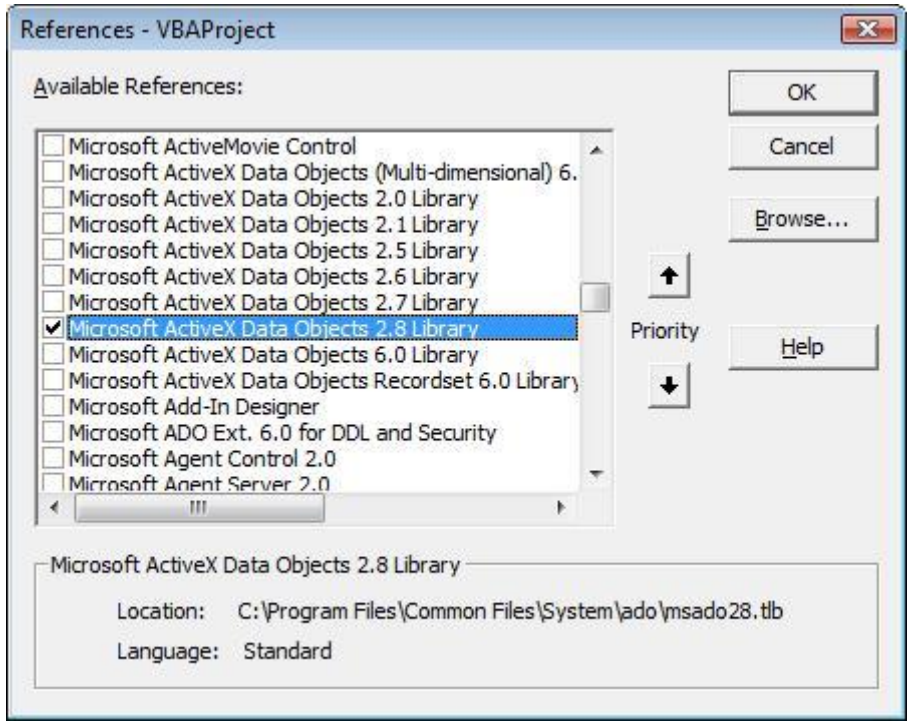
There are many properties and methods of **DAO** which are not covered in this guide. This library allows many ways to produce the same effect which include writing SQL (structured query language).

ADO

ADO stands for **A***ctiveX* **D***ata* **O***bjects* is an alternative method of connecting to a database.

In order to use this feature, users will need to add the **ADO** library to the project.

Choose from the **Tools** menu and select **Reference...**



Note: You may have noticed that there several versions of ActiveX Data Objects in the illustration above. Generally, you should choose the latest version but depending on which version of Excel (or more accurately Windows operating system) try and pick the best fit version. For example 2.8 is for those running on Windows XP where as users would choose 6.0 for Windows Vista.

This library will than allow objects to be created to interrogate a database, tables, fields and return information to populate a spreadsheet. This will also allow users to add, edit, update and delete data to an external file without the need to open the associated application.

An advanced feature of this library (**ADOX**) will even allow users to create, modify and delete structures of a database whether a table, query, stored procedure or fields.

Using the control flow techniques as discussed in this manual, the user can fully control how data should be handled - opening the potential power of VBA.

Note: In order to test this section, users will need an Access database and will need to familiarise themselves with the database. It is not essential to have Microsoft Access loaded as this reference uses the *backdoor* but it will be difficult to check the database without it!

Example - Connecting to a database:

```
Sub ConnectExcelDB()
    Dim cn as ADODB.Connection

    Set cn = New ADODB.Connection

    With cn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.ConnectionString = "Data Source=C:\pivot data.xls;" & _
            "Extended Properties=Excel 8.0;"
.Open
    End With
End Sub
```

The above example creates a connection and open the workbook 'pivot data'. It requires the **Extended Properties=Excel 8.0** argument (which users need to adjust for their own version of Excel).

```
Sub ConnectAccessDB()
    Dim cn As ADODB.Connection

    Set cn = New ADODB.Connection

    With cn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.ConnectionString = "Data Source=C:\db2.mdb;"
.Open
    End With
End Sub
```

The above example connects to an Access database (db2).

There other ways to connect as well as setting optional arguments which control the method of connection (using **ODBC** or **DSN**-Less etc) which is beyond this article.

Example - Reading from a database:

Using an Access database, users can connect to table, query or write SQL (structured query language) into the calling object.

```
Sub ReadingData()
    Dim cn As ADODB.Connection
    Dim rs As ADODB.Recordset

    Set cn = New ADODB.Connection

    With cn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.ConnectionString = "Data Source=C:\db2.mdb;"
.Open
    End With

    Set rs = New ADODB.Recordset
    'opens a connection to a table called customers.
    rs.Open "Customers", cn, adOpenKeyset, adLockOptimistic, adCmdTable
    'show the second fields value, first record (column 2, row 1).
    Debug.Print rs.Fields(1).Value 'second columns - starts at 0

    rs.Close
    cn.Close

    Set rs = Nothing
    Set cn = Nothing
End Sub
```

The above example creates a connection. It then creates another new object (**rs**) which the recordset of a table, query or SQL source and opens it too.

Now you have a collection of data (all records in that file). Using a property (**Fields**), you can pass either an index or string name into it to refer to any field in that source file and return one of several values (in this case the data value).

Make sure you close and dispose of the objects when finished (and in the correct order) though it will clear and dispose of all objects when the procedure comes to an end - *just good habits of programming!*

To refer to an actual field instead of an index, use **Fields("Customer Name")**.

It is good practice to narrow down the recordset to the smallest amount of data in memory which the above example fails to do (all records). Instead, consider passing a query or SQL statement instead:

```
rs.Open "Select * From Customers Where Country='UK';", cn .....
```

There are optional arguments which also help performance and restrictions to an open connection which I've used in my example above **adOpenKeyset, adLockOptimistic, adCmdTable** and will require further investigation to help establish the rule (refer to VBA help for more information).

Example - Writing to a database:

Create a connection and open a recordset (table) to add, edit and delete records.

```
Sub EditingData()
    Dim cn As ADODB.Connection
    Dim rs As ADODB.Recordset

    Set cn = New ADODB.Connection

    With cn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.ConnectionString = "Data Source=C:\db2.mdb;"
.Open
    End With

    Set rs = New ADODB.Recordset
    rs.Open "Customers", cn, adOpenDynamic, adLockOptimistic
    'edit the second field, first record's value.
    rs.Fields(1).Value = "Always Open QM"
    rs.Update 'save the changes.

    rs.Close
    cn.Close

    Set rs = Nothing
    Set cn = Nothing
End Sub
```

Using the **rs.Update** property enforces any changes to be saved and written to the database.

If you wish add a new record, you can use **rs.AddNew** method but it will still need to use **rs.Update** to save the changes.

Example:

```
Sub NewRecord()
    Dim cn As ADODB.Connection
    Dim rs As ADODB.Recordset

    Set cn = New ADODB.Connection

    With cn
.Provider = "Microsoft.Jet.OLEDB.4.0"
.ConnectionString = "Data Source=C:\db2.mdb;"
.Open
    End With

    Set rs = New ADODB.Recordset

    rs.Open "Customers", cn, adOpenDynamic, adLockOptimistic
    'adding a new record.
    rs.AddNew
    rs.Fields(0).Value = "XYZ" 'customer ID field
    rs.Fields(1).Value = "XYZ Limited" 'customer name field
    rs.Fields(5).Value = "London" 'city field
    rs.Fields(8).Value = "UK" 'country field
    rs.Update 'save the changes.

    rs.Close
    cn.Close

    Set rs = Nothing
    Set cn = Nothing
End Sub
```

The above example populates new values to four fields and then saves the changes. Make sure any record being added satisfies the rules of the data source which is being used to store the data which will include indexing (which is generally a mandatory field).

Other useful methods include **EOF** (*end of file*) and **BOF** (*beginning of file*) which allows you to iterate through records using loops. - *look at the help for more information.*

There is much, much more on this subject (*I've not done this justice*) and users should now be confident to go off and investigate further using various other resources (books and the web!).

Finally, which one to use **DAO** or **ADO**?

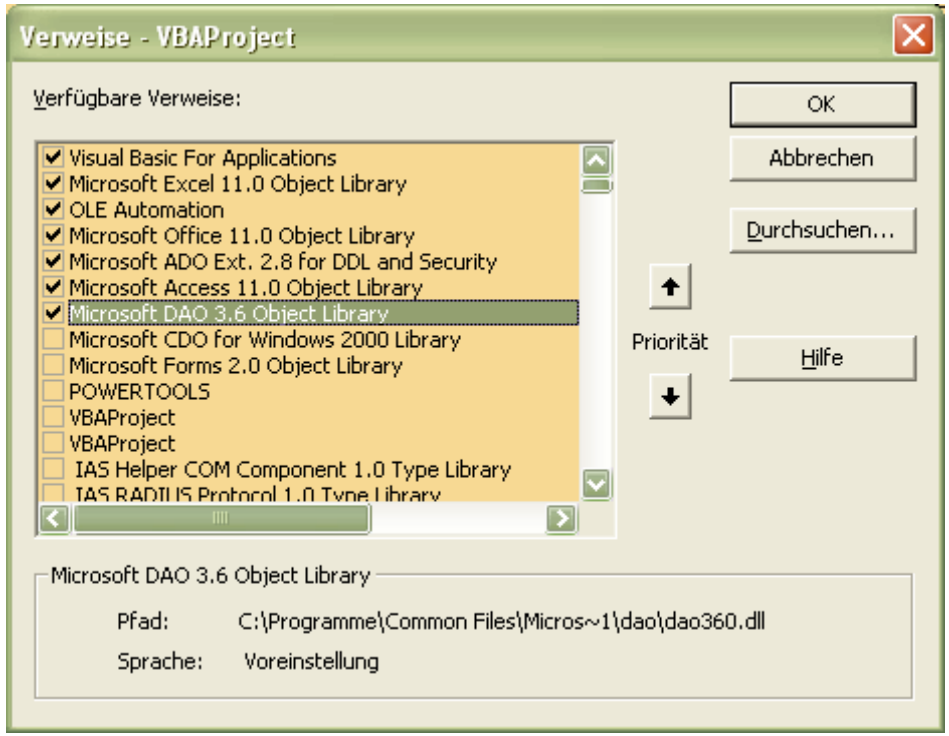
There are many arguments which one should use but as a general rule if you are going to communicate with Microsoft '*Jet*' engine (Access, SQL etc) then using **DAO** is quicker and easier to master.

Consider using **ADO** for across platform applications typically over the web (server) and non-Microsoft Window environments which have the capability to create DSN-less connections. It also handles multiple databases at the same time and is considered the standard with other programming languages.

Both have similar members (methods and properties) and can conflict if both are being referenced in the same module.

ACCESS-Datenbank erstellen aus Exceldatei

Es sollten immer unbedingt folgende Verweise eingestellt werden:




```
Sub DAOFromExcelToAccess()
' exports data from the active worksheet to a table in an Access database
' this procedure must be edited before use
Dim db As Database, rs As Recordset, r As Long
    Set db = OpenDatabase("C:\FolderName\DataBaseName.mdb")
    ' open the database
    Set rs = db.OpenRecordset("TableName", dbOpenTable)
    ' get all records in a table
    r = 3 ' the start row in the worksheet
    Do While Len(Range("A" & r).Formula) > 0
    ' repeat until first empty cell in column A
        With rs
            .AddNew ' create a new record
            ' add values to each field in the record
            .Fields("FieldName1") = Range("A" & r).Value
            .Fields("FieldName2") = Range("B" & r).Value
            .Fields("FieldNameN") = Range("C" & r).Value
```

```
            ' add more fields if necessary...
            .Update ' stores the new record
        End With
        r = r + 1 ' next row
    Loop
    rs.Close
    Set rs = Nothing
    db.Close
    Set db = Nothing
End Sub
```

2.MÖGLICHKEIT

```
'als Verweis: -> MS DAO 3.6. xxx
' -> MS Access xxx

Dim db As Access.Application

Set db = CreateObject("Access.Application") ' Objekt erstellen
db.OpenCurrentDatabase (TextBox2.Value) ' Datenbank öffnen

DoCmd.TransferSpreadsheet , acSpreadsheetTypeExcel9, tablename:="Tabellenname", Filename:="Pfad der Datenbank", Hasfieldnames:=True

DoCmd.Close
```

3. MÖGLICHKEIT

Eine CSV-Datei umwandeln nach ACCESS-Datenbank

```
Dim appAccess As Access.Application



Sub CSV_DATEI_NACH_ACCESS()
   Dim dbs As Database, tdf As TableDef, fld As Field, s%
```

```
Dim strDB As String
' Pfad der Datenbank
strDB = "C:\OPLeXor\Export\Test.mdb"
' Eine neue Instanz von Microsoft Access erstellen - 11 ist für Access 2003
Set appAccess = CreateObject("Access.Application.11")
' Die Datenbank im Microsoft Access-Fenster öffnen.
Application.DisplayAlerts = False
Kill strDB
appAccess.NewCurrentDatabase strDB
Application.DisplayAlerts = True
' Die Database-Objektvariable einstellen.
Set dbs = appAccess.CurrentDb
' Eine neue Tabelle erstellen.
Set tdf = dbs.CreateTableDef("Offene Posten")
' Felder für die neue Tabelle erstellen und anfügen
Set fld = tdf.CreateField("Konto", dbLong): tdf.Fields.Append fld
Set fld = tdf.CreateField("KtoBez", dbText): tdf.Fields.Append fld
Set fld = tdf.CreateField("RechNr", dbLong): tdf.Fields.Append fld
Set fld = tdf.CreateField("Datum", dbText): tdf.Fields.Append fld
Set fld = tdf.CreateField("BetragS", dbDouble): tdf.Fields.Append fld
Set fld = tdf.CreateField("BetragH", dbDouble): tdf.Fields.Append fld
Set fld = tdf.CreateField("Saldo", dbDouble): tdf.Fields.Append fld
'Das TableDef-Objekt anfügen.
dbs.TableDefs.Append tdf
' CSV - Datei importieren
Application.DisplayAlerts = False
' Wichtig: wenn ACCESS von User parallel bereits geöffnet ist - auch wenn ohne Datenbank
' führt folgende Zeile zu einem Fehler - daher unbedingt ACCESS vor Ausführen dieses Codes schließen
DoCmd.TransferText , , "Offene Posten", "C:\OPLeXor\Export\OP_Export.csv", True
' Fehler-Tabelle löschen, die von Access automatisch erzeugt wird
DoCmd.DeleteObject acTable, "OP_Export_Importfehler"
tdf.Fields("Konto").Name = "Konto"
tdf.Fields("KtoBez").Name = "KtoBez"
tdf.Fields("RechNr").Name = "RechNr"
tdf.Fields("Datum").Name = "Datum"
tdf.Fields("BetragS").Name = "BetragS"
tdf.Fields("BetragH").Name = "BetragH"
tdf.Fields("Saldo").Name = "Saldo"

' Datenbank schließen
appAccess.CloseCurrentDatabase
'Access beenden
Access.Application.Quit
Set appAccess = Nothing
```

```
End Sub



Sub EXPORT_Excel_nach_Access()

' Daten aus der aktuellen Tabelle in eine ACCESS-Datenbank exportieren

Dim db As Database, rs As Recordset, r As Long
   Set db = OpenDatabase("C:\OPLexor\Export\EX_Demo.mdb")
   ' Datenbank öffnen
   Set rs = db.OpenRecordset("TableName", dbOpenTable)
   ' Datensätze in der Tabelle
   r = 1 ' die Startzeile in der Tabelle
   Do While Len(Range("A" & r).Formula) > 0
   ' Schleife bis erste leere Zelle in Spalte A
      With rs
         .AddNew ' neuen Datensatz erstellen
         ' füge Werte in die Datenfelder ein
         .Fields("FieldName1") = Range("A" & r).Value
         .Fields("FieldName2") = Range("B" & r).Value
         .Fields("FieldNameN") = Range("C" & r).Value

         .Update ' Datenbank beschreiben
      End With
      r = r + 1 ' Nächste Zeile
   Loop
   rs.Close ' Schließen der Tabelle
   Set rs = Nothing
   db.Close ' Schließen der Datenbank
   Set db = Nothing
End Sub
```

ACCESS-Datenbank auslesen

Copy data from an Access database into Excel with ADO

Ron de Bruin (last update 18 Feb 2006)
Go to the Excel tips page

Most examples on the internet are not so easy to work with for a normal user like me.
I will try to change that on this webpage.

We use small macro's that are not so difficult to use/change that call one big macro named GetDataFromAccess.
You can download a zip file on the bottom of this page with two Excel workbooks and the OrderDatabase.mdb.
In one workbook you can find all code from this page and in the other one it is very easy get the info you want
because you can save the criteria (100 or more) and can use Data>Validation cells to select your criteria.

Note: the workbooks and the OrderDatabase.mdb must be in the same folder

In the OrderDatabase.mdb there is a table named Orders with the following fields:

OrderNumber
OrderDate
RequiredDate
ShippedDate
Freight
ShipVia
ShipCountry
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode

Below you find a few example macro's that you can use to retrieve only the records you want

First line: Path/name of the Access file, Table name
Second-Eighth line: You can fill in seven criteria, and if you not fill in any criteria it return all records
The first three criteria are only for Text fields
The fourth and fifth are for numbers fields
The sixth and seventh are for date fields
Line nine: Destination sheet/range
Line ten: Which fields (* = all), Copy field names, clear all cells on Destination sheet first

Note: If you use criteria 4-7 (number or Date fields) you can change >, <, >=, <= to get the result you want.

```
Sub Test1()
'This example retrieves the data for the records in which ShipCountry = Germany
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
```

```
                "ShipCountry", "=", "Germany", _
                "", "=", "", _
                "", "=", "", _
                "", ">", "", _
                "", "<", "", _
                "", ">=", "", _
                "", "<=", "", _
                Sheets("test").Range("A8"), _
                "*", True, True
End Sub


Sub Test2()
'This example retrieves also the data for the records in which ShipCountry = Germany
'It only retrieves this four fields: OrderNumber, ShipName, ShipAddress, ShipPostalCode
'I changed the "*" for WhichFields in the code to the names of the fields
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
                "ShipCountry", "=", "Germany", _
                "", "=", "", _
                "", "=", "", _
                "", ">", "", _
                "", "<", "", _
                "", ">=", "", _
                "", "<=", "", _
                Sheets("test").Range("A8"), _
                "OrderNumber, ShipName, ShipAddress, ShipPostalCode", True, True
End Sub


Sub Test3()
'This example retrieves the data for the records in which
'ShipCountry = Germany and ShipVia = Speedy Express
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
                "ShipCountry", "=", "Germany", _
                "ShipVia", "=", "Speedy Express", _
                "", "=", "", _
                "", ">", "", _
                "", "<", "", _
                "", ">=", "", _
                "", "<=", "", _
                Sheets("test").Range("A8"), _
                "*", True, True
End Sub


Sub Test4()
```

```
'This example retrieves the data for the records in which
'ShipCountry = Germany and ShipVia = Speedy Express
'and Freight = between 100 and 300
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
                      "ShipCountry", "=", "Germany", _
                      "ShipVia", "=", "Speedy Express", _
                      "", "=", "", _
                      "Freight", ">", "100", _
                      "Freight", "<", "300", _
                      "", ">=", "", _
                      "", "<=", "", _
                      Sheets("test").Range("A8"), _
                      "*", True, True
End Sub


Sub Test5()
'This example retrieves the data for the records in which
'ShipCountry = Germany and ShipVia = Speedy Express
'and ShippedDate = between 1/1/1998 and 3/1/1998
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
                      "ShipCountry", "=", "Germany", _
                      "ShipVia", "=", "Speedy Express", _
                      "", "=", "", _
                      "", ">", "", _
                      "", "<", "", _
                      "ShippedDate", ">=", "1/1/1998", _
                      "ShippedDate", "<=", "3/1/1998", _
                      Sheets("test").Range("A8"), _
                      "*", True, True
End Sub


Sub Test6()
'This example retrieves all records
    GetDataFromAccess ThisWorkbook.Path & "/OrderDatabase.mdb", "Orders", _
                      "", "=", "", _
                      "", "=", "", _
                      "", "=", "", _
                      "", ">", "", _
                      "", "<", "", _
                      "", ">=", "", _
                      "", "<=", "", _
                      Sheets("test").Range("A8"), _
                      "*", True, True
End Sub
```

The Big macro

```
Public Sub GetDataFromAccess(MyDatabaseFilePathAndName As String, MyTable As String, _
                    MyTableField1 As String, S1 As String, MyFieldValue1 As String, _
                    MyTableField2 As String, S2 As String, MyFieldValue2 As String, _
                    MyTableField3 As String, S3 As String, MyFieldValue3 As String, _
                    MyTableField4 As String, S4 As String, MyFieldValue4 As String, _
                    MyTableField5 As String, S5 As String, MyFieldValue5 As String, _
                    MyTableField6 As String, S6 As String, MyFieldValue6 As String, _
                    MyTableField7 As String, S7 As String, MyFieldValue7 As String, _
                    DestSheetRange As Range, WhichFields As String, _
                    FieldNames As Boolean, ClearRange As Boolean)

'Date changed : 18 Feb 2006
'Add the WhichFields option to copy only the fields you want

    Dim MyConnection As String
    Dim MySQL As String
    Dim MyDatabase As Object
    Dim col As Integer
    Dim I As Integer
    Dim str1 As Variant
    Dim str2 As Variant
    Dim str3 As Variant

    'Select the DestSheetRange where you paste the records
    Application.Goto DestSheetRange

    'If ClearRange = True it clear all cells on that sheet first
    If ClearRange Then Range(DestSheetRange.Address, "IV" & Rows.Count).ClearContents

    'Create connection string
    MyConnection = "Provider=Microsoft.Jet.OLEDB.4.0;"
    MyConnection = MyConnection & "Data Source=" & MyDatabaseFilePathAndName & ";"

    ' Create MySQL string
    str1 = Array(MyTableField1, MyTableField2, MyTableField3, MyTableField4, MyTableField5, MyTableField6, MyTableField7)
    str2 = Array(S1, S2, S3, S4, S5, S6, S7)
    str3 = Array(MyFieldValue1, MyFieldValue2, MyFieldValue3, MyFieldValue4, MyFieldValue5, MyFieldValue6, MyFieldValue7)
    MySQL = ""
```

```
For I = LBound(str1) To UBound(str1)
   If str3(I) <> "" Then
      If MySQL = "" Then
         If I <= 2 Then
            MySQL = "SELECT " & WhichFields & " FROM " & MyTable & " WHERE [" _
                  & str1(I) & "] " & str2(I) & " '" & str3(I) & "'"
         ElseIf I = 3 Or I = 4 Then
            MySQL = "SELECT " & WhichFields & " FROM " & MyTable & " WHERE [" _
                  & str1(I) & "] " & str2(I) & " " & str3(I)
         ElseIf I = 5 Or I = 6 Then
            MySQL = "SELECT " & WhichFields & " FROM " & MyTable & " WHERE [" _
                  & str1(I) & "] " & str2(I) & " #" & str3(I) & "#"
         End If
      Else
         If I <= 2 Then
            MySQL = MySQL & " and [" & str1(I) & "] " & str2(I) & " '" & str3(I) & "'"
         ElseIf I = 3 Or I = 4 Then
            MySQL = MySQL & " and [" & str1(I) & "] " & str2(I) & " " & str3(I)
         ElseIf I = 5 Or I = 6 Then
            MySQL = MySQL & " and [" & str1(I) & "] " & str2(I) & " #" & str3(I) & "#"
         End If
      End If
   End If
Next I

'If MySQL is empty copy all records
If MySQL = "" Then MySQL = "SELECT " & WhichFields & " FROM " & MyTable & ";"

' Open the database and copy the data
On Error GoTo SomethingWrong
Set MyDatabase = CreateObject("adodb.recordset")
MyDatabase.Open MySQL, MyConnection, 0, 1, 1

' Check to make sure we received data and copy the data
If Not MyDatabase.EOF Then

   'If FieldNames = True copy the field names and records
   'If = False copy only records
   If FieldNames Then
      For col = 0 To MyDatabase.Fields.Count - 1
         DestSheetRange.Offset(0, col).Value = MyDatabase.Fields(col).Name
      Next
      DestSheetRange.Offset(1, 0).CopyFromRecordset MyDatabase
   Else
      DestSheetRange.CopyFromRecordset MyDatabase
   End If
```

```
    Else
        MsgBox "No records returned from : " & MyDatabaseFilePathAndName, vbCritical
    End If

    MyDatabase.Close
    Set MyDatabase = Nothing
    Exit Sub

SomethingWrong:
    On Error GoTo 0
    Set MyDatabase = Nothing
    MsgBox "Error copying data", vbCritical, "Test Access data to Excel"
End Sub
```

Tip

Instead of enter field values in the code you can also use a cell value
"ShipVia", "=", Sheets("Sheet1").Range("A2").Value

Check out "Example to save criteria.xls" where I use data validation cells with unique values from the fields and you can also save your criteria (100 or more).

Download

Download the Example workbooks and database

Tips

After you retrieve the records into Excel you can always use
the Excel filters or my EasyFilter Add-in to filter/Copy to a different Sheet/Workbook.
http://www.rondebruin.nl/easyfilter.htm

See also this page from Ole P. Erlandsen's
http://www.erlandsendata.no/english/index.php?t=envbadac

ACCESS-Datenbank: Datensatz hinzufügen

Mit folgendem Code kann ein Datensatz an eine Access-Datenbank angefügt werden: Annahmen: Datenbank = Test.mdb

```
Sub TestAdd()
Dim db As Database
Dim rs As Recordset
Set db = OpenDatabase("C:\Test.mdb")
Set rs = db.OpenRecordset(Name:="Test", Type:=dbOpenDynaset)
With rs
.AddNew
.Fields("Name").Value = Range("A1")
.Fields("Alter").Value = Range("A2")
.Update
End With
rs.Close
db.Close
Set rs = Nothing
End Sub
```

Export SQL-Data to CSV

```
' ------------------------------------------------------------------------
'
'  Title: Export sql data to a CSV File
'


' ------------------------------------------------------------------------

' ********************************************************
' * Programmer Name  : Waty Thierry
' * Module Name      : Database_Module
' * Module Filename  : Database.bas
' * Procedure Name   : CSVExport
' * Parameters       :
' *                    db As DAO.Database
' *                    sSQL As String
' *                    sDest As String
' ********************************************************
' * Comments         : Export sql data to a CSV File
' *
' *
' ********************************************************

Public Function CSVExport(db As DAO.Database, sSQL As String, sDest As String) As Boolean
```

```
    Dim record        As Recordset
    Dim nI            As Long
    Dim nJ            As Long
    Dim nFile         As Integer
    Dim sTmp          As String

    On Error GoTo Err_Handler

    Set record = db.OpenRecordset(sSQL, DAO.dbOpenDynaset, DAO.dbReadOnly)

    ' *** Open output file
    nFile = FreeFile

    Open sDest For Output As #nFile

    ' *** Export fields name
    For nI = 0 To record.Fields.Count - 1
       sTmp = "" & (record.Fields(nI).Name)
       Write #nFile, sTmp;
    Next
    Write #nFile,

    If record.RecordCount > 0 Then
       record.MoveLast
       record.MoveFirst

       For nI = 1 To record.RecordCount
          For nJ = 0 To record.Fields.Count - 1
             sTmp = "" & (record.Fields(nJ))
             Write #nFile, sTmp;
          Next
          Write #nFile,
          record.MoveNext
       Next
    End If

    Close #nFile
    CSVExport = True

    Exit Function

Err_Handler:
```

```
   MsgBox ("Error: " & Err.Description)
   CSVExport = False
End Function
```

Retreive ALL the information about Access Table & Fields using ADO

```
' ---------------------------------------------------------------------------
'  Title: Retreive ALL the information about Access Table & Fields using ADO
' ' ---------------------------------------------------------------------------

' Declarations

Option Explicit

'Properties of the Catalog
Private Catalog As ADOX.Catalog
Private Col     As ADOX.Column
Private Cols    As ADOX.Columns
Private Grp     As ADOX.Group
Private Grps    As ADOX.Groups
Private Ndx     As ADOX.Index
Private Ndxs    As ADOX.Indexes
Private Key     As ADOX.Key
Private Keys    As ADOX.Keys
Private Proc    As ADOX.Procedure
Private Procs   As ADOX.Procedures
Private Prop    As ADOX.Property
Private Props   As ADOX.Properties
Private Table   As ADOX.Table
Private Tables  As ADOX.Tables
Private User    As ADOX.User
Private Users   As ADOX.Users
Private View    As ADOX.View
Private Views   As ADOX.Views
```

```
Public Enum TblProps
    tblTempTable = 0
    tblValidationText = 1
    tblValidationRule = 2
    tblCacheLinkNamePassword = 3
    tblRemoteTableName = 4
    tblLinkProviderString = 5
    tblLinkDataSource = 6
    tblExclusiveLink = 7
    tblCreateLink = 8
    tblTableHiddenInAccess = 9
End Enum

Public Enum ColProps
    colAutoincrement = 0
    colDefault = 1
    colDescription = 2
    colNullable = 3
    colFixedLength = 4
    colSeed = 5
    colIncrement = 6
    colValidationText = 7
    colValidationRule = 8
    colIISNotLastColumn = 9
    colAutoGenerate = 10
    colOneBlobPerPage = 11
    colCompressedUnicode = 12
    colAllowZeroLength = 13
    colHyperlink = 14
End Enum


' Code

Public Function ColumnFormat(TableName As String, Column As Variant) As Variant
    'return variant because we do not
    'know the type of data that is going
    'to be returned to calling method
    On Error GoTo ErrHandler

    Set Table = Tables(TableName)
```

```
    Set Cols = Table.Columns
    Set Col = Cols(Column)

    ColumnFormat = NumberFormat(Col.Type)
ExitHere:
    Set Table = Nothing
    Set Cols = Nothing
    Set Col = Nothing
Exit Function
ErrHandler:
    ColumnFormat = ""
    Resume ExitHere
End Function

Public Function ColumnProperty(TableName As String, Column As Variant, Property As ColProps) As Variant
    'return variant because we do not
    'know the type of data that is going
    'to be returned to calling method
    On Error GoTo ErrHandler

    Set Table = Tables(TableName)
    Set Cols = Table.Columns
    Set Col = Cols(Column)

    ColumnProperty = Col.Properties(Property).Value

ExitHere:
    Set Table = Nothing
    Set Cols = Nothing
    Set Col = Nothing

    Exit Function

ErrHandler:
    ColumnProperty = ""
    Resume ExitHere
End Function

Public Function TableProperty(TableName As String, Property As TblProps) As Variant
    'return variant because we do not
    'know the type of data that is going
    'to be returned to calling method
```

```
    On Error GoTo ErrHandler

    Set Table = Tables(TableName)
    Set Props = Table.Properties
    TableProperty = Table.Properties(Property).Value

ExitHere:
    Set Table = Nothing
    Set Props = Nothing

    Exit Function

ErrHandler:
    TableProperty = Nothing
    Resume ExitHere
End Function

Private Function NumberFormat(ColType As ADODB.DataTypeEnum) As String
    Select Case ColType
        Case adEmpty                                                    '  0 - No value was specified (DBTYPE_EMPTY).
        Case adSmallInt:          NumberFormat = "General Number"       '  2 - A 2-byte signed integer (DBTYPE_I2).
        Case adInteger:           NumberFormat = "General Number"       '  3 - A 4-byte signed integer (DBTYPE_I4).
        Case adSingle:            NumberFormat = "General Number"       '  4 - A single-precision floating point value
(DBTYPE_R4).
        Case adDouble:            NumberFormat = "General Number"       '  5 - A double-precision floating point value
(DBTYPE_R8).
        Case adCurrency:          NumberFormat = "Currency"             '  6 - A currency value (DBTYPE_CY). Currency is a
fixed-point number with four digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by
10,000.
        Case adDate:              NumberFormat = "General Date"         '  7 - A Date value (DBTYPE_DATE). A date is stored
as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is
the fraction of a day.
        Case adBSTR                                                     '  8 - A null-terminated character string (Unicode)
(DBTYPE_BSTR).
        Case adIDispatch                                                '  9 - A pointer to an IDispatch interface on an OLE
object (DBTYPE_IDISPATCH).
        Case adError                                                    ' 10 - A 32-bit error code (DBTYPE_ERROR).
        Case adBoolean:           NumberFormat = "True/False"           ' 11 - A Boolean value (DBTYPE_BOOL).
        Case adVariant                                                  ' 12 - An Automation Variant (DBTYPE_VARIANT).
        Case adIUnknown                                                 ' 13 - A pointer to an IUnknown interface on an OLE
object (DBTYPE_IUNKNOWN).
```

```
        Case adDecimal:          NumberFormat = "Standard"          ' 14 - An exact numeric value with a fixed precision
and scale (DBTYPE_DECIMAL).
        Case adTinyInt:          NumberFormat = "General Number"    ' 16 - A 1-byte signed integer (DBTYPE_I1).
        Case adUnsignedTinyInt:  NumberFormat = "General Number"    ' 17 - A 1-byte unsigned integer (DBTYPE_UI1).
        Case adUnsignedSmallInt: NumberFormat = "General Number"    ' 18 - A 2-byte unsigned integer (DBTYPE_UI2).
        Case adUnsignedInt:      NumberFormat = "General Number"    ' 19 - A 4-byte unsigned integer (DBTYPE_UI4).
        Case adUnsignedBigInt:   NumberFormat = "General Number"    ' 21 - An 8-byte unsigned integer (DBTYPE_UI8).
        Case adBigInt:           NumberFormat = "General Number"    ' 20 - An 8-byte signed integer (DBTYPE_I8).
        Case adGUID                                                 ' 72 - A globally unique identifier (GUID)
(DBTYPE_GUID).
        Case adBinary                                               '128 - A binary value (DBTYPE_BYTES).
        Case adChar                                                 '129 - A String value (DBTYPE_STR).
        Case adWChar                                                '130 - A null-terminated Unicode character string
(DBTYPE_WSTR).
        Case adNumeric:          NumberFormat = "General Number"    '131 - An exact numeric value with a fixed precision
and scale (DBTYPE_NUMERIC).
        Case adUserDefined                                          '132 - A user-defined variable (DBTYPE_UDT).
        Case adDBDate:           NumberFormat = "General Date"      '133 - A date value (yyyymmdd) (DBTYPE_DBDATE).
        Case adDBTime:           NumberFormat = "Long Time"         '134 - A time value (hhmmss) (DBTYPE_DBTIME).
        Case adDBTimeStamp:      NumberFormat = "General Date"      '135 - A date-time stamp (yyyymmddhhmmss plus a
fraction in billionths) (DBTYPE_DBTIMESTAMP).
        Case adVarChar                                              '200 - A String value (Parameter object only).
        Case adLongVarChar                                          '201 - A long String value (Parameter object only).
        Case adVarWChar                                             '202 - A null-terminated Unicode character string
(Parameter object only).
        Case adLongVarWChar                                         '203 - A long null-terminated string value (Parameter
object only).
        Case adVarBinary                                            '204 - A binary value (Parameter object only).
        Case adLongVarBinary                                        '205 - A long binary value (Parameter object only).
    End Select
End Function

Private Function SetCatalog() As ADOX.Catalog
    'Retrieves the description of the field
    Cat.Tables(1).Columns(1).Properties(2).Value
    Set DBCatalog = Cat
    Set Cat = Nothing

    If Not Catalog Is Nothing Then
    End If
End Function
```

```
Private Sub Class_Initialize()
    'Create the Catlog
    Set Catalog = New ADOX.Catalog

    Catalog.ActiveConnection = cnADO

    Set Tables = Catalog.Tables
    Set Users = Catalog.Users
    Set Views = Catalog.Views
    Set Procs = Catalog.Procedures
    Set Grps = Catalog.Groups
End Sub

Private Sub Class_Terminate()
    Set Col = Nothing
    Set Cols = Nothing
    Set Grp = Nothing
    Set Grps = Nothing
    Set Ndx = Nothing
    Set Ndxs = Nothing
    Set Key = Nothing
    Set Keys = Nothing
    Set Proc = Nothing
    Set Procs = Nothing
    Set Prop = Nothing
    Set Props = Nothing
    Set Table = Nothing
    Set Tables = Nothing
    Set User = Nothing
    Set Users = Nothing
    Set View = Nothing
    Set Views = Nothing
    Set Catalog = Nothing
End Sub
```

DIAGRAMME - CHARTS

Wichtig:

Diagramme können in Excel direkt in Tabellenblättern stehen oder eine selbständige Arbeitsmappen-Einheit CHART sein

Schleifen die mit WORKSHEET arbeiten, finden diese eigenständigen CHART-Arbeitsmappen-Elemente nicht. Schleifen, die mit SHEET arbeiten, finden es schon.

--- GRUNDLAGEN ---

Inserting A Chart

Method 1:

```
Sub CreateChart()
'PURPOSE: Create a chart (chart dimensions are not required)

Dim rng As Range
Dim cht As Object

'Your data range for the chart
  Set rng = ActiveSheet.Range("A24:M27")

'Create a chart
  Set cht = ActiveSheet.Shapes.AddChart2

'Give chart some data
  cht.Chart.SetSourceData Source:=rng

'Determine the chart type
  cht.Chart.ChartType = xlXYScatterLines

End Sub
```

Method 2:

```
Sub CreateChart()
'PURPOSE: Create a chart (chart dimensions are required)

Dim rng As Range
Dim cht As ChartObject

'Your data range for the chart
  Set rng = ActiveSheet.Range("A24:M27")

'Create a chart
  Set cht = ActiveSheet.ChartObjects.Add( _
    Left:=ActiveCell.Left, _
    Width:=450, _
    Top:=ActiveCell.Top, _
    Height:=250)

'Give chart some data
  cht.Chart.SetSourceData Source:=rng

'Determine the chart type
  cht.Chart.ChartType = xlXYScatterLines

End Sub
```

Looping Through Charts & Series

```
Sub LoopThroughCharts()
'PURPOSE: How to cycle through charts and chart series

Dim cht As ChartObject
Dim ser As Series

'Loop Through all charts on ActiveSheet
  For Each cht In ActiveSheet.ChartObjects

  Next cht

'Loop through all series in a chart
  For Each ser In grph.Chart.SeriesCollection

  Next ser

'Loop Through all series on Activesheet
  For Each cht In ActiveSheet.ChartObjects
    For Each ser In grph.Chart.SeriesCollection

    Next ser
  Next cht

End Sub
```

Adding & Modifying A Chart Title

```
Sub AddChartTitle()
'PURPOSE: Add a title to a specific chart

Dim cht As ChartObject

Set cht = ActiveSheet.ChartObjects("Chart 1")

'Ensure chart has a title
  cht.Chart.HasTitle = True

'Change chart's title
  cht.Chart.ChartTitle.Text = "My Graph"

End Sub
```

```
Sub RepositionChartTitle()
'PURPOSE: Reposition a chart's title

Dim cht As ChartObject

Set cht = ActiveSheet.ChartObjects("Chart 1")

'Reposition title
  With cht.Chart.ChartTitle
    .Left = 100
    .Top = 50
  End With

End Sub
```

Adding & Modifying A Graph Legend

```
Sub InsertChartLegend()

Dim cht As Chart

Set cht = ActiveSheet.ChartObjects("Chart 1").Chart

'Add Legend to the Right
  cht.SetElement (msoElementLegendRight)

'Add Legend to the Left
  cht.SetElement (msoElementLegendLeft)

'Add Legend to the Bottom
  cht.SetElement (msoElementLegendBottom)

'Add Legend to the Top
  cht.SetElement (msoElementLegendTop)

'Add Overlaying Legend to the Left
  cht.SetElement (msoElementLegendLeftOverlay)

'Add Overlaying Legend to the Right
  cht.SetElement (msoElementLegendRightOverlay)

End Sub
```

```
Sub DimensionChartLegend()

Dim lgd As Legend

Set lgd = ActiveSheet.ChartObjects("Chart 1").Chart.Legend

lgd.Left = 240.23
lgd.Top = 6.962
lgd.Width = 103.769
lgd.Height = 25.165

End Sub
```

Adding Various Chart Attributes

```
Sub AddStuffToChart()

Dim cht As Chart

Set cht = ActiveSheet.ChartObjects("Chart 1").Chart

'Add X-axis
  cht.HasAxis(xlCategory, xlPrimary) = True '[Method #1]
  cht.SetElement (msoElementPrimaryCategoryAxisShow) '[Method #2]

'Add X-axis title
  cht.Axes(xlCategory, xlPrimary).HasTitle = True '[Method #1]
  cht.SetElement (msoElementPrimaryCategoryAxisTitleAdjacentToAxis) '[Method #2]

'Add y-axis
  cht.HasAxis(xlValue, xlPrimary) = True '[Method #1]
  cht.SetElement (msoElementPrimaryValueAxisShow) '[Method #2]

'Add y-axis title
  cht.Axes(xlValue, xlPrimary).HasTitle = True '[Method #1]
  cht.SetElement (msoElementPrimaryValueAxisTitleAdjacentToAxis) '[Method #2]

'Add Data Labels (Centered)
  cht.SetElement (msoElementDataLabelCenter)

'Add Major Gridlines
  cht.SetElement (msoElementPrimaryValueGridLinesMajor)

'Add Linear Trend Line
  cht.SeriesCollection(1).Trendlines.Add Type:=xlLinear

End Sub
```

Modifying Various Chart Attributes

```
Sub ChangeChartFormatting()

Dim cht As Chart

Set cht = ActiveSheet.ChartObjects("Chart 1").Chart

'Adjust y-axis Scale
  cht.Axes(xlValue).MinimumScale = 40
  cht.Axes(xlValue).MaximumScale = 100

'Adjust x-axis Scale
  cht.Axes(xlCategory).MinimumScale = 1
  cht.Axes(xlCategory).MaximumScale = 10

'Adjust Bar Gap
  cht.ChartGroups(1).GapWidth = 60

'Format Font Size
  cht.ChartArea.Format.TextFrame2.TextRange.Font.Size = 12

'Format Font Type
  cht.ChartArea.Format.TextFrame2.TextRange.Font.Name = "Arial"

'Make Font Bold
  cht.ChartArea.Format.TextFrame2.TextRange.Font.Bold = msoTrue

'Make Font Italicized
  cht.ChartArea.Format.TextFrame2.TextRange.Font.Italic = msoTrue

End Sub
```

Removing Various Chart Attributes

```
Sub RemoveChartFormatting()

Dim cht As Chart

Set cht = ActiveSheet.ChartObjects("Chart 1").Chart

'Remove Chart Series
  cht.SeriesCollection(2).Delete

'Remove Gridlines
  cht.Axes(xlValue).MajorGridlines.Delete
  cht.Axes(xlValue).MinorGridlines.Delete

'Remove X-axis
  cht.Axes(xlCategory).Delete

'Remove Y-axis
  cht.Axes(xlValue).Delete

'Remove Legend
  cht.Legend.Delete

'Remove Title
  cht.ChartTitle.Delete

'Remove ChartArea border
  cht.ChartArea.Border.LineStyle = xlNone

'No background color fill
  cht.ChartArea.Format.Fill.Visible = msoFalse
  cht.PlotArea.Format.Fill.Visible = msoFalse

End Sub
```

Change Your Colors

```
Sub ChangeChartColors()

Dim cht As Chart

Set cht = ActiveSheet.ChartObjects("Chart 1").Chart

'Change first bar chart series fill color
  cht.SeriesCollection(1).Format.Fill.ForeColor.RGB = RGB(91, 155, 213)

'Change X-axis label color
  cht.Axes(xlCategory).TickLabels.Font.Color = RGB(91, 155, 213)

'Change Y-axis label color
  cht.Axes(xlValue).TickLabels.Font.Color = RGB(91, 155, 213)

'Change Plot Area border color
  cht.PlotArea.Format.Line.ForeColor.RGB = RGB(91, 155, 213)

'Change Major gridline color
  cht.Axes(xlValue).MajorGridlines.Format.Line.ForeColor.RGB = RGB(91, 155, 213)

'Change Chart Title font color
  cht.ChartTitle.Format.TextFrame2.TextRange.Font.Fill.ForeColor.RGB = RGB(91, 155, 213)

'No background color fill
  cht.ChartArea.Format.Fill.Visible = msoFalse
  cht.PlotArea.Format.Fill.Visible = msoFalse

End Sub
```

--- Allgemeines ---

http://www.computorcompanion.com/LPMArticle.asp?ID=221

Chart Events in Microsoft Excel

by Jon Peltier

When you use a computer, you continuously interact with objects and programs, through *events*. An event is anything that happens, such as moving the

mouse, clicking on objects, changing data, or activating windows. Programs can raise events of their own as well. A program can respond to events using event procedures. For example, if you click a scroll button, Excel responds by scrolling the window down.

As a programmer, you can write event procedures that will respond to (or *trap*) events by users. These event procedures can trap events in the worksheet, in the workbook, at the application level, or in a chart. In this article, I introduce you to chart event procedures in Microsoft Excel, which you can use for charts on a chart sheet and embedded charts.

Why Use Chart Events?

Chart events can make it much easier for people to use your programs. Perhaps you need users to select a point in a series. If you create an event procedure that detects a click on the chart, users won't have to type in a series name and a point number.

After the click the program can perform one or more useful tasks:

- Display a customized chart tip;
- Extract information from the chart and place it in a worksheet;
- Activate another chart or worksheet;
- Identify points to be included in a subsequent analysis.

Although Excel can accept data in a variety of formats and arrangements, you can do certain things with the data to ensure your charts are easy to build and maintain. "Good data" has these characteristics:

1. Data for each series is arranged in Columns (preferable to Rows)
2. First column is used for Categories (X axis labels)
3. First row is used for Series Names
4. Top left cell is blank
5. Data below series names is numeric (right-adjusted with no alignment specified)
6. There are no blank rows or blank columns within the data range

7. There is a buffer row or column between the table and any other non-blank cells.

	Alpha	Beta
A	5	5
B	8	2
C	2	7
D	7	5
E	6	4

In this case, I use sample data above to demonstrate chart event procedures in a workbook named EventCharts.xls.

Create A Chart Sheet

Select the entire range you want included in your chart, or select a single cell within this range. If you followed guidelines 6 and 7 about Good Data, Excel selects the range of continuous data that includes the selected cell.

Start the Chart Wizard by clicking on the Chart Wizard button on Excel's Standard toolbar, or by choosing Insert|Chart. Follow the steps in the chart wizard, and in Step 4 - Chart Location, select the As New Sheet option. If you want you can and if desired, enter a new name for the sheet, although in this example uses the default name, Chart1.

Press the F11 function key, to quickly create a chart sheet from a selected range, or from the continuous range surrounding the selected cell.

The Chart Code Module

Right-click the chart sheet tab, and select View Code. This opens the Visual Basic Editor (VBE). Within it, a code module appears with the caption ChartEvents.xls - Chart1 (code), as shown in Figure 1.

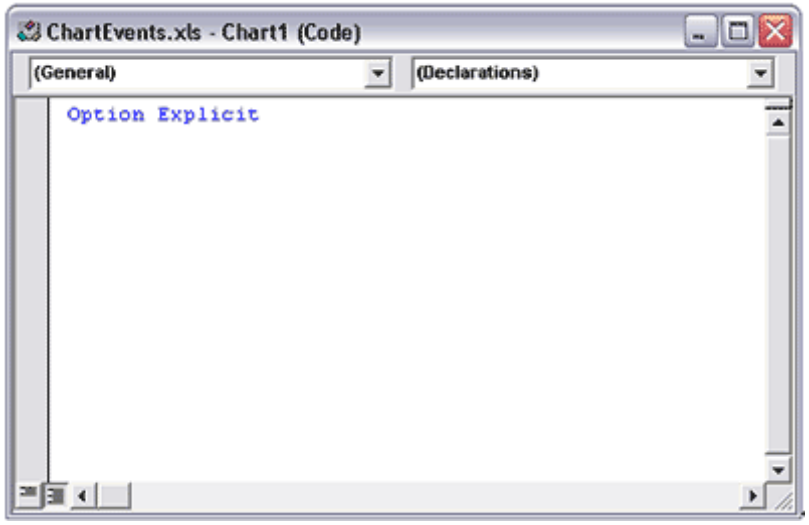


Figure 1 - Chart Code Module

The code module should contain only the line

```
Option Explicit
```

This line means any variable used in the module must be explicitly declared prior to use. This approach is slightly more work, but it assures that no variable will be accidentally defined by a typographical error.

If the module does not include the Option Explicit line, type it in, then go to the VBE's Tools menu, select Options..., and on the Editor tab, check the Require Variable Declarations box. Every module you create after setting this option will automatically contain the Option Explicit line.

Chart Event Procedure

At the top of the code module, you see two dropdowns. One contains the items (General) and Chart, the other contains (Declarations). While you can type your entire event procedure from scratch, it's easier to use these dropdowns when creating procedures.

Select Chart from the left hand dropdown. The following shell of an event procedure appears in the code module:

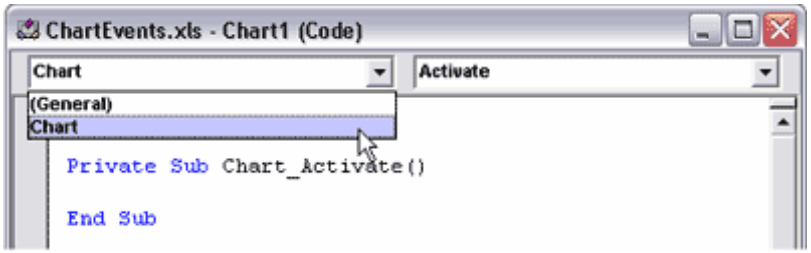


Figure 2 - Event Procedure.

```
Private Sub Chart Activate()


End Sub
```

This procedure runs whenever the chart is activated. Between these lines, type whatever code you want to be executed when the chart is activated. For example:

```
Private Sub Chart Activate()

    MsgBox "Welcome to my Chart!"

End Sub
```

Now when you activate the chart, the following message box appears:

Figure 3 - Message Box

More Chart Events

Chart_Activate is good for illustration of chart events, but other events can be much more useful. The chart events available to you are listed below. These can be accessed from the right hand dropdown in the chart code module when Chart is selected in the left hand dropdown.

- Activate
- BeforeDoubleClick
- BeforeRightClick
- Calculate
- Deactivate
- DragOver
- DragPlot
- MouseDown
- MouseMove
- MouseUp
- Resize
- Select
- SeriesChange

Next you can use an event to tell which chart element has been selected. Choose Chart from the left hand dropdown of the code module, then choose Select from the right hand dropdown, to produce the following procedure shell:

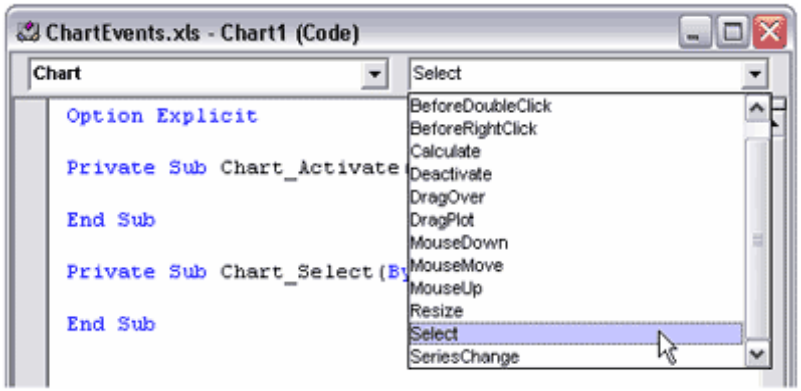


Figure 4 - Procedure Shell

```
Private Sub Chart_Select(ByVal ElementID As Long,

        ByVal Arg1 As Long, ByVal Arg2 As Long)


End Sub
```

Note that the line continuation character (an underscore) has been inserted in the first line to wrap the line at a convenient place.

The Chart_Select event fires whenever a chart element has been selected. Three arguments are passed to the procedure: ElementID, which identifies the selected chart element, and Arg1 and Arg2, which provide additional information. For example, if ElementID identifies that a series has been selected, Arg1 identifies which series has been selected, and Arg2 identifies the point within the series which has been selected, or Arg1 = -1 to indicate that the entire series is selected. If Arg1 or Arg2 are not relevant to the selected ElementID, they are assigned the value zero.

When you select Chart from the left hand dropdown, the Chart_Activate procedure shell is created in the code module (Activate is the default event for a chart). You can ignore it, delete it, or use it for another purpose.

This table lists the ElementID values, and the meanings of their associated Arg1 and Arg2 parameters.

ElementID	**Arg1**	**Arg2**
xlChartArea	None	None

xlChartTitle	None	None
xlPlotArea	None	None
xlLegend	None	None
xlFloor	None	None
xlWalls	None	None
xlCorners	None	None
xlDataTable	None	None
xlSeries	SeriesIndex	PointIndex
xlDataLabel	SeriesIndex	PointIndex
xlTrendline	SeriesIndex	TrendLineIndex
xlErrorBars	SeriesIndex	None
xlXErrorBars	SeriesIndex	None
xlYErrorBars	SeriesIndex	None
xlLegendEntry	SeriesIndex	None
xlLegendKey	SeriesIndex	None

xlAxis	AxisIndex	AxisType
xlMajorGridlines	AxisIndex	AxisType
xlMinorGridlines	AxisIndex	AxisType
xlAxisTitle	AxisIndex	AxisType
xlDisplayUnitLabel	AxisIndex	AxisType
xlUpBars	GroupIndex	None
xlDownBars	GroupIndex	None
xlSeriesLines	GroupIndex	None
xlHiLoLines	GroupIndex	None
xlDropLines	GroupIndex	None
xlRadarAxisLabels	GroupIndex	None
xlShape	ShapeIndex	None
xlPivotChartDropZone	DropZoneType	None
xlPivotChartFieldButton	DropZoneType	PivotFieldIndex
xlNothing	None	None

Now you can add a line to the procedure to identify which chart element has been selected, and show the values of its associated arguments:

```
Private Sub Chart_Select(ByVal ElementID As Long, _
        ByVal Arg1 As Long, ByVal Arg2 As Long)


    MsgBox "Element: " & ElementID & vbCrLf & "  Arg 1: " & Arg1
        & vbCrLf & "  Arg 2: " & Arg2


End Sub
```

To test your event procedure, select the chart's title. Click OK in the message box, and then test your event procedure with other elements, such as a series, and a point in a series.

The following three message boxes appear when selecting the Chart Title (ElementID = 4), Series 1 (ElementID = 3, Arg1 = 1, and Arg2 = -1), and Series 1 Point 4 (ElementID = 3, Arg1 = 1, Arg2 = 4).

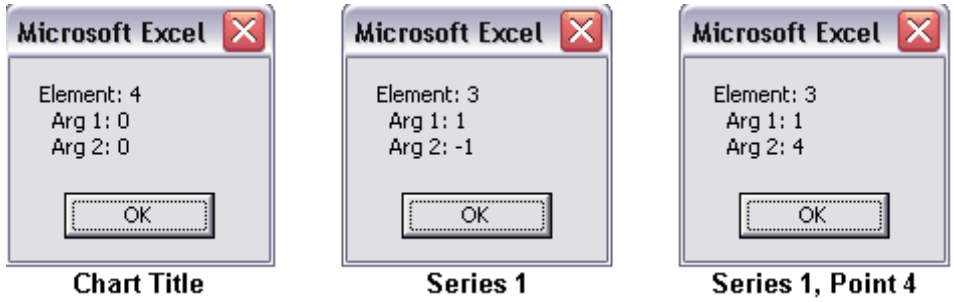


Figure 5 - Message boxes

These message boxes aren't very informative – text descriptions would be more useful. Replace the Chart_Select event code with the following more extensive procedure. It converts the values to text, and creates a more informative message:

```
Private Sub Chart Select(ByVal ElementID As Long,
        ByVal Arg1 As Long, ByVal Arg2 As Long)
```

```
Dim sElement As String
Dim sArg As String

Select Case ElementID
    Case xlChartArea
        sElement = "Chart Area"
    Case xlChartTitle
        sElement = "Chart Title"
    Case xlPlotArea
        sElement = "Plot Area"
    Case xlLegend
        sElement = "Legend"
    Case xlFloor
        sElement = "Floor"
    Case xlWalls
        sElement = "Walls"
    Case xlCorners
        sElement = "Corners"
    Case xlDataTable
        sElement = "Data Table"
    Case xlSeries
        sElement = "Series " & Arg1
        If Arg2 > 0 Then sArg = "Point " & Arg2
    Case xlDataLabel
        sElement = "Data Label"
```

```
        sArg = "Series " & Arg1
        If Arg2 > 0 Then sArg = sArg & ", Point " & Arg2
    Case xlTrendline
        sElement = "Trendline"
        sArg = "Series " & Arg1 & ", Trendline " & Arg2
    Case xlErrorBars
        sElement = "Error Bars"
        sArg = "Series " & Arg1
    Case xlXErrorBars
        sElement = "X Error Bars"
        sArg = "Series " & Arg1
    Case xlYErrorBars
        sElement = "Y Error Bars"
        sArg = "Series " & Arg1
    Case xlLegendEntry
        sElement = "Legend Entry"
        sArg = "Series " & Arg1
    Case xlLegendKey
        sElement = "Legend Key"
        sArg = "Series " & Arg1
    Case xlAxis
        sElement = IIf(Arg1 = 1, "Primary ", "Secondary ")
        sElement = sElement & IIf(Arg2 = 1, "Category ", "Value ")
        sElement = sElement & "Axis"
    Case xlMajorGridlines
        sElement = IIf(Arg1 = 1, "Primary ", "Secondary ")
```

```
    sElement = sElement & IIf(Arg2 = 1, "Category ", "Value ")
    sElement = sElement & "Major Gridlines"
Case xlMinorGridlines
    sElement = IIf(Arg1 = 1, "Primary ", "Secondary ")
    sElement = sElement & IIf(Arg2 = 1, "Category ", "Value ")
    sElement = sElement & "Minor Gridlines"
Case xlAxisTitle
    sElement = IIf(Arg1 = 1, "Primary ", "Secondary ")
    sElement = sElement & IIf(Arg2 = 1, "Category ", "Value ")
    sElement = sElement & "Axis Title"
Case xlDisplayUnitLabel
    sElement = IIf(Arg1 = 1, "Primary ", "Secondary ")
    sElement = sElement & IIf(Arg2 = 1, "Category ", "Value ")
    sElement = sElement & "Axis Display Unit Label"
Case xlUpBars
    sElement = "Up Bars"
    sArg = "Group Index " & Arg1
Case xlDownBars
    sElement = "Down Bars"
    sArg = "Group Index " & Arg1
Case xlSeriesLines
    sElement = "Series Lines"
    sArg = "Group Index " & Arg1
Case xlHiLoLines
    sElement = "High-Low Lines"
    sArg = "Group Index " & Arg1
```

```
        Case xlDropLines
            sElement = "Drop Lines"
            sArg = "Group Index " & Arg1
        Case xlRadarAxisLabels
            sElement = "Radar Axis Labels"
            sArg = "Group Index " & Arg1
        Case xlShape
            sElement = "Shape"
            sArg = "Shape Number " & Arg1
        Case xlNothing
            sElement = "Nothing"
    End Select

    MsgBox sElement & IIf(Len(sArg) > 0, vbCrLf & sArg, "")
End Sub
```

Test your new code, by selecting elements in the chart. Here are the revised message boxes, made more informative by analyzing the ElementID and other arguments:

Figure 6 - Revised Message Boxes

This is a more useful procedure, because it tells us in human words what has been selected. On the other hand, it is less useful than it could be, because

it pops up a message box every time a new chart element is selected, forcing the user to clear the message box before proceeding.

Identify the Point Clicked By the User

Another useful event procedure tells you which point was clicked on, and display its values. Chart_Select is not the best event to trap to detect clicking on a point, because two Chart_Select events occur when a point is selected: the series is selected by the first click, and the point by the second. The MouseUp event is a better event to trap.

To create a new event procedure, choose Chart in the left dropdown and MouseUp in the right, to create this shell:

```
Private Sub Chart MouseUp(ByVal Button As Long, ByVal Shift As Long,

        ByVal x As Long, ByVal y As Long)


End Sub
```

Four arguments are passed to this procedure.

Button tells which button was released (xlNoButton, xlPrimaryButton, xlSecondaryButton, or xlMiddleButton).

Shift tells which keys are depressed when the mouse button was released. The value of Shift can be one or a sum of several of the following values:

0 (zero)	No keys
1	SHIFT key
2	CTRL key
4	ALT key

If Shift = 2, the CTRL key was depressed when the mouse button was clicked; if Shift = 5, both the SHIFT and ALT keys were depressed.

The coordinates of the cursor, x and y, are in chart object client coordinate units. It's not necessary to be concerned about these, since you just pass them along to other functions to get more useful information.

Test the following event procedure, which uses Chart_MouseUp to determine where the mouse click occurred. The procedure:

- traps the release of the button,
- determines where on the chart the click occurred (x, y),
- passes these coordinates to the GetChartElement function to determine what chart element is located at these coordinates,
- determines whether a data point (or data label) is located at the mouse click,
- extracts the X and Y values for the data point, if the click occurred at a data point or data label,
- displays this information in a message box.

```
Private Sub Chart MouseUp(ByVal Button As Long, ByVal Shift As Long,
        ByVal x As Long, ByVal y As Long)

    Dim ElementID As Long, Arg1 As Long, Arg2 As Long
    Dim myX As Variant, myY As Double

    With ActiveChart
        ' Pass x & y, return ElementID and Args
        .GetChartElement x, y, ElementID, Arg1, Arg2

        ' Did we click over a point or data label?
        If ElementID = xlSeries Or ElementID = xlDataLabel Then
            If Arg2 > 0 Then
                ' Extract x value from array of x values
                myX = WorksheetFunction.Index
                    (.SeriesCollection(Arg1).XValues, Arg2)
                ' Extract y value from array of y values
```

```
                myY = WorksheetFunction.Index _
                    (.SeriesCollection(Arg1).Values, Arg2)

                ' Display message box with point information
                MsgBox "Series " & Arg1 & vbCrLf _
                    & """" & .SeriesCollection(Arg1).Name & """" & vbCrLf
                    & "Point " & Arg2 & vbCrLf
                    & "X = " & myX & vbCrLf
                    & "Y = " & myY
            End If
        End If
    End With

End Sub
```

You could do many things with the data from this procedure. A message box requires a user response to continue. Replace this by drawing a textbox with the same information, as a smarter chart tip. Use the MouseMove event instead, to mimic Excel's normal chart tip behavior.

Place this data into a worksheet range. After clicking on several points, you would have a small table of values from the selected points. These could be used for subsequent analysis.

Identify the clicked point, then jump to another appropriate chart. This approach has the effect of drilling down deeper into the data. Suppose, for example, the values for the categories A through E in the chart reflected sums of other values. You could have several other chart sheets, Chart A through Chart E that show detail for the selected categories. An event procedure such as the following produces this kind of drill down effect (to test the procedure, create chart sheets named Chart A, Chart B, etc.):

```
Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, _
        ByVal x As Long, ByVal y As Long)

    Dim ElementID As Long, Arg1 As Long, Arg2 As Long
```

```
    Dim myX As Variant


    With ActiveChart
        ' Pass x & y, return ElementID and Args
        .GetChartElement x, y, ElementID, Arg1, Arg2


        ' Did we click over a point or data label?
        If ElementID = xlSeries Or ElementID = xlDataLabel Then
            If Arg2 > 0 Then
                ' Extract x value from array of x values
                myX = WorksheetFunction.Index
                    (.SeriesCollection(Arg1).XValues, Arg2)


                ' Don't crash if chart doesn't exist
                On Error Resume Next
                ' Activate the appropriate chart
                ThisWorkbook.Charts("Chart " & myX).Select
                On Error GoTo 0
            End If
        End If
    End With


End Sub
```

To return to the starting chart on another mouse click, you can add the following Chart_MouseUp event procedure to the code modules of the detail charts. For example, add the following procedure to the code module for the Chart A sheet. When you click an element in Chart A, you'll return to the main chart:

```
Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, _
```

```
        ByVal x As Long, ByVal y As Long)


    ' Don't crash if chart doesn't exist
    On Error Resume Next
    ' Return to calling chart
    ThisWorkbook.Charts("Chart1").Select
    On Error GoTo 0


End Sub
```

The VBE Project Explorer

You may have noticed the Project Explorer window of the Visual Basic Editor. By default it is docked in the upper left of the VB window. You can open the VB Editor from Excel by selecting the Tools|Macro|Visual Basic Editor, or by pressing Alt +F11. If the Project Explorer window is not open, from within the VBE, choose View|Project Explorer or press Ctrl+R. Figure 7 shows an undocked Project Explorer window.

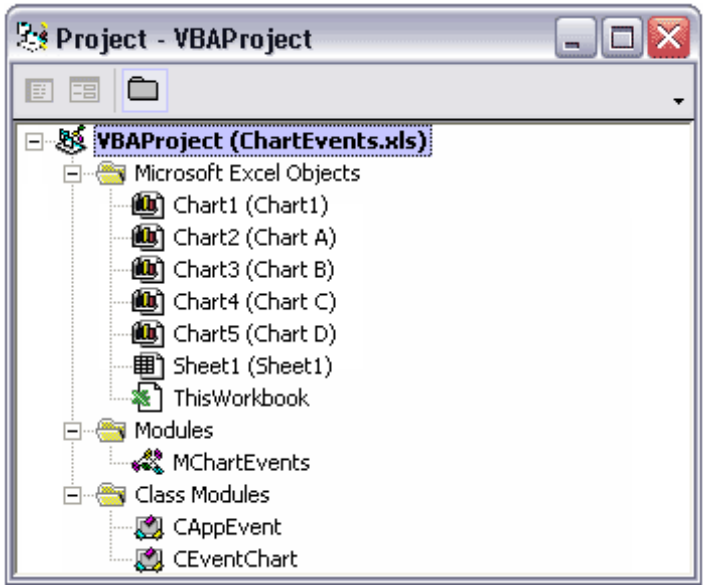


Figure 7 - Project Explorer Window

Every VBA Project that you have open in Excel is listed in the Project Explorer, including open workbooks and installed add-ins. If the project is unprotected, its constituent objects are listed in a tree view structure underneath it. If the folder icon in the toolbar is selected, the objects are grouped. If the folder icon is not selected, the objects are listed in alphabetical order without grouping.

You can see three of the four groups in Figure 7. All chart sheets and worksheets are listed under the Microsoft Excel Objects group, with the ThisWorkbook object that represents the workbook itself. Regular code modules are listed under the Modules group, where we see the module named MChartEvents. The Class Modules group contains class modules, such as CAppEvent and CEventChart, which will be constructed later in this article. If a project has any user forms, these will be listed under the Forms group.

Under Microsoft Excel Objects, the sheets and workbook are listed in alphabetical order by their "Code Name". The Code Name is the name of the object in the VB Editor; the default Chart1 through Chart5, Sheet1 and ThisWorkbook are shown above. The Excel name of an object is shown in parentheses after its Code Name. You can change the Code Name by changing the (Name) property in the VBE's Properties window; if you change the Name property, the name displayed on the sheet tab in Excel will be changed.

To access a sheet's code module, double click it in the Project Explorer window, or right-click a sheet tab in Excel and choose View Code. To access the workbook's code module, double-click it in the Project Explorer window, or right-click the small Excel icon to the left of the File menu in Excel, and choose View Code.

New Forms, Modules, and Class Modules are added to a project using the VBE's Insert menu. A new chart sheet or worksheet object is added whenever a sheet is created within Excel. A new project is added to the Project Explorer's list when a workbook is created or opened, or an add-in is installed, within Excel.

Embedded Chart Events

The approach I have described so far is effective, but only for a single chart sheet. To apply to multiple chart sheets, the event code has to be copied to each chart's code module. It does not work at all for embedded charts.

A more flexible and powerful technique is to use a class module to trap chart events. A class module can be used for any chart, whether it's an embedded chart or chart sheet. And it eliminates the need to place the code in a chart sheet's code module.

To create a new class module, switch to the VB Editor. Make sure the ChartEvents.xls workbook is selected in the Project Explorer window.

Within the VBE, choose Insert|Class Module. A new code module opens, with the caption ChartEvents.xls - Class1 (Code). If the Properties window is not open, choose View|Properties, or press the F4 function key. In the Properties window, change the class name to CEventChart, following the convention that the name of a class module begins with capital C.

The class module should begin with this line:

```
Option Explicit
```

If not, follow the instructions in "The Chart Code Module" subsection in the "Chart Sheets" section of this article, above.

Next, add this line in the Declarations section of the class module, to declare a Chart type object with event trapping enabled.

```
Public WithEvents EvtChart As Chart
```

After adding this line, the class module contains the EvtChart item in the left hand dropdown, in addition to (General) and Class. When the EvtChart item is selected, all of the chart events described in the first half of this article appear in the right hand dropdown.

To create the equivalent chart event procedure as you created in the "More Chart Events" subsection above, select EvtChart from the left dropdown and Select from the right dropdown, to produce this procedure shell:

```
Private Sub EvtChart_Select(ByVal ElementID As Long, _
        ByVal Arg1 As Long, ByVal Arg2 As Long)


End Sub
```

The shell of the EvtChart_Activate procedure is produced when the EvtChart item is selected in the left dropdown. This can be deleted or ignored. Add the Msgbox command within this procedure to identify the element and arguments for the selected chart element. The entire class module code should look like this:

```
Option Explicit


' Declare object of type "Chart" with events
Public WithEvents EvtChart As Chart


Private Sub EvtChart Select(ByVal ElementID As Long,
        ByVal Arg1 As Long, ByVal Arg2 As Long)


    MsgBox "Element: " & ElementID & vbCrLf & "  Arg 1: " & Arg1
        & vbCrLf & "  Arg 2: " & Arg2
```

```
End Sub
```

The code is almost ready to trap chart events, but first you need to activate the chart.

Activating the Chart

Make sure the ChartEvents.xls workbook is selected in the Project Explorer window. From the VBE Insert menu, select Module. A new code module opens, with the caption ChartEvents.xls - Module1 (Code). In the Properties window, change the module name to MChartEvents, following the convention that the name of a regular module begins with capital M.

The module should begin with this line:

```
Option Explicit
```

If not, follow the instructions in "The Chart Code Module" section. Now you must declare an instance of the chart events class, by adding this line in the Declarations section of the class module, immediately below the Option Explicit line.

```
Dim clsEventChart As New CEventChart
```

Next, you create two procedures, Set_This_Chart and Reset_Chart, To activate a chart for events, you assign the active chart to the EvtChart property of the instance of the class:

```
Set clsEventChart.EvtChart = ActiveChart
```

To deactivate the chart for events, you assign the EvtChart property of the instance of the class to Nothing:

```
Set clsEventChart.EvtChart = Nothing
```

These two lines are wrapped in the Set_This_Chart and Reset_Chart procedures, with a check within the Set_This_Chart procedure to prevent errors if no chart is selected. The entire module looks like this:

```
Option Explicit


Dim clsEventChart As New CEventChart


Sub Set This Chart()

    ' Skip if no chart is selected to prevent an error

    If Not ActiveChart Is Nothing Then
```

```
        ' Enable events for the active chart
        ' Works for chart sheets and embedded charts
        Set clsEventChart.EvtChart = ActiveChart
    End If
End Sub


Sub Reset Chart()
    ' Disable events for chart previously enabled as active chart
    Set clsEventChart.EvtChart = Nothing
End Sub
```

To test the procedures, select any chart, embedded or chart sheet, and run Set_This_Chart. Now select any chart element, and a message box pops up to tell you the ElementID, Arg1, and Arg2.

Activating Chart(s)

This approach is fine for a single embedded chart, but it can be broadened to include all charts embedded in the active sheet. Instead of a variable clsEventChart, you declare an array clsEventCharts() to contain all of the enabled charts, and wrap them within loops so all embedded charts on the active sheet are included.

Adjust the MChartEvents module to look like this:

```
Option Explicit


Dim clsEventCharts() As New CEventChart


Sub Set All Charts()
    ' Enable events for all charts embedded on a sheet
    ' Works for embedded charts on a worksheet or chart sheet
    If ActiveSheet.ChartObjects.Count > 0 Then
```

```
        ReDim clsEventCharts(1 To ActiveSheet.ChartObjects.Count)
        Dim chtObj As ChartObject
        Dim chtnum As Integer

        chtnum = 1
        For Each chtObj In ActiveSheet.ChartObjects
            ' Debug.Print chtObj.Name, chtObj.Parent.Name
            Set clsEventCharts(chtnum).EvtChart = chtObj.Chart
            chtnum = chtnum + 1
        Next ' chtObj
    End If
End Sub

Sub Reset All Charts()
    ' Disable events for all charts previously enabled together
    Dim chtnum As Integer
    On Error Resume Next
    Set clsEventChart.EvtChart = Nothing
    For chtnum = 1 To UBound(clsEventCharts)
        Set clsEventCharts(chtnum).EvtChart = Nothing
    Next ' chtnum
End Sub
```

Activate a sheet with embedded charts and run Set_All_Charts. All embedded charts will now respond to chart events as the single embedded chart did in the previous section. This approach works for any embedded charts on a worksheet or a chart sheet (yes, you can embed chart objects on a chart sheet).

The previous code won't activate the parent chart sheet, however. You reinstate the clsEventChart declaration, to be used with the chart sheet, and adjust the Set_All_Charts procedure to activate a chart sheet and then activate the embedded charts.

Modify the module to look like this:

```
Option Explicit


Dim clsEventChart As New CEventChart
Dim clsEventCharts() As New CEventChart


Sub Set_All_Charts()
    ' Enable events for active sheet if sheet is a chart sheet
    If TypeName(ActiveSheet) = "Chart" Then
        Set clsEventChart.EvtChart = ActiveSheet
    End If


    ' Enable events for all charts embedded on a sheet
    ' Works for embedded charts on a worksheet or chart sheet
    If ActiveSheet.ChartObjects.Count > 0 Then
        ReDim clsEventCharts(1 To ActiveSheet.ChartObjects.Count)
        Dim chtObj As ChartObject
        Dim chtnum As Integer


        chtnum = 1
        For Each chtObj In ActiveSheet.ChartObjects
            ' Debug.Print chtObj.Name, chtObj.Parent.Name
            Set clsEventCharts(chtnum).EvtChart = chtObj.Chart
            chtnum = chtnum + 1
        Next ' chtObj
    End If
```

```
End Sub


Sub Reset_All_Charts()
    ' Disable events for all charts previously enabled together
    Dim chtnum As Integer
    On Error Resume Next
    Set clsEventChart.EvtChart = Nothing
    For chtnum = 1 To UBound(clsEventCharts)
        Set clsEventCharts(chtnum).EvtChart = Nothing
    Next ' chtnum
End Sub
```

Using Event Procedures to Enable Chart Events

Event procedures in Microsoft Excel have different levels of influence. In the earlier example, the event procedures in the code module behind a chart sheet only detect events within the chart sheet. Similarly, event procedures on a worksheet's code module only detect events in that worksheet. Event procedures on a workbook's code module trap events that occur in all sheets within the workbook. Finally, application-level event procedures watch over all sheets and workbooks in Excel. There is no application code module, however, so you need to create a special class module to handle application events.

Using these events is a more convenient way to activate chart events than running the Set_All_Charts and Reset_All_Charts procedures used above. The following code examples demonstrate how you can use events to set different levels of activation for chart events:

1. all charts in one worksheet
2. all charts in one chart sheet
3. all charts on all sheets in one workbook
4. all charts on all sheets in all workbooks

You can decide which level of activation you need, based on the purpose of your chart event code. If you are only concerned with a single chart in one worksheet, you can stop with the first example. If you have built an extensive add-in that processes events for any charts, you need a full-blown

application events class module.

1. Worksheet Events to Enable All Charts In One Worksheet

You must trap the Activate and Deactivate events for the desired worksheet, and use these events to run the Set_All_Charts and Reset_All_Charts procedures. Right-click the sheet tab, and choose View Code from the pop up menu to open the VBE. You see a new code module with the caption ChartEvents.xls - Sheet1 (Code), assuming this is Sheet1. Choose Worksheet on the left hand dropdown and Activate from the right hand dropdown. Now type the line Set_All_Charts inside the Worksheet_Activate procedure shell. Then choose Deactivate from the right hand dropdown, and type the line Reset_All_Charts inside the Worksheet_Deactivate procedure shell. You see the following in the code module:

```
Option Explicit


Private Sub Worksheet_Activate()

    Set_All_Charts

End Sub


Private Sub Worksheet Deactivate()

    Reset All Charts

End Sub
```

This code turns on chart events for all charts embedded in the worksheet when it is activated, then turns off chart events when the worksheet is deactivated. If your program executes additional code in the Worksheet_Activate and _Deactivate events, include the lines above with the other code, in the order that works best for your application.

2. Chart Sheet Events to Enable All Charts In One Chart Sheet

You need to trap the Activate and Deactivate events for the desired chart sheet, and use these events to run the Set_All_Charts and Reset_All_Charts procedures. To open the VBE, right-click the sheet tab, and choose View Code from the pop up menu. It opens the code module with the caption ChartEvents.xls - Chart1 (Code), assuming this is Chart1. Choose Chart on the left dropdown and Activate from the right dropdown, and type the line Set_All_Charts inside the Chart_Activate procedure shell. Then choose Deactivate from the right dropdown, and type the line Reset_All_Charts inside the Chart_Deactivate procedure shell. You see the following in the code module:

```
Option Explicit
```

```
Private Sub Chart_Activate()

    Set_All_Charts

End Sub


Private Sub Chart Deactivate()

    Reset All Charts

End Sub
```

This code turns on chart events for the chart sheet itself and for all charts embedded in the chart sheet when the sheet is activated, then turns off chart events when the sheet is deactivated. If your program executes additional code in the Chart_Activate and _Deactivate events, include the lines above with the other code, in the order that works best for your application.

3. Workbook Events to Enable All Charts On All Sheets In One Workbook

Remove the Worksheet_Activate, Worksheet_Deactivate, Chart_Activate, and Chart_Deactivate code that enables chart events for specific worksheets or chart sheets. These procedures are rendered redundant by the following Workbook_SheetActivate and _SheetDeactivate event procedures.

You must trap the Activate and Deactivate events for all sheets (worksheets and chart sheets) in the desired workbook, and use these events to run the Set_All_Charts and Reset_All_Charts procedures. Right-click the small Excel icon to the left of the File menu, and choose View Code from the pop up menu to open the VBE. It opens the code module with the caption ChartEvents.xls - ThisWorkbook (Code). Choose Workbook on the left dropdown, SheetActivate from the right dropdown, and type the line Set_All_Charts inside the Workbook_SheetActivate procedure shell. Then choose SheetDeactivate from the right dropdown, and type the line Reset_All_Charts inside the Workbook_SheetDeactivate procedure shell. You now see the following in the code module:

```
Option Explicit


Private Sub Workbook SheetActivate(ByVal Sh As Object)

    Set All Charts

End Sub


Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
```

```
    Reset_All_Charts

End Sub
```

This code turns on chart events for all charts embedded in the newly activated sheet, and for the sheet itself if it's a chart sheet, then turns off chart events when the sheet is deactivated. If your program executes additional code in the Workbook_SheetActivate and _SheetDeactivate events, include the lines above with the other code, in the order that works best for your application.

4. Application Events to Enable All Charts on All Sheets In All Workbooks

You must trap the Activate and Deactivate events for all sheets (worksheets and chart sheets) in any workbook, and the Activate and Deactivate events for all workbooks, and use these events to run the Set_All_Charts and Reset_All_Charts procedures. This process is a little more involved than using sheet or workbook events because you must create a class module to detect application events.

To create the Application Event Class Module, switch to the VB Editor to create a new class module. Make sure the ChartEvents.xls workbook is selected in the Project Explorer window. In the VBE, choose Insert|Class Module. A new code module opens with the caption ChartEvents.xls - Class1 (Code). If the Properties window is not open, choose View|Properties or press the F4 function key. In the Properties window, change the class name to CAppEvent (following the convention that the name of a class module begins with a capital C). Now declare an Application type object with event trapping enabled, by adding this line in the Declarations section of the class module.

```
Public WithEvents EventApp As Excel.Application
```

After adding this line, the class module contains the EventApp item in the left dropdown, in addition to (General) and Class. Select EventApp from the left dropdown, and select in turn SheetActivate, SheetDeactivate, WorkbookActivate, and WorkbookDeactivate from the right dropdown. Insert the line Set_All_Charts in the Activate event procedures, and Reset_All_Charts in the Deactivate event procedures. The CAppEvent class module should now contain this code:

```
Option Explicit


Public WithEvents EventApp As Excel.Application


Private Sub EventApp_SheetActivate(ByVal Sh As Object)

    Set All Charts

End Sub
```

```
Private Sub EventApp_SheetDeactivate(ByVal Sh As Object)
    Reset_All_Charts
End Sub

Private Sub EventApp_WorkbookActivate(ByVal Wb As Workbook)
    Set All Charts
End Sub

Private Sub EventApp WorkbookDeactivate(ByVal Wb As Workbook)
    Reset All Charts
End Sub
```

You must create an instance of the application class before it will respond to events. To activate the application event class, in the MChartEvents module, add a declaration line:

```
Dim clsAppEvent As New CAppEvent
```

Insert the following procedures into the MChartEvents module. The first tells the class that it is responding to events in the current application, and enables events in the active sheet's charts. The second tells the class that it is responding to events in Nothing (no events!), and disables events in the active sheet's charts.

```
Sub InitializeAppEvents()
  Set clsAppEvent.EventApp = Application
  Set_All_Charts
End Sub

Sub TerminateAppEvents()
  Set clsAppEvent.EventApp = Nothing
  Reset_All_Charts
End Sub
```

The last step in activating application-level events is to run the InitializeAppEvents procedure. You could use a command bar button or menu command for this, but those approaches both require the user to remember to go click something. Since you now are an expert on events, you can use workbook events to activate the application events.

The application events and chart events class modules likely reside within an add-in or workbook you've built to trap events from the entire application. Open the ThisWorkbook code module for your workbook. If your program is a regular workbook, use the dropdowns at the top of the code module to insert Workbook_Open and Workbook_BeforeClose procedures, and enter InitializeAppEvents and TerminateAppEvents where appropriate, as shown below:

```
Private Sub Workbook Open()

    InitializeAppEvents

End Sub


Private Sub Workbook BeforeClose(Cancel As Boolean)

    TerminateAppEvents

End Sub
```

If your program is an add-in, use the dropdowns at the top of the code module to insert Workbook_AddinInstall and Workbook_AddinUninstall procedures, and enter InitializeAppEvents and TerminateAppEvents where appropriate, as shown below:

```
Private Sub Workbook AddinInstall()

    InitializeAppEvents

End Sub


Private Sub Workbook AddinUninstall()

    TerminateAppEvents

End Sub
```

When the workbook is opened or the add-in installed, the application event class is instantiated, so it can begin watching for chart events. When the workbook is closed or the add-in uninstalled, the application event class in terminated, and chart events are no longer trapped.

In a perfect world, this would be sufficient to keep the application event procedures working. In our programming environment, however, many things can break the application event hooks. If your program encounters a run time error, for example, you may find yourself without application events, and

therefore, without chart events. You should insert the InitializeAppEvents line into other procedures where appropriate. For example, you could place it at the top or bottom of every procedure run from your program's command bar and menu buttons.

The Final Event

In this article, you found out how useful chart events can be. They can extend the user interface of the programs you write, making it easy for users to identify points for your program to work on. With a little ingenuity, you can create powerful applications. For example, recently, I made a utility that draws a trend line for only part of a plotted series using the first and last points indicated by two mouse clicks. I made another utility that lets the user zoom in and out on the basis of mouse actions. Once you've experimented with chart events, you'll think of many ways to enhance your projects.

Acknowledgment *- Thanks to my colleague, Excel MVP Debra Dalgleish, for reviewing this article and helping make it a coherent view of a complicated technique.*

VBA Code to Add Chart Objects and Series

Add a Chart

When you record a macro to add a chart object to a worksheet, Excel comes up with the following code:

```
Sub RecordedAddChartObject()
'
' RecordedAddChartObject Macro
' Macro recorded 5/2/02 by Jon Peltier
'
    Charts.Add
    ActiveChart.ChartType = xlXYScatterLines
    ActiveChart.SetSourceData Source:=Sheets("Sheet1").Range("A3:G14")
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Sheet1"
End Sub
```

Excel uses `Charts.Add` followed later by `ActiveChart.Location` to create a chart object, and uses `ActiveChart.SetSourceData` to set all the series data in one shot. The coding is efficient in terms of the small length of the code, but inflexxible in terms of your control over the output.

In my examples I use `ChartObjects.Add`, which also requires (or ***allows***) me to state the position and size of the chart. This example does almost exactly what the recorded macro above does:

```
Sub AddChartObject()
'
    With ActiveSheet.ChartObjects.Add _
```

```
            (Left:=100, Width:=375, Top:=75, Height:=225)
        .Chart.SetSourceData Source:=Sheets("Sheet1").Range("A3:G14")
        .Chart.ChartType = xlXYScatterLines
    End With
End Sub
```

The difference in the charts produced by the recorded macro and by the amended code are slight. The amended code positions and sizes the chart according to `(Left:=100, Width:=375, Top:=75, Height:=225)`, where these dimensions are in pixels. The Chart Wizard creates a chart roughly half as wide and half as tall as the visible part of the worksheet window, centered within the window (if you have frozen panes in the sheet, the chart is half the size of the active pane, subject to certain minimum dimensions).

Errors in recorded macros

There are some instances when a recorded macro will contain code that will not work properly. For example, a recorded macro always puts `ActiveChart.ChartType` ahead of `ActiveChart.SetSourceData`, but I have reversed them in the code above. The steps given by the recorder work fine for most chart types, but a few chart types cannot be correctly assigned until the chart has been populated with sufficient data. In particular, code which creates **stock charts** and **bubble charts** will fail if you do not reverse the order of the steps.

When working with a **surface** or **contour chart**, Excel will let you manually delete a series in the chart (via the Source Data dialog, because you cannot select a single series in such a chart). You can record a macro to see the steps you took. When you try to run the macro, however, it will crash on the `.Delete` line in the macro. In most chart types the series can be independently formatted, and they don't even have to be the same type, leading to the ability to create custom combination charts. The problem with surface or contour charts is that their series are not treated by VBA as independent series. The trick in this case is to temporarily convert the surface chart to another chart type, say, a line chart, delete the series, then convert back to a surface chart.

In what I think of as "Marker charts", that is, **XY** and **Line charts**, VBA cannot access certain series properties, including `.Values`, `.XValues`, and `.Formula` if the source range of the series contains no chartable data (i.e., it consists of blanks or errors). This severe inconvenience can be avoided if you change the series type to an area or column chart type before accessing the forbidden properties, then change it back to a Line or XY chart series.

Inefficiencies in recorded macros

Aside from error-raising macro problems, a recorded macro is less efficient, because it mimics all the mouseclicks and keystrokes (every cough and camera flash) that occurred while the recording was taking place. A recorded macro clicks on every object to select it, then performs an action on the selection:

```
    ActiveChart.Axes(xlValue).Select
```

```
    Selection.TickLabels.Font.Bold = True
    Selection.TickLabels.NumberFormat = "0.0"
```

Streamline your code by replacing all the `Object.Select` plus `Select.Property` sequences with shorter `Object.Property` statements:

```
    ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True
    ActiveChart.Axes(xlValue).TickLabels.NumberFormat = "0.0"
```

If you have two or more property or method statements that work on the same object, wrap them in a `With/End With` block:

```
    With ActiveChart.Axes(xlValue).TickLabels
        .Font.Bold = True
        .NumberFormat = "0.0"
    End With
```

There are other inefficiencies specific to charting, such as the `Charts.Add . . . ActiveChart.Location` sequence which can be shortened to a single `ChartObjects.Add` command, as discussed earlier in this page.

You may find it more convenient, in a longer procedure, to define some object variables. The next procedure does the same as the two above, but it uses a ChartObject variable for the new chart object that we create. If we need to refer to this chart object later in the procedure, we can conveniently use the variable `myChtObj`.

```
Sub AddChartObject()
Dim myChtObj As ChartObject
'
    Set myChtObj = ActiveSheet.ChartObjects.Add _
        (Left:=100, Width:=375, Top:=75, Height:=225)
    myChtObj.Chart.SetSourceData Source:=Sheets("Sheet1").Range("A3:G14")
    myChtObj.Chart.ChartType = xlXYScatterLines
End Sub
```

You are not limited to using `SetSourceData` to define the data being charted. You can add the series one-by-one, selecting the precise data you want, not what Excel will assume you want.

Top of Page

Add a Series

The following is a macro I recorded while adding a series using the Add command in the Source Data dialog:

```
Sub RecordedAddSeries()
'
' RecordedAddSeries Macro
' Macro recorded 5/2/02 by Jon Peltier
'
    ActiveChart.SeriesCollection.NewSeries
    ActiveChart.SeriesCollection(6).XValues = "=Sheet1!R4C1:R14C1"
```

```
    ActiveChart.SeriesCollection(6).Values = "=Sheet1!R4C7:R14C7"
    ActiveChart.SeriesCollection(6).Name = "=Sheet1!R3C7"
End Sub
```

This is basically my preferred syntax, although the following has been neatened up in a few ways. First, the index of the series is not mentioned in the code, so it's more readily reused. Second, I can use the familiar A1 cell address notation, or any VBA range reference technique.

```
Sub AddNewSeries()
    With ActiveChart.SeriesCollection.NewSeries
        .Name = ActiveSheet.Range("G3")
        .Values = ActiveSheet.Range("G4:G14")
        .XValues = ActiveSheet.Range("A4:A14")
    End With
End Sub
```

And as shown below, this is very flexible. For example, you can adjust the code to insert a string for the series name, a VBA array for X or Y values (X values in the following code), or a reference to a defined range name (`Y_Range` for Y values).

```
Sub AddNewSeries()
    With ActiveChart.SeriesCollection.NewSeries
        .Name = "Fred"
        .Values = "=Sheet1!Y_Range"
        .XValues = Array(1, 2, 3)
    End With
End Sub
```

As with the chart object variable above, you can define an object variable for the new chart series being added. The following procedure assigns the variable `MyNewSrs` to the new chart series it creates.

```
Sub AddNewSeries()
    Dim MyNewSrs As Series
    Set MyNewSrs = ActiveChart.SeriesCollection.NewSeries
    With MyNewSrs
        .Name = "Fred"
        .Values = "=Sheet1!Y_Range"
        .XValues = Array(1, 2, 3)
    End With
End Sub
```

Top of Page

Too Many Series?

When you create a chart, Excel looks at the selection, and tries to determine how many series you want in the chart. In the Chart Wizard, you see this behavior in step 2, where the Data Range is tentatively filled in for you. When creating a chart in code, you don't get this chance to make it right, and your chart may have any number of series. It is best to clear out all of these initial series, and start from scratch with the series you intend to add. This macro clears the chart:

```
Sub RemoveUnwantedSeries()
    With ActiveChart
        Do Until .SeriesCollection.Count = 0
```

```
            .SeriesCollection(1).Delete
        Loop
    End With
End Sub
```

The chart now appears completely blank, and the only object you can currently access is the chart area. But you can now add series as shown above.

Top of Page

Putting It All Together

Let's put the last few steps together, to produce a robust little procedure that will create a chart sheet with exactly the right number of series in the right place, using the selected range as the chart's data source. The first row contains the series labels, the first column contains the X values, and the rest of the columns contain the Y values for each series.

```
Sub EmbeddedChartFromScratch()
    Dim myChtObj As ChartObject
    Dim rngChtData As Range
    Dim rngChtXVal As Range
    Dim iColumn As Long

    ' make sure a range is selected
    If TypeName(Selection) <> "Range" Then Exit Sub

    ' define chart data
    Set rngChtData = Selection

    ' define chart's X values
    With rngChtData
        Set rngChtXVal = .Columns(1).Offset(1).Resize(.Rows.Count - 1)
    End With

    ' add the chart
    Set myChtObj = ActiveSheet.ChartObjects.Add _
        (Left:=250, Width:=375, Top:=75, Height:=225)
    With myChtObj.Chart

        ' make an XY chart
        .ChartType = xlXYScatterLines

        ' remove extra series
        Do Until .SeriesCollection.Count = 0
            .SeriesCollection(1).Delete
        Loop

        ' add series from selected range, column by column
        For iColumn = 2 To rngChtData.Columns.Count
            With .SeriesCollection.NewSeries
                .Values = rngChtXVal.Offset(, iCOlumn - 1)
                .XValues = rngChtXVal
                .Name = rngChtData(1, iColumn)
            End With
```

```
        Next

    End With

End Sub
```

A similar approach is used in Interactive Chart Creation, which provides dialogs for the user to select the range which will be covered by the chart and the range containing the data to be charted. The series-by-series definition of X and Y ranges is described in Excel XY Chart Variations with VBA to allow much more flexibility in the arrangement of the chart's source data range. The Quick Chart Utility is based on this approach.

Top of Page

VBA Code to Resize and Reposition Chart Objects

Chart Object Size and Position

We learned above that it is easy to define the size and position of a chart at the time of its creation:

```
    Set myChtObj = ActiveSheet.ChartObjects.Add _
        (Left:=100, Width:=375, Top:=75, Height:=225)
```

In the same way, we can position or size the chart, by changing the appropriate properties of the ChartObject:

```
Sub ResizeAndRepositionChart()
    ' The ChartObject is the Chart's parent
    With ActiveChart.Parent
        .Left = 100
        .Width = 375
        .Top = 75
        .Height = 225
    End With
End Sub
```

To adjust a particular chart on the sheet, use `With ActiveSheet.ChartObjects(1)` in place of `With ActiveChart.Parent` in the procedure above.

Top of Page

Cover a Range with a Chart

You can easily configure the chart to cover a specific range of cells on the worksheet. To cover the range D5:K25 with the active chart, run this procedure:

```
Sub CoverRangeWithChart()
    Dim cht As Chart Object
    Dim rng As Range

    Set cht = ActiveChart.Parent
    Set rng = ActiveSheet.Range("D5:K25")

    cht.Left = rng.Left
```

```
    cht.Width = rng.Width
    cht.Top = rng.Top
    cht.Height = rng.Height
End Sub
```

Top of Page

Create an Array of Charts

Suppose you have a lot of charts on a worksheet, and you'd like to arrange them neatly. The following procedure loops through the charts, resizes them to consistent dimensions, and arranges them in systematic rows and columns:

```
Sub ArrangeMyCharts()
    Dim iChart As Long
    Dim nCharts As Long
    Dim dTop As Double
    Dim dLeft As Double
    Dim dHeight As Double
    Dim dWidth As Double
    Dim nColumns As Long

    dTop = 75      ' top of first row of charts
    dLeft = 100    ' left of first column of charts
    dHeight = 225  ' height of all charts
    dWidth = 375   ' width of all charts
    nColumns = 3   ' number of columns of charts
    nCharts = ActiveSheet.ChartObjects.Count

    For iChart = 1 To nCharts
        With ActiveSheet.ChartObjects(iChart)
            .Height = dHeight
            .Width = dWidth
            .Top = dTop + Int((iChart - 1) / nColumns) * dHeight
            .Left = dLeft + ((iChart - 1) Mod nColumns) * dWidth
        End With
    Next
End Sub
```

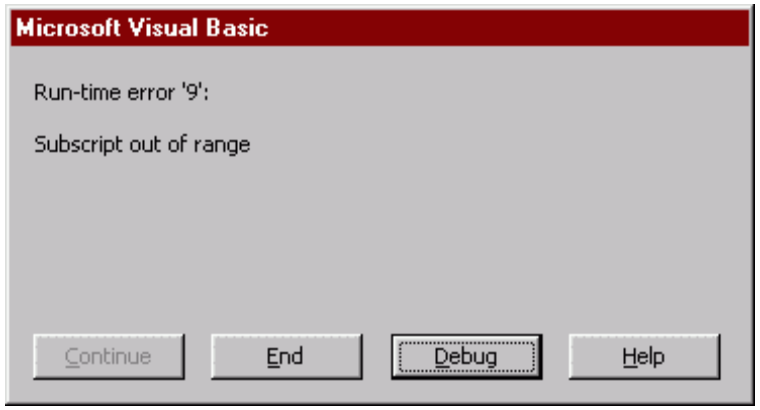
Now that you've had a good laugh at the title of this page ("Error Free VBA"), I'll point out that although you cannot make your VBA code error-free, you can at least try to make it error-resistant. In my Quick Chart VBA page and other pages on this site, I've presented some VBA procedures to help create and modify your charts. In this page I will outline a few basic techniques to reduce the effects of errors in the use of your code.

On Error Resume Next

For our first example, let's try to activate a chart sheet. The following code will activate the chart sheet named "Chart1".

```
Sub ActivateChartSheet1()
    Charts("Chart1").Activate
End Sub
```

This works just fine, as long as there is a chart sheet named "Chart1". But if there is no such chart, you get the following error message:



If this message is unclear to you and me, it must be completely inscrutable to the unfortunate user who only thought he was activating a chart. You can use a simple statement, `On Error Resume Next`, which ignores the error and continues processing with the step after the line that caused the error. Then you can determine what the error was, and react accordingly.

```
Sub ActivateChartSheet2()
    On Error Resume Next
    Charts("Chart1").Activate
    If Err.Number <> 0 Then
        MsgBox "Requested chart does not exist.", _
            vbExclamation, "No Such Chart"
    End If
    On Error Goto 0
End Sub
```

The `If` statement determines whether there was an error, and what the error was. If there is no error up until that point, `Err.Number` is zero, and the message box is not shown. All we did here was notify the user that the chart sheet was not found, as shown in the following message box:

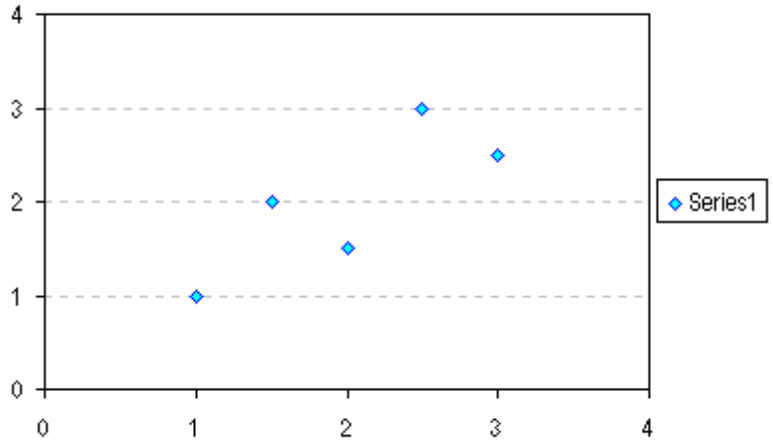
The message box shows that the programmer gave at least some small thought to how the procedure might fall short, and might actually reduce the frustration felt by the user.

We certainly could get fancier; some statements might lead to more than one error, and we would need to provide a different message, or follow a different set of subsequent actions, depending on the error that was encountered. The following procedure illustrates the use of `Select Case` to provide error-specific messages to the user.

```
Sub ActivateChartSheet3()
    On Error Resume Next
    Charts("Chart1").Activate
    If Err.Number <> 0 Then
        Select Case Err.Number
            Case 9
                MsgBox "Requested chart does not exist.", _
                    vbExclamation, "No Such Chart"
            Case Else
                MsgBox "Couldn't activate chart sheet.", _
                    vbExclamation, "I Don't Know Why!"
        End Select
    End If
End Sub
```

Is There an Active Chart?

Many of the charting macros I've demonstrated on these pages work on the active chart. For example, to add a chart title and axis titles to the following chart:

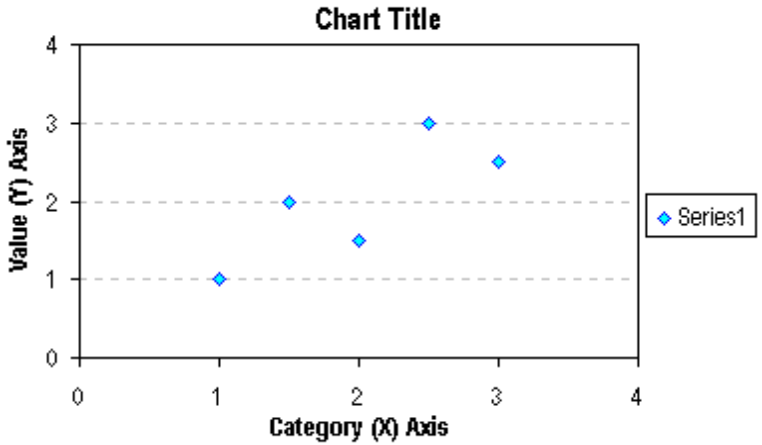


I would write something like this macro:

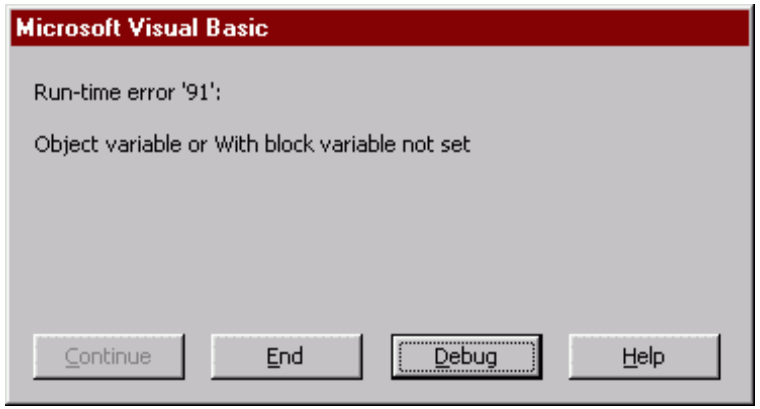
```
Sub AddTitle1()
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Text = "Chart Title"
```

```
        With .Axes(xlCategory, xlPrimary)
            .HasTitle = True
            .AxisTitle.Text = "Category (X) Axis"
        End With
        With .Axes(xlValue, xlPrimary)
            .HasTitle = True
            .AxisTitle.Text = "Value (Y) Axis"
        End With
    End With
End Sub
```

with the following result:



If the user has not selected a chart prior to running the code, however, there is no `ActiveChart`, and the code will fail, with the following error message:



"Oh, no," thinks the user, "what is wrong with this awful code?" You can check for an active chart, and either work on the active chart, or remind the user that he needs to select a chart first.

```
Sub AddTitle2()
    If Not ActiveChart Is Nothing Then
        With ActiveChart
            .HasTitle = True
            .ChartTitle.Text = "Chart Title"
            With .Axes(xlCategory, xlPrimary)
                .HasTitle = True
                .AxisTitle.Text = "Category (X) Axis"
            End With
            With .Axes(xlValue, xlPrimary)
                .HasTitle = True
                .AxisTitle.Text = "Value (Y) Axis"
            End With
        End With
    Else
        MsgBox "Please select a chart and try again.", _
            vbExclamation, "No Chart Selected"
    End If
End Sub
```

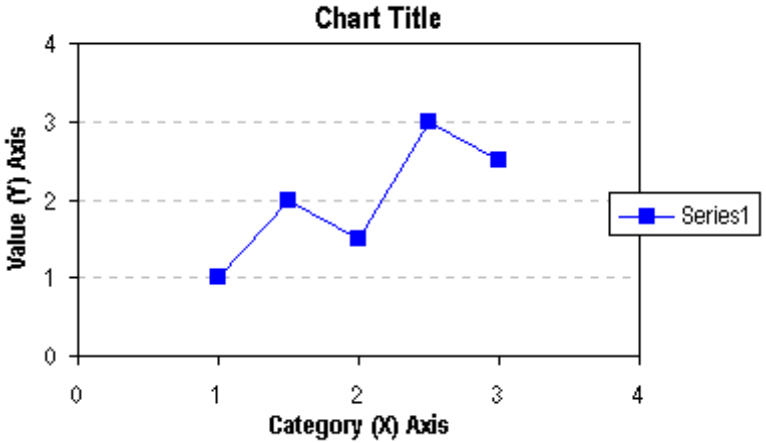
When this message box pops up, the user now thinks, "I forgot to select a chart *again*!"

Is the Correct Object Selected?

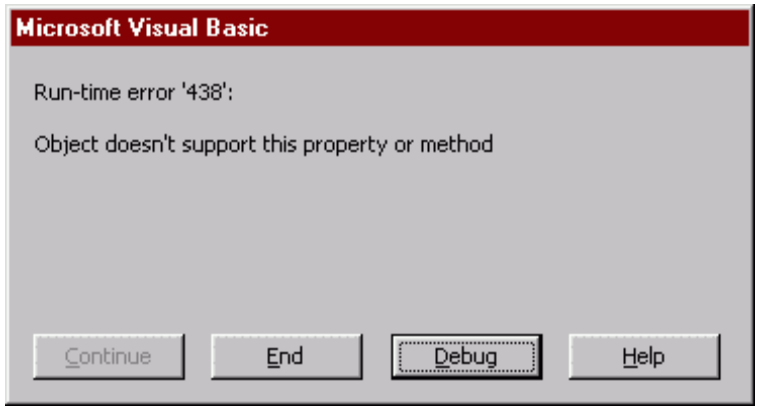
Suppose we take the nice chart above, and change to series formatting so we have a thick blue line connecting large blue square markers. This macro will make short work of our task:

```
Sub BlueSquares1()
    With Selection
        .MarkerBackgroundColorIndex = 5
        .MarkerForegroundColorIndex = 5
        .MarkerStyle = xlSquare
        .Smooth = False
        .MarkerSize = 6
        .Shadow = False
        With .Border
            .ColorIndex = 5
            .Weight = xlThin
            .LineStyle = xlContinuous
        End With
    End With
End Sub
```

And here's the chart:



But if no series has been selected, the code will crash:



Says the user, "What object doesn't support *WHAT* property or method??" Using the `TypeName` function, we can check whether a chart series has been selected.

```
Sub BlueSquares2()
    If TypeName(Selection) = "Series" Then
        With Selection
            .MarkerBackgroundColorIndex = 5
            .MarkerForegroundColorIndex = 5
            .MarkerStyle = xlSquare
            .Smooth = False
            .MarkerSize = 6
```

```
            .Shadow = False
            With .Border
                .ColorIndex = 5
                .Weight = xlThin
                .LineStyle = xlContinuous
            End With
        End With
    Else
        MsgBox "Please select a series and try again.", _
            vbExclamation, "No Series Selected"
    End If
End Sub
```

The subsequent message box reminds the user that he has forgotten an important step.

Conclusion

There is certainly more to error checking than always reminding the user that he forgot to do something. Well-considered code should follow a more organized process so that the user never feels he has done something to cause him to start over from scratch. This requires substantial testing and anticipation about how the user might actually use the macros. A typical approach might follow these steps:

```
' PseudoCode
Is a Chart Selected?
  If not:
    Display UserForm with List of Charts on the Active Sheet
    User Selects a Chart
Is a Series Selected?
  If not:
    Display UserForm with List of Series in the Active Chart
    User Selects a Series
Perform the Intended Actions on the Selected Series of the Selected Chart
```

Diagramm erzeugen (eigenes Blatt und in aktuellem Blatt)

Excel Charts erstellen mit Hilfe von VBA

von: Karl

Geschrieben am: 19.11.2009 15:20:40

Hallo zusammen,

ich bin gerade dabei ein Makro zu erstellen das Daten in einem Chart darstellt.

Soweit eigentlich ganz simpel, aber ich kriege es nicht hin ein Chart in dem aktuellen Arbeitsblatt zu erstellen. Separates Arbeitsblatt funktioniert.

Ich bekomme folgende Fehlermeldung: RTE 438 - Object does not support this property or method

Anbei "mein - John Walkenbachs Excel 2007 Power Programming" Code (sorry falls ich die _ Konventionen mit dem Code einpasten nicht beachte, keine Ahnung wie das geht):

```
Sub CreateCharts()

Dim MyChart As Chart
Dim DataRange As Range

test88 = Selection.Address

Set DataRange = ActiveSheet.Range(test88)

Set MyChart = ActiveSheet.Shapes.AddChart.Add

MyChart.SetSourceData Source:=DataRange
ActiveChart.ChartType = xlColumnClustered


End Sub
```

Vielen Dank schonmal,

Karl

Betrifft: AW: Excel Charts erstellen mit Hilfe von VBA

von: F1

Geschrieben am: 19.11.2009 17:47:06

```
Sub CreateCharts()
Dim MyChart As Chart
Dim DataRange As Range
Dim Test88 As String
Test88 = Selection.Address
Set DataRange = ActiveSheet.Range(Test88)
Set MyChart = Charts.Add
MyChart.SetSourceData Source:=DataRange
ActiveChart.ChartType = xlColumnClustered
ActiveChart.Location Where:=2, Name:="Tabelle1"
End Sub
```

F1

Betrifft: Ergänzung

von: F1

Geschrieben am: 19.11.2009 17:48:43

oder ...

ActiveChart.Location Where:=2, Name:=ActiveSheet.Name

F1

Betrifft: AW: Ergänzung

von: Karl

Geschrieben am: 19.11.2009 18:20:01

F1 auch Dir ein Dankeschoen. Funktioniert super. Weisst Du ob die Methode .ClearToMatchStyle in Excel 2003 funktioniert (bei mir tut sie es naemlich nicht!).

danke,

Karl

Betrifft: ClearToMatchStyle Nicht in Excel 2003

von: F1

Geschrieben am: 19.11.2009 19:48:35

oT

Betrifft: AW: ClearToMatchStyle Nicht in Excel 2003

von: Karl

Geschrieben am: 20.11.2009 11:36:56

Danke!

Betrifft: AW: Excel Charts erstellen mit Hilfe von VBA

von: fcs

Geschrieben am: 19.11.2009 18:09:33

Hall Karl,

wenn du mit Excel 2003 arbeitest, dann könnte Mr. Walkenbach's Excel 2007-Lösung evtl. nicht funktionieren.

Meine unter Excel 2003 funktionierende Methode Daten eines selektierten Zellbereichs in einem Diagramm darzustellen:

```
Sub CreateCharts()
    Dim wks As Worksheet
    Dim MyChart As Chart
    Dim DataRange As Range
    Set wks = ActiveSheet
    Set DataRange = Selection
    Charts.Add
    ActiveChart.Location Where:=xlLocationAsObject, Name:=wks.Name
    Set MyChart = wks.ChartObjects(wks.ChartObjects.Count).Chart
    With MyChart
      .ChartType = xlColumnClustered
      .SetSourceData Source:=DataRange  ', PlotBy:=xlRows
    End With
End Sub
```

Gruß
Franz

Abfragen Mausklick auf Diagramm/Punkte

```
Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
   Dim ElementID As Long, Arg1 As Long, Arg2 As Long
   Dim myX As Variant, myY As Double

   With ActiveChart
      ' Pass x & y, return ElementID and Args
      .GetChartElement x, y, ElementID, Arg1, Arg2
      ' Did we click over a point or data label?
      If ElementID = xlSeries Or ElementID = xlDataLabel Then
         If Arg2 > 0 Then
            ' Extract x value from array of x values
            myX = WorksheetFunction.Index(.SeriesCollection(Arg1).XValues, Arg2)
```

```
            ' Extract y value from array of y values
            myY = WorksheetFunction.Index(.SeriesCollection(Arg1).Values, Arg2)
            ' Display message box with point information
            MsgBox "Button " & Button & vbCrLf & _
                "Series " & Arg1 & vbCrLf _
               & """" & .SeriesCollection(Arg1).Name & """" & vbCrLf _
               & "Point " & Arg2 & vbCrLf _
               & "X = " & myX & vbCrLf _
               & "Y = " & myY
          End If
        End If
    End With
End Sub


Private Sub Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
  MsgBox "Button = " & Button & Chr$(13) & _
       "Shift = " & Shift & Chr$(13) & _
       "X = " & x & " Y = " & y

       If Shift = 0 Then Exit Sub

       a = ActiveChart().Index

       For x = 1 To Charts.Count
       If ActiveChart().Name = Charts(x).Name Then a = x
       Next x


       If Button = 1 And Shift = 1 Then a = a + 1
       If Button = 2 And Shift = 1 Then a = a - 1
       If Charts.Count < a Then a = 2
       If 1 > a Then a = Charts.Count
       If Shift = 1 Then
       Charts(a).Activate
       Exit Sub
       End If

            If Shift = 4 Then
      If Button = 1 Then
        Worksheets("Control").Activate
        Excel.Application.DisplayAlerts = False
        Exit Sub
        End If
      If Button = 2 Then
        Worksheets("Selected").Activate
```

```
    Excel.Application.DisplayAlerts = False
    Exit Sub
    End If

   End If


 If ActiveChart.Axes(xlValue).ScaleType = xlLinear And ActiveChart.Axes(xlCategory).ScaleType = xlLinear Then z = 1
 If ActiveChart.Axes(xlValue).ScaleType = xlLinear And ActiveChart.Axes(xlCategory).ScaleType = xlLogarithmic Then z = 2
 If ActiveChart.Axes(xlValue).ScaleType = xlLogarithmic And ActiveChart.Axes(xlCategory).ScaleType = xlLinear Then z = 3
 If ActiveChart.Axes(xlValue).ScaleType = xlLogarithmic And ActiveChart.Axes(xlCategory).ScaleType = xlLogarithmic Then z = 4

  If z = 1 Then
  ActiveChart.Axes(xlValue).ScaleType = xlLinear
  ActiveChart.Axes(xlCategory).ScaleType = xlLogarithmic
  End If
  If z = 2 Then
  ActiveChart.Axes(xlValue).ScaleType = xlLogarithmic
  ActiveChart.Axes(xlCategory).ScaleType = xlLinear
  End If
  If z = 3 Then
  ActiveChart.Axes(xlValue).ScaleType = xlLogarithmic
  ActiveChart.Axes(xlCategory).ScaleType = xlLogarithmic
  End If
  If z = 4 Then
  ActiveChart.Axes(xlValue).ScaleType = xlLinear
  ActiveChart.Axes(xlCategory).ScaleType = xlLinear
  End If
    Excel.Application.DisplayAlerts = False
End Sub
```

Auslesen angeklickten Diagrammpunkt und bearbeiten Serienname und Datenquelle

Für Seibersdorf schrieb ich eine Routine, die das Anklicken eines Diagrammpunktes erlauben sollte (nur linker Mausklicks – rechte sollten das Kontextmenü weiter auslösen). Wenn man auf JA klickt, soll

- die zweite Serie von Datenpunkten (Backup-Data) entfernt werden
- diese leere zweite Serie soll nun als Auffangbecken für eine neue Serie dienen, die der User durch weitere Mausklicks aus der Serie 1 einzeln befüllt
- und diese zweite Serie soll auch einen neuen Namen in allen Charts erhalten (alle Charts zeigten von den selben Quelldatenzeilen, verschiedene Spaltenwerte) und zudem sollten anschließend gewählte Datenpunkte der Serie 1)

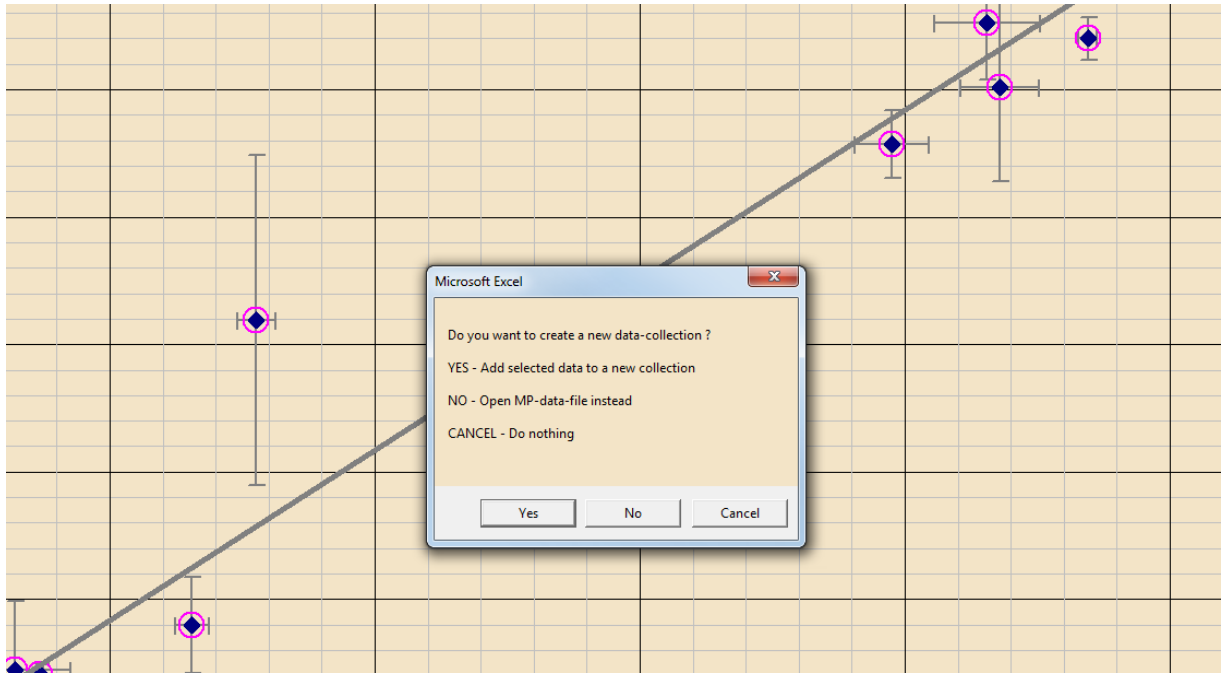
Hier der Code

```
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer
Const VK_LBUTTON = &H1 ' left Mousebutton
Const VK_RBUTTON = &H2 ' right Mousebutton



Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
    Dim ElementID As Long, Arg1 As Long, Arg2 As Long
    Dim myX As Variant, myY As Double
    Dim l As Long, c As Long ' line and column
    Dim wks As Sheets
    Dim ch As Chart


    Dim ANSWER

' --- leave routine if right mouse-button was clicked

    If Button = 2 Then ' a pressed down right mousebutton has the value 2
        Exit Sub ' don't execute procedure when right mousekey is pressed down for context-menu
    End If

' --- get the data of the element clicked on

    With ActiveChart
        ' Pass x & y, return ElementID and Args
        .GetChartElement x, y, ElementID, Arg1, Arg2
        ' Did we click over a point or data label?
        If ElementID = xlSeries Or ElementID = xlDataLabel Then
            If Arg2 > 0 Then
                ' Extract x value from array of x values
                myX = WorksheetFunction.Index(.SeriesCollection(Arg1).XValues, Arg2)
                ' Extract y value from array of y values
                myY = WorksheetFunction.Index(.SeriesCollection(Arg1).Values, Arg2)
            End If
        End If
    End With

 ' --- ask user what to do next

    ANSWER = MsgBox("Do you want to create a new data-collection ?" & vbCrLf & vbCrLf & _
    "YES - Add selected data to a new collection" & vbCrLf & vbCrLf & _
    "NO - Open MP-data-file instead" & vbCrLf & vbCrLf & _
    "CANCEL - Do nothing" & vbCrLf & vbCrLf, vbYesNoCancel)
```

```
' --------------------------
' - Cancelling
' --------------------------

If ANSWER = vbCancel Then
    Exit Sub
End If

' --------------------------
' - Opening MP-Data
' --------------------------

If ANSWER = vbNo Then

 Workbooks.Open Filename:=""
End
End If

' --------------------------
' - Creating new data-collection
' --------------------------

If ANSWER = vbYes Then

 ' check if new collection has been previously created already
 If Sheets("Cts_Data").Range("IV1") = "NEW_COLLECTION_WAS_ADDED" Then GoTo CONTINUE

 ' if new collection will be created now, save this information in sheet "Cts_Data" in Cell IV1
 Sheets("Cts_Data").Range("IV1") = "NEW_COLLECTION_WAS_ADDED"

 ' Ask for collection name
 ANSWER = InputBox("Type the name for the new collection", "Collection-name:", "New")

    ' renaming Backup-Series with new Collection-Name (this MUST be done before Backup-data are removed, because
    For Each ch In ActiveWorkbook.Charts
        ch.SeriesCollection(2).Name = "=" & """" & ANSWER & """"
    Next ch

    ' deleting backup-data
    Sheets("Cts_Data").Range("E2:G65000").ClearContents
    Sheets("Cts_Data").Range("E1") = Replace(Sheets("Cts_Data").Range("E1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("F1") = Replace(Sheets("Cts_Data").Range("F1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("G1") = Replace(Sheets("Cts_Data").Range("G1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("L2:N65000").ClearContents
    Sheets("Cts_Data").Range("L1") = Replace(Sheets("Cts_Data").Range("L1"), "Backup", ANSWER)
```

```
    Sheets("Cts_Data").Range("M1") = Replace(Sheets("Cts_Data").Range("M1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("N1") = Replace(Sheets("Cts_Data").Range("N1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("S2:U65000").ClearContents
    Sheets("Cts_Data").Range("S1") = Replace(Sheets("Cts_Data").Range("S1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("T1") = Replace(Sheets("Cts_Data").Range("T1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("U1") = Replace(Sheets("Cts_Data").Range("U1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("Z2:AB65000").ClearContents
    Sheets("Cts_Data").Range("Z1") = Replace(Sheets("Cts_Data").Range("Z1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("AA1") = Replace(Sheets("Cts_Data").Range("AA1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("AB1") = Replace(Sheets("Cts_Data").Range("AB1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("AG2:AH65000").ClearContents
    Sheets("Cts_Data").Range("AG1") = Replace(Sheets("Cts_Data").Range("AG1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("AH1") = Replace(Sheets("Cts_Data").Range("AH1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("AM2:AN65000").ClearContents
    Sheets("Cts_Data").Range("AM1") = Replace(Sheets("Cts_Data").Range("AM1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("AN1") = Replace(Sheets("Cts_Data").Range("AN1"), "Backup", ANSWER)

    Sheets("Cts_Data").Range("AS2:AT65000").ClearContents
    Sheets("Cts_Data").Range("AS1") = Replace(Sheets("Cts_Data").Range("AS1"), "Backup", ANSWER)
    Sheets("Cts_Data").Range("AT1") = Replace(Sheets("Cts_Data").Range("AT1"), "Backup", ANSWER)

CONTINUE:

    If Arg2 < 1 Then Exit Sub ' if no chart-element was selected: end

    ' Display message box with point information (when backup-datapoint is selected, an error occurs, as the backupdata are already removed)

    ' MsgBox "Series " & Arg1 & vbCrLf & vbCrLf _
                & """" & ActiveChart.SeriesCollection(Arg1).Name & """" & vbCrLf & vbCrLf _
                & "Point " & Arg2 & vbCrLf & vbCrLf _
                & "X = " & myX & vbCrLf & vbCrLf _
                & "Y = " & myY

    If ElementID <> 3 Then Exit Sub ' if no chart-element was selected: end

    ' --- copy selected chart-point to new collection

    l = Arg2 + 1 ' line-number in sheet "Cts_Data" is always the number of the selected element +1

    ' transferring data of the selected element from column B-D into columng E-G
    For c = 2 To 4
       Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
```

```
    Next c

    ' transferring data of the selected element from column I-K into columng L-N
    For c = 9 To 11
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c

    ' transferring data of the selected element from column P-R into columng S-U
    For c = 16 To 18
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c

    ' transferring data of the selected element from column W-Y into columng Z-AB
    For c = 23 To 25
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c

    ' transferring data of the selected element from column AD-AE into columng AG-AH
    For c = 30 To 31
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c

    ' transferring data of the selected element from column AJ-AK into columng AM-AN
    For c = 36 To 37
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c

    ' transferring data of the selected element from column AP-AQ into columng AS-AT
    For c = 42 To 43
        Sheets("Cts_Data").Cells(l, c + 3) = Sheets("Cts_Data").Cells(l, c)
    Next c


End If

End Sub
```



Schleife durch alle Diagramme in einem Tabellenblatt

```
Dim oShape As Shape
For Each oShape In ActiveSheet.Shapes
   If Left(oShape.Name, 5) = "Chart" Then
      Debug.Print oShape.Name
   End If
Next oShape
```

Schleife durch alle Diagramme in allen Tabellenblättern

Folgender Code arbeitet nur für Diagramme, die in Tabellenblättern eingebettet sind und nicht, die eigenständige Arbeitsmappenelemente sind, (Im VBA.Editor erkennt man Tabellenblätter an den Bezeichnungen Sheet1, Sheet2 … und Diagramme (die eigene Arbeitsmappenelemente sind) an Namen wie Chart1, Chart2… - für die Schleife durch eigenständige Chart-Arbeitsmappen-Elemente siehe nächsten Code

```
Dim wks As Worksheet
Dim cht As ChartObject

For Each wks In ActiveWorkbook.Worksheets
wks.Activate
For Each cht In ActiveSheet.ChartObjects
Debug.Print cht.Name
Next cht
Next wks
```

(note that this won't pick up charts that are their own sheet).

Schleife durch alle Diagramme die ein eigenständiges Arbeitsmappenelement sind

Wie schon ganz zu Beginn hier bei den DIAGRAMMEN erklärt können Diagramme

- ein eingebettes Objekt in ein Tabellenblatt sein
- oder ein eigenständiges Arbeitsmappen-Element, das eine eigenständige Registerkarte ist

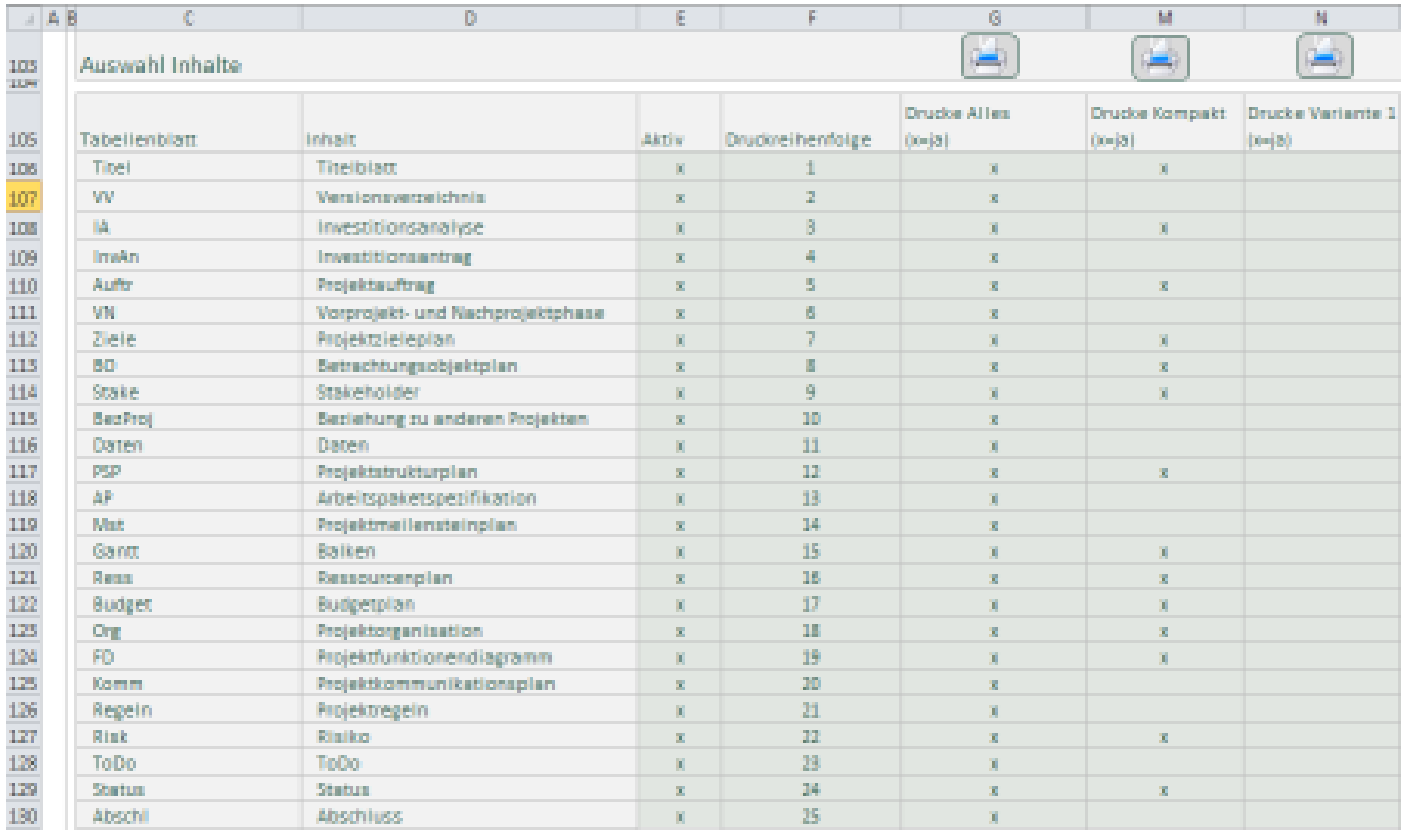
Schleifen, die durch alle Registerkarten-Elemente gehen sollen - sowohl Tabellenblätter als auch Diagramm-Register - arbeiten nicht mit der Worksheet-Methode, sondern mit der Sheet-Methode. (Siehe auch meinen Code im Bereich TABELLENBLÄTTER / Sortieren der Tabellenblätter in Arbeitsmappen)

Damit eine Schleife nun durch alle Diagramme geht, die ein eigenständiges Arbeitsmappen-Element sind, müssen wir vorigen Code abändern:

```
Dim wks As Sheets
Dim ch as Chart

For each ch in ActiveWorkbook.Charts
        Msgbox ch.name
Next ch
```

DRUCKEN + PDF

Auswählbare Seiten ausdrucken



Auswahl Inhalte

Tabellenblatt	Inhalt	Aktiv	Druckreihenfolge	Drucke Alles (x=ja)	Drucke Kompakt (x=ja)	Drucke Variante 1 (x=ja)
Titel	Titelblatt	x	1	x	x	
VV	Versionsverzeichnis	x	2	x		
IA	Investitionsanalyse	x	3	x	x	
InvAn	Investitionsantrag	x	4	x		
Auftr	Projektauftrag	x	5	x	x	
VN	Vorprojekt- und Nachprojektphase	x	6	x		
Ziele	Projektzieleplan	x	7	x	x	
BO	Betrachtungsobjektplan	x	8	x	x	
Stake	Stakeholder	x	9	x	x	
BezProj	Beziehung zu anderen Projekten	x	10	x		
Daten	Daten	x	11	x		
PSP	Projektstrukturplan	x	12	x	x	
AP	Arbeitspaketspezifikation	x	13	x		
Mst	Projektmeilensteinplan	x	14	x		
Gantt	Balken	x	15	x	x	
Ress	Ressourcenplan	x	16	x	x	
Budget	Budgetplan	x	17	x	x	
Org	Projektorganisation	x	18	x	x	
FD	Projektfunktionendiagramm	x	19	x	x	
Komm	Projektkommunikationsplan	x	20	x		
Regeln	Projektregeln	x	21	x		
Risk	Risiko	x	22	x	x	
ToDo	ToDo	x	23	x		
Status	Status	x	24	x	x	
Abschl	Abschluss	x	25	x		

```
Sub Print_Alles()

Dim Reihenfolge As Integer
Dim Anzahltabellen As Integer

Anzahltabellen = 25 ' derzeit max. 25 auszudruckende Blätter

Application.EnableEvents = False

For T = 1 To Anzahltabellen ' 25 mal müssen wir durch die gesamte Auflistung von Tabellen, da wir 25 mal die nächste Reihenfolge suchen müssen

    For Z = 106 To 106 + Anzahltabellen - 1 ' Schleife durch die Auflistung der 25 Tabellen
        Reihenfolge = Reihenfolge + 1 ' bei jeden Durchlauf suchen wir die nächsten Reihenfolgenummer
        If Cells(Z, 6) = Reihenfolge Then ' wenn wir sie gefunden haben ...
            If Cells(Z, 7) <> "" Then ' ... schauen, ob die aktuelle Tabelle überhaupt ausgedruckt werden soll
```

```
            Worksheets(Cells(Z, 3).Value).PrintOut
          End If
        End If
    Next Z

Next T

Application.EnableEvents = True

End Sub
```

Alle Seiten mit einem Wert in Zelle A1 ausdrucken

With this macro you loop through every worksheet and if there Is a
value in a certain cell it will add the sheet to the array and print it.
You can also test a for a word like Sh.Range("A1").Value = "PrintMe"

```
Sub Print_All_Worksheets_With_Value_In_A1()
    Dim Sh As Worksheet
    Dim Arr() As String
    Dim N As Integer
    N = 0
    For Each Sh In ActiveWorkbook.Worksheets
        If Sh.Visible = xlSheetVisible And Sh.Range("A1").Value <> "" Then
            N = N + 1
            ReDim Preserve Arr(1 To N)
            Arr(N) = Sh.Name
        End If
    Next
    With ActiveWorkbook
        .Worksheets(Arr).PrintOut
    End With
End Sub
```

Alle X Zeilen einen Seitenumbruch einfügen

If row 1 is a header row and you want to print it on every page then
change RW + 1 to RW + 2 and use File>Page Setup>Sheet to fill in $1:$1
in the "Rows to repeat at top: " box.

This example will add breaks every 20 rows from row 1 till the last row with data in column A.

```
Sub Insert_PageBreaks()
    Dim Lastrow As Long
```

```
    Dim Row_Index As Long
    Dim RW As Long

    'How many rows do you want between each page break
    RW = 20

    With ActiveSheet
        'Remove all PageBreaks
        .ResetAllPageBreaks

        'Search for the last row with data in Column A
        Lastrow = .Cells(Rows.Count, "A").End(xlUp).Row

        For Row_Index = RW + 1 To Lastrow Step RW
            .HPageBreaks.Add Before:=.Cells(Row_Index, 1)
        Next
    End With
End Sub
```

Ausdruck von Arbeitsmappe, Tabellenblatt, Auswahl ...

Look in the VBA help for PrintOut and see that you can use the following arguments.
expression.PrintOut(From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate, PrToFileName)

Note: Remember that you can't print sheets that are hidden
If you print the whole workbook with the first example there is no problem but the example
for Sheets or worksheets (example 2 and 3) will not work if there are hidden sheets.

```
ActiveWorkbook.PrintOut
'the whole workbook

Worksheets.PrintOut
'all worksheets

Sheets.PrintOut
'all sheets

Sheets(Array("Sheet1", "Sheet3")).PrintOut
'all sheets in the array

ActiveWindow.SelectedSheets.PrintOut
'print all selected sheets

ActiveSheet.PrintOut
'only the activesheet
```

```
Sheets("Sheet1").PrintOut
'only "Sheet1"

Selection.PrintOut
'print only the selection

Range("C1:D5").PrintOut
'print range
```

Druckbereich einer Tabelle automatisch auf letzte Zeile einstellen

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)

' --- Druckbereich Reparieren --
    ' Reisekostentabelle RK
    If ActiveSheet.Name = RK.Name Then
        RK.PageSetup.PrintArea = "$E$$:$AD$" & RK.Range("E65536").End(xlUp).Row
 ' oder LETZTEZELLE(RK.Name).Row - aber das nimmt auch leere Zellen !
    End If

End Sub
```

Benötigt wird meine Funktion LETZTEZELLE

```
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' ermittelt die letzte Zelle einer Tabelle mit Inhalt oder Zellformat

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False
```

```
On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function
```

Drucken-Dialog öffnen

```
Application.Dialogs(xlDialogPrint).Show
```

Druck verhindern

Die einfachste Lösung besteht im Definieren eines Druckbereiches, der nur eine leere Zelle umfasst und dann den Blattschutz zu setzen

LÖSUNG 1

Man blendet bei jedem Speichern mit VBA die Tabelle aus - damit sie ohne VBA nicht eingeblendet werden kann (veryhidden).

Dann fügt man folgenden Code ein:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Cancel = True
    MsgBox "Drucken nicht erlaubt !", vbCritical
End Sub
```

Man kann das Tool zwar ohne VBA starten (Pause / deaktivieren), aber ohne Code kommt man nicht zum gewünschten Blatt.

Und sobald der VBA-Code läuft, unterbindet man den Ausdruck

Eine bestimmte Anzahl von Ausdruckexemplaren auslesen in Zelle, Fußzeile oder Kopfzeile

This example will print ? copies of the same sheet (It use a Input box to ask you how many)
It will copy the page number in cell A1 or in the Header or Footer.

Note: you can use this for testing .PrintOut preview:=True

```
Sub PrintCopies_ActiveSheet_1()
   Dim CopiesCount As Long
   Dim CopieNumber As Long
   CopiesCount = Application.InputBox("How many copies do you want", Type:=1)

   For CopieNumber = 1 To CopiesCount
      With ActiveSheet
         ' This example print the number in cell A1
         .Range("A1").Value = CopieNumber & " of " & CopiesCount

         'If you want the number in the footer use this line
         '.PageSetup.LeftFooter = CopieNumber & " of " & CopiesCount

         'Print the sheet
         .PrintOut
      End With
   Next CopieNumber
End Sub
```

The example below continue printing where It left off, such as today you print
numbered pages 1-25 and the next time when you enter 10 in the input box it print 26-35.

```
Sub PrintCopies_ActiveSheet_2()
' This example print the number in cell A1
   Dim CopiesCount As Long
```

```
    Dim CopieNumber As Long
    CopiesCount = Application.InputBox("How many copies do you want", Type:=1)

    With ActiveSheet
        If Not IsNumeric(.Range("A1").Value) Then .Range("A1").Value = 0

        For CopieNumber = 1 To CopiesCount
            .Range("A1").Value = .Range("A1").Value + 1

            'Print the sheet
            .PrintOut

        Next CopieNumber
    End With
End Sub
```

Formeln ausdrucken

If you want to print your formulas then you can toggle the view with this

Excel 97-2003: Tools - Options - View and check Formulas
Excel 2007: "Show Formulas" in the Formula Auditing group on the Formula tab

Or the shortcut CTRL ` in all Excel versions

Check out also this example from John Walkenbach.
http://j-walk.com/ss/excel/tips/tip37.htm

And David McRitchie's site
http://www.mvps.org/dmcritchie/excel/formula.htm

Kopfzeile / Fußzeile nicht ausdrucken auf allen Seiten

The example below will only print the right header on the first page of the ActiveSheet.

You have these options
(LeftHeader, CenterHeader, RightHeader, LeftFooter, CenterFooter, RightFooter)

Check out the VBA help for all formatting codes.
Look for "Formatting Codes for Headers and Footers"

You can use something like this :

```
.CenterFooter = "&8Page &P & of &N"
.RightFooter = "&8Last Saved : &B" & ActiveWorkbook.BuiltinDocumentProperties("Last Save Time")
.LeftFooter = "&8" & ActiveWorkbook.FullName & Chr(10) & "Sheetname : &B" & ActiveSheet.Name
```

```
Sub Test()
    Dim TotPages As Long
    TotPages = Application.ExecuteExcel4Macro("GET.DOCUMENT(50)")
    With ActiveSheet.PageSetup
        .RightHeader = "Your Header info"
        ActiveSheet.PrintOut From:=1, To:=1
        .RightHeader = ""
        ActiveSheet.PrintOut From:=2, To:=TotPages
    End With
End Sub
```

Tip: If you not want to print the last page you can do this

```
ActiveSheet.PrintOut From:=2, To:=TotPages -1
```

You can also make a different header for the last page if you want

Letztes Speicherdatum in Fußzeile jeder Seite andrucken

If you copy this in the ThisWorkbook module it will print the Last Save Time
in the Right Footer of every sheet when you use one of the Print options in Excel.
Note: The Property is not working correct in Excel 97.

Where do I paste the code that I want to use in my workbook
http://www.rondebruin.nl/code.htm

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Dim wkSht As Worksheet
    For Each wkSht In ThisWorkbook.Worksheets
        wkSht.PageSetup.RightFooter = "&8Last Saved : " & _
            Format(ThisWorkbook.BuiltinDocumentProperties("Last Save Time"), _
            "yyyy-mmm-dd hh:mm:ss")
    Next wkSht
End Sub
```

If you use Excel 97 you can use this to add the Date in a worksheet cell

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
                                Cancel As Boolean)
    'If you save the file the date/time will be placed in cell A1 of Sheet1
    Sheets("Sheet1").Range("A1").Value = Format(Now, "yyyy-mmm-dd hh:mm:ss")
End Sub
```

Mehrere Tabellenblätter als PDF exportieren

```
Sub EXPORT_PDF()

Dim DATEINAME As String

' Markieren der gewünschten Tabellen
Sheets(Array(DIR_ORGA.Name, DIR_MA.Name, DIR_PK.Name, DIR_PENS.Name)).Select

' Speichern als PDF
DATEINAME = ActiveWorkbook.Path & "\PDF"


    With ActiveSheet
        .ExportAsFixedFormat Type:=xlTypePDF, Filename:=DATEINAME & ".pdf", _
            Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False, OpenAfterPublish:=True
    End With

End Sub
```

Nicht fortlaufende Bereiche Ausdrucken

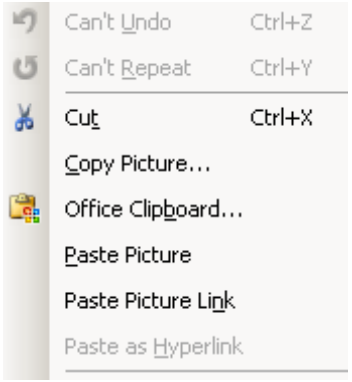
Insert a new worksheet (you're going to use this for printing).
Do this for each area you want to print on the same page.
(The areas don’t have to be on the same sheet)

If you change the original range, the picture will change too (values and formatting!).
You can use this sheet always to print because it will always update if you change

The original data.

Excel 97-2003

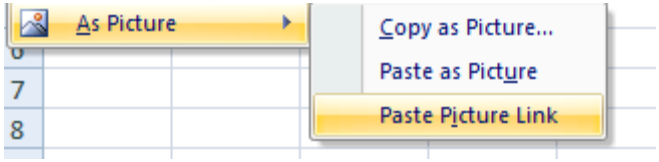
Select the range
Edit | Copy in the menu bar (or use Ctrl-C)
Go to the new worksheet and with the Shift key pressed click on Edit | Paste Picture Link.
Go back and do the same for the other areas.



Note: You see that the edit menu will change when you press the shift button when you click on Edit

Excel 2007

Select the range
Home tab | Clipboard group | Copy (or use Ctrl-C)
Go to the new worksheet and Click on the arrow on the Paste button in the Clipboard group on the Home tab and choose As Picture >Paste Picture Link.
Go back and do the same for the other areas.



Print selection or range with one or more areas with a macro.

The macro will add a new sheet and copy all the selection areas on it.
Then it will print and delete the sheet.

```
Sub Test()
    Dim Destrange As Range
    Dim Smallrng As Range
    Dim Newsh As Worksheet
    Dim Ash As Worksheet
    Dim Lr As Long

    Application.ScreenUpdating = False

    Set Ash = ActiveSheet
    Set Newsh = Worksheets.Add
    Ash.Select

    Lr = 1

    'You can also use a range with more areas like this
    'For Each smallrng In Ash.Range("A1:C1,D10:G20,A30").Areas

    For Each Smallrng In Selection.Areas
        Smallrng.Copy
        Set Destrange = Newsh.Cells(Lr, 1)
        Destrange.PasteSpecial xlPasteValues
        Destrange.PasteSpecial xlPasteFormats
        Lr = Lr + Smallrng.Rows.Count
    Next Smallrng

    Newsh.Columns.AutoFit

    Newsh.PrintOut

    Application.DisplayAlerts = False
    Newsh.Delete
    Application.DisplayAlerts = True

    Application.ScreenUpdating = True

End Sub
```

PDF-Datei Öffnen

Code der unter 32 Bit und 64 Bit gleichermaßen läuft:

```
Sub OEFFNE_DATEI(Dateipfad As String)


If Dir(Dateipfad) = "" Then
       MsgBox "Die Datei gibt's leider nicht"
      Else
       ActiveWorkbook.FollowHyperlink Dateipfad
End If



End Sub

Sub TESTE()

OEFFNE_DATEI ("C:\13.pdf")

End Sub
```

PDF-Export aktuelles Tabellenblatt + Vermailen

```
Sub FAKTURSPEICHERN()
' Der SPEICHER-Button in den 6 Fakturen + Angebot + Lieferschein

If Sheets("EINSTELLUNGEN").Range("H7") = 2 Then On Error Resume Next

    Dim PFAD ' Nur der Speicherpfad
    Dim DATEINAME ' nur der Dateiname
    Dim GESAMTPFAD ' Gesamter Speicherpfad inkl Dateiname
    Dim RECHNUNGSNUMMER
    Dim DATUM
    Dim BRUTTOBETRAG
    Dim KUNDE As String

    Dim App, Itm ' Variablen für Email
    Dim EMAIL ' Email-Adresse

    KUNDE = ActiveSheet.Range("B4")
    If InStr(KUNDE, " ") > 0 Then ' wenn es ein Leerzeichen gibt, nimm nur den Namen bis zum Leerzeichen
```

```
        KUNDE = Left(KUNDE, InStr(KUNDE, " "))
    End If

    PFAD = "D:\EIGENE DATEIEN\SONSTIGES\' MY KASSA BANK\AR " & Year(Now) & "\"
    RECHNUNGSNUMMER = ActiveSheet.Range("B17").Value
    BRUTTOBETRAG = ActiveSheet.Range("H41").Value
    DATUM = ActiveSheet.Range("D17").Value
    DATEINAME = "AR" & RECHNUNGSNUMMER & " " & KUNDE & " " & BRUTTOBETRAG & ".pdf"
    GESAMTPFAD = PFAD & DATEINAME

    With ActiveSheet
        .ExportAsFixedFormat Type:=xlTypePDF, Filename:=GESAMTPFAD, Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False,
OpenAfterPublish:=False
    End With

    MsgBox "Die Datei wurde gespeichert unter " & vbCrLf & vbCrLf & GESAMTPFAD

    EMAIL = Range("D11")

    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        With Itm
            .Subject = "Aktuelle Honorarnote"
            .to = EMAIL
            '.Cc =
            .Attachments.Add (GESAMTPFAD)
            .Body = Range("L3") & vbCrLf & vbCrLf & "Ich erlaube mir im Anhang die aktuelle Honorarnote einzufügen." & vbCrLf & vbCrLf & "Für Fragen dazu
stehe ich Ihnen natürlich sehr gerne zur Verfügung."
            .display
        End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunden warten
    Itm.display

    Set App = Nothing
    Set Itm = Nothing

    Exit Sub


KEINOUTLOOK:
```

```
    MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
    Exit Sub

End Sub
```

PDF-Export Allgemeines

Ab 2007 gibt’s intern einen PDF-Export - davor nur über Tools.

Wobei 2007 war es nur kurz - und nun gibt es ein eigenes Addin, das erst installiert werden muss:

http://www.microsoft.com/en-us/download/details.aspx?id=9943

Use the "ExportAsFixedFormat" function to perform a "Save to PDF." This function can be invoked from any worksheet (e.g., ActiveSheet.ExportAsFixedFormat) or for the workbook as a whole (e.g., ActiveWorkbook.ExportAsFixedFormat).

You can set a number of options with this command; the option values shown below are Excel's default values.

[Mandatory] Type:= xlTypePDF

[Mandatory] Filename:= <path and name of file with .pdf suffix>

[Optional] Quality:= xlQualityStandard (xlQualityStandard or xlQualityMinimum)

[Optional] IncludeDocProperties:= True ("True" means that the document properties, such as author name and document title, will be included in the PDF file; "False" means that these properties will not be set in the PDF file.)

[Optional] IgnorePrintAreas:= False ("True" means that all content in the spreadsheet should be included; "False" indicates that only the information within your set printing areas will be included.)

[Optional] From:= 1 (This is the page number where Excel should start the save. If this is omitted, then Excel begins at the first page.)

[Optional] To:= 5 (This is the page number where Excel should finish the save. If this is omitted, then Excel ends on the last page.)

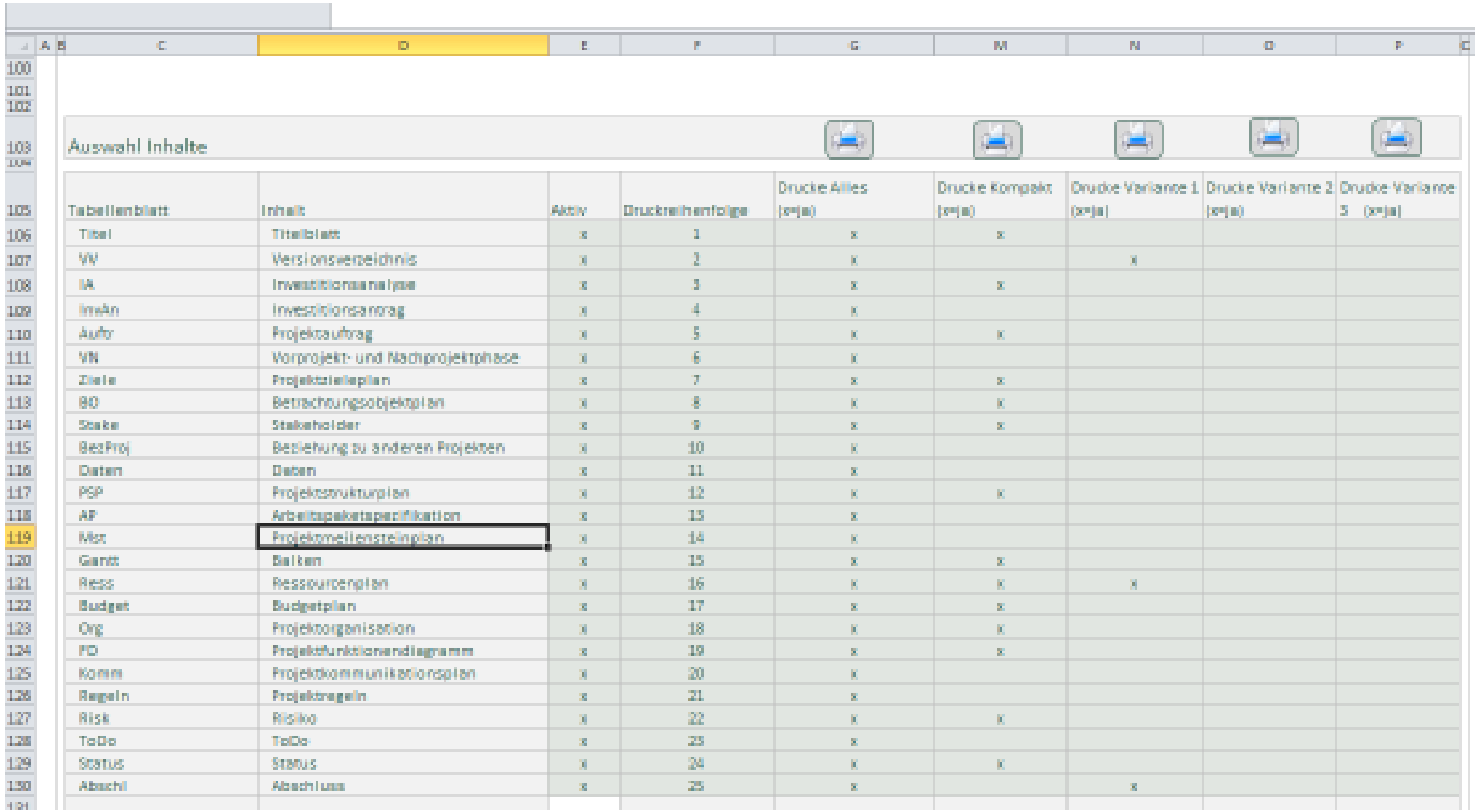
[Optional] OpenAfterPublish:= False ("True" indicates that Excel should launch your default reader for PDF files with the PDF file open; "False" indicates that the new PDF file should not be automatically opened.)

PDF aus allen Tabellen machen (ab 2007)

Ab 2007 gibt’s intern einen PDF-Export - davor nur über Tools.

Wobei 2007 war es nur kurz - und nun gibt es ein eigenes Addin, das erst installiert werden muss:

http://www.microsoft.com/en-us/download/details.aspx?id=9943



Auswahl Inhalte

Tabellenblatt	Inhalt	Aktiv	Druckreihenfolge	Drucke Alles (x=ja)	Drucke Kompakt (x=ja)	Drucke Variante 1 (x=ja)	Drucke Variante 2 (x=ja)	Drucke Variante 3 (x=ja)
Titel	Titelblatt	x	1	x	x			
VV	Versionsverzeichnis	x	2	x		x		
IA	Investitionsanalyse	x	3	x	x			
InvAn	Investitionsantrag	x	4	x				
Auftr	Projektauftrag	x	5	x	x			
VN	Vorprojekt- und Nachprojektphase	x	6	x				
Ziele	Projektzielplan	x	7	x	x			
BO	Betrachtungsobjektplan	x	8	x	x			
Stake	Stakeholder	x	9	x	x			
BezProj	Beziehung zu anderen Projekten	x	10	x				
Daten	Daten	x	11	x				
PSP	Projektstrukturplan	x	12	x	x			
AP	Arbeitspaketspezifikation	x	13	x				
Mst	Projektmeilensteinplan	x	14	x				
Gantt	Balken	x	15	x	x			
Ress	Ressourcenplan	x	16	x	x	x		
Budget	Budgetplan	x	17	x	x			
Org	Projektorganisation	x	18	x	x			
FD	Projektfunktionendiagramm	x	19	x	x			
Komm	Projektkommunikationsplan	x	20	x				
Regeln	Projektregeln	x	21	x				
Risk	Risiko	x	22	x	x			
ToDo	ToDo	x	23	x				
Status	Status	x	24	x	x			
Abschl	Abschluss	x	25	x		x		

Wichtig bei meinem Code waren zwei Dinge, die sonst zu kommentarlosen ERROR 400 Fehlern führten:

1.) Die Tabellen mussten sichtbar sein, damit sie die Zeile " Sheets(Auswahl).Select " auswählen konnte vor dem Export
2.) Auf das Verzeichnis musste man Schreibrechte haben (C:\123.pdf geht ja z.B. ab Vista nicht mehr) - ich speicherte daher gerne ins selbe Verzeichnis wie die XLS(m)-Datei auch das PDF mit DATEINAME = Left(ActiveWorkbook.FullName, InStrRev(ActiveWorkbook.FullName, ".") - 1) & ".pdf"

```
Sub Export_PDF_All()


Dim Auswahl(1 To 25)
Dim Reihenfolge As Integer
Dim Anzahltabellen As Integer
Dim wks As Worksheet

Anzahltabellen = 25 ' derzeit max. 25 auszudruckende Blätter

Reihenfolge = 1

' Feststellen der Tabellenblattnamen und speichern im Array Auswahl
For T = 1 To Anzahltabellen ' 25 mal müssen wir durch die gesamte Auflistung von Tabellen, da wir 25 mal die nächste Reihenfolge suchen müssen

    For Z = 106 To 106 + Anzahltabellen - 1 ' Schleife durch die Auflistung der 25 Tabellen
        If Cells(Z, 6) = Reihenfolge Then ' wenn wir sie gefunden haben ...
            If Cells(Z, 7) <> "" Then ' ... schauen, ob die aktuelle Tabelle überhaupt ausgedruckt werden soll
                Auswahl(Reihenfolge) = Cells(Z, 3).Value
            End If
            Reihenfolge = Reihenfolge + 1 ' bei jeden Durchlauf suchen wir die nächsten Reihenfolgenummer
            Exit For
        End If
    Next Z

Next T

Application.EnableEvents = False

' Markieren der gewünschten Tabellen
Sheets(Auswahl).Select

DATEINAME = Left(ActiveWorkbook.FullName, InStrRev(ActiveWorkbook.FullName, ".") - 1) & ".pdf"

' Speichern als PDF

    With ActiveSheet
        .ExportAsFixedFormat Type:=xlTypePDF, Filename:=DATEINAME, _
            Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False, OpenAfterPublish:=True
    End With

Application.EnableEvents = True

End Sub
```

PDF aus einzelnen Tabellen machen (ab 2007)

Siehe Bild von PDF AUS ALLEN TABELLEN MACHEN (ab 2007)

```
Sub Export_PDF_Kompakt()

Dim Auswahl()
Dim Reihenfolge As Integer
Dim Anzahltabellen As Integer
Dim wks As Worksheet

Anzahltabellen = 25 ' derzeit max. 25 auszudruckende Blätter

Reihenfolge = 1

' Feststellen der Tabellenblattnamen und speichern im Array Auswahl
For T = 1 To Anzahltabellen ' 25 mal müssen wir durch die gesamte Auflistung von Tabellen, da wir 25 mal die nächste Reihenfolge suchen müssen

   For Z = 106 To 106 + Anzahltabellen - 1 ' Schleife durch die Auflistung der 25 Tabellen
      If Cells(Z, 6) = Reihenfolge Then ' wenn wir sie gefunden haben ...

         If Cells(Z, 13) <> "" Then ' ... schauen, ob die aktuelle Tabelle überhaupt ausgedruckt werden soll
               If arraysize = 0 Then
                  ReDim Preserve Auswahl(arraysize)
                  Auswahl(0) = Cells(Z, 3).Value
                  arraysize = arraysize + 1
               Else
                  ReDim Preserve Auswahl(arraysize)
                  Auswahl(arraysize) = Cells(Z, 3).Value
                  arraysize = arraysize + 1
               End If
         End If
         Reihenfolge = Reihenfolge + 1 ' bei jeden Durchlauf suchen wir die nächsten Reihenfolgenummer
         If Reihenfolge > Anzahltabellen Then GoTo FERTIG
         Exit For
      End If
   Next Z

Next T

FERTIG:
```

```
Application.EnableEvents = False

' Markieren der gewünschten Tabellen
Sheets(Auswahl).Select


' Speichern als PDF

DATEINAME = ActiveWorkbook.Name

If InStr(1, DATEINAME, ".") > 0 Then

   DATEINAME = Left(DATEINAME, InStr(1, DATEINAME, ".") - 1)

End If


   With ActiveSheet
      .ExportAsFixedFormat Type:=xlTypePDF, Filename:=Tabelle4.Range("G102") & DATEINAME & ".pdf", _
         Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False, OpenAfterPublish:=True
   End With

Tabelle4.Activate

Application.EnableEvents = True
End Sub
```

PDF aus Exceltabelle machen

```
Sub PDF_erstellen()
Dim sPrinter As String
Dim NewDateiname As String
Dim NewPfad As String
NewDateiname = "TEST"
NewPfad = "C:\"
Application.ScreenUpdating = False
sPrinter = Application.ActivePrinter

Application.ActivePrinter = "BroadGun pdfMachine auf Ne02:"
Drucker = Application.ActivePrinter
prtcmd = NewPfad & NewDateiname
Application.SendKeys (prtcmd), True
Application.SendKeys "{ENTER}", True
```

```
ActiveWindow.SelectedSheets.PrintOut ActivePrinter:=Drucker
Application.ActivePrinter = sPrinter
Application.ScreenUpdating = True
MsgBox "Datei " & prtcmd & " erstellt!"
End Sub
```

Sichtbare, ausgeblendete oder alle Blätter drucken

If you want to print a whole workbook you can use this code line

```
ThisWorkbook.PrintOut
```
Or
```
ActiveWorkbook.PrintOut
```

But this will not print hidden Worksheets.
You can use this macro to print hidden and visible Worksheets

```
Sub Print_Hidden_And_Visible_Worksheets()
'Dave Peterson
    Dim CurVis As Long
    Dim sh As Worksheet
    For Each sh In ActiveWorkbook.Worksheets
        With sh
            CurVis = .Visible
            .Visible = xlSheetVisible
            .PrintOut
            .Visible = CurVis
        End With
    Next sh
End Sub
```

To print only hidden sheets use

```
        With Sh
            CurVis = .Visible
            If CurVis >= 0 Then
                .Visible = xlSheetVisible
                .PrintOut
                .Visible = CurVis
            End If
        End With
```

Ungerade und gerade Seiten ausdrucken

This option is not available in Excel but you can use a macro to do it.

```
Sub Print_Odd_Even()
   Dim Totalpages As Long
   Dim StartPage As Long
   Dim Page As Integer

   StartPage = 1  '1 = Odd and 2 = Even

   'Or use the InputBox suggestion from Gord Dibben
   'StartPage = InputBox("Enter 1 for Odd, 2 for Even")

   Totalpages = Application.ExecuteExcel4Macro("GET.DOCUMENT(50)")
   For Page = StartPage To Totalpages Step 2
      ActiveSheet.PrintOut from:=Page, To:=Page, _
                     Copies:=1, Collate:=True
   Next
End Sub
```

Zeilen, Spalten und Zellen ausblenden beim Ausdruck

AutoFilter or Advanced Filter is also a very good way to print only the things you want in the sheet, filter the range with your criteria and print the sheet.
Check out this site for examples http://www.contextures.com/tiptech.html

But it is not always possible to get the result with a filter, See the examples below for another way to hide Rows/Cells.

If you use one of the Print options in Excel the event below will check the ActiveSheet name and run the code. This example will run if the ActiveSheet name = “Sheet1”

1) It will hide row 10:15
2) Print
3) Unhide row 10:15

Copy/Paste this event in the Thisworkbook module

Where do I paste the code that I want to use in my workbook

http://www.rondebruin.nl/code.htm

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    If ActiveSheet.Name = "Sheet1" Then
        Cancel = True
        Application.EnableEvents = False
        Application.ScreenUpdating = False
        With ActiveSheet
            .Rows("10:15").EntireRow.Hidden = True
            .PrintOut
            .Rows("10:15").EntireRow.Hidden = False
        End With
        Application.EnableEvents = True
        Application.ScreenUpdating = True
    End If
End Sub
```

Or hide columns (this example hide column B and D)

```
        With ActiveSheet
            .Range("B1,D1").EntireColumn.Hidden = True
            .PrintOut
            .Range("B1,D1").EntireColumn.Hidden = False
        End With
```

Or hide all rows with a blank cell in column A

```
        With ActiveSheet
            On Error Resume Next
            .Columns("A").SpecialCells(xlCellTypeBlanks).EntireRow.Hidden = True
            .PrintOut
            .Columns("A").SpecialCells(xlCellTypeBlanks).EntireRow.Hidden = False
            On Error GoTo 0
        End With
```

Make the Font white of a range

You can replace the red lines in the BeforePrint event above
with the example code below if you want to use it.
Or you can use the examples below in a macro.

1) It will make the font white
2) Print
3) Make the font black again

```
    'Range with one area
    With ActiveSheet
        .Range("B10:B14").Font.ColorIndex = 2
        .PrintOut
```

```
    .Range("B10:B14").Font.ColorIndex = 1
End With

'Range with more areas
With ActiveSheet
    .Range("A1:A3,B10:B14,C12").Font.ColorIndex = 2
    .PrintOut
    .Range("A1:A3,B10:B14,C12").Font.ColorIndex = 1
End With

'All cells with a error
With ActiveSheet
    .Cells.SpecialCells(xlCellTypeFormulas, xlErrors).Font.ColorIndex = 2
    .PrintOut
    .Cells.SpecialCells(xlCellTypeFormulas, xlErrors).Font.ColorIndex = 1
End With
```

Use conditional formatting

Make the Font white with conditional formatting
http://www.contextures.com/xlCondFormat03.html#Print

If you only want to print unprotected cells then use Dave Peterson's example below

Pick out a cell that you can use for a flag (say X1--but you can use any cell
you want, but keep it out of the print range).

Then select your range to hide/print.

Excel 97-2003: Format|Conditional formatting
Excel 2007: "Conditional formatting" In the Styles group on the Home tab
Click on New Rule and choose "Use a formula to determine which cells to format"

The formula is:
=AND($X$1="hide",CELL("Protect",A1)=1)
make it white font on white fill (or blue on blue or ...)
Note: A1 in the formula is the first cell in your selection

When you want to print, enter hide in X1 to hide, print and clear X1 when you're ready.

Hide Empty rows, Print and unhide the rows

This example will loop through every row in the range
Set rng = Sheets("Sheet1").Range("A1:A30")

If every cell in column A:G is empty it will hide that row.
After the loop it print the sheet and then unhide the rows.

Change "A1:G1" in the macro to the cells you want.
You can also use this with non contiguous ranges like "B1,D1:G1"

```
Sub Hide_Print_Unhide()
    Dim rw As Long
    Dim rng As Range
    Dim cell As Range

    Application.ScreenUpdating = False

    Set rng = Sheets("Sheet1").Range("A1:A30")

    With rng.Columns(1)
        For Each cell In rng
            If Application.WorksheetFunction.CountA( _
               .Parent.Cells(cell.Row, 1).Range("A1:G1")) = 0 Then _
               .Parent.Rows(cell.Row).Hidden = True
        Next cell
        .Parent.PrintOut
        .EntireRow.Hidden = False
    End With

    Application.ScreenUpdating = True
End Sub
```

E-MAIL + OUTLOOK

ALLGEMEINER CODE

Email ActiveWorkbook As Outlook Attachment

This VBA macro code with add the entire ActiveWorkbook as an attachment to a brand new Outlook message.

```
Sub EmailWorkbook()
'PURPOSE: Create email message with ActiveWorkbook attached
'SOURCE: www.TheSpreadsheetGuru.com

Dim SourceWB As Workbook
Dim DestinWB As Workbook
Dim OutlookApp As Object
Dim OutlookMessage As Object
Dim TempFileName As Variant
Dim ExternalLinks As Variant
Dim TempFilePath As String
Dim FileExtStr As String
Dim DefaultName As String
Dim UserAnswer As Long
Dim x As Long

Set SourceWB = ActiveWorkbook

'Check for macro code residing in
  If Val(Application.Version) >= 12 Then
    If SourceWB.FileFormat = 51 And SourceWB.HasVBProject = True Then
      UserAnswer = MsgBox("There was VBA code found in this xlsx file. " & _
        "If you proceed the VBA code will not be included in your email attachment. " & _
        "Do you wish to proceed?", vbYesNo, "VBA Code Found!")

    If UserAnswer = vbNo Then Exit Sub 'Handle if user cancels

    End If
  End If

'Determine Temporary File Path
  TempFilePath = Environ$("temp") & "\"

'Determine Default File Name for InputBox
  If SourceWB.Saved Then
    DefaultName = Left(SourceWB.Name, InStrRev(SourceWB.Name, ".") - 1)
  Else
    DefaultName = SourceWB.Name
  End If

'Ask user for a file name
  TempFileName = Application.InputBox("What would you like to name your attachment? (No Special Characters!)", _
    "File Name", Type:=2, Default:=DefaultName)
```

```
    If TempFileName = False Then Exit Sub 'Handle if user cancels

'Determine File Extension
  If SourceWB.Saved = True Then
    FileExtStr = "." & LCase(Right(SourceWB.Name, Len(SourceWB.Name) - InStrRev(SourceWB.Name, ".", , 1)))
  Else
    FileExtStr = ".xlsx"
  End If

'Optimize Code
  Application.ScreenUpdating = False
  Application.EnableEvents = False
  Application.DisplayAlerts = False

'Save Temporary Workbook
  SourceWB.SaveCopyAs TempFilePath & TempFileName & FileExtStr
  Set DestinWB = Workbooks.Open(TempFilePath & TempFileName & FileExtStr)

'Break External Links
  ExternalLinks = DestinWB.LinkSources(Type:=xlLinkTypeExcelLinks)

    'Loop Through each External Link in ActiveWorkbook and Break it
      On Error Resume Next
        For x = 1 To UBound(ExternalLinks)
          DestinWB.BreakLink Name:=ExternalLinks(x), Type:=xlLinkTypeExcelLinks
        Next x
      On Error GoTo 0

'Save Changes
  DestinWB.Save

'Create Instance of Outlook
  On Error Resume Next
    Set OutlookApp = GetObject(class:="Outlook.Application") 'Handles if Outlook is already open
  Err.Clear
    If OutlookApp Is Nothing Then Set OutlookApp = CreateObject(class:="Outlook.Application") 'If not, open Outlook

    If Err.Number = 429 Then
      MsgBox "Outlook could not be found, aborting.", 16, "Outlook Not Found"
      GoTo ExitSub
    End If
  On Error GoTo 0

'Create a new email message
```

```
  Set OutlookMessage = OutlookApp.CreateItem(0)

'Create Outlook email with attachment
  On Error Resume Next
    With OutlookMessage
     .To = ""
     .CC = ""
     .BCC = ""
     .Subject = TempFileName
     .Body = "Please see attached." & vbNewLine & vbNewLine & "Chris"
     .Attachments.Add DestinWB.FullName
     .Display
    End With
  On Error GoTo 0

'Close & Delete the temporary file
  DestinWB.Close SaveChanges:=False
  Kill TempFilePath & TempFileName & FileExtStr

'Clear Memory
  Set OutlookMessage = Nothing
  Set OutlookApp = Nothing

'Optimize Code
ExitSub:
  Application.ScreenUpdating = True
  Application.EnableEvents = True
  Application.DisplayAlerts = True

End Sub
```

Email Selected Worksheets As Outlook Attachment

This VBA macro will attach only the selected tabs within the ActiveWorkbook to a new Outlook email message.

```
Sub EmailSelectedSheets()
'PURPOSE: Create email message with only Selected Worksheets attached
'SOURCE: www.TheSpreadsheetGuru.com

Dim SourceWB As Workbook
Dim DestinWB As Workbook
Dim OutlookApp As Object
Dim OutlookMessage As Object
Dim TempFileName As Variant
Dim ExternalLinks As Variant
Dim TempFilePath As String
Dim FileExtStr As String
Dim DefaultName As String
Dim UserAnswer As Long
Dim x As Long

'Optimize Code
  Application.ScreenUpdating = False
  Application.EnableEvents = False
  Application.DisplayAlerts = False

'Copy only selected sheets into new workbook
  Set SourceWB = ActiveWorkbook
  SourceWB.Windows(1).SelectedSheets.Copy
  Set DestinWB = ActiveWorkbook

'Check for macro code residing in
  If Val(Application.Version) >= 12 Then
    If SourceWB.FileFormat = 51 And SourceWB.HasVBProject = True Then
      UserAnswer = MsgBox("There was VBA code found in this xlsx file. " & _
        "If you proceed the VBA code will not be included in your email attachment. " & _
        "Do you wish to proceed?", vbYesNo, "VBA Code Found!")

    'Handle if user cancels
      If UserAnswer = vbNo Then
        DestinWB.Close SaveChanges:=False
        GoTo ExitSub
      End If

    End If
  End If

'Determine Temporary File Path
  TempFilePath = Environ$("temp") & "\"
```

```
'Determine Default File Name for InputBox
  If SourceWB.Saved Then
    DefaultName = Left(SourceWB.Name, InStrRev(SourceWB.Name, ".") - 1)
  Else
    DefaultName = SourceWB.Name
  End If

'Ask user for a file name
  TempFileName = Application.InputBox("What would you like to name your attachment? (No Special Characters!)", _
    "File Name", Type:=2, Default:=DefaultName)

    If TempFileName = False Then GoTo ExitSub 'Handle if user cancels

'Determine File Extension
  If SourceWB.Saved = True Then
    FileExtStr = "." & LCase(Right(SourceWB.Name, Len(SourceWB.Name) - InStrRev(SourceWB.Name, ".", , 1)))
  Else
    FileExtStr = ".xlsx"
  End If

'Break External Links
  ExternalLinks = DestinWB.LinkSources(Type:=xlLinkTypeExcelLinks)

    'Loop Through each External Link in ActiveWorkbook and Break it
      On Error Resume Next
        For x = 1 To UBound(ExternalLinks)
          DestinWB.BreakLink Name:=ExternalLinks(x), Type:=xlLinkTypeExcelLinks
        Next x
      On Error GoTo 0

'Save Temporary Workbook
  DestinWB.SaveCopyAs TempFilePath & TempFileName & FileExtStr

'Create Instance of Outlook
  On Error Resume Next
    Set OutlookApp = GetObject(class:="Outlook.Application") 'Handles if Outlook is already open
  Err.Clear
    If OutlookApp Is Nothing Then Set OutlookApp = CreateObject(class:="Outlook.Application") 'If not, open Outlook

    If Err.Number = 429 Then
      MsgBox "Outlook could not be found, aborting.", 16, "Outlook Not Found"
      GoTo ExitSub
    End If
```

```
  On Error GoTo 0

'Create a new email message
  Set OutlookMessage = OutlookApp.CreateItem(0)

'Create Outlook email with attachment
  On Error Resume Next
    With OutlookMessage
     .To = ""
     .CC = ""
     .BCC = ""
     .Subject = TempFileName
     .Body = "Please see attached." & vbNewLine & vbNewLine & "Chris"
     .Attachments.Add TempFilePath & TempFileName & FileExtStr
     .Display
    End With
  On Error GoTo 0

'Close & Delete the temporary file
  DestinWB.Close SaveChanges:=False
  Kill TempFilePath & TempFileName & FileExtStr

'Clear Memory
  Set OutlookMessage = Nothing
  Set OutlookApp = Nothing

'Optimize Code
ExitSub:
  Application.ScreenUpdating = True
  Application.EnableEvents = True
  Application.DisplayAlerts = True

End Sub
```

Aktuelles Tabellenblatt als PDF-Datei speichern und vermailen

```
Sub FAKTURSPEICHERN()
' Der SPEICHER-Button in den 6 Fakturen + Angebot + Lieferschein

If Sheets("EINSTELLUNGEN").Range("H7") = 2 Then On Error Resume Next
```

```
    Dim PFAD ' Nur der Speicherpfad
    Dim DATEINAME ' nur der Dateiname
    Dim GESAMTPFAD ' Gesamter Speicherpfad inkl Dateiname
    Dim RECHNUNGSNUMMER
    Dim DATUM
    Dim BRUTTOBETRAG
    Dim KUNDE As String

    Dim App, Itm ' Variablen für Email
    Dim EMAIL ' Email-Adresse

    KUNDE = ActiveSheet.Range("B4")
    If InStr(KUNDE, " ") > 0 Then ' wenn es ein Leerzeichen gibt, nimm nur den Namen bis zum Leerzeichen
       KUNDE = Left(KUNDE, InStr(KUNDE, " "))
    End If

    PFAD = "D:\EIGENE DATEIEN\SONSTIGES\' MY KASSA BANK\AR " & Year(Now) & "\"
    RECHNUNGSNUMMER = ActiveSheet.Range("B17").Value
    BRUTTOBETRAG = ActiveSheet.Range("H41").Value
    DATUM = ActiveSheet.Range("D17").Value
    DATEINAME = "AR" & RECHNUNGSNUMMER & " " & KUNDE & " " & BRUTTOBETRAG & ".pdf"
    GESAMTPFAD = PFAD & DATEINAME

    With ActiveSheet
       .ExportAsFixedFormat Type:=xlTypePDF, Filename:=GESAMTPFAD, Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False,
OpenAfterPublish:=False
    End With

    MsgBox "Die Datei wurde gespeichert unter " & vbCrLf & vbCrLf & GESAMTPFAD

    EMAIL = Range("D11")

    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

       With Itm
          .Subject = "Aktuelle Honorarnote"
          .to = EMAIL
          '.Cc =
          .Attachments.Add (GESAMTPFAD)
          .Body = Range("L3") & vbCrLf & vbCrLf & "Ich erlaube mir im Anhang die aktuelle Honorarnote einzufügen." & vbCrLf & vbCrLf & "Für Fragen dazu
stehe ich Ihnen natürlich sehr gerne zur Verfügung."
```

```
        .display
      End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunden warten
    Itm.display

    Set App = Nothing
    Set Itm = Nothing

    Exit Sub


KEINOUTLOOK:
    MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
    Exit Sub


End Sub
```

DSGVO-Mailer

```
' Variablen für Datei
    Public KOPIE As Workbook
    Public DATEINAME As String ' nur der Dateiname - zB Test.xlsx
    Public DATEIORIG As String ' der Dateiname im Tempordner inkl. Pfad - zB C:\Users\Me\AppData\Local\Temp\Test.xlsx
    Public DATEIKOPIE As String ' gleich wie DATEIORIG aber mit dem Interims-Unterstrich - zB C:\Users\Me\AppData\Local\Temp\Test_.xlsx
    Public EMAIL As String ' Die Emailadresse des Empfängers
    Public PASSWORT As String ' Das Passwort für die Exceldatei
    Public PAUSENMODUS ' 1=Pausenmodus =Bearbeitenmodus ist an - 0 oder leer: Standardmodus für Emailversand


' Allgemeine Funktion zum Finden der letzten Zelle einer Tabelle
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)
```

```
   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function

Sub Mail_I()

' Dies ist der Code, wenn man auf den MAILEN-Button klickt, um die aktuelle Mappe per Mail zu versenden
' Der Code macht den ersten Teil des Emailversandes: er speichert die aktuelle Datei als temporäre Kopie in den TEMP-Ordner
' Dann übergibt er an das Fenster mit den Kunden und den Emailadressen und dem Passwort der Klienten

' Fehlerüberprüfung (wenn man zB auf das Mail-Icon klickt, aber gar keine Arbeitsmappe offen ist)
    On Error GoTo FEHLER

' Festlegen des reinen Dateinamens
    DATEINAME = ActiveWorkbook.Name

' Festlegen der originalen Datei im Temp-Ordner
```

```
    DATEIORIG = Environ("TEMP") & "\" & ActiveWorkbook.Name

' Prüfen, ob Datei schon gespeichert worden ist als Exceldatei
    If InStr(DATEIORIG, ".xl") = 0 Then DATEIORIG = DATEIORIG & ".xlsx" ' falls nicht, mach sie zu einer Standard-Exceldatei

' Festlegen der temporären Datei (sie muss einen Unterstrich haben, da man nicht zwei Dateien mit gleichem Namen gleichzeitig öffnen kann)
    DATEIKOPIE = Replace(DATEIORIG, ".xl", "_.xl")

' Löschen eventueller alten Dateien im TEMP-Ordner von früheren Emailversandvorgängen
    If Len(Dir(DATEIORIG)) > 0 Then Kill DATEIORIG
    If Len(Dir(DATEIKOPIE)) > 0 Then Kill DATEIKOPIE

' Speichern der Datei als temporäre Kopie
    ActiveWorkbook.SaveCopyAs Filename:=DATEIKOPIE

' DSGVO-Mailer-Fenster einblenden
    Windows("Excel-DSGVO-Mailer.xlsm").Visible = True
    Workbooks("Excel-DSGVO-Mailer.xlsm").Activate

    Exit Sub

FEHLER:
    MsgBox "Der Befehl kann nicht ausgeführt werden - vermutlich gibt es keine Arbeitsmappe"

End Sub

Sub MAIL_II()

' Dies ist der zweite Teil zum Vermailen
' er wird automatisch dann ausgeführt, wenn man in der Liste der Kunden eine Zeile mit einem Kunden angeklickt hat
' Achtung: Der Code muss unterscheiden, ob die Mandantenauswahl in Excel für eine Exceldatei erfolgt, oder ob der Aufruf auf Word erfolgte

' Variablen für WORD-Datei
    Dim WORDDATEIKOPIE As String ' Pfad und Name der Word-Dateikopie
    Dim WORDDATEIORIG As String 'Pfad und Name der originalen Word-Datei
    Dim WORDDATEINAME As String ' Der reine Dateiname für das Emailbetreff
    Dim MYWORD As Object ' Word-Application

' Variablen für Email
    Dim App, Itm

' Auslesen der Daten des gewählten Klienten

    EMAIL = Cells(ActiveCell.Row, 6)
```

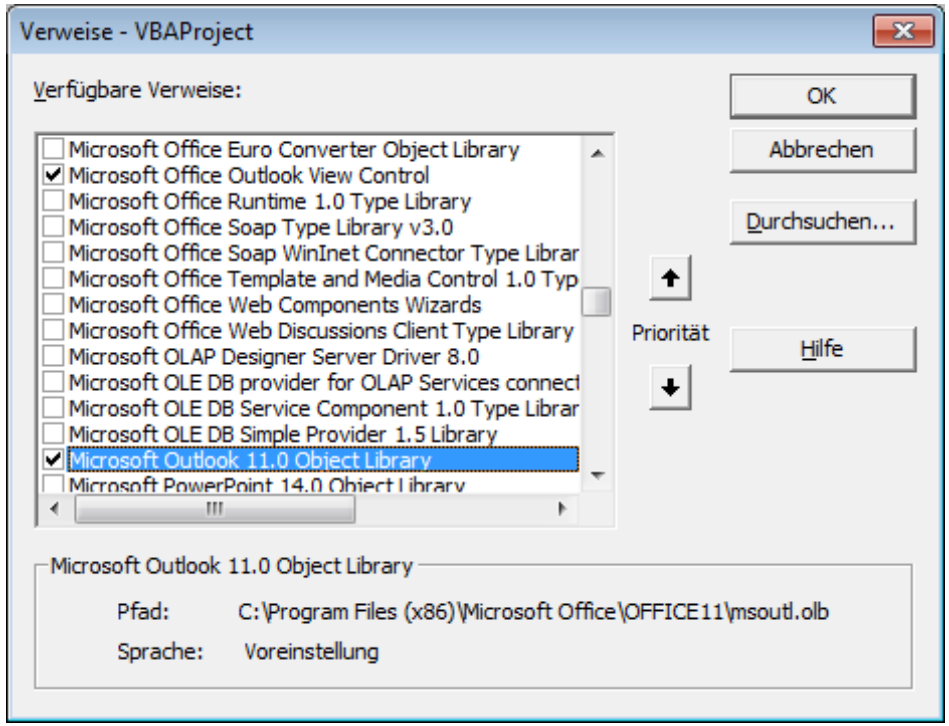
```
    PASSWORT = Cells(ActiveCell.Row, 7)

' Prüfen, ob der Aufruf von Word aus erfolgte => dann zum ganz unten befindlichen Code für Word verzweigen
    If GetSetting(appname:="DSGVO", section:="Excel", Key:="Word") = 1 Then GoTo WORDAUFRUF

' Bildschirmaktualisierung aufheben
    Application.ScreenUpdating = False

' Fehler abfangen
    On Error GoTo FEHLER

' Öffnen der temporären Kopie
    Set KOPIE = Workbooks.Open(DATEIKOPIE, False, False) ' Verknüpfungen nicht aktualisieren (false) und Datei nicht nur lesend öffnen (false)

' Speichern der Kopie mit Passwort
    Application.DisplayAlerts = False ' Nicht warnen beim Überschreiben
    KOPIE.SaveAs Filename:=DATEIKOPIE, Password:=PASSWORT, WriteResPassword:="", ReadOnlyRecommended:=False, CreateBackup:=False
    KOPIE.Close
    Application.DisplayAlerts = True ' Wieder normal warnen beim Überschreiben

' Umbenennen der Kopie auf Namen ohne Unterstrich
    Name DATEIKOPIE As DATEIORIG

' E-Mailprogramm ansteuern
    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        With Itm
            .Subject = DATEINAME
            .to = EMAIL
            '.Cc =
            .Attachments.Add (DATEIORIG)
            .display
        End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunden warten
    Itm.display

    Set App = Nothing
    Set Itm = Nothing

' Ausblenden des DSGVO-Mailers
```

```
    Windows("Excel-DSGVO-Mailer.xlsm").Visible = False
    Application.ScreenUpdating = True

    Exit Sub

FEHLER:
    Application.ScreenUpdating = True
    Exit Sub

KEINOUTLOOK:
    MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
    Application.ScreenUpdating = True
    Exit Sub


' ------------------
' --- WORD-AUFRUF ---
' ------------------

' Dies ist der Code, der abgearbeitet wird, wenn die Excelpasswortliste von Word aufgerufen wurde

WORDAUFRUF:

' Zurücksetzen des Aufrufparameters in der Registrierung
    SaveSetting "DSGVO", "Excel", "Word", "0"

' Auslesen der Dateipfade
    WORDDATEIKOPIE = GetSetting(appname:="DSGVO", section:="Excel", Key:="Worddatei")
    WORDDATEIORIG = Replace(WORDDATEIKOPIE, "_.docx", ".docx")
    WORDDATEINAME = Mid(WORDDATEIORIG, InStrRev(WORDDATEIORIG, "\") + 1, 99)

' Öffnen der Kopie der Worddatei
    Set MYWORD = CreateObject("Word.Application") ' starte MS Word
    With MYWORD
      .Visible = True
      .Documents.Open Filename:=WORDDATEIKOPIE
      .Activedocument.SaveAs2 Filename:=WORDDATEIORIG, Password:=PASSWORT
      .Activedocument.Close
      .Application.Quit
    End With

' Umbenennen der Worddateikopie in den originalen Dateinamen
'    Name WORDDATEIKOPIE As WORDDATEIORIG
```

```
' E-Mailprogramm ansteuern
    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        With Itm
            .Subject = WORDDATEINAME
            .to = EMAIL
            '.Cc =
            .Attachments.Add (WORDDATEIORIG)
            .display
        End With

    Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunde warten
    Itm.display

    Set App = Nothing
    Set Itm = Nothing


' Schließen des DSGVO-Mailers
    Application.WindowState = -4137
    Application.Quit

End Sub
```

Emailversand mit Auswahl Mailkonto

Für allen Code hier: die MS Outlook Object Library einblenden



Grundtheorie: es geht nur ab Outlook 2007 wirklich gut

- mit Exchange im Hintergrund kann man das Mailaccount auswählen mit SentOnBehalfOfName
- ansonsten mit Standalone Outlook (ab 2007) geht es mit " Set .SendUsingAccount = .Session.Accounts.Item("Kontoname") "

BSP-Code:

```
Sub test()
Dim olApp As Object
Dim olOldBody As String

Set olApp = CreateObject("Outlook.Application")

    With olApp.CreateItem(0)
          Set .SendUsingAccount = .Session.Accounts.Item("Kontoname") ' Sendekonto vorwählen. Ab Outlook 2007 möglich
              .GetInspector.Display
              olOldBody = .htmlBody
```

```
            .To = Tabelle1.Cells(6, 2).Value
            .Subject = "test "
            .htmlBody = "dies ist ein test<br><br>" & olOldBody
    End With

End Sub
```

Hier mal ein umfangreicheres Beispiel:

```
ub Mailtest()
Dim oApp As Object
Dim olOutAddressIn As String
Dim olOutAddressOut As String
Dim olOutAddressName As String
Dim vntTempArray As Variant
Dim olOldBody As String


Rem Emailadresse und Kontoname (angezeigter Name) auslesen

Set oApp = CreateObject("OUTLOOK.Application")

With oApp.Session.Accounts

    For i = 1 To .Count

        With .Item(i)
            olOutAddressIn = .SmtpAddress
            olOutAddressName = .DisplayName
        End With

            vntTempArray = Split(olOutAddressIn, "@")
            olOutAddressOut = vntTempArray(1)
               If olOutAddressOut = "firmaY.de" Then Exit For

    Next i

End With
```

```
Set oApp = Nothing

Rem Email erstellen

Set oApp = CreateObject("OUTLOOK.Application")
    With oApp.CreateItem(0)
              .GetInspector.Display
              olOldBody = .HTMLBody
              .To = "Test@server.de"
              .Subject = "Test"
              .HTMLBody = "Liebe Kollegin,<br><br>Dies ist nur ein Test.<br><br>" & _
                          "Gruß, Max.<br><br>" & olOldBody
         Set .SendUsingAccount = .Session.Accounts.Item(olOutAddressName)

    End With

End Sub
```

Noch ein weiteres Beispiel:

```
Sub SendMailWithOutlook()

'Call ToggleSheetProtection

    Dim OlApp As Object
    Dim OlMail As Object 'or as MailItem?
    Dim OlAccounts As Outlook.Accounts
    Dim OlAccount As Outlook.Account
    Dim OlAccountTemp As Outlook.Account
    Dim FoundAccount As Boolean

    Dim strMailBody As String
    Dim strToMail As String, strCcMail As String, strSubjMail As String, strGreetingMail As String, strMailBodyAddendum As String, strAttachPath As String

    Dim wbXl As Workbook
    Dim wsXl As Worksheet

    Set wbXl = ActiveWorkbook
    Set wsXl = wbXl.Sheets("MailCtrlCntr")
```

```
    Application.ScreenUpdating = False
    Application.DisplayAlerts = False
    Application.EnableEvents = False

    Dim shpMailBodyTextBox As Variant
    Set shpMailBodyTextBox = ActiveWorkbook.Sheets("MailCtrlCntr").Shapes("MailBodyTextBox")
    strTextBox = shpMailBodyTextBox.TextFrame2.TextRange.Text

    With wsXl
        strToMail = "sw8@gmx.at" ' .Range("MailTo").Value
        strCcMail = "sw8@gmx.at" '.Range("MailCc").Value
        strSubjMail = "test" ' .Range("MailSubj").Value
        strGreetingMail = "gruß" ' .Range("MailGreeting").Value
        ' strMailBodyAddendum = .Range("MailBodyAddendum").Value
        ' strAttachPath = .Range("AttachPath").Value
        ' strMailBodySalutation = .Range("MailBodySalutation").Value
        ' strMailBodySalutationNames = .Range("MailBodySalutationNames").Value
    End With

    If strMailBodyAddendum = "" Then
       strMailBody = strGreetingMail & vbNewLine & vbNewLine & _
                strTextBox & vbNewLine & vbNewLine & _
                strMailBodySalutation & vbNewLine & vbNewLine & _
                strMailBodySalutationNames
       Else

       strMailBody = strGreetingMail & vbNewLine & vbNewLine & _
                strTextBox & vbNewLine & vbNewLine & _
                strMailBodyAddendum & vbNewLine & vbNewLine & _
                strMailBodySalutation & vbNewLine & vbNewLine & _
                strMailBodySalutationNames

     End If

'loop through and find Outlook account based on from email address
Set OlApp = CreateObject("Outlook.Application") 'New Outlook.Application
Set OlMail = OlApp.CreateItem(0)
FoundAccount = False

'set email account to search for:
' strFrom = "stefan.wenninger@medplan.at"  ' wsXl.Range("EmailFrom").Value
strFrom = "info@golfversicherung.at"  ' wsXl.Range("EmailFrom").Value
```

```
Set OlAccounts = OlApp.Application.Session.Accounts
For Each OlAccountTemp In OlAccounts
If (OlAccountTemp.SmtpAddress = strFrom) Then
Set OlAccount = OlAccountTemp
FoundAccount = True
Exit For
End If
Next

If (FoundAccount) Then
Set OlMail = OlApp.CreateItem(olMailItem)
    With OlMail
        'On Error Resume Next
        Set .SendUsingAccount = OlAccount 'not a direct assignment, needs the Set verb
        .To = strToMail
        .CC = strCcMail
        .BCC = ""
        .Subject = strSubjMail
        .Body = strMailBody
        '.Attachments.Add (strAttachPath)
        .Display  'you could also use display and send manually
    End With
'On Error GoTo 0
Else
    MsgBox ("Could not find the selected Outlook email account: " & strFrom & ".")
End If


    Set OlMail = Nothing
    Set OlApp = Nothing
    Set OlAccount = Nothing

    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Application.EnableEvents = True

'Call ToggleSheetProtection

End Sub


VERSION 1
```

Important read this :

The code on this page is only working when you use Outlook as your mail program.
Copy the code in a Standard module of your workbook, if you just started with VBA see this page.
Where do I paste the code that I find on the internet

Change Sender name and Reply address

The easiest way is to add this code line that change the sender name and reply address in the Outlook macro examples on my site.

'The receiver can see the original mail address in the properties if he want
.SentOnBehalfOfName = """SenderName"" <Reply@Address.com>"

Use SendUsingAccount in Excel/Outlook 2007 or higher

If you want to mail from another account then your default mail account in Outlook 2007-2013 then you can use SendUsingAccount, this is added to the object model in Outlook 2007.

First add a reference to the Microsoft Outlook Library in your Excel workbook

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook **?** Object Library
? is the Outlook version number

First we must know the account number that we want to use.
Run the macro below so you know the account number that you must use in the mail macro.

```
Sub Which_Account_Number()
'Don't forget to set a reference to Outlook in the VBA editor
    Dim OutApp As Outlook.Application
    Dim I As Long

    Set OutApp = CreateObject("Outlook.Application")

    For I = 1 To OutApp.Session.Accounts.Count
        MsgBox OutApp.Session.Accounts.Item(I) & " : This is account number " & I
    Next I
End Sub
```

Now you know the number of the account that you want to use in your mail macro you can use the following test subroutine that send a small text in an e-mail message. Change the mail address and the Item number "**Item(1)**" in the macro before you run it.

```
Sub Mail_small_Text_Change_Account()
'Only working in Office 2007-2013
'Don't forget to set a reference to Outlook in the VBA editor
    Dim OutApp As Outlook.Application
    Dim OutMail As Outlook.MailItem
    Dim strbody As String

    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(olMailItem)

    strbody = "Hi there" & vbNewLine & vbNewLine & _
              "This is line 1" & vbNewLine & _
              "This is line 2" & vbNewLine & _
              "This is line 3" & vbNewLine & _
              "This is line 4"

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = strbody

        'SendUsingAccount is new in Office 2007
        'Change Item(1)to the account number that you want to use
        .SendUsingAccount = OutApp.Session.Accounts.Item(1)

        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Note : Got mails from 2013 users that say they need to remove .Item

VERSION 2

```
Public Sub test()
   Dim OutApp As Object, oAccount As Object, Nachricht As Object
   Set OutApp = CreateObject("Outlook.Application")
   Dim Anlage As String
```

```
    'Aktive Arbeitsmappe wird als Mail gesendet
    Anlage = Pfad & Jahr & "\" & Monat & "\" & Range("E2") & " - " & Range("E1") & " in " & Range("H2") & ".pdf"
    For Each oAccount In OutApp.Session.Accounts
       If oAccount.AccountType = 2 Then
          If oAccount.DisplayName = "kam" Then Exit For
       End If
    Next
    Set Nachricht = OutApp.CreateItem(0)
    Nachricht.Display
    With Nachricht
          .To = emailFirma
          .Subject = ArtderMeldung & " " & Date & Time
          .Attachments.Add Anlage
          .Body = "Das ist ein Test." & vbCrLf & "Bitte ignorieren."
          'Hier wird die Mail nochmals angezeigt
          .Display
          ' Alternativ Hier wird die Mail gleich in den Postausgang gelegt
          ' Set .SendUsingAccount = oAccount
          '.Send
          ' SendKeys "%S"
    End With
    'Outlook schliessen
    Set OutApp = Nothing
    Set Nachricht = Nothing
End Sub
```

VERSION 3

Code:

```
Sub Serienmail()
   Dim olApp As Object
   Dim i As Integer
   Dim lz As Integer
   Dim Pfad As String
   Dim oldBody As String

   Rem Hier Pfad anpassen
   Pfad = "W:\Users\Dave\Desktop\"

   Rem Zählen wie viel Zellen belegt sind
   lz = Tabelle1.Cells(Rows.Count, 1).End(xlUp).Row

   Rem Start der Sendeschleife an Empfänger
   For i = 2 To lz

      Set olApp = CreateObject("Outlook.Application")

     With olApp.CreateItem(0)
       Rem Insdpector-Objekt erstellen
            .GetInspector.Display
       Rem Sendekonto wählen. Sendekonto in Anführungszeichen
          Set .SendUsingAccount = .Session.Accounts.Item("Info - Drinkinggames")

       Rem Originalbody zwischenspeichern
      oldBody = .htmlBody

       Rem E-Mail Adresse. Die Empfänger stehen in Spalte A ab Zeile 1
            .To = Tabelle1.Cells(i, 4) '

       Rem Der Betreff in Spalte B
            .Subject = Tabelle1.Cells(i, 5)
```

```
        Rem _
           Der zu sendende Text in Spalte C _
           Maximal 1024 Zeichen _
           Der Text wird ohne Formatierung übernommen

           .htmlBody = "Hallo " & Tabelle1.Cells(i, 2).Value & ",<br><br>" & _
                       "Hier kommt der Text " & Tabelle1.Cells(i, 5).Value & "<br><br>" & _
                       "Hier kommt die Info " & Tabelle1.Cells(i, 6).Value & "<br><br><br>" & _
                       "Mit freundlichen Grüßen<br><br><br>" & oldBody

        Rem Lesebetsätigung anfordern
           .ReadReceiptRequested = True

        Rem Abfrage ob Anhang vorhanden
        If Pfad & Tabelle1.Cells(i, 6).Value <> "" Then
           .Attachments.Add Pfad & Tabelle1.Cells(i, 6).Value
           Else
        End If

        Rem Hier wird die Mail gleich in den Postausgang gelegt
        '.Send
      End With


      Rem _
        Sendepause einschalten. Outlook kann die Aufträge _
        nicht schnell genug verarbeiten
      Application.Wait (Now + TimeValue("0:00:05"))
    Next i

End Sub
```

```
Sub Serienmail()
```

```
Dim olApp As Object
Dim i As Integer
Dim lz As Integer
Dim Pfad As String
Dim oldBody As String
em Hier Pfad anpassen
Pfad = "W:\Users\Dave\Desktop\"
Rem Zählen wie viel Zellen belegt sind
lz = Tabelle1.Cells(Rows.Count, 1).End(xlUp).Row
Rem Start der Sendeschleife an Empfänger
For i = 2 To lz
Set olApp = CreateObject("Outlook.Application")
With olApp.CreateItem(0)
Rem Insdpector-Objekt erstellen
.GetInspector.Display
Rem Sendekonto wählen. Sendekonto in Anführungszeichen
Set .SendUsingAccount = .Session.Accounts.Item("Info - Drinkinggames")
Rem Originalbody zwischenspeichern
oldBody = .htmlBody
Rem E-Mail Adresse. Die Empfänger stehen in Spalte A ab Zeile 1
.To = Tabelle1.Cells(i, 4) '
Rem Der Betreff in Spalte B
.Subject = Tabelle1.Cells(i, 5)
Rem _                  Der zu sendende Text in Spalte C _                Maximal 1024 Zeichen _                  Der Text
wird ohne Formatierung übernommen
.htmlBody = "Hallo " & Tabelle1.Cells(i, 2).Value & ",<br><br>" & _
                        "Hier kommt der Text " & Tabelle1.Cells(i, 5).Value & "<br><br>" & _
                      "Hier kommt die Info " & Tabelle1.Cells(i, 6).Value & "<br><br><br>" & _
"Mit freundlichen Grüßen<br><br><br>" & oldBody
Rem Lesebetsätigung anfordern
.ReadReceiptRequested = True
Rem Abfrage ob Anhang vorhanden
If Pfad & Tabelle1.Cells(i, 6).Value <> "" Then
.Attachments.Add Pfad & Tabelle1.Cells(i, 6).Value
End If
Rem Hier wird die Mail gleich in den Postausgang gelegt
'.Send
End With
Rem _            Sendepause einschalten. Outlook kann die Aufträge _           nicht schnell genug verarbeiten
Application.Wait (Now + TimeValue("0:00:05"))
Next i
```

```
End Sub
```

E-Mailadresse in einer Zelle ändern

```
Sub MAILADRESSE()

    ' Um in der Tabelle Anleitung die E-Mail-Adresse in Zelle K26 ändern / einstellen zu können

    Dim BUTTONTEXT As String
    Dim Wahl As Integer
    Dim MAILADDY As String

    MAILADDY = Sheets("Anleitung").Range("K26").Value ' Auslesen vorige Adresse

    BUTTONTEXT = "NEUE E-MAIL-ADRESSE EINGEBEN" & vbCrLf
    BUTTONTEXT = BUTTONTEXT & vbCrLf & "(ABBRUCH= BISHERIGE ADRESSE BEIBEHALTEN)"

    MAILADDY = InputBox(BUTTONTEXT, "ÄNDERUNG E-MAIL-ADRESSE", MAILADDY)

    If MAILADDY = "" Then Exit Sub

    Sheets("Anleitung").Range("K26").Value = MAILADDY
    Sheets("Anleitung").Range("K26").Select
    ActiveSheet.Hyperlinks.Add Anchor:=Selection, Address:="mailto:" & MAILADDY

End Sub
```

Email-Adresse prüfen ob gültig

```
' ---------------------------------------------------------------------------
'  Title: Comprehensive e-mail address syntax validation check
'
' ---------------------------------------------------------------------------

' ----------------------------------------------------- '
' Purpose: Validate a given e-mail address.
' In:          strAddress - string to be validated
```

```
' Out:
' Returns: True if strAddress is a valid e-mail address; False otherwise.
' Date: 8/23/1999
' Programmer: Ian Lent and Melvin Tucker
' ------------------------------------------------- '

Public Function ValidateEmail(ByVal strAddress As String) As Boolean
    Dim lngIndex As Long            ' Position in strAddress
    Dim lngCountAt As Long          ' Number of "@"
    Dim lngLastDotPos As Long       ' Position of the previous dot in the string
    Dim strCurrentChar As String    ' Buffer that holds the contents of the string one char at a time.

    On Error GoTo Fail_Validation

    ValidateEMail = True            ' Prove me wrong!
    strAddress = Trim(strAddress)
    lngLastDotPos = 0
    lngCountAt = 0

    ' If the address isn't at least this (a@b.com) long,
    ' it's not a valid address.
    If Len(strAddress) < 7 Then GoTo Fail_Validation

    ' Check for certain generably allowable characters in the leading position.
    ' If found, it's not a valid address.
    strCurrentChar = Left$(strAddress, 1)
    If strCurrentChar = "." Or strCurrentChar = "@" Or strCurrentChar = "_" Or _
       strCurrentChar = "-" Then GoTo Fail_Validation

    ' Check the string for non-allowable characters.
    For lngIndex = 1 To Len(strAddress)
       strCurrentChar = Mid$(strAddress, lngIndex, 1)

       ' Count the number of "@".
       If strCurrentChar = "@" Then lngCountAt = lngCountAt + 1

       ' If there are two consecutive dots, it's not a valid address.
       If strCurrentChar = "." Then
          If lngIndex = lngLastDotPos + 1 Then
             GoTo Fail_Validation
          Else
             lngLastDotPos = lngIndex
          End If
       End If
```

```
        Select Case Asc(strCurrentChar)
            ' These characters are not allowable in e-mail addresses.
            Case 1 To 44, 47, 58 To 63, 91 To 94, 96, 123 To 127, 128 To 255
                GoTo Fail_Validation
        End Select
    Next lngIndex

    ' If there isn't one, and only one "@", then it's not a valid address.
    If lngCountAt <> 1 Then GoTo Fail_Validation

    ' If the extension isn't a known one, it's not a valid address.
    Select Case Right$(strAddress, 4)
        Case ".com", ".org", ".net", ".edu", ".mil", ".gov"
            ' Yes, it's valid.
        Case Else
            GoTo Fail_Validation
    End Select

ValidateEMail_Exit:
    Exit Function

Fail_Validation:
    ValidateEMail = False
    GoTo ValidateEMail_Exit
End Function
```

Alle Dateien in Ordner/Unterordner vermailen

```
Sub EMailVerschicken_MA()

' Variablen für Outlook und Mail
Dim outObj As Object
Dim Mail As Object
Dim i As Integer

' Variablen für Ordner und Dateien
Dim Verzeichnis As String ' das Hauptverzeichnis, das durchsucht wird
Dim objFSO As Object ' das Filesystemobjekt
Dim objDir As Object ' das Verzeichnisobjekt

Dim Unterverzeichnis As Variant ' die Unterverzeichnisse im Hauptverzeichnis
Dim Datei As Variant ' die einzelnen Dateien
```

```
Dim Datei_Gesamtpfad As String ' der gesamte Pfad zur Datei und der Dateiname : zB C:\TEST\Mappe1.xls

' Erzeugen des Ordner-Objektes

Set objFSO = CreateObject("scripting.filesystemobject")
Verzeichnis = "C:\Test\Mitarbeiter"
Set objDir = objFSO.GetFolder(Verzeichnis)

' Erzeugen Email
Set outObj = CreateObject("Outlook.Application")
Set Mail = outObj.CreateItem(0)
With Mail
    .Subject = "Stundenlisten MA " & Range("C3") & "-" & Range("B3") & " Mitarbeiter"
    .Body = "Sehr geehrte Frau Nasr" & Chr(13) & Chr(13) & _
    "Im Anhang dieser Email finden Sie die aktuellen Stundenlisten unserer Mitarbeiter." & Chr(13) & Chr(13) & _
    "Liebe Grüße " & Chr(13) & Chr(13) & _
    Application.UserName
    .To = Range("K3")
End With

' Einfügen der Dateien

For Each Datei In objDir.Files
    Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
    If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) führen zu Fehlermeldung und werden nicht
versandt
        Mail.Attachments.Add Datei_Gesamtpfad
    End If
Next Datei

Mail.Display


' Speicher sparen

Set Mail = Nothing
Set outObj = Nothing

End Sub
```

Für Unterordner

Obiger Code geht ja nur für die Dateien, die direkt im Hauptordner sind. Nachfolgend mein Code um Dateien aus Unterordnern anzuzeigen. Mit

Hilfe dieses Codes sollte der obige Code einfach erweiterbar sein um auch Dateien aus Unterordnern einzufügen.

```
Sub Dateisuche()

' Variablen für Ordner und Dateien
Dim Verzeichnis As String ' das Hauptverzeichnis, das durchsucht wird
Dim objFSO As Object ' das Filesystemobjekt
Dim objDir As Object ' das Verzeichnisobjekt

Dim Unterverzeichnis As Variant ' die Unterverzeichnisse im Hauptverzeichnis
Dim Datei As Variant ' die einzelnen Dateien
Dim Datei_Gesamtpfad As String ' der gesamte Pfad zur Datei und der Dateiname : zB C:\TEST\Mappe1.xls

' Erzeugen des Ordner-Objektes

Set objFSO = CreateObject("scripting.filesystemobject")
Verzeichnis = "C:\Test"
Set objDir = objFSO.GetFolder(Verzeichnis)

' Auflisten der Dateien direkt im Hauptordner

For Each Datei In objDir.Files
   Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
   If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) wollen wir nicht anzeigen
      MsgBox Datei_Gesamtpfad & " aus dem Ordner " & objDir.Name
   End If
Next Datei

MsgBox "Das war die letzte Datei direkt im Hauptordner - nun kommen die Dateien in den Unterordnern"

' Auflisten der Dateien in den Unterordnern

For Each Unterverzeichnis In objDir.subfolders
   For Each Datei In Unterverzeichnis.Files
      Datei_Gesamtpfad = Datei ' Umwandeln Dateiobjekt in String
      If InStr(Datei_Gesamtpfad, "~") < 1 Then ' Tempdateien von aktuell geöffneten Dateien (erkennbar an ~) wollen wir nicht anzeigen
         MsgBox Datei_Gesamtpfad & " aus dem Unterordner " & Unterverzeichnis.Name
      End If
   Next Datei
Next Unterverzeichnis

' Speicher sparen
```

```
Set Mail = Nothing
Set outObj = Nothing

End Sub
```

Gesamte Arbeitsmappe vermailen

```
Sub MAILEN()


' UM MAIL ZU ERZEUGEN
    Dim App, Itm
    Dim DATEI As String

' Speichern damit aktuelle Version versandt wird
    Application.DisplayAlerts = False
    ActiveWorkbook.Save
    Application.DisplayAlerts = True

' E-Mailprogramm ansteuern
    On Error GoTo KEINOUTLOOK
    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
    On Error GoTo 0

        DATEI = ActiveWorkbook.FullName
        With Itm
            .Subject = "Stundenliste " & Range("O6").Value & " - " & Range("F6").Value
            .To = Sheets("Parameter").Range("C16").Value
            '.Cc =
            .Attachments.Add (DATEI)
            .display
        End With


    Set App = Nothing
    Set Itm = Nothing

    Exit Sub

KEINOUTLOOK:
        MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
End Sub
```

2.Lösung

Mit ActiveWorkbook.SendMail wird die gesamte Excel-Datei verschickt,

Outlook-Mail öffnen

Meine 2 BSP:

1.) Mit Übernahme von Text aus einer Exeltabelle (Monatsabrechnung Connect)

' Folgende Funktion wird von SUB MAILEN gebraucht
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Sub MAILEN()

' UM MAIL ZU ERZEUGEN
Dim App, Itm
Dim DATEI As String
Dim TN As String

' Hinweisfenster

If Sheets("Einstellungen").Range("A1") = "" Then
Sheets("Einstellungen").Range("A1") = 1
MsgBox "EINMALIGER HINWEIS:" & vbCrLf & vbCrLf & "DER TEXT FÜR DIE E-MAIL IST BEREITS IN DIE ZWISCHENABLAGE KOPIERT UND SIE KÖNNEN" & vbCrLf & "IN DER SICH NUN ÖFFNENDEN E-MAIL DEN TEXT DIREKT EINFÜGEN" & vbCrLf & "(IM MENÜ 'BEARBEITEN' DEN PUNKT 'EINFÜGEN' WÄHLEN)"
End If

' Kopieren des Mailtextes

Rows("47:102").Select
Range("C47").Activate
Selection.Copy
ActiveWindow.ScrollRow = 1

' Öffnen der Mail

```
    On Error GoTo KEINOUTLOOK

    Set App = CreateObject("Outlook.Application")
    Set Itm = App.CreateItem(0)
        DATEI = ActiveWorkbook.FullName
        With Itm
            .Subject = "Monatsabrechnung " & Range("J5") & "/" & Sheets("Einstellungen").Range("B5").Value & " - " & Sheets("Einstellungen").Range("C5").Value
& " " & Sheets("Einstellungen").Range("D5").Value
            .To = Sheets("Einstellungen").Range("B32").Value
            '.Attachments.Add (DATEI)
            .display
        End With

    Set App = Nothing
    Set Itm = Nothing

    Exit Sub

KEINOUTLOOK:
    Dim Ret As Long, URL As String
    URL = "mailto:" & Sheets("Einstellungen").Range("B32").Value & "?subject=" & "Monatsabrechnung " & ActiveSheet.Name & " " &
Sheets("Einstellungen").Range("B5").Value
    ShellExecute 0&, vbNullString, URL, vbNullString, vbNullString, vbNormalFocus
End Sub
```

2. Version)

```
' Folgende Funktion wird von SUB MAILEN gebraucht
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal
lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Sub MAILEN()

Dim App, Itm
Dim DATEI As String

On Error GoTo KEINOUTLOOK
Set App = CreateObject("Outlooks.Application") ' Wenn Outlook nicht existiert=>Fehler wird ausgelöst und verzweigt

Set Itm = App.CreateItem(0)
On Error GoTo 0
```

```
' ---------------------------------------------------------------
' WENN OUTLOOK EXISTIERT WIRD DAS HIER AUSGEFÜHRT
   If Sheets("SK").Range("C3").Value = False Then
      DATEI = ActiveWorkbook.FullName
      With Itm
         .Subject = "KASSABUCH " & ActiveSheet.Name
         .To = Sheets("SK").Range("G6").Value
         .Attachments.Add (DATEI)
         .display
      End With

     Else

      DATEI = Left(ActiveWorkbook.FullName, Len(ActiveWorkbook.FullName) - 3) & "zip"
      On Error GoTo KEINEZIPDATEI
      With Itm
         .Subject = "KASSABUCH " & ActiveSheet.Name
         .To = Sheets("SK").Range("G6").Value
         .Attachments.Add (DATEI)
         .display
      End With
      On Error GoTo 0
   End If

   Set App = Nothing
   Set Itm = Nothing

On Error GoTo 0
Exit Sub

KEINEZIPDATEI:
MsgBox "DIE  DATEI  " & vbCrLf & vbCrLf & DATEI & vbCrLf & vbCrLf & "EXISTIERT  LEIDER  NOCH  NICHT ." & vbCrLf & vbCrLf _
& "SIE  MÜSSEN  ERST  EINE  ZIP-DATEI ERSTELLEN" & vbCrLf _
& "ODER  DIE  ZIP-FUNKTION  IM  TABELLENBLATT  'SK'  DEAKTIVIEREN."
Exit Sub
' ---------------------------------------------------------------
' WENN KEIN OUTLOOK EXISTIERT, SONDER NUR OUTLOOK EXPRESS etc. :

KEINOUTLOOK:

Dim Ret As Long, URL As String

   URL = "mailto:" & Sheets("SK").Range("G6").Value & "?subject=" & "KASSABUCH " & ActiveSheet.Name
   ShellExecute 0&, vbNullString, URL, vbNullString, vbNullString, vbNormalFocus
```

```
    MsgBox "FÜGEN SIE IHRE DATEIEN BITTE NOCH ALS ANHANG EIN."

End Sub
```

FREMDE BEISPIELE

BSP 1:

```
Sub NewMail()
Dim App, Itm
Set App = CreateObject("Outlook.Application")
Set Itm = App.CreateItem(0)

With Itm
'.Subject = "Link senden"
'.To = "Jedermann@nirgendwo.de"
'.Body = "HuHu"
'.Attachments.Add (NewFileName) ' Must be complete path
.display ' This property is used when you want the user to see email
' and manually send. Then comment out rest of code except "End With" statement
' .send
End With
Set App = Nothing
Set Itm = Nothing
End Sub
```

BSP 2:

Beschreibung
Der nachfolgende Beispielcode zeigt,wie sich EMails mit Anlagen unter VBA (Word/Excel/Access) per Knopfdruck versenden lassen. Benötigt wird in jedem Fall Microsoft Outlook.

Einen Button in Excel erstellen

In VBA wechseln (ALT+F11) und nachfolgenden Code ins Modul kopieren

Unter **Extras/Verweise... / Verfügbare Verweise** folgende Eintrag aktivieren:
"Mircosoft Outlook 9.0 Object Library" (MSOUTL9.OLB)

Speichern
Ausprobieren

Der Code ist entsprechend auskommentiert und erklärt sich somit von selbst.

```
Sub Schaltfläche1_BeiKlick()
  ' Aufruf Unterprogramm "senden"
  Call senden
End Sub

'-------------------------------------------------------------

Private Sub senden()
  ' Outlook Applikation
  Dim ool As Outlook.Application
  Dim oInspector As Outlook.Inspector
  Dim oMail As Outlook.mailitem
  Dim myattachments As Variant

  ' Für Inputbox "EMailadresse-Änderung"
  Dim Mldg, Titel, Voreinstellung, MailAdress


  ' Adresse anzeigen und Änderung ermöglichen
  Mldg = "Ist die angegebene Emailadresse richtig?"
  Titel = "Mailadresse"
  Voreinstellung = "liste@chello.at"
  MailAdress = InputBox(Mldg, Titel, Voreinstellung)

  ' Wurde Abbrechen gedrückt, dann alles beenden
  If MailAdress = "" Then Exit Sub

  ' Verweis zu Outlook + neue Nachricht
  Set ool = CreateObject("Outlook.Application")
  Set oMail = ool.CreateItem(olMailItem)
  Set myattachments = oMail.Attachments

  ' Befreff-Zeile
  oMail.Subject = "Stand dieser Liste: " & _
    Format(Date, "Long Date") & " !"

  ' An-Zeile (Empfänger)
  oMail.To = MailAdress   ' "g.huber@unilog.at"
  oMail.Recipients.ResolveAll
  oMail.Display

  ' Texteingabe (Nachricht selbst)
  oMail.Body = "Hier die Exceldatei, bitteschön..."
```

```
    ' Anhang
    ' Nachfolgend ein Beispiel. Suchen Sie sich eine Datei auf
    ' Ihrem Rechner aus - vollständiger Pfad muß mitangegeben
    ' sein.
    ' Es können auch weitere Dateien angegeben werden.
    ' Hierzu einfach mit myattachments.Add "???" fortsetzen.
    myattachments.Add "D:\Eigene Dateien\Beispiel.xls"

    ' Speicher freigeben
    Set ool = Nothing
    Set oInspector = Nothing
    Set oMail = Nothing
  End Sub
```

Einzelnes Tabellenblatt mailen

eingesetzt bei Independence-Tool

```
Sub MAIL_SENDEN()

' UM MAIL ZU ERZEUGEN:
' das Tabellenblatt "MAIL" wird dann vermailt


   Dim strTabelle As String              ' Variable für den Tabellennamen
   Dim wsTabelle As Worksheet              ' Variable für die Tabelle als Objekt



   strTabelle = "MAIL"

            Application.ScreenUpdating = False

' Tabelle kurz einblenden
            Sheets(strTabelle).Visible = True

' neue Arbeitsmappe erstellen als Kopie
            Sheets(strTabelle).Copy

' Email-Addy ist auszulesen inTabelle Einstellungen
```

```
ActiveWorkbook.SendMail ThisWorkbook.Sheets("Einstellungen").Range("F17").Value, "Bitte neuen  Klient überprüfen"

'               aktive neue Arbeitsmappe schließen ohne Speichern
                ActiveWorkbook.Close False

' Tabellenblatt, das gemailt wurde, wieder ausblenden
                Sheets(strTabelle).Visible = False

'               Bildschirmaktualisierung ein
                Application.ScreenUpdating = True
End Sub
```

2.Lösung

```
Sub einzelnes_blatt_senden()
'************************************************
'* 24.08.06, 22.04.07                           *
'* erstellt von Karin, http://beverly.excelhost.de*
'* beverly@excelhost.de                         *
'************************************************
    Dim strTabelle As String              ' Variable für den Tabellennamen
    Dim wsTabelle As Worksheet            ' Variable für die Tabelle als Objekt
'   Tabelle2 als Standard festlegen
    strTabelle = "Tabelle versenden"
'   Name der zu versendenen Tabelle abfragen
    strTabelle = InputBox("Welches Blatt möchten Sie senden?" & vbCrLf & _
        vbCrLf & "Bitte den Tabellennamen eingeben", , strTabelle)
'   kein Abbruch der Eingabe
    If strTabelle <> "" Then
'       Schleife über alle Arbeitsblätter
        For Each wsTabelle In ThisWorkbook.Sheets
'           Name der Tabelle entspricht dem der zu versendenen Tabelle
            If wsTabelle.Name = strTabelle Then
'               Bildschirmaktualisierung aus
                Application.ScreenUpdating = False
'               Tabelle komplett kopieren
                Sheets(strTabelle).Copy
'               aktive Arbeitsmappe mit Mailbenachrichtigung "Diese Tabelle wurde als Mail versandt" versenden
                ActiveWorkbook.SendMail ThisWorkbook.Worksheets("Tabelle1").Cells(5, 1), "Diese Tabelle wurde als Mail versandt"
'               aktive Arbeitsmappe schließen ohne Speichern
                ActiveWorkbook.Close False
'               Bildschirmaktualisierung ein
                Application.ScreenUpdating = True
```

```
'             Schleife verlassen
              Exit For
           Else
'             Tabelle mit dem eingegebenen Namen ist nicht vorhanden
              If wsTabelle.Name = ThisWorkbook.Sheets(ThisWorkbook.Sheets.Count).Name Then MsgBox "Diese Tabelle gibt es nicht"
           End If
        Next wsTabelle
     End If
End Sub
```

3.Lösung

```
Sub BlattMailen()
Dim AktBlatt As String
Dim Pfad As String
Dim AktDatei As String
Dim Dateiname As String

AktBlatt = ActiveSheet.Name
AktDatei = ActiveWorkbook.Name
Pfad = ActiveWorkbook.Path

' TEMP-Dateiname festlegen
Dateiname = Pfad & Application.PathSeparator & AktBlatt & ".xls"

Sheets(AktBlatt).Cells.Copy
Workbooks.Add
Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, SkipBlanks:= _
False, Transpose:=False
Selection.PasteSpecial Paste:=xlFormats, Operation:=xlNone, SkipBlanks:= _
False, Transpose:=False
Range("A6").Select

ActiveSheet.Name = AktBlatt
Application.CutCopyMode = False

ActiveWindow.SelectedSheets.PrintOut From:=1, To:=1, Copies:=1, Collate _
:=True

' Datei speichern
ActiveWorkbook.SaveAs _
FileName:=Dateiname, _
FileFormat:=xlNormal, _
password:="", _
```

```
WriteResPassword:="", _
ReadOnlyRecommended:=False, _
CreateBackup:=False

' Datei versenden
sAddress = Range("D16").Value
' ACHTUNG in D16 oder andere Zelle muss die Empfaengeradresse
' hinein
Dim Empfaenger As String
Dim Betreff As String
Empfaenger = sAddress
Betreff = "dein Betreff"

Application.Dialogs(xlDialogSendMail).Show Empfaenger, Betreff

' TEMP-Datei schließen und löschen
ActiveWorkbook.Close False
'If MsgBox("Temporäre Datei " & Dateiname & " löschen?", vbYesNo) = vbYes Then
Kill Dateiname
'End If
End Sub
```

---

4. Lösung

Hier sind es allerdings bis zu 3 Tabellenbereiche aus einer Tabelle. Den Code musst Du natürlich entsprechen Deinen Bedürfnissen anpassen.

Gruß

```
Private Sub Mailversand_3_Tabellenbereiche()
Dim rng1 As Range
Dim rng2 As Range
Dim rng3 As Range
Dim sAdress As String

Application.ScreenUpdating = False
Set rng1 = Range("A63:J86")
Set rng2 = Range("A125:J187")
Set rng3 = Range("A384:J408")
sAdress = Range("B1").Value
Workbooks.Add 1
```

```
rng1.Copy Range("A24")
rng2.Copy Range("A47")
rng3.Copy Range("A1")
Range("A5:I5").Select
ActiveWorkbook.SendMail Worksheets "@xyz.de", " Betreff"
Application.DisplayAlerts = False
ActiveWindow.Close
Application.DisplayAlerts = True
Set rng3 = Nothing
Set rng2 = Nothing
Set rng1 = Nothing
Application.ScreenUpdating = True
End Sub
```

5.Lösung

```
Sub SendMail()
Dim outObj As Object
Dim Mail As Object
ActiveWorkbook.ActiveSheet.Copy
ActiveWorkbook.SaveAs "C:\windows\temp\anhang.xls"
Set outObj = CreateObject("Outlook.Application")
Set Mail = outObj.CreateItem(0)
With Mail
.Subject = "Hier den Betreff eingeben"
.Body = "Hier, falls du einen Text im Body haben möchtest"
.To = "Hier die E-Mail Adresse eintragen"
.Attachments.Add "C:\windows\temp\anhang.xls"
End With
Mail.Display
Set Mail = Nothing
Set outObj = Nothing
End Sub
```

Tabellenbereich kopieren und in Mail einfügen

Zuerst die beiden bekannten, nicht zufrieden stellenden Codes:

```
Sub Send_OriginalRange_from_Excel()
'Ohne Select geht es nicht :-))
```

```
Range("C47:C101").Select
'Das anzeigen der Envelope Commandbar ist unabdingbar (die bietet die Funktion: markierten Bereich mailen)
ActiveWorkbook.EnvelopeVisible = True
'Nun werden die Adressen vergeben
With ActiveSheet.MailEnvelope
      .Introduction = "Das ist der Einleitungstext." & vbCrLf & "mit einer zweiten Zeile"
      .Item.To = "sw8@gmx.at"
      .Item.Subject = "Die aktuellen Daten" ' betreff
       ' wenn man es gleich senden will
      '.Item.Send
       ' ansonsten, wenn man es noch bearbeiten will, deaktivere Item.Send und aktiviere Display
      '.Item.Display
   End With
End Sub
```

Ergebnis ist schon recht fein, da auch die Formatierungen mit übernommen warden - aber wenn man mit Item.Display die Datei noch nicht senden möchte, sondern noch zur Bearbeitung freigeben möchte, dann hat man noch keine echte E-Mail , sondern ist noch in der Exceldatei drin und kann nicht so gut die Mail fertig stellen.

Code 2:

```
Sub Excel_Range_via_Outlook_Senden()
    Dim OutApp As Object, Mail As Object, i
    Dim Nachricht
    'Verweis auf "Microsoft Forms 2.0 Object Library" aktivieren !!
    'sonst geht es nicht
    'Dataobject wird gebraucht wegen der Zwischenablage
    Dim ClpObj As DataObject
    'For i = 1 To 10 ' was die for next schleife soll weiß ich nicht
        Set ClpObj = New DataObject
        Set OutApp = CreateObject("Outlook.Application")
        Set Nachricht = OutApp.CreateItem(0)
        'Excelbereich der versendet werden soll.
'Wenn kein Bereich versendet werden soll sondern
'der Bereich bereits kopiert wurde, können sie die
'nächsten beiden Zeilen auskommentieren
        Range("C47:C101").Select
        'Bereich wird in die Zwischenablage kopiert
        Selection.Copy
        With Nachricht
            .Subject = "Betreffzeile Header"
            'Zwischenablage wird eingefügt
            ClpObj.GetFromClipboard
            .Body = ClpObj.GetText(1)
```

```
            .To = "sw8@gmx.at"
            'Hier wird die Mail angezeigt
            .Display
            'Hier wird die Mail gleich gesendet
            '.Send
        End With
        Set OutApp = Nothing
        Set Nachricht = Nothing
        'Auf Outlook warten. Ist nicht schnell genug :-))
        Application.Wait (Now + TimeValue("0:00:05"))
    'Next i '
End Sub
```

Auch eine nette Sache dieser Code, da eine echte Mail aufgeht - Nachteil: die Formate fehlen ☹

Nun meine Lösung

Es wird zuerst eine neue Mail geöffnet und an diese dann die Tastenkombination STRG-V bzw Alt-B-I übergeben, womit der zuvor in die Zwischenablage kopierte Teil eingefügt wird:

```
Sub MAILEN()


    Dim OutApp As Object, Mail As Object, i
    Dim Nachricht
    Dim ClpObj As DataObject


' Öffnen der Mail
        Set ClpObj = New DataObject
        Set OutApp = CreateObject("Outlook.Application")
        Set Nachricht = OutApp.CreateItem(0)
        Range("C47:C101").Select
        Selection.Copy
        Range("A1").Select
        With Nachricht
            .Subject = "betreff"
            ' Wichtig, eine E-Mail addy muss da sein, sonst bleibt der Cursor im AN-Feld und der Text wird hier eingefügt
            If Sheets("Einstellungen").Range("B32").Value <> "" Then .To = Sheets("Einstellungen").Range("B32").Value
        .Display ' Mail anzeigen - und nicht senden: .Send
        End With
        Set OutApp = Nothing
```

```
    Set Nachricht = Nothing


'Kurz warten, damit die Mail Zeit zum Öffnen hat
    Application.Wait (Now + TimeValue("0:00:05"))

' Dann die Zwischenablage einfügen
    'Application.SendKeys ("%bi") ' im Menü BEARBEITEN (Alt-B) das e-I-nfügen wählen
    Application.SendKeys ("^v") ' Strg-V Anweisung ist die 2.Möglichkeit statt Alt-B + I

End Sub
```

Theorie: siehe auch hier in der VBA-Sammlung die SENDKEYS-Anweisung

Tabellenblatt per Email versenden

VERSION 1

Hallo!
Hier mal ein Beispielcode. Funktioniert bei mir bestens.

```
Sub MailSenden()
Application.ScreenUpdating = False
On Error Resume Next
Dim empfänger As String
Dim i As Integer
Dim aws As String
Dim olapp As Object
For i = 1 To 50
Sheets(i).Activate
empfänger = Sheets(i).Range("C3").Value
ActiveWorkbook.ActiveSheet.Copy
ActiveWorkbook.SaveAs ActiveSheet.Name
aws = ActiveWorkbook.FullName
Set olapp = CreateObject("Outlook.Application")
With olapp.CreateItem(0)
    .To = empfänger
    '.CC = "" 'Optional Kopie an
    '.BCC = "" 'Optional Blindkopie an
    .Subject = "Abrechnung vom " & Date
    .HtmlBody = "Sehr geehrte Damen und Herren,
```

```
anbei die Abrechnung.

Mit _
freundlichen Grüßen,

Unterschrift"
    .attachments.Add aws
    .Display
  SendKeys "%s", True 'Mail sofort senden
  ActiveWorkbook.Close
   Set olapp = Nothing
End With
Next i
Sheets(1).Activate
Application.ScreenUpdating = True
End Sub
```

VERSION 2

Die folgende Nachricht zum Thema stammt von: Vinzenz Mai, 19. 11. 2007, 23:39
Hallo Alex,

»» »» mit welchem E-Mail-Client soll das funktionieren?

»» Ich dachte das ist unabhängig und das Script greift wie bei der manuellen Auswahl (Datei->Senden) auf dem als Standard festgelegten Client zurück.

also mit einem MAPI-fähigen Client :-)
Folgender Code (im Modul des Workbook-Objektes) sollte es tun:

```
Public Sub MapiSendMail()
   Dim xlWB As Workbook

   On Error GoTo Err_Sub

   ' Kopiere das aktuelle Tabellenblatt in eine neue Arbeitsmapppe
   Me.ActiveSheet.Copy

   ' Greife auf diese neue Arbeitsmappe
   Set xlWB = ActiveWorkbook

   ' Versende Mail über MAPI - Bestätigung typischerweise erforderlich :-)
   ' Anlage enthält eine Excel-Mappe Mappe<nr>.xls mit dem aktuellen Blatt
```

```
    ' Mailbody ist leer - beste Voraussetzungen, um als Spam klassifiziert
    ' zu werden.
    xlWB.SendMail "max.mustermann@example.org", "Hier der Betreff"

    ' Schließe die Datei, ohne Änderungen zu speichern
    xlWB.Close False

    ' Gebe Ressourcen frei
    Set xlWB = Nothing

    Exit Sub

Err_Sub:
    ' Primitive nichtfunktionale Fehler-"behandlung"
    MsgBox Err.Number & vbCrLf & Err.Description
    Exit Sub

End Sub
```

Für Serienmail ohne so etwas wie "ClickYes" nicht verwendbar :-)

--- Mail-Sammlung von Ron De Bruin ---

GRUNDSÄTZLICHES

http://www.rondebruin.nl/sendmail.htm:

The code is tested in Outlook Express, Windows Mail and Outlook and It may or may not work with other email clients. All code on this page will also work in Excel 2007.

There are three Microsoft mail programs:
Microsoft Outlook, Outlook Express, Windows Mail (new in Vista, replace Outlook Express).
Microsoft Outlook is a part of Microsoft Office and Outlook Express and Windows Mail are a part of the operating system.

The code in the links in the first section of this tips page are working with all three programs and uses the SendMail Method. The code in the second and third section of this tips page are only working with Microsoft Outlook. Because there is an object model for Outlook you have much more options when you use the code in this section.
A good alternative for the Outlook object model code is CDO for Win 2000 on my CDO page (no security warnings)

If you want it easy without using VBA code then try one of the Add-ins on this page
Mail Add-ins for Excel (Free and very easy to use)

Check out also the worksheet templates
Mail Worksheet Templates

New: Outlook Mail Worksheet Template for Excel 2007-2010
http://www.rondebruin.nl/mail/outlooksheettemplate.htm

Sending mail from Excel with CDO

Ron de Bruin (last update 31-Oct-2007)
Go back to the mail tips page

What is CDO doing

The example code is using CDOSYS (CDO for Windows 2000).
It does not depend on MAPI or CDO and hence is dialog free
and does not use your mail program to send email.
<You can send mail without a mail program>

Briefly to explain, this code builds the message and drops it
in the pickup directory, and SMTP service running on the machine
picks it up and send it out to the internet.

Why using CDO code instead of Outlook automation or SendMail in VBA.

1: It doesn't matter what Mail program you are using (It only use the SMTP server).
2: It doesn't matter what Office version you are using (97…2007)
3: You can send a range/sheet in the body of the mail (some mail programs can’t do this)
4: You can send any file you like (Word, PDF, PowerPoint, TXT files,….)
5: No Security warnings anymore, really great if you are sending a lot of mail in a loop.

Read this!!!

This code will not work in Win 98 and ME.
You must be connected to the internet when you run a example.

It is possible that you get a Send error when you use one of the examples.
AFAIK : This will happen if you haven't setup an account in Outlook Express or Windows Mail.
In that case the system doesn't know the name of your SMTP server.

If this happens you can use the commented green lines in each example.
Don't forget to fill in the SMTP server name in each code sample where
it says "Fill in your SMTP server here"

When you also get the Authentication Required Error you can add this three lines.
.Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
.Item("http://schemas.microsoft.com/cdo/configuration/sendusername") = "username"
.Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = "password"

Don't remove the TextBody line in the code. If you do you can't open the attachment (bug in CDO).
If you don't want to have text in the body use this then .TextBody = ""

Note: It is always possible that your firewall block the code (Check your firewall settings)

Can you use CDO on your machine?

Let's try a basic example first.

The code below will send four text lines in the body of the mail to the person in this line
.To = "ron@debruin.nl"

Change ron@debruin.nl to your own mail address before you test the code.
If you read the information above you know that if you have a account in Outlook Express or
Windows Mail you can Run the code below after changing the mail address.
But if you not have a account in Outlook Express or Windows Mail you also need the commented
green lines in the code. Remove every ' before every green line and fill in the name of your SMTP server
where it says "Fill in your SMTP server here"

1) Open a new workbook
2) Alt F11 (to open the VBA editor)
3) Insert>Module

4) Paste the code in this module
5) Make your changes
6) Alt q to go back to Excel

When you use Alt F8 you can select the macro and press Run.
Now wait a moment and see if you receive the mail in your inbox.

```
Sub CDO_Mail_Small_Text()
    Dim iMsg As Object
    Dim iConf As Object
    Dim strbody As String
    '    Dim Flds As Variant

    Set iMsg = CreateObject("CDO.Message")
    Set iConf = CreateObject("CDO.Configuration")

    '    iConf.Load -1    ' CDO Source Defaults
    '    Set Flds = iConf.Fields
    '    With Flds
    '        .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
    '        .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") _
    '                    = "Fill in your SMTP server here"
    '        .Item("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 25
    '        .Update
    '    End With

    strbody = "Hi there" & vbNewLine & vbNewLine & _
             "This is line 1" & vbNewLine & _
             "This is line 2" & vbNewLine & _
             "This is line 3" & vbNewLine & _
             "This is line 4"

    With iMsg
       Set .Configuration = iConf
       .To = "ron@debruin.nl"
       .CC = ""
       .BCC = ""
       .From = """Ron"" <ron@something.nl>"
       .Subject = "Important message"
       .TextBody = strbody
       .Send
    End With
```

```
End Sub
```

Use the GMail SMTP server from Google.
http://gmail.google.com

You can find the code in the workbook with examples that you can download below.
There is more information about the code in the workbook.
Note: You must have a Gmail account to try this example.

Download workbook with more examples

You can download a example workbook with eighth examples.
Download Example workbook with all the code

Attachment examples:
Module file1 = Workbook
Module file2 = One worksheet or more
Module file3 = Every sheet with a mail address in cell A1

Body examples:
Module body1 = Selection/Range or whole worksheet
Module body2 = Personalized Mail
Module body3 = Every sheet with a mail address in cell A1
Module body4 = Small text and text from a txt file

Note: the body examples in the workbook are using the function RangetoHTML in the "bodyfunction" module of the workbook.

Gmail example:

Module gmail = Use the smtp.gmail.com server from Gmail to send mail

Tips and links

CDO sheet template

Check out this sheet template if you want to send every sheet to a different person.
Or want to send one or more sheets to one or more recipient.
http://www.rondebruin.nl/mail/templates.htm

Set importance/priority and request read receipt

For importance/priority and read receipt you can add this in the With iMsg part of the macro before .Send

```
' Set importance or Priority to high
.Fields("urn:schemas:httpmail:importance") = 2
.Fields("urn:schemas:mailheader:X-Priority") = 1

' Request read receipt
.Fields("urn:schemas:mailheader:return-receipt-to") = "ron@debruin.nl"
.Fields("urn:schemas:mailheader:disposition-notification-to") = "ron@debruin.nl"

' Update fields
.Fields.Update
```

Changing the To line

If you want to mail to all E-mail addresses in a range then use this code
instead of .To = "ron@debruin.nl"

The example below will use the cells from sheets("Sheet1") in ThisWorkbook (workbook with the code)
It is possible that you must use ActiveWorkbook or something else in your code to use it.

```
Dim cell As Range
Dim strto As String
On Error Resume Next
For Each cell In ThisWorkbook.Sheets("Sheet1") _
    .Range("A1:A10").Cells.SpecialCells(xlCellTypeConstants)
    If cell.Value Like "?*@?*.?*" Then
        strto = strto & cell.Value & ";"
    End If
Next cell
On Error GoTo 0
If Len(strto) > 0 Then strto = Left(strto, Len(strto) - 1)
```

Change the To line to .To = strto

Or to more people
.To = "Jon@something.com;ron@something.com"

Or you can use a address in a cell like this
.To = Sheets("Sheet1").Range("C1").Value

Change the Body line

Plain text :

Note: see also the example in the workbook to send all text from a txt file (Module body4)

If you want to add more text to the body then you can use the code below.
Instead of .TextBody = "This is the body text" use .TextBody = strbody then.

```
Dim strbody As String
strbody = "Hi there" & vbNewLine & vbNewLine & _
    "This is line 1" & vbNewLine & _
    "This is line 2" & vbNewLine & _
    "This is line 3" & vbNewLine & _
    "This is line 4"
```

Or use this if you want to use cell values

```
Dim cell As Range
Dim strbody As String
For Each cell In Sheets("Sheet1").Range("C1:C20")
    strbody = strbody & cell.Value & vbNewLine
Next
```

Or this one

```
Dim strbody As String
With Sheets("Sheet1")
    strbody = "Hi there" & vbNewLine & vbNewLine & _
        .Range("A1") & vbNewLine & _
        .Range("A2") & vbNewLine & _
        .Range("A3") & vbNewLine & _
        .Range("A4")
End With
```

Links

```
.TextBody = "file://Yourcomputer/YourFolder/Week2.xls"

'If there are spaces use %20
.TextBody = "file://Yourcomputer/YourFolder/Week%202.xls"

'Example for a file on a website
.TextBody = "http://www.rondebruin.nl/files/EasyFilter.zip"
```

HTML text :

If you want to create emails that are formatted you can use HTMLBody (Office 2000 and up) instead of TextBody. You can find a lot of WebPages on the internet with more HTML tags examples.

```
.HTMLBody = "<H3><B>Dear Ron de Bruin</B></H3>" & _
"Please visit this website to download an update.<BR>" & _
"<A HREF=""http://www.rondebruin.nl/"">Ron's Excel Page</A>"
```

Tip: Or send a complete webpage, instead of HTMLBody or TextBody use

```
.CreateMHTMLBody "http://www.rondebruin.nl/copy1.htm"

Or file on your computer
.CreateMHTMLBody "file://C:/test.htm"
```

Copy the cells as values

If you want to paste as values the sheet must be unprotected!!!!!
Or Unprotect and Protect the sheet in the Sub also.

See this page for example code that you can use
http://www.rondebruin.nl/values.htm

Test if you are online

You can use code like this in your subroutine to avoid errors if you run the code when you are not online (example below is for a dial up connection)

For checking other connections check out this great website.
http://vbnet.mvps.org/

```
Public Declare Function InternetGetConnectedState _
                    Lib "wininet.dll" (lpdwFlags As Long, _
                                ByVal dwReserved As Long) As Boolean

Function IsConnected() As Boolean
   Dim Stat As Long
   IsConnected = (InternetGetConnectedState(Stat, 0&) <> 0)
End Function

Sub Test()
' Randy Birch
   If IsConnected = True Then
      MsgBox "Copy your mail code here"
   Else
      MsgBox "You can't use this subroutine because you are not online"
   End If
End Sub
```

Mail the whole workbook With SendMail

Ron de Bruin (last update 25-Jan-2009)
Go back to the mail index page
Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

The following subroutine sends the active workbook in an e-mail message.
Change the mail address and subject in the macro before you run it.

```
Sub Mail_workbook_1()
'Working in 97-2007
    Dim wb As Workbook
    Set wb = ActiveWorkbook

    If Val(Application.Version) >= 12 Then
        If wb.FileFormat = 51 And wb.HasVBProject = True Then
```

```
        MsgBox "There is VBA code in this xlsx file, there will" & vbNewLine & _
              "be no VBA code in the file you send. Save the" & vbNewLine & _
              "file first as xlsm and then try the macro again.", vbInformation
        Exit Sub
      End If
   End If

   On Error Resume Next
   wb.SendMail "ron@debruin.nl", _
            "This is the Subject line"
   On Error GoTo 0
End Sub
```

To send a workbook other than the active workbook, change the assignment to the wb variable.
For example Set wb = ThisWorkbook
Note: It doesn't have to be the active workbook used at that time.

Example 2

This sub will send a newly created workbook (copy of the ActiveWorkbook).
It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.
Change the mail address and subject in the macro before you run it.

```
Sub Mail_Workbook_2()
'Working in 2000-2007
   Dim wb1 As Workbook
   Dim wb2 As Workbook
   Dim TempFilePath As String
   Dim TempFileName As String
   Dim FileExtStr As String

   Set wb1 = ActiveWorkbook

   If Val(Application.Version) >= 12 Then
      If wb1.FileFormat = 51 And wb1.HasVBProject = True Then
         MsgBox "There is VBA code in this xlsx file, there will" & vbNewLine & _
              "be no VBA code in the file you send. Save the" & vbNewLine & _
              "file first as xlsm and then try the macro again.", vbInformation
         Exit Sub
```

```
        End If
    End If

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Make a copy of the file/Open it/Mail it/Delete it
    'If you want to change the file name then change only TempFileName
    TempFilePath = Environ$("temp") & "\"
    TempFileName = "Copy of " & wb1.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")
    FileExtStr = "." & LCase(Right(wb1.Name, _
                             Len(wb1.Name) - InStrRev(wb1.Name, ".", , 1)))

    wb1.SaveCopyAs TempFilePath & TempFileName & FileExtStr
    Set wb2 = Workbooks.Open(TempFilePath & TempFileName & FileExtStr)

    With wb2
        On Error Resume Next
        .SendMail "ron@debruin.nl", _
                "This is the Subject line"
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    'Delete the file
    Kill TempFilePath & TempFileName & FileExtStr

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Mail one Sheet With SendMail

Ron de Bruin (last update 25-Jan-2009)
Go back to the mail index page
Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

The following subroutine sends a newly created workbook with just the ActiveSheet.
It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.
Change the mail address and subject in the macro before you run it.

Important: Read also the information below the macro

```
Sub Mail_ActiveSheet()
'Working in 97-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim Sourcewb As Workbook
    Dim Destwb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    Set Sourcewb = ActiveWorkbook

    'Copy the sheet to a new workbook
    ActiveSheet.Copy
    Set Destwb = ActiveWorkbook

    'Determine the Excel version and file extension/format
    With Destwb
        If Val(Application.Version) < 12 Then
            'You use Excel 97-2003
```

```
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007, we exit the sub when your answer is
        'NO in the security dialog that you only see  when you copy
        'an sheet from a xlsm file with macro's disabled.
        If Sourcewb.Name = .Name Then
            With Application
                .ScreenUpdating = True
                .EnableEvents = True
            End With
            MsgBox "Your answer is NO in the security dialog"
            Exit Sub
        Else
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
                End If
            Case 56: FileExtStr = ".xls": FileFormatNum = 56
            Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
            End Select
        End If
    End If
End With

'    'Change all cells in the worksheet to values if you want
'    With Destwb.Sheets(1).UsedRange
'        .Cells.Copy
'        .Cells.PasteSpecial xlPasteValues
'        .Cells(1).Select
'    End With
'    Application.CutCopyMode = False

'Save the new workbook/Mail it/Delete it
TempFilePath = Environ$("temp") & "\"
TempFileName = "Part of " & Sourcewb.Name & " " _
           & Format(Now, "dd-mmm-yy h-mm-ss")

With Destwb
    .SaveAs TempFilePath & TempFileName & FileExtStr, _
          FileFormat:=FileFormatNum
```

```
        On Error Resume Next
        .SendMail "ron@debruin.nl", _
                "This is the Subject line"
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    'Delete the file you have send
    Kill TempFilePath & TempFileName & FileExtStr

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

You can also use the following line if you know the sheet you want to mail :
Sheets("Sheet5").Copy
It doesn't have to be the active sheet used at that time.

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 97-2003

If you run the code in Excel 2007 it will look at the FileFormat of the parent workbook
and save the new file in that format.
Only if the parent workbook is an xlsm file and if there is no code in the new workbook
it will save the new file as xlsx, this way the receiver knows that this is a macro free file.
If the parent workbook is not an xlsx, xlsm, or xls then it will be saved as xlsb.

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

If you always want to save in a certain format you can replace this part of the macro

```
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
```

```
        End If
    Case 56: FileExtStr = ".xls": FileFormatNum = 56
    Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
    End Select
```

With one of the one liners from this list

```
FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56
```

Or maye you want to save the one sheet workbook to csv, txt ot prn.
(you can use this also if you run it in 97-2003)

```
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
```

Example 2

The example below will send each sheet in the Shname Array to a person In the Addr Array.
In this example four separate mails will be send with one sheet.

Sheet1 to ron@test.nl
Sheet2 to jelle@test.nl
Sheet3 to judith@test.nl
Sheet4 to nicolet@test.nl

If you run the macro in Excel 2007 the files will be saved/send as xlsm files.
You can change that to another format if you want (See information of Example 1)
Copy this macro in a module of the file with the sheets in the array (not in your personal.xls(b)).

See also the mail index page for other code examples to do this or install the sheet template.

```
Sub Mail_Sheets()
'Working in 97-2007
    Dim wb As Workbook
    Dim Shname As Variant
    Dim Addr As Variant
    Dim N As Integer
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    Shname = Array("Sheet1", "Sheet2", "Sheet3", "Sheet4")
```

```
    Addr = Array("ron@test.nl", "jelle@test.nl", "judith@test.nl", "nicolet@test.nl")

    If Val(Application.Version) >= 12 Then
        'You run Excel 2007
        FileExtStr = ".xlsm": FileFormatNum = 52
    Else
        'You run Excel 97-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    End If

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    TempFilePath = Environ$("temp") & "\"

    'Create the new workbooks/Mail it/Delete it
    For N = LBound(Shname) To UBound(Shname)

        TempFileName = "Sheet " & Shname(N) & " " & Format(Now, "dd-mmm-yy h-mm-ss")

        ThisWorkbook.Sheets(Shname(N)).Copy
        Set wb = ActiveWorkbook

        With wb
            .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormatNum
            On Error Resume Next
            .SendMail Addr(N), _
                    "This is the Subject line"
            On Error Resume Next
            .Close SaveChanges:=False
        End With

        Kill TempFilePath & TempFileName & FileExtStr

    Next N

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Mail more then one sheet with SendMail

Ron de Bruin (last update 25-Jan-2009)
Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

The following subroutine sends a newly created workbook with just the sheets in the Array.
.Sheets(Array("Sheet1", "Sheet3")).Copy

Use this if you want to send the selected sheets
TheActiveWindow.SelectedSheets.Copy

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the macro

```
Sub Mail_Sheets_Array()
'Working in 97-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim Sourcewb As Workbook
    Dim Destwb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim sh As Worksheet
    Dim TheActiveWindow As Window
    Dim TempWindow As Window

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    Set Sourcewb = ActiveWorkbook

    'Copy the sheets to a new workbook
    'We add a temporary Window to avoid the Copy problem
    'if there is a List or Table in one of the sheets and
    'if the sheets are grouped
    With Sourcewb
        Set TheActiveWindow = ActiveWindow
        Set TempWindow = .NewWindow
        .Sheets(Array("Sheet1", "Sheet3")).Copy
    End With

    'Close temporary Window
    TempWindow.Close

    Set Destwb = ActiveWorkbook

    'Determine the Excel version and file extension/format
    With Destwb
        If Val(Application.Version) < 12 Then
            'You use Excel 97-2003
            FileExtStr = ".xls": FileFormatNum = -4143
        Else
            'You use Excel 2007, we exit the sub when your answer is
```

```
        'NO in the security dialog that you only see  when you copy
        'an sheet from a xlsm file with macro's disabled.
        If Sourcewb.Name = .Name Then
            With Application
                .ScreenUpdating = True
                .EnableEvents = True
            End With
            MsgBox "Your answer is NO in the security dialog"
            Exit Sub
        Else
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
                End If
            Case 56: FileExtStr = ".xls": FileFormatNum = 56
            Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
            End Select
        End If
    End If
End With

'    'Change all cells in the worksheets to values if you want
'    For Each sh In Destwb.Worksheets
'        sh.Select
'        With sh.UsedRange
'            .Cells.Copy
'            .Cells.PasteSpecial xlPasteValues
'            .Cells(1).Select
'        End With
'        Application.CutCopyMode = False
'        Destwb.Worksheets(1).Select
'    Next sh

'Save the new workbook/Mail it/Delete it
TempFilePath = Environ$("temp") & "\"
TempFileName = "Part of " & Sourcewb.Name _
           & " " & Format(Now, "dd-mmm-yy h-mm-ss")

With Destwb
    .SaveAs TempFilePath & TempFileName & FileExtStr, _
```

```
            FileFormat:=FileFormatNum
        On Error Resume Next
        .SendMail "ron@debruin.nl", _
                "This is the Subject line"
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    Kill TempFilePath & TempFileName & FileExtStr

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 97-2003

If you run the code in Excel 2007 it will look at the FileFormat of the parent workbook
and save the new file in that format.
Only if the parent workbook is an xlsm file and if there is no code in the new workbook
it will save the new file as xlsx, this way the receiver knows that this is a macro free file.
If the parent workbook is not an xlsx, xlsm, or xls then it will be saved as xlsb.

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro’s, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

If you always want to save in a certain format you can replace this part of the macro

```
Select Case Sourcewb.FileFormat
Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
Case 52:
   If .HasVBProject Then
      FileExtStr = ".xlsm": FileFormatNum = 52
   Else
      FileExtStr = ".xlsx": FileFormatNum = 51
   End If
Case 56: FileExtStr = ".xls": FileFormatNum = 56
Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
End Select
```

With one of the one liners from this list

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Or maye you want to save the one sheet workbook to csv, txt ot prn.
(you can use this also if you run it in 97-2003)

FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36

Mail Range or Selection with SendMail

Ron de Bruin (last update 25-Jan-2009)
Go back to the mail index page
Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

The following subroutine sends a newly created workbook with just the visible cells
in the Range("A1:K50").The cells will be PasteSpecial as values in the workbook you send.

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the macro

```
Sub Mail_Range()
'Working in 2000-2007
    Dim Source As Range
    Dim Dest As Workbook
    Dim wb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    Set Source = Nothing
    On Error Resume Next
    Set Source = Range("A1:K50").SpecialCells(xlCellTypeVisible)
    On Error GoTo 0

    If Source Is Nothing Then
        MsgBox "The source is not a range or the sheet is protected, " & _
               "please correct and try again.", vbOKOnly
        Exit Sub
    End If
```

```
With Application
   .ScreenUpdating = False
   .EnableEvents = False
End With

Set wb = ActiveWorkbook
Set Dest = Workbooks.Add(xlWBATWorksheet)

Source.Copy
With Dest.Sheets(1)
   .Cells(1).PasteSpecial Paste:=8
   .Cells(1).PasteSpecial Paste:=xlPasteValues
   .Cells(1).PasteSpecial Paste:=xlPasteFormats
   .Cells(1).Select
   Application.CutCopyMode = False
End With

TempFilePath = Environ$("temp") & "\"
TempFileName = "Selection of " & wb.Name & " " _
          & Format(Now, "dd-mmm-yy h-mm-ss")

If Val(Application.Version) < 12 Then
   'You use Excel 2000-2003
   FileExtStr = ".xls": FileFormatNum = -4143
Else
   'You use Excel 2007
   FileExtStr = ".xlsx": FileFormatNum = 51
End If

With Dest
   .SaveAs TempFilePath & TempFileName & FileExtStr, _
         FileFormat:=FileFormatNum
   On Error Resume Next
   .SendMail "ron@debruin.nl", _
          "This is the Subject line"
   On Error GoTo 0
   .Close SaveChanges:=False
End With

Kill TempFilePath & TempFileName & FileExtStr

With Application
   .ScreenUpdating = True
   .EnableEvents = True
```

```
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 2000-2003

If you run the code in Excel 2007 it will save the new file as xlsx
But you can change that if you want

Options for all Excel versions :

Save the one sheet workbook to csv, txt or prn.
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36

Options only for Excel 2007 :

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Example 2

The following subroutine sends a newly created workbook with just the visible cells
in the Selection. The cells will be PasteSpecial as values in the workbook you send.

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the first macro on this page.

```
Sub Mail_Selection()
'Working in 2000-2007
    Dim Source As Range
    Dim Dest As Workbook
    Dim wb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    Set Source = Nothing
    On Error Resume Next
    Set Source = Selection.SpecialCells(xlCellTypeVisible)
    On Error GoTo 0

    If Source Is Nothing Then
        MsgBox "The source is not a range or the sheet is protected, " & _
               "please correct and try again.", vbOKOnly
        Exit Sub
    End If

    If ActiveWindow.SelectedSheets.Count > 1 Or _
       Selection.Cells.Count = 1 Or _
       Selection.Areas.Count > 1 Then
        MsgBox "An Error occurred :" & vbNewLine & vbNewLine & _
               "You have more than one sheet selected." & vbNewLine & _
               "You only selected one cell." & vbNewLine & _
               "You selected more than one area." & vbNewLine & vbNewLine & _
               "Please correct and try again.", vbOKOnly
        Exit Sub
    End If

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    Set wb = ActiveWorkbook
    Set Dest = Workbooks.Add(xlWBATWorksheet)
```

```
    Source.Copy
    With Dest.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
        .Cells(1).PasteSpecial Paste:=xlPasteValues
        .Cells(1).PasteSpecial Paste:=xlPasteFormats
        .Cells(1).Select
        Application.CutCopyMode = False
    End With

    TempFilePath = Environ$("temp") & "\"
    TempFileName = "Selection of " & wb.Name & " " _
               & Format(Now, "dd-mmm-yy h-mm-ss")

    If Val(Application.Version) < 12 Then
        'You use Excel 2000-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007
        FileExtStr = ".xlsx": FileFormatNum = 51
    End If

    With Dest
        .SaveAs TempFilePath & TempFileName & FileExtStr, _
               FileFormat:=FileFormatNum
        On Error Resume Next
        .SendMail "ron@debruin.nl", _
                 "This is the Subject line"
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    Kill TempFilePath & TempFileName & FileExtStr

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Mail every WorkSheet with address in A1 with SendMail

Ron de Bruin (last update 25-Jan-2009)

Go back to the mail index page

Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

This procedure will mail every Worksheet with an address in cell A1. It does this by
cycling through each worksheet in the workbook and checking cell A1 for the @ character.
If found, a copy of the worksheet is made and saved with a date/time stamp, and then sent
by e-mail to the address in cell A1. And finally, the file is deleted from your hard disk

Note: Copy this macro in a module of the file with the sheets you want to send (not in your personal.xls(b))

Important: Read also the information below the macro

```
Sub Mail_Every_Worksheet()
'Working in 97-2007
    Dim sh As Worksheet
```

```
Dim wb As Workbook
Dim FileExtStr As String
Dim FileFormatNum As Long
Dim TempFilePath As String
Dim TempFileName As String

TempFilePath = Environ$("temp") & "\"

If Val(Application.Version) < 12 Then
   'You use Excel 97-2003
   FileExtStr = ".xls": FileFormatNum = -4143
Else
   'You use Excel 2007
   FileExtStr = ".xlsm": FileFormatNum = 52
End If

With Application
   .ScreenUpdating = False
   .EnableEvents = False
End With

For Each sh In ThisWorkbook.Worksheets
   If sh.Range("A1").Value Like "?*@?*.?*" Then

      sh.Copy
      Set wb = ActiveWorkbook

      TempFileName = "Sheet " & sh.Name & " of " _
                & ThisWorkbook.Name & " " _
                & Format(Now, "dd-mmm-yy h-mm-ss")

      With wb
         .SaveAs TempFilePath & TempFileName & FileExtStr, _
               FileFormat:=FileFormatNum
         On Error Resume Next
         .SendMail sh.Range("A1").Value, _
                  "This is the Subject line"
         On Error GoTo 0
         .Close SaveChanges:=False
      End With

      Kill TempFilePath & TempFileName & FileExtStr

   End If
```

```
    Next sh

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 2000-2003

If you run the code in Excel 2007 it will save the new file as xlsm
But you can change that if you want

Options for all Excel versions :

Save the one sheet workbook to csv, txt or prn.
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36

Options only for Excel 2007 :

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro’s, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Mail a row or rows to each person in a range with SendMail

Ron de Bruin (last update 25-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is working with Outlook, Outlook Express and Windows Mail.
With SendMail it is not possible to :

1) Send text in the Body of the mail
2) Use the CC or BCC field
3) Attach other files

If you want to have the options above and more and use Outlook you
can use one of the Outlook object model examples on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips1.htm

Example 1

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have :

In column A : Names of the students or ?
In column B:H : Information about the student or ?

We filter the range A1:H? for every unique name in the name column (column A in this example)
For every unique name we create a new file with only the data of that person and send it to the
mail address it find with the VLookup function in the worksheet "Mailinfo".

Important: You must create this worksheet manual and add the names and mail addresses one time.
Add a worksheet to your workbook with the name "Mailinfo" with in column A the names
and in column B the mail addresses of every possible person in your Name column..

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

```
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 1      'Filter column = A because the filter range start in A
```

Copy the macro below in a Standard module.

```
Sub Send_Row_Or_Rows_Attachment_1()
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
    Dim Rnum As Long
    Dim FilterRange As Range
    Dim FieldNum As Integer
    Dim mailAddress As String
    Dim NewWB As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    On Error GoTo cleanup

    With Application
        .EnableEvents = False
```

```
    .ScreenUpdating = False
End With

'Set filter sheet, you can also use Sheets("MySheet")
Set Ash = ActiveSheet

'Set filter range and filter column (column with names)
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 1    'Filter column = A because the filter range start in column A

'Add a worksheet for the unique list and copy the unique list in A1
Set Cws = Worksheets.Add
FilterRange.Columns(FieldNum).AdvancedFilter _
        Action:=xlFilterCopy, _
        CopyToRange:=Cws.Range("A1"), _
        CriteriaRange:="", Unique:=True

'Count of the unique values + the header cell
Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

'If there are unique values start the loop
If Rcount >= 2 Then
    For Rnum = 2 To Rcount

        'Look for the mail address in the MailInfo worksheet
        mailAddress = ""
        On Error Resume Next
        mailAddress = Application.WorksheetFunction. _
                    VLookup(Cws.Cells(Rnum, 1).Value, _
                    Worksheets("Mailinfo").Range("A1:B" & _
                    Worksheets("Mailinfo").Rows.Count), 2, False)
        On Error GoTo 0

        If mailAddress <> "" Then

            'Filter the FilterRange on the FieldNum column
            FilterRange.AutoFilter Field:=FieldNum, _
                            Criteria1:=Cws.Cells(Rnum, 1).Value

            'Copy the visible data in a new workbook
            With Ash.AutoFilter.Range
                On Error Resume Next
                Set rng = .SpecialCells(xlCellTypeVisible)
                On Error GoTo 0
```

```
        End With

        Set NewWB = Workbooks.Add(xlWBATWorksheet)

        rng.Copy
        With NewWB.Sheets(1)
            .Cells(1).PasteSpecial Paste:=8
            .Cells(1).PasteSpecial Paste:=xlPasteValues
            .Cells(1).PasteSpecial Paste:=xlPasteFormats
            .Cells(1).Select
            Application.CutCopyMode = False
        End With

        'Create a file name
        TempFilePath = Environ$("temp") & "\"
        TempFileName = "Your data of " & Ash.Parent.Name _
                  & " " & Format(Now, "dd-mmm-yy h-mm-ss")

        If Val(Application.Version) < 12 Then
            'You use Excel 2000-2003
            FileExtStr = ".xls": FileFormatNum = -4143
        Else
            'You use Excel 2007
            FileExtStr = ".xlsx": FileFormatNum = 51
        End If

        'Save, Mail, Close and Delete the file
        With NewWB
            .SaveAs TempFilePath & TempFileName _
                 & FileExtStr, FileFormat:=FileFormatNum
            On Error Resume Next
            .SendMail mailAddress, _
                    "This is the Subject line"
            On Error GoTo 0
            .Close savechanges:=False
        End With

        Kill TempFilePath & TempFileName & FileExtStr
    End If

    'Close AutoFilter
    Ash.AutoFilterMode = False

Next Rnum
```

```
    End If

cleanup:
    Application.DisplayAlerts = False
    Cws.Delete
    Application.DisplayAlerts = True

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
```

Example 2

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel 2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have :

In column A : Names of the students or ?
In column B : E-mail addresses
In column C:H : Information about the student or ?

Note: Every row must have a mail address in column B

We filter the range A1:H? for every unique mail address in column B.
For every unique mail address we create a new file with only the records with that
mail address and send it to that mail address.

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

```
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 2        'Filter column = B because the filter range start in A
```

Copy the macro below in a Standard module.

```
Sub Send_Row_Or_Rows_Attachment_2()
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
    Dim Rnum As Long
    Dim FilterRange As Range
    Dim FieldNum As Integer
    Dim NewWB As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    On Error GoTo cleanup

    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    'Set filter sheet, you can also use Sheets("MySheet")
    Set Ash = ActiveSheet

    'Set filter range and filter column (column with e-mail addresses)
    Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
    FieldNum = 2    'Filter column = B because the filter range start in column A

    'Add a worksheet for the unique list and copy the unique list in A1
    Set Cws = Worksheets.Add
    FilterRange.Columns(FieldNum).AdvancedFilter _
          Action:=xlFilterCopy, _
          CopyToRange:=Cws.Range("A1"), _
          CriteriaRange:="", Unique:=True

    'Count of the unique values + the header cell
    Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

    'If there are unique values start the loop
    If Rcount >= 2 Then
        For Rnum = 2 To Rcount
```

```
'If the unique value is a mail addres create a mail
If Cws.Cells(Rnum, 1).Value Like "?*@?*.?*" Then

    'Filter the FilterRange on the FieldNum column
    FilterRange.AutoFilter Field:=FieldNum, _
                        Criteria1:=Cws.Cells(Rnum, 1).Value

    'Copy the visible data in a new workbook
    With Ash.AutoFilter.Range
       On Error Resume Next
       Set rng = .SpecialCells(xlCellTypeVisible)
       On Error GoTo 0
    End With

    Set NewWB = Workbooks.Add(xlWBATWorksheet)

    rng.Copy
    With NewWB.Sheets(1)
       .Cells(1).PasteSpecial Paste:=8
       .Cells(1).PasteSpecial Paste:=xlPasteValues
       .Cells(1).PasteSpecial Paste:=xlPasteFormats
       .Cells(1).Select
       Application.CutCopyMode = False
    End With

    'Create a file name
    TempFilePath = Environ$("temp") & "\"
    TempFileName = "Your data of " & Ash.Parent.Name _
               & " " & Format(Now, "dd-mmm-yy h-mm-ss")

    If Val(Application.Version) < 12 Then
       'You use Excel 2000-2003
       FileExtStr = ".xls": FileFormatNum = -4143
    Else
       'You use Excel 2007
       FileExtStr = ".xlsx": FileFormatNum = 51
    End If

    'Save, Mail, Close and Delete the file
    With NewWB
       .SaveAs TempFilePath & TempFileName _
            & FileExtStr, FileFormat:=FileFormatNum
       On Error Resume Next
```

```
                .SendMail Cws.Cells(Rnum, 1).Value, _
                        "This is the Subject line"
                On Error GoTo 0
                .Close savechanges:=False
            End With

            Kill TempFilePath & TempFileName & FileExtStr
        End If

        'Close AutoFilter
        Ash.AutoFilterMode = False

    Next Rnum
  End If

cleanup:
  Application.DisplayAlerts = False
  Cws.Delete
  Application.DisplayAlerts = True

  With Application
    .EnableEvents = True
    .ScreenUpdating = True
  End With
End Sub
```

- OUTLOOK OBJECT MODEL - ATTACHMENT - Mail the whole workbook

Ron de Bruin (last update 25-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.

http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends the last saved version of the active workbook in an e-mail message.
Change the mail address and subject in the macro before you run it.

```
Sub Mail_workbook_Outlook_1()
'Working in 2000-2007
'This example send the last saved version of the Activeworkbook
    Dim OutApp As Object
    Dim OutMail As Object

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add ActiveWorkbook.FullName
        'You can add other files also like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Example 2

This sub will send a newly created workbook (copy of the ActiveWorkbook).
It is saving the workbook before mailing it with a date/time stamp.

After the file is sent the workbook will be deleted from your hard disk.
Change the mail address and subject in the macro before you run it.

```
Sub Mail_workbook_Outlook_2()
'Working in 2000-2007
    Dim wb1 As Workbook
    Dim wb2 As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim OutApp As Object
    Dim OutMail As Object

    Set wb1 = ActiveWorkbook

    If Val(Application.Version) >= 12 Then
        If wb1.FileFormat = 51 And wb1.HasVBProject = True Then
            MsgBox "There is VBA code in this xlsx file, there will" & vbNewLine & _
                  "be no VBA code in the file you send. Save the" & vbNewLine & _
                  "file first as xlsm and then try the macro again.", vbInformation
            Exit Sub
        End If
    End If

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Make a copy of the file/Open it/Mail it/Delete it
    'If you want to change the file name then change only TempFileName
    TempFilePath = Environ$("temp") & "\"
    TempFileName = "Copy of " & wb1.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")
    FileExtStr = "." & LCase(Right(wb1.Name, _
                            Len(wb1.Name) - InStrRev(wb1.Name, ".", , 1)))

    wb1.SaveCopyAs TempFilePath & TempFileName & FileExtStr
    Set wb2 = Workbooks.Open(TempFilePath & TempFileName & FileExtStr)

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    On Error Resume Next
```

```
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add wb2.FullName
        'You can add other files also like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
    End With
    On Error GoTo 0

    wb2.Close SaveChanges:=False

    'Delete the file
    Kill TempFilePath & TempFileName & FileExtStr

    Set OutMail = Nothing
    Set OutApp = Nothing

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11

2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail one worksheet

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends a newly created workbook with just the ActiveSheet.
It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.
Change the mail address and subject in the macro before you run it.

Important: Read also the information below the macro

```
Sub Mail_ActiveSheet()
'Working in 2000-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim Sourcewb As Workbook
    Dim Destwb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim OutApp As Object
    Dim OutMail As Object

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    Set Sourcewb = ActiveWorkbook

    'Copy the sheet to a new workbook
    ActiveSheet.Copy
    Set Destwb = ActiveWorkbook

    'Determine the Excel version and file extension/format
    With Destwb
        If Val(Application.Version) < 12 Then
            'You use Excel 2000-2003
            FileExtStr = ".xls": FileFormatNum = -4143
        Else
            'You use Excel 2007, we exit the sub when your answer is
            'NO in the security dialog that you only see  when you copy
            'an sheet from a xlsm file with macro's disabled.
            If Sourcewb.Name = .Name Then
                With Application
```

```
            .ScreenUpdating = True
            .EnableEvents = True
        End With
        MsgBox "Your answer is NO in the security dialog"
        Exit Sub
      Else
        Select Case Sourcewb.FileFormat
        Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
        Case 52:
          If .HasVBProject Then
            FileExtStr = ".xlsm": FileFormatNum = 52
          Else
            FileExtStr = ".xlsx": FileFormatNum = 51
          End If
        Case 56: FileExtStr = ".xls": FileFormatNum = 56
        Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
        End Select
      End If
   End If
End With

'    'Change all cells in the worksheet to values if you want
'    With Destwb.Sheets(1).UsedRange
'        .Cells.Copy
'        .Cells.PasteSpecial xlPasteValues
'        .Cells(1).Select
'    End With
'    Application.CutCopyMode = False

'Save the new workbook/Mail it/Delete it
TempFilePath = Environ$("temp") & "\"
TempFileName = "Part of " & Sourcewb.Name & " " _
           & Format(Now, "dd-mmm-yy h-mm-ss")

Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon
Set OutMail = OutApp.CreateItem(0)

With Destwb
   .SaveAs TempFilePath & TempFileName & FileExtStr, _
         FileFormat:=FileFormatNum
   On Error Resume Next
   With OutMail
      .To = "ron@debruin.nl"
```

```
            .CC = ""
            .BCC = ""
            .Subject = "This is the Subject line"
            .Body = "Hi there"
            .Attachments.Add Destwb.FullName
            'You can add other files also like this
            '.Attachments.Add ("C:\test.txt")
            .Send   'or use .Display
        End With
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    'Delete the file you have send
    Kill TempFilePath & TempFileName & FileExtStr

    Set OutMail = Nothing
    Set OutApp = Nothing

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

You can also use the following line if you know the sheet you want to mail :
Sheets("Sheet5").Copy
It doesn't have to be the active sheet used at that time.

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 97-2003

If you run the code in Excel 2007 it will look at the FileFormat of the parent workbook
and save the new file in that format.

Only if the parent workbook is an xlsm file and if there is no code in the new workbook
it will save the new file as xlsx, this way the receiver knows that this is a macro free file.
If the parent workbook is not an xlsx, xlsm, or xls then it will be saved as xlsb.

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

If you always want to save in a certain format you can replace this part of the macro

```
Select Case Sourcewb.FileFormat
Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
Case 52:
   If .HasVBProject Then
      FileExtStr = ".xlsm": FileFormatNum = 52
   Else
      FileExtStr = ".xlsx": FileFormatNum = 51
   End If
Case 56: FileExtStr = ".xls": FileFormatNum = 56
Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
End Select
```

With one of the one liners from this list

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Or maye you want to save the one sheet workbook to csv, txt ot prn.
(you can use this also if you run it in 97-2003)

FileExtStr = ".csv": FileFormatNum = 6

```
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem
```

Set OutMail = OutApp.CreateItem(olMailItem)

Mail more then one worksheet

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends a newly created workbook with just the sheets in the Array.
.Sheets(Array("Sheet1", "Sheet3")).Copy

Use this if you want to send the selected sheets
TheActiveWindow.SelectedSheets.Copy

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the macro

```
Sub Mail_Sheets_Array()
'Working in 2000-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim Sourcewb As Workbook
    Dim Destwb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim OutApp As Object
    Dim OutMail As Object
    Dim sh As Worksheet
    Dim TheActiveWindow As Window
    Dim TempWindow As Window

    With Application
        .ScreenUpdating = False
```

```
    .EnableEvents = False
End With

Set Sourcewb = ActiveWorkbook

'Copy the sheets to a new workbook
'We add a temporary Window to avoid the Copy problem
'if there is a List or Table in one of the sheets and
'if the sheets are grouped
With Sourcewb
    Set TheActiveWindow = ActiveWindow
    Set TempWindow = .NewWindow
    .Sheets(Array("Sheet1", "Sheet3")).Copy
End With

'Close temporary Window
TempWindow.Close

Set Destwb = ActiveWorkbook

'Determine the Excel version and file extension/format
With Destwb
    If Val(Application.Version) < 12 Then
        'You use Excel 97-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007, we exit the sub when your answer is
        'NO in the security dialog that you only see  when you copy
        'an sheet from a xlsm file with macro's disabled.
        If Sourcewb.Name = .Name Then
            With Application
                .ScreenUpdating = True
                .EnableEvents = True
            End With
            MsgBox "Your answer is NO in the security dialog"
            Exit Sub
        Else
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
```

```
            End If
          Case 56: FileExtStr = ".xls": FileFormatNum = 56
          Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
          End Select
        End If
    End If
End With

'    'Change all cells in the worksheets to values if you want
'    For Each sh In Destwb.Worksheets
'        sh.Select
'        With sh.UsedRange
'            .Cells.Copy
'            .Cells.PasteSpecial xlPasteValues
'            .Cells(1).Select
'        End With
'        Application.CutCopyMode = False
'        Destwb.Worksheets(1).Select
'    Next sh


'Save the new workbook/Mail it/Delete it
TempFilePath = Environ$("temp") & "\"
TempFileName = "Part of " & Sourcewb.Name & " " _
           & Format(Now, "dd-mmm-yy h-mm-ss")

Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon
Set OutMail = OutApp.CreateItem(0)

With Destwb
    .SaveAs TempFilePath & TempFileName & FileExtStr, _
          FileFormat:=FileFormatNum
    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add Destwb.FullName
        'You can add other files also like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
```

```
        End With
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    'Delete the file you have send
    Kill TempFilePath & TempFileName & FileExtStr

    Set OutMail = Nothing
    Set OutApp = Nothing

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 97-2003

If you run the code in Excel 2007 it will look at the FileFormat of the parent workbook
and save the new file in that format.
Only if the parent workbook is an xlsm file and if there is no code in the new workbook
it will save the new file as xlsx, this way the receiver knows that this is a macro free file.
If the parent workbook is not an xlsx, xlsm, or xls then it will be saved as xlsb.

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

If you always want to save in a certain format you can replace this part of the macro

```
            Select Case Sourcewb.FileFormat
            Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
            Case 52:
                If .HasVBProject Then
                    FileExtStr = ".xlsm": FileFormatNum = 52
                Else
                    FileExtStr = ".xlsx": FileFormatNum = 51
                End If
            Case 56: FileExtStr = ".xls": FileFormatNum = 56
            Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
```

```
End Select
```

With one of the one liners from this list

```
FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56
```

Or maye you want to save the one sheet workbook to csv, txt ot prn.
(you can use this also if you run it in 97-2003)

```
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail Range or Selection

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page

Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends a newly created workbook with just the visible cells
in the Range("A1:K50").The cells will be PasteSpecial as values in the workbook you send.

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the macro

```
Sub Mail_Range()
'Working in 2000-2007
    Dim Source As Range
    Dim Dest As Workbook
    Dim wb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim OutApp As Object
    Dim OutMail As Object

    Set Source = Nothing
    On Error Resume Next
    Set Source = Range("A1:K50").SpecialCells(xlCellTypeVisible)
    On Error GoTo 0
```

```
If Source Is Nothing Then
    MsgBox "The source is not a range or the sheet is protected, " & _
           "please correct and try again.", vbOKOnly
    Exit Sub
End If

With Application
    .ScreenUpdating = False
    .EnableEvents = False
End With

Set wb = ActiveWorkbook
Set Dest = Workbooks.Add(xlWBATWorksheet)
Source.Copy
With Dest.Sheets(1)
    .Cells(1).PasteSpecial Paste:=8
    .Cells(1).PasteSpecial Paste:=xlPasteValues
    .Cells(1).PasteSpecial Paste:=xlPasteFormats
    .Cells(1).Select
    Application.CutCopyMode = False
End With

TempFilePath = Environ$("temp") & "\"
TempFileName = "Selection of " & wb.Name & " " _
           & Format(Now, "dd-mmm-yy h-mm-ss")

If Val(Application.Version) < 12 Then
    'You use Excel 2000-2003
    FileExtStr = ".xls": FileFormatNum = -4143
Else
    'You use Excel 2007
    FileExtStr = ".xlsx": FileFormatNum = 51
End If

Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon
Set OutMail = OutApp.CreateItem(0)

With Dest
    .SaveAs TempFilePath & TempFileName & FileExtStr, _
          FileFormat:=FileFormatNum
    On Error Resume Next
    With OutMail
```

```
            .To = "ron@debruin.nl"
            .CC = ""
            .BCC = ""
            .Subject = "This is the Subject line"
            .Body = "Hi there"
            .Attachments.Add Dest.FullName
            'You can add other files also like this
            '.Attachments.Add ("C:\test.txt")
            .Send   'or use .Display
        End With
        On Error GoTo 0
        .Close SaveChanges:=False
    End With

    Kill TempFilePath & TempFileName & FileExtStr

    Set OutMail = Nothing
    Set OutApp = Nothing

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 2000-2003

If you run the code in Excel 2007 it will save the new file as xlsx
But you can change that if you want

Options for all Excel versions :

Save the one sheet workbook to csv, txt or prn.
FileExtStr = ".csv": FileFormatNum = 6

FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36

Options only for Excel 2007 :

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Example 2

The following subroutine sends a newly created workbook with just the visible cells in the Selection. The cells will be PasteSpecial as values in the workbook you send.

It is saving the workbook before mailing it with a date/time stamp.
After the file is sent the workbook will be deleted from your hard disk.

Important: Read also the information below the first macro on this page.

```
Sub Mail_Selection()
'Working in 2000-2007
    Dim Source As Range
    Dim Dest As Workbook
    Dim wb As Workbook
    Dim TempFilePath As String
```

```
Dim TempFileName As String
Dim FileExtStr As String
Dim FileFormatNum As Long
Dim OutApp As Object
Dim OutMail As Object

Set Source = Nothing
On Error Resume Next
Set Source = Selection.SpecialCells(xlCellTypeVisible)
On Error GoTo 0

If Source Is Nothing Then
    MsgBox "The source is not a range or the sheet is protected, " & _
           "please correct and try again.", vbOKOnly
    Exit Sub
End If

If ActiveWindow.SelectedSheets.Count > 1 Or _
   Selection.Cells.Count = 1 Or _
   Selection.Areas.Count > 1 Then
    MsgBox "An Error occurred :" & vbNewLine & vbNewLine & _
           "You have more than one sheet selected." & vbNewLine & _
           "You only selected one cell." & vbNewLine & _
           "You selected more than one area." & vbNewLine & vbNewLine & _
           "Please correct and try again.", vbOKOnly
    Exit Sub
End If

With Application
    .ScreenUpdating = False
    .EnableEvents = False
End With

Set wb = ActiveWorkbook
Set Dest = Workbooks.Add(xlWBATWorksheet)
Source.Copy
With Dest.Sheets(1)
    .Cells(1).PasteSpecial Paste:=8
    .Cells(1).PasteSpecial Paste:=xlPasteValues
    .Cells(1).PasteSpecial Paste:=xlPasteFormats
    .Cells(1).Select
    Application.CutCopyMode = False
End With
```

```
TempFilePath = Environ$("temp") & "\"
TempFileName = "Selection of " & wb.Name & " " _
          & Format(Now, "dd-mmm-yy h-mm-ss")

If Val(Application.Version) < 12 Then
   'You use Excel 2000-2003
   FileExtStr = ".xls": FileFormatNum = -4143
Else
   'You use Excel 2007
   FileExtStr = ".xlsx": FileFormatNum = 51
End If

Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon
Set OutMail = OutApp.CreateItem(0)

With Dest
   .SaveAs TempFilePath & TempFileName & FileExtStr, _
         FileFormat:=FileFormatNum
   On Error Resume Next
   With OutMail
      .To = "ron@debruin.nl"
      .CC = ""
      .BCC = ""
      .Subject = "This is the Subject line"
      .Body = "Hi there"
      .Attachments.Add Dest.FullName
      'You can add other files also like this
      '.Attachments.Add ("C:\test.txt")
      .Send   'or use .Display
   End With
   On Error GoTo 0
   .Close SaveChanges:=False
End With

Kill TempFilePath & TempFileName & FileExtStr

Set OutMail = Nothing
Set OutApp = Nothing

With Application
   .ScreenUpdating = True
   .EnableEvents = True
End With
```

```
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail every worksheet with address in A1

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

This procedure will mail every Worksheet with an address in cell A1. It does this by
cycling through each worksheet in the workbook and checking cell A1 for the @ character.
If found, a copy of the worksheet is made and saved with a date/time stamp, and then sent
by e-mail to the address in cell A1. And finally, the file is deleted from your hard disk

Note: Copy this macro in a module of the file with the sheets you want to send (not in your personal.xls(b))

Important: Read also the information below the macro

```
Sub Mail_Every_Worksheet()
'Working in 2000-2007
    Dim sh As Worksheet
    Dim wb As Workbook
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim OutApp As Object
    Dim OutMail As Object

    TempFilePath = Environ$("temp") & "\"

    If Val(Application.Version) < 12 Then
        'You use Excel 97-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007
        FileExtStr = ".xlsm": FileFormatNum = 52
    End If

    With Application
```

```
    .ScreenUpdating = False
    .EnableEvents = False
End With

Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon

For Each sh In ThisWorkbook.Worksheets
    If sh.Range("A1").Value Like "?*@?*.?*" Then

        sh.Copy
        Set wb = ActiveWorkbook

        TempFileName = "Sheet " & sh.Name & " of " _
                   & ThisWorkbook.Name & " " _
                   & Format(Now, "dd-mmm-yy h-mm-ss")

        Set OutMail = OutApp.CreateItem(0)
        With wb
            .SaveAs TempFilePath & TempFileName & FileExtStr, _
                  FileFormat:=FileFormatNum
            On Error Resume Next
            With OutMail
                .To = sh.Range("A1").Value
                .CC = ""
                .BCC = ""
                .Subject = "This is the Subject line"
                .Body = "Hi there"
                .Attachments.Add wb.FullName
                'You can add other files also like this
                '.Attachments.Add ("C:\test.txt")
                .Send   'or use .Display
            End With
            On Error GoTo 0
            .Close SaveChanges:=False
        End With
        Set OutMail = Nothing

        Kill TempFilePath & TempFileName & FileExtStr
    End If
Next sh

Set OutApp = Nothing
```

```
    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Information

In the macro you see that if Val(Application.Version) < 12 is True that I use
FileExtStr = ".xls": FileFormatNum = -4143
This is the normal Excel workbook format in 2000-2003

If you run the code in Excel 2007 it will save the new file as xlsm
But you can change that if you want

Options for all Excel versions :

Save the one sheet workbook to csv, txt or prn.
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36

Options only for Excel 2007 :

This are the main formats in Excel 2007 :

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library

? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail a different file(s) to each person in a range

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

Make a list in Sheets("Sheet1") with :

In column A : Names of the people
In column B : E-mail addresses
In column C:Z : Filenames like this C:\Data\Book2.xls (don't have to be Excel files)

The Macro will loop through each row in "Sheet1" and if there is a E-mail address in column B
and file name(s) in column C:Z it will create a mail with this information and send it.

```
Sub Send_Files()
```

```
'Working in 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim sh As Worksheet
    Dim cell As Range, FileCell As Range, rng As Range

    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    Set sh = Sheets("Sheet1")

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon

    For Each cell In sh.Columns("B").Cells.SpecialCells(xlCellTypeConstants)

        'Enter the file names in the C:Z column in each row
        Set rng = sh.Cells(cell.Row, 1).Range("C1:Z1")

        If cell.Value Like "?*@?*.?*" And _
           Application.WorksheetFunction.CountA(rng) > 0 Then
            Set OutMail = OutApp.CreateItem(0)

            With OutMail
                .To = cell.Value
                .Subject = "Testfile"
                .Body = "Hi " & cell.Offset(0, -1).Value

                For Each FileCell In rng.SpecialCells(xlCellTypeConstants)
                    If Trim(FileCell) <> "" Then
                        If Dir(FileCell.Value) <> "" Then
                            .Attachments.Add FileCell.Value
                        End If
                    End If
                Next FileCell

                .Send  'Or use Display
            End With

            Set OutMail = Nothing
        End If
    Next cell
```

```
    Set OutApp = Nothing

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail chart or chart sheet as picture

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

This example sends a chart with the name "Chart 1" from "Sheet1" of the ActiveWorkbook.
It will save My_Sales1.gif in the temp folder, send the mail and delete My_Sales1.gif after that.

```
Sub SaveSend_Embedded_Chart()
'Working in 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim Fname As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    'fill in the file path/name of the gif file
    Fname = Environ$("temp") & "\My_Sales1.gif"

    'if you hold down the CTRL key when you select the chart
    'in 2000-2007 you see the name in the name box(formula bar)
    ActiveWorkbook.Worksheets("Sheet1").ChartObjects("Chart 1").Chart.Export _
```

```
        Filename:=Fname, FilterName:="GIF"

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add Fname
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Kill Fname
    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Example 2

This example sends a chart sheet with the name "Chart1" from the ActiveWorkbook.
It will save My_Sales2.gif the temp folder, send the mail and delete My_Sales2.gif after that.

```
Sub SaveSend_Chart_Sheet()
'Working in 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim Fname As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    'fill in the file path/name of the gif file
    Fname = Environ$("temp") & "\My_Sales2.gif"

    'Save Chart sheet as Gif file
    ActiveWorkbook.Sheets("Chart1").Export _
        Filename:=Fname, FilterName:="GIF"

    On Error Resume Next
    With OutMail
```

```
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add Fname
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Kill Fname
    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)

With this three

Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)

Mail a row or rows to each person in a range (Attachment)

Ron de Bruin (last update 25-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have:

In column A : Names of the students or ?
In column B:H : Information about the student or ?

We filter the range A1:H? for every unique name in the name column (column A in this example)
For every unique name we create a new file with only the data of that person and send it to the
mail address it find with the VLookup function in the worksheet "Mailinfo".

Important: You must create this worksheet manual and add the names and mail addresses one time.

Add a worksheet to your workbook with the name "Mailinfo" with in column A the names
and in column B the mail addresses of every possible person in your Name column..

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

```
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 1      'Filter column = A because the filter range start in A
```

Tip : For testing I use .Display, change it to .Send if it is working OK.

```
Sub Send_Row_Or_Rows_Attachment_1()
    Dim OutApp As Object
    Dim OutMail As Object
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
    Dim Rnum As Long
    Dim FilterRange As Range
    Dim FieldNum As Integer
    Dim mailAddress As String
    Dim NewWB As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String
    Dim FileExtStr As String
    Dim FileFormatNum As Long

    On Error GoTo cleanup
    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon

    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    'Set filter sheet, you can also use Sheets("MySheet")
    Set Ash = ActiveSheet

    'Set filter range and filter column (column with names)
    Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
    FieldNum = 1    'Filter column = A because the filter range start in column A
```

```
'Add a worksheet for the unique list and copy the unique list in A1
Set Cws = Worksheets.Add
FilterRange.Columns(FieldNum).AdvancedFilter _
       Action:=xlFilterCopy, _
       CopyToRange:=Cws.Range("A1"), _
       CriteriaRange:="", Unique:=True

'Count of the unique values + the header cell
Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

'If there are unique values start the loop
If Rcount >= 2 Then
   For Rnum = 2 To Rcount

       'Look for the mail address in the MailInfo worksheet
       mailAddress = ""
       On Error Resume Next
       mailAddress = Application.WorksheetFunction. _
                  VLookup(Cws.Cells(Rnum, 1).Value, _
                    Worksheets("Mailinfo").Range("A1:B" & _
                      Worksheets("Mailinfo").Rows.Count), 2, False)
       On Error GoTo 0

       If mailAddress <> "" Then

          'Filter the FilterRange on the FieldNum column
          FilterRange.AutoFilter Field:=FieldNum, _
                           Criteria1:=Cws.Cells(Rnum, 1).Value

          'Copy the visible data in a new workbook
          With Ash.AutoFilter.Range
             On Error Resume Next
             Set rng = .SpecialCells(xlCellTypeVisible)
             On Error GoTo 0
          End With

          Set NewWB = Workbooks.Add(xlWBATWorksheet)

          rng.Copy
          With NewWB.Sheets(1)
             .Cells(1).PasteSpecial Paste:=8
             .Cells(1).PasteSpecial Paste:=xlPasteValues
             .Cells(1).PasteSpecial Paste:=xlPasteFormats
             .Cells(1).Select
```

```
            Application.CutCopyMode = False
        End With

        'Create a file name
        TempFilePath = Environ$("temp") & "\"
        TempFileName = "Your data of " & Ash.Parent.Name _
                  & " " & Format(Now, "dd-mmm-yy h-mm-ss")

        If Val(Application.Version) < 12 Then
            'You use Excel 2000-2003
            FileExtStr = ".xls": FileFormatNum = -4143
        Else
            'You use Excel 2007
            FileExtStr = ".xlsx": FileFormatNum = 51
        End If

        'Save, Mail, Close and Delete the file
        Set OutMail = OutApp.CreateItem(0)

        With NewWB
            .SaveAs TempFilePath & TempFileName _
                 & FileExtStr, FileFormat:=FileFormatNum
            On Error Resume Next
            With OutMail
                .to = mailAddress
                .Subject = "Test mail"
                .Attachments.Add NewWB.FullName
                .Body = "Hi there"
                .Display  'Or use Send
            End With
            On Error GoTo 0
            .Close savechanges:=False
        End With

        Set OutMail = Nothing
        Kill TempFilePath & TempFileName & FileExtStr
    End If

    'Close AutoFilter
    Ash.AutoFilterMode = False

  Next Rnum
End If
```

```
cleanup:
    Set OutApp = Nothing
    Application.DisplayAlerts = False
    Cws.Delete
    Application.DisplayAlerts = True

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
```

Example 2

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have:

In column A : Names of the students or ?
In column B : E-mail addresses
In column C:H : Information about the student or ?

Note: Every row must have a mail address in column B

We filter the range A1:H? for every unique mail address in column B.
For every unique mail address we create a new file with only the records with that
mail address and send it to that mail address.

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

```
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 2        'Filter column = B because the filter range start in A
```

Tip : For testing I use .Display, change it to .Send if it is working OK.

```
Sub Send_Row_Or_Rows_Attachment_2()
    Dim OutApp As Object
    Dim OutMail As Object
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
```

```
Dim Rnum As Long
Dim FilterRange As Range
Dim FieldNum As Integer
Dim NewWB As Workbook
Dim TempFilePath As String
Dim TempFileName As String
Dim FileExtStr As String
Dim FileFormatNum As Long

On Error GoTo cleanup
Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon

With Application
   .EnableEvents = False
   .ScreenUpdating = False
End With

'Set filter sheet, you can also use Sheets("MySheet")
Set Ash = ActiveSheet

'Set filter range and filter column (column with e-mail addresses)
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 2    'Filter column = B because the filter range start in column A

'Add a worksheet for the unique list and copy the unique list in A1
Set Cws = Worksheets.Add
FilterRange.Columns(FieldNum).AdvancedFilter _
       Action:=xlFilterCopy, _
       CopyToRange:=Cws.Range("A1"), _
       CriteriaRange:="", Unique:=True

'Count of the unique values + the header cell
Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

'If there are unique values start the loop
If Rcount >= 2 Then
   For Rnum = 2 To Rcount

       'If the unique value is a mail addres create a mail
       If Cws.Cells(Rnum, 1).Value Like "?*@?*.?*" Then

           'Filter the FilterRange on the FieldNum column
           FilterRange.AutoFilter Field:=FieldNum, _
```

```
                Criteria1:=Cws.Cells(Rnum, 1).Value

'Copy the visible data in a new workbook
With Ash.AutoFilter.Range
   On Error Resume Next
   Set rng = .SpecialCells(xlCellTypeVisible)
   On Error GoTo 0
End With

Set NewWB = Workbooks.Add(xlWBATWorksheet)

rng.Copy
With NewWB.Sheets(1)
   .Cells(1).PasteSpecial Paste:=8
   .Cells(1).PasteSpecial Paste:=xlPasteValues
   .Cells(1).PasteSpecial Paste:=xlPasteFormats
   .Cells(1).Select
   Application.CutCopyMode = False
End With

'Create a file name
TempFilePath = Environ$("temp") & "\"
TempFileName = "Your data of " & Ash.Parent.Name _
          & " " & Format(Now, "dd-mmm-yy h-mm-ss")

If Val(Application.Version) < 12 Then
   'You use Excel 2000-2003
   FileExtStr = ".xls": FileFormatNum = -4143
Else
   'You use Excel 2007
   FileExtStr = ".xlsx": FileFormatNum = 51
End If

'Save, Mail, Close and Delete the file
Set OutMail = OutApp.CreateItem(0)

With NewWB
   .SaveAs TempFilePath & TempFileName _
        & FileExtStr, FileFormat:=FileFormatNum
   On Error Resume Next
   With OutMail
      .to = Cws.Cells(Rnum, 1).Value
      .Subject = "Test mail"
      .Attachments.Add NewWB.FullName
```

```
                .Body = "Hi there"
                .Display  'Or use Send
            End With
            On Error GoTo 0
            .Close savechanges:=False
        End With

        Set OutMail = Nothing
        Kill TempFilePath & TempFileName & FileExtStr
    End If

    'Close AutoFilter
    Ash.AutoFilterMode = False

  Next Rnum
 End If

cleanup:
 Set OutApp = Nothing
 Application.DisplayAlerts = False
 Cws.Delete
 Application.DisplayAlerts = True

 With Application
  .EnableEvents = True
  .ScreenUpdating = True
 End With
End Sub
```

Create and Mail PDF files with Excel 2007/2010

Ron de Bruin (last update 25-Oct-2009)
Go back to the Excel tips page

A new feature of Microsoft Excel 2007 (with Microsoft Office Service Pack 2 installed) is the ability to create and mail Acrobat Reader PDF files. If you do not wish to install Microsoft Office SP2, you can install just the add-in. You can download it here :
2007 Microsoft Office Add-in: Microsoft Save as PDF

After the add-in is installed you can use the code below or do a manual Save As PDF.
Office Button >Save As ….PDF
Office Button >Send ….PDF

Tips / warnings :

1) If you have also installed Acrobat Reader you can change OpenAfterPublish in the code to
   True to open the PDF file after you create it.
2) The mail code example is not working with Outlook Express or Windows Mail.
3) If you set OpenAfterPublish in the code to True then you can do a manual send in
   Acrobat Reader (also with Outlook Express or Windows Mail).
4) If there is no printer installed the add-in will not work. You only have to install a printer driver
   of one of the printers in the default printer list, you not need a real printer to use the add-in.
5) When you use a hyperlink to another place in the workbook or if you use the Hyperlink
   worksheet function the hyperlinks are not working in the PDF.

Example code to Create and Mail PDF files

You can copy/paste the code from the TXT file to your own workbook or download the example workbook,
I suggest that you download the example file below.

Download a workbook with all the code to test the macros
Download PDF-Examples.zip

Or:

Copy the macros and functions from this TXT file in a Standard module of your workbook.
See this page if you not know how how to do this.
http://www.rondebruin.nl/code.htm

In the workbook you find three modules with eigth macros and three functions.
Note: You not have to edit or change the functions in the module named "FunctionsModule"

The macros to create a PDF in the "CreatePDF" module

Macro 1 : Create a PDF of the workbook
Macro 2 : Create a PDF of the ActiveSheet or selected sheets
Macro 3 : Create a PDF of the selection or range
Macro 4 : Create a PDF with every sheet with a specific sheet level name

Tip: In Macro 2 you can also use Sheets("Sheet3") instead of ActiveSheet in the code.
The worksheet not have to be active then to create the the PDF.

The first three macros calls a function named RDB_Create_PDF
Macro four calls a function named Create_PDF_Sheet_Level_Names

You see there are four arguments in both functions.

1: What do we want to publish (In macro four this is the sheet level name)
2: Path/File name of the PDF file, when you use "" you can enter a file name in the GetSaveAsFilename dialog
3: Overwrite the file you choose in the GetSaveAsFilename dialog if it already exist :True or False
4: OpenAfterPublish : True or False

In the macros in the workbook you see that we use one code line to create the PDF

This will create a PDF of the ActiveWorkbook and overwrite the file if it exist and open the PDF.
FileName = RDB_Create_PDF(ActiveWorkbook, "", True, True)

Note: For a fixed file name and overwrite it each time you run the macro use this
FileName = RDB_Create_PDF(ActiveWorkbook, "C:\Users\Ron\Test\YourPdfFile.pdf", True, True)

To create a PDF of the sheets with a sheet level named range "addtopdf" and
overwrite the file if it exist and open the PDF.
FileName = Create_PDF_Sheet_Level_Names("addtopdf", "", True, True)

Note: For a fixed file name and overwrite it each time you run the macro use this
FileName = Create_PDF_Sheet_Level_Names("addtopdf", "C:\Users\Ron\Test\YourPdfFile.pdf", True, True)

How do I add sheet level names to the sheets I want ?

1: Ctrl-F3 to open the Name manager
2: Click on New
3: Enter addtopdf as name
4: Change scope to the sheet you want
5: Refers to is not important because we only use the name
6: OK
7: Repeat the steps above for every sheet you want in the PDF.

The macros that create and mail the PDF in the "CreatePDFMail" module

Important : The code is only working if you use Outlook
Note : Example 5 in this module wil create a PDF of each sheet that have a mail address in A1
and Mail this sheet to the address in A1.

The macros in this module are almost the same as in the "CreatePDF" module
I replaced this two comments lines in the macros

```
'Ok, you find the PDF where you saved it
'You can call the mail macro here if you want
```

With this code line :

```
RDB_Mail_PDF_Outlook Filename, "ron@debruin.nl", "This is the subject", _
    "See the attached PDF file with the last figures" _
      & vbNewLine & vbNewLine & "Regards Ron de bruin", False

  'If you also want to delete the PDF file from your system after it create the mail add also this line:
  Kill FileName
```

You see there are five arguments in the function call.

1: FileName (do not change this)
2: To who do we want to send the mail ?
3: What is the subject ?
4: What do we want in the body of the mail ?
5: Do we want to Display (False) the mail or send it directly(True)

Also set the last argument to False of the code line that create the PDF so it not open the PDF.
FileName = RDB_Create_PDF(ActiveWorkbook, "", True, False)

Insert Outlook Signature in mail

Zip file or files with the default Windows zip program (VBA)

Ron de Bruin (last update 26-May-2009)
Go back to the Excel tips page
Information

Warning: The code below is not supported by Microsoft
It is not possible to hide the copy dialog when you copy to a zip folder (only working with normal folders).
Also there is no possibility to avoid that someone can cancel the CopyHere operation or that your VBA code will be notified that the operation has been cancelled.

Note: Do not Dim for example FileNameZip as String in the code examples.
This must be a Variant, if you change this the code will not work.

If you want to Unzip files see this page on my site.
http://www.rondebruin.nl/windowsxpunzip.htm

See also the the Zip (compress) section on my site for examples for 7-zip and WinZip.

Important

Before we can try one of the macro examples we must copy the following code in a normal module of your workbook. Every macro use the sub NewZip and the first example also use both functions.
See this page how : http://www.rondebruin.nl/code.htm

```
Sub NewZip(sPath)
'Create empty Zip File
'Changed by keepITcool Dec-12-2005
    If Len(Dir(sPath)) > 0 Then Kill sPath
    Open sPath For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
End Sub


Function bIsBookOpen(ByRef szBookName As String) As Boolean
' Rob Bovey
    On Error Resume Next
    bIsBookOpen = Not (Application.Workbooks(szBookName) Is Nothing)
End Function


Function Split97(sStr As Variant, sdelim As String) As Variant
'Tom Ogilvy
    Split97 = Evaluate("{""" & _
                    Application.Substitute(sStr, sdelim, """,""") & """}")
End Function
```

Examples

There are five examples on this page that you can copy in a normal module of your workbook.
Please read the information good above before you start testing the code below.

Browse to the folder you want and select the file or files

```
Sub Zip_File_Or_Files()
    Dim strDate As String, DefPath As String, sFName As String
    Dim oApp As Object, iCtr As Long, I As Integer
    Dim FName, vArr, FileNameZip

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
        DefPath = DefPath & "\"
    End If
```

```
    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    'Browse to the file(s), use the Ctrl key to select more files
    FName = Application.GetOpenFilename(filefilter:="Excel Files (*.xl*), *.xl*", _
                MultiSelect:=True, Title:="Select the files you want to zip")
    If IsArray(FName) = False Then
        'do nothing
    Else
        'Create empty Zip File
        NewZip (FileNameZip)
        Set oApp = CreateObject("Shell.Application")
        I = 0
        For iCtr = LBound(FName) To UBound(FName)
            vArr = Split97(FName(iCtr), "\")
            sFName = vArr(UBound(vArr))
            If bIsBookOpen(sFName) Then
                MsgBox "You can't zip a file that is open!" & vbLf & _
                      "Please close it and try again: " & FName(iCtr)
            Else
                'Copy the file to the compressed folder
                I = I + 1
                oApp.Namespace(FileNameZip).CopyHere FName(iCtr)

                'Keep script waiting until Compressing is done
                On Error Resume Next
                Do Until oApp.Namespace(FileNameZip).items.Count = I
                    Application.Wait (Now + TimeValue("0:00:01"))
                Loop
                On Error GoTo 0
            End If
        Next iCtr

        MsgBox "You find the zipfile here: " & FileNameZip
    End If
End Sub
```

Browse to a folder and zip all files in it

```
Sub Zip_All_Files_in_Folder_Browse()
    Dim FileNameZip, FolderName, oFolder
    Dim strDate As String, DefPath As String
```

```
    Dim oApp As Object

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
       DefPath = DefPath & "\"
    End If

    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    Set oApp = CreateObject("Shell.Application")

    'Browse to the folder
    Set oFolder = oApp.BrowseForFolder(0, "Select folder to Zip", 512)
    If Not oFolder Is Nothing Then
       'Create empty Zip File
       NewZip (FileNameZip)

       FolderName = oFolder.Self.Path
       If Right(FolderName, 1) <> "\" Then
          FolderName = FolderName & "\"
       End If

       'Copy the files to the compressed folder
       oApp.Namespace(FileNameZip).CopyHere oApp.Namespace(FolderName).items

       'Keep script waiting until Compressing is done
       On Error Resume Next
       Do Until oApp.Namespace(FileNameZip).items.Count = _
       oApp.Namespace(FolderName).items.Count
          Application.Wait (Now + TimeValue("0:00:01"))
       Loop
       On Error GoTo 0

       MsgBox "You find the zipfile here: " & FileNameZip

    End If
End Sub
```

Zip all files in the folder that you enter in the code

Note: Before you run the macro below change the folder in this macro line
FolderName = "C:\Users\Ron\test\"

```
Sub Zip_All_Files_in_Folder()
    Dim FileNameZip, FolderName
    Dim strDate As String, DefPath As String
    Dim oApp As Object

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
       DefPath = DefPath & "\"
    End If

    FolderName = "C:\Users\Ron\test\"    '<< Change

    strDate = Format(Now, " dd-mmm-yy h-mm-ss")
    FileNameZip = DefPath & "MyFilesZip " & strDate & ".zip"

    'Create empty Zip File
    NewZip (FileNameZip)

    Set oApp = CreateObject("Shell.Application")
    'Copy the files to the compressed folder
    oApp.Namespace(FileNameZip).CopyHere oApp.Namespace(FolderName).items

    'Keep script waiting until Compressing is done
    On Error Resume Next
    Do Until oApp.Namespace(FileNameZip).items.Count = _
       oApp.Namespace(FolderName).items.Count
        Application.Wait (Now + TimeValue("0:00:01"))
    Loop
    On Error GoTo 0

    MsgBox "You find the zipfile here: " & FileNameZip
End Sub
```

Zip the ActiveWorkbook

This sub will make a copy of the Activeworkbook and zip it in "C:\Users\Ron\test\" with a date-time stamp.
Change this folder or use your default path Application.DefaultFilePath

```
Sub Zip_ActiveWorkbook()
```

```
Dim strDate As String, DefPath As String
Dim FileNameZip, FileNameXls
Dim oApp As Object
Dim FileExtStr As String

DefPath = "C:\Users\Ron\test\"    '<< Change
If Right(DefPath, 1) <> "\" Then
   DefPath = DefPath & "\"
End If

'Create date/time string and the temporary xl* and Zip file name
If Val(Application.Version) < 12 Then
   FileExtStr = ".xls"
Else
   Select Case ActiveWorkbook.FileFormat
   Case 51: FileExtStr = ".xlsx"
   Case 52: FileExtStr = ".xlsm"
   Case 56: FileExtStr = ".xls"
   Case 50: FileExtStr = ".xlsb"
   Case Else: FileExtStr = "notknown"
   End Select
   If FileExtStr = "notknown" Then
      MsgBox "Sorry unknown file format"
      Exit Sub
   End If
End If

strDate = Format(Now, " yyyy-mm-dd h-mm-ss")

FileNameZip = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & ".zip"

FileNameXls = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & FileExtStr

If Dir(FileNameZip) = "" And Dir(FileNameXls) = "" Then

   'Make copy of the activeworkbook
   ActiveWorkbook.SaveCopyAs FileNameXls

   'Create empty Zip File
   NewZip (FileNameZip)

   'Copy the file in the compressed folder
```

```
        Set oApp = CreateObject("Shell.Application")
        oApp.Namespace(FileNameZip).CopyHere FileNameXls

        'Keep script waiting until Compressing is done
        On Error Resume Next
        Do Until oApp.Namespace(FileNameZip).items.Count = 1
           Application.Wait (Now + TimeValue("0:00:01"))
        Loop
        On Error GoTo 0
        'Delete the temporary xls file
        Kill FileNameXls

        MsgBox "Your Backup is saved here: " & FileNameZip

    Else
        MsgBox "FileNameZip or/and FileNameXls exist"

    End If
End Sub
```

Zip and mail the ActiveWorkbook

SIEHE AUCH KAPITEL DATEIZUGRIFFE - ZIPPEN / ENTZIPPEN

This will only work if you use Outlook as your mail program

This sub will send a newly created workbook (copy of the Activeworkbook).
It save and zip the workbook before mailing it with a date/time stamp.
After the zip file is sent the zip file and the workbook will be deleted from your hard disk.

```
Sub Zip_Mail_ActiveWorkbook()
    Dim strDate As String, DefPath As String, strbody As String
    Dim oApp As Object, OutApp As Object, OutMail As Object
    Dim FileNameZip, FileNameXls
    Dim FileExtStr As String

    DefPath = Application.DefaultFilePath
    If Right(DefPath, 1) <> "\" Then
        DefPath = DefPath & "\"
    End If
```

```
'Create date/time string and the temporary xl* and zip file name
If Val(Application.Version) < 12 Then
   FileExtStr = ".xls"
Else
   Select Case ActiveWorkbook.FileFormat
   Case 51: FileExtStr = ".xlsx"
   Case 52: FileExtStr = ".xlsm"
   Case 56: FileExtStr = ".xls"
   Case 50: FileExtStr = ".xlsb"
   Case Else: FileExtStr = "notknown"
   End Select
   If FileExtStr = "notknown" Then
      MsgBox "Sorry unknown file format"
      Exit Sub
   End If
End If

strDate = Format(Now, " yyyy-mm-dd h-mm-ss")

FileNameZip = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & ".zip"

FileNameXls = DefPath & Left(ActiveWorkbook.Name, _
Len(ActiveWorkbook.Name) - 4) & strDate & FileExtStr


If Dir(FileNameZip) = "" And Dir(FileNameXls) = "" Then

   'Make copy of the activeworkbook
   ActiveWorkbook.SaveCopyAs FileNameXls

   'Create empty Zip File
   NewZip (FileNameZip)

   'Copy the file in the compressed folder
   Set oApp = CreateObject("Shell.Application")
   oApp.Namespace(FileNameZip).CopyHere FileNameXls

   'Keep script waiting until Compressing is done
   On Error Resume Next
   Do Until oApp.Namespace(FileNameZip).items.Count = 1
      Application.Wait (Now + TimeValue("0:00:01"))
   Loop
   On Error GoTo 0
```

```
        'Create the mail
        Set OutApp = CreateObject("Outlook.Application")
        Set OutMail = OutApp.CreateItem(0)
        strbody = "Hi there" & vbNewLine & vbNewLine & _
                  "This is line 1" & vbNewLine & _
                  "This is line 2" & vbNewLine & _
                  "This is line 3" & vbNewLine & _
                  "This is line 4"

        On Error Resume Next
        With OutMail
            .To = "ron@debruin.nl"
            .CC = ""
            .BCC = ""
            .Subject = "This is the Subject line"
            .Body = strbody
            .Attachments.Add FileNameZip
            .Display   'or use .Send
        End With
        On Error GoTo 0

        'Delete the temporary Excel file and Zip file you send
        Kill FileNameZip
        Kill FileNameXls
    Else
        MsgBox "FileNameZip or/and FileNameXls exist"
    End If
End Sub
```

Insert Outlook Signature in mail

Ron de Bruin (last update 27-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Information

If you create a signature in Outlook it will save three files (HTM, TXT and RTF) into

```
SigString = "C:\Documents and Settings\" & Environ("username") & _
        "\Application Data\Microsoft\Signatures\Mysig.txt"
```

In Vista use this

```
SigString = "C:\Users\" & Environ("username") & _
        "\AppData\Roaming\Microsoft\Signatures\Mysig.txt"
```

Note: "Application Data" and "AppData" are hidden folders (Use Tools>Folder Options to change it)

In the two examples on this page we use the HTML and TXT file.
You must change the file name of the signature to your signature name in the code;
I use the name Mysig in the examples.

Important : This will not work if Word is your mail editor, you can turn that of in the options in Outlook

Example 1 : Add signature within an plain message
Example 2 : Add HTML signature within an HTML message

Both examples use this Function

```
Function GetBoiler(ByVal sFile As String) As String
'Dick Kusleika
    Dim fso As Object
```

```
    Dim ts As Object
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(sFile).OpenAsTextStream(1, -2)
    GetBoiler = ts.readall
    ts.Close
End Function
```

Example 1

This example add a signature to a mail with a small plain message.
Change the mail address and the name of the signature file in the code before you run it.

```
Sub Mail_Outlook_With_Signature_Plain()
' Don't forget to copy the function GetBoiler in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strbody As String
    Dim SigString As String
    Dim Signature As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strbody = "Hi there" & vbNewLine & vbNewLine & _
            "This is line 1" & vbNewLine & _
            "This is line 2" & vbNewLine & _
            "This is line 3" & vbNewLine & _
            "This is line 4"

    'Use the second SigString if you use Vista as operating system

    SigString = "C:\Documents and Settings\" & Environ("username") & _
              "\Application Data\Microsoft\Signatures\Mysig.txt"

    'SigString = "C:\Users\" & Environ("username") & _
     "\AppData\Roaming\Microsoft\Signatures\Mysig.txt"

    If Dir(SigString) <> "" Then
       Signature = GetBoiler(SigString)
    Else
       Signature = ""
```

```
    End If

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = strbody & vbNewLine & vbNewLine & Signature
        'You can add files also like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Example 2

This example add a html signature to a html mail.
Change the mail address and the name of the signature file in the code before you run it.

```
Sub Mail_Outlook_With_Signature_Html()
' Don't forget to copy the function GetBoiler in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strbody As String
    Dim SigString As String
    Dim Signature As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strbody = "<H3><B>Dear Customer</B></H3>" & _
            "Please visit this website to download the new version.<br>" & _
            "Let me know if you have problems.<br>" & _
            "<A HREF=""http://www.rondebruin.nl/tips.htm"">Ron's Excel Page</A>" & _
            "<br><br><B>Thank you</B>"
```

```
    'Use the second SigString if you use Vista as operating system

    SigString = "C:\Documents and Settings\" & Environ("username") & _
                "\Application Data\Microsoft\Signatures\Mysig.htm"

    'SigString = "C:\Users\" & Environ("username") & _
     "\AppData\Roaming\Microsoft\Signatures\Mysig.htm"

    If Dir(SigString) <> "" Then
       Signature = GetBoiler(SigString)
    Else
       Signature = ""
    End If

    On Error Resume Next
    With OutMail
       .To = "ron@debruin.nl"
       .CC = ""
       .BCC = ""
       .Subject = "This is the Subject line"
       .HTMLBody = strbody & "<br><br>" & Signature
       'You can add files also like this
       '.Attachments.Add ("C:\test.txt")
       .Send   'or use .Display
    End With

    On Error GoTo 0
    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

- OUTLOOK OBJECT MODEL - BODY -

Mail worksheet in the body of the mail

Ron de Bruin (last update 27-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends the whole ActiveSheet in the body of the mail without pictures.
Don't forget to copy the function RangetoHTML in the same module.
You only have to change the mail address in the macro before you can run the macro.

If you use Office 2002 - 2007 see this page for an example with pictures.
http://www.rondebruin.nl/mail/folder3/mailenvelope.htm

Important: Read also the information below the macro

```
Sub Mail_Sheet_Outlook_Body()
' Don't forget to copy the function RangetoHTML in the module.
' Working in Office 2000-2007
    Dim rng As Range
    Dim OutApp As Object
    Dim OutMail As Object
    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    Set rng = Nothing
    Set rng = ActiveSheet.UsedRange
    'You can also use a sheet name
    'Set rng = Sheets("YourSheet").UsedRange

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)
```

```
    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .HTMLBody = RangetoHTML(rng)
        .Send   'or use .Display
    End With
    On Error GoTo 0

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
Function RangetoHTML(rng As Range)
' Changed by Ron de Bruin 28-Oct-2006
' Working in Office 2000-2007
    Dim fso As Object
    Dim ts As Object
    Dim TempFile As String
    Dim TempWB As Workbook

    TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

    'Copy the range and create a new workbook to past the data in
    rng.Copy
    Set TempWB = Workbooks.Add(1)
    With TempWB.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
        .Cells(1).Select
        Application.CutCopyMode = False
        On Error Resume Next
        .DrawingObjects.Visible = True
        .DrawingObjects.Delete
        On Error GoTo 0
    End With
```

```
    'Publish the sheet to a htm file
    With TempWB.PublishObjects.Add( _
         SourceType:=xlSourceRange, _
         Filename:=TempFile, _
         Sheet:=TempWB.Sheets(1).Name, _
         Source:=TempWB.Sheets(1).UsedRange.Address, _
         HtmlType:=xlHtmlStatic)
        .Publish (True)
    End With

    'Read all data from the htm file into RangetoHTML
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)
    RangetoHTML = ts.ReadAll
    ts.Close
    RangetoHTML = Replace(RangetoHTML, "align=center x:publishsource=", _
                          "align=left x:publishsource=")

    'Close TempWB
    TempWB.Close savechanges:=False

    'Delete the htm file we used in this function
    Kill TempFile

    Set ts = Nothing
    Set fso = Nothing
    Set TempWB = Nothing
End Function
```

Tips

If you want to add a few text lines above the HTML body you can add this to the macro.
Note: This is not working if Word is your mail editor in Outlook 2000-2003, you can change this setting in Outlook: Tools>Options>…Mail Format tab

Add this Dim line

```
    Dim StrBody As String
```

Build the string you want to add

```
StrBody = "This is line 1" & "<br>" & _
        "This is line 2" & "<br>" & _
        "This is line 3" & "<br><br><br>"
```

Or use this for cell values

```
StrBody = Sheets("Sheet2").Range("A1").Value & "<br>" & _
        Sheets("Sheet2").Range("A2").Value & "<br>" & _
        Sheets("Sheet2").Range("A3").Value & "<br><br><br>"
```

And change the HTMLBody line to this

```
.HTMLBody = StrBody & RangetoHTML(rng)
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)

Mail Range or Selection in the body of the mail

Ron de Bruin (last update 27-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

The following subroutine sends the visible cells in the selection in the body of the mail without pictures.
Don't forget to copy the function RangetoHTML in the same module.
You only have to change the mail address in the macro and select a few cells before you can run the macro.

If you use Office 2002 - 2007 see this page for an example with pictures.
http://www.rondebruin.nl/mail/folder3/mailenvelope.htm

Important: Read also the information below the macro

```
Sub Mail_Selection_Range_Outlook_Body()
' Don't forget to copy the function RangetoHTML in the module.
' Working in Office 2000-2007
    Dim rng As Range
    Dim OutApp As Object
    Dim OutMail As Object
```

```
    Set rng = Nothing
    On Error Resume Next
    'Only the visible cells in the selection
    Set rng = Selection.SpecialCells(xlCellTypeVisible)
    'You can also use a range if you want
    'Set rng = Sheets("YourSheet").Range("D4:D12").SpecialCells(xlCellTypeVisible)
    On Error GoTo 0

    If rng Is Nothing Then
        MsgBox "The selection is not a range or the sheet is protected" & _
               vbNewLine & "please correct and try again.", vbOKOnly
        Exit Sub
    End If

    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .HTMLBody = RangetoHTML(rng)
        .Send   'or use .Display
    End With
    On Error GoTo 0

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
Function RangetoHTML(rng As Range)
```

```
' Changed by Ron de Bruin 28-Oct-2006
' Working in Office 2000-2007
    Dim fso As Object
    Dim ts As Object
    Dim TempFile As String
    Dim TempWB As Workbook

    TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

    'Copy the range and create a new workbook to past the data in
    rng.Copy
    Set TempWB = Workbooks.Add(1)
    With TempWB.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
        .Cells(1).Select
        Application.CutCopyMode = False
        On Error Resume Next
        .DrawingObjects.Visible = True
        .DrawingObjects.Delete
        On Error GoTo 0
    End With

    'Publish the sheet to a htm file
    With TempWB.PublishObjects.Add( _
         SourceType:=xlSourceRange, _
         Filename:=TempFile, _
         Sheet:=TempWB.Sheets(1).Name, _
         Source:=TempWB.Sheets(1).UsedRange.Address, _
         HtmlType:=xlHtmlStatic)
        .Publish (True)
    End With

    'Read all data from the htm file into RangetoHTML
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)
    RangetoHTML = ts.ReadAll
    ts.Close
    RangetoHTML = Replace(RangetoHTML, "align=center x:publishsource=", _
                          "align=left x:publishsource=")

    'Close TempWB
    TempWB.Close savechanges:=False
```

```
    'Delete the htm file we used in this function
    Kill TempFile

    Set ts = Nothing
    Set fso = Nothing
    Set TempWB = Nothing
End Function
```

Tips

If you want to add a few text lines above the HTML body you can add this to the macro.
Note: This is not working if Word is your mail editor in Outlook 2000-2003, you can change this setting in Outlook: Tools>Options>…Mail Format tab

Add this Dim line

```
    Dim StrBody As String
```

Build the string you want to add

```
    StrBody = "This is line 1" & "<br>" & _
              "This is line 2" & "<br>" & _
              "This is line 3" & "<br><br><br>"
```

Or use this for cell values

```
    StrBody = Sheets("Sheet2").Range("A1").Value & "<br>" & _
              Sheets("Sheet2").Range("A2").Value & "<br>" & _
              Sheets("Sheet2").Range("A3").Value & "<br><br><br>"
```

And change the HTMLBody line to this

```
        .HTMLBody = StrBody & RangetoHTML(rng)
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail selection, range or worksheet in the body of a mail with MailEnvelope

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail Index page
Manual

In Excel 2002-2007 you have a option in the User Interface to mail a selection or worksheet
in the body of the mail.

Excel 2002-2003 : You find this button next to the save icon in the Standard toolbar
or use File>Send to>mail recipient.

Excel 2007 : Microsoft hide it (It is not in the Office button>Send menu)

But you can add this option to the QAT (Quick Access Toolbar)

1) Office Button > Excel Options
2) Customize
3) Choose "Commands Not in the Ribbon" in the "Choose Commands from" list
4) Select the command "Send to Mail Recepient"
5) Add
6) OK

Note: it is a toggle button and if you look closely, you'll see that that icon is depressed

when you see the envelope.

Important:
It is not working if your Outlook version is newer then your Excel version.
For example Excel 2002 and Outlook 2003
Error <Excel could not start the E-mail program>

If you use Conditional Formatting Icons or Data bars in your Excel 2007 workbook the
receiver will not see them in the mail.

VBA code

Important: MailEnvelope code is not working if your Outlook version is newer then your Excel version.
The two links below are working if you are in that situation

If you use Conditional Formatting Icons or Data bars in your Excel 2007 workbook the
receiver will not see them in the mail.

I personally not like to use the VBA MailEnvelope code because it will use the Excel window instead of
creating a separate Outlook mail. When you use the code examples below you can use .Display instead of .Send in the code and it will create a separate Outlook mail and you can view it first if you want.

Mail worksheet in the body
http://www.rondebruin.nl/mail/folder3/mail2.htm

Mail Range or Selection in the body
http://www.rondebruin.nl/mail/folder3/mail4.htm

But the big advantage of MailEnvelope code is that if you have a logo or picture on your worksheet
it will also send them in the body of the mail, if you use the examples in the links above not.

Note : It is possible that Outlook must be your default mail program if you want to use the code.

Close Excel first and check out if Outlook is your mail program for Office.
Start>Settings>Control Panel....Internet options (Program Tab)
In Vista : Start>Default programs

Basic code examples

Selection or whole sheet

If the selection is one cell it will send the whole worksheet.
Read the information in the code.

```
Sub Send_Selection_Or_ActiveSheet_with_MailEnvelope()
    Dim Sendrng As Range

    On Error GoTo StopMacro
```

```
    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Note: if the selection is one cell it will send the whole worksheet
    Set Sendrng = Selection

    'Create the mail and send it
    With Sendrng

        ActiveWorkbook.EnvelopeVisible = True
        With .Parent.MailEnvelope

            ' Set the optional introduction field thats adds
            ' some header text to the email body.
            .Introduction = "This is a test mail."

            ' In the "With .Item" part you can add more options
            ' See the tips on this Outlook example page.
            ' http://www.rondebruin.nl/mail/tips2.htm
            With .Item
                .To = "ron@debruin.nl"
                .Subject = "My subject"
                .Send
            End With

        End With
    End With

StopMacro:
    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
    ActiveWorkbook.EnvelopeVisible = False

End Sub
```

Fixed range or worksheet

Note: The worksheet/range not have to be active when you run the code.

Read the information in the code.

```
Sub Send_Range_Or_Whole_Worksheet_with_MailEnvelope()
    Dim AWorksheet As Worksheet
    Dim Sendrng As Range
    Dim rng As Range

    On Error GoTo StopMacro

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Fill in the Worksheet/range you want to mail
    'Note: if you use one cell it will send the whole worksheet
    Set Sendrng = Worksheets("Sheet1").Range("A1:B15")

    'Remember the activesheet
    Set AWorksheet = ActiveSheet

    'Create the mail and send it
    With Sendrng

        ' Select the worksheet with the range you want to send
        .Parent.Select

        'Remember the ActiveCell on that worksheet
        Set rng = ActiveCell

        'Select the range you want to mail
        .Select

        ' Create the mail and send it
        ActiveWorkbook.EnvelopeVisible = True
        With .Parent.MailEnvelope

            ' Set the optional introduction field thats adds
            ' some header text to the email body.
            .Introduction = "This is a test mail."

            ' In the "With .Item" part you can add more options
            ' See the tips on this Outlook example page.
            ' http://www.rondebruin.nl/mail/tips2.htm
            With .Item
```

```
            .To = "ron@debruin.nl"
            .Subject = "My subject"
            .Send
        End With

    End With

    'select the original ActiveCell
    rng.Select
  End With

  'Activate the sheet that was active before you run the macro
  AWorksheet.Select

StopMacro:
  With Application
    .ScreenUpdating = True
    .EnableEvents = True
  End With
  ActiveWorkbook.EnvelopeVisible = False

End Sub
```

Mail a small message

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page

Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1 : send a small text message
Example 2 : send the text from a txt file in the body of the mail

Example 1

The following subroutine sends a small text in an e-mail message.
Change the mail address and subject in the macro before you run it.

```
Sub Mail_small_Text_Outlook()
```

```
'Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strbody As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strbody = "Hi there" & vbNewLine & vbNewLine & _
              "This is line 1" & vbNewLine & _
              "This is line 2" & vbNewLine & _
              "This is line 3" & vbNewLine & _
              "This is line 4"

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = strbody
        'You can add a file like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Example 2

The following subroutine sends the text from a txt file in the body of the mail.
Change the mail address, subject and path/name of the txt file in the macro before you run it.
I use C:\test.txt in this example. Note: this example also use the GetBoiler function, you can
find it below the macro. Copy this function also in your standard module.

```
Sub Mail_Text_From_Txtfile_Outlook()
'Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object

    Set OutApp = CreateObject("Outlook.Application")
```

```
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = GetBoiler("C:\test.txt")
        'You can add a file like this
        '.Attachments.Add ("C:\test.txt")
        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub


Function GetBoiler(ByVal sFile As String) As String
'Dick Kusleika
    Dim fso As Object
    Dim ts As Object
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(sFile).OpenAsTextStream(1, -2)
    GetBoiler = ts.readall
    ts.Close
End Function
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail a message to each person in a range

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page

Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

Make a list on the ActiveSheet with :
In column A : Names of the people
In column B : E-mail addresses
In column C : yes or no (if the value is yes it will create a mail)

The Macro will loop through each row on the Activesheet and if there is a E-mail address in column B
and "yes" in column C it will create a mail with a reminder like the one below for each person.
If you have duplicate addresses in the column check out this example.

Dear Jelle (Jelle is a name in column A for example)

Please contact us to discuss bringing your account up to date

```
Sub TestFile()
    Dim OutApp As Object
```

```
    Dim OutMail As Object
    Dim cell As Range

    Application.ScreenUpdating = False
    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon

    On Error GoTo cleanup
    For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
       If cell.Value Like "?*@?*.?*" And _
          LCase(Cells(cell.Row, "C").Value) = "yes" Then

          Set OutMail = OutApp.CreateItem(0)
          On Error Resume Next
          With OutMail
             .To = cell.Value
             .Subject = "Reminder"
             .Body = "Dear " & Cells(cell.Row, "A").Value _
                  & vbNewLine & vbNewLine & _
                    "Please contact us to discuss bringing " & _
                    "your account up to date"
             'You can add files also like this
             '.Attachments.Add ("C:\test.txt")
             .Send  'Or use Display
          End With
          On Error GoTo 0
          Set OutMail = Nothing
       End If
    Next cell

cleanup:
    Set OutApp = Nothing
    Application.ScreenUpdating = True
End Sub
```

Tips

You can also use the values of cells in a range as the body text.
This example will add all the text/values that are in the range G1:G20 to the body.

Add this code to the sub before the loop start

```
    Dim strbody As String
    For Each cell In Range("G1:G20")
       strbody = strbody & cell.Value & vbNewLine
```

```
    Next
```

And replace the body line with this one

```
    .Body = "Dear " & Cells(cell.Row, "A").Value & vbNewLine & vbNewLine & strbody
```

If you want to create emails that are formatted you can use HTMLBody (Office 2000 and up) instead of Body.
You can find a lot of WebPages on the internet with more HTML tags examples.

```
    .HTMLBody = "<H3><B>Dear " & cell.Offset(0, -1).Value & "</B></H3>" & _
        "Please contact us to discuss bringing your account up to date.<BR><BR>" & _
            "<B>Regards Ron de Bruin</B>"
```

Example 2

Make a list on the ActiveSheet with :
In column A : Names of the people
In column B : E-mail addresses
In column C :yes or no (if the value is yes it will create a mail)

Note: You can also use a formula in column C that display yes if a condition is true, this example will display "yes" if the date in column E is within 90 days of today's date. =IF((E1-TODAY())>90,"No","Yes")

In column D :The macro will add "send" in D when you send the mail
(next time you run the macro it will not mailed again)
In column E :Date that you want to check with the formula in C to decide if you want to send the mail or not

The Macro will loop through each row on the Activesheet and if there is a E-mail address in column B
and "yes" in column C and not "send" in column D it will create a mail with a reminder like the one below
for each person in the range

Dear Jelle (Jelle is a name in column A for example)

Please contact us to discuss bringing your account up to date

```
Sub TestFile_2()
    Dim OutApp As Object
    Dim OutMail As Object
    Dim cell As Range

    Application.ScreenUpdating = False
    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon

    On Error GoTo cleanup
    For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
        If cell.Value Like "?*@?*.?*" And _
```

```
        LCase(Cells(cell.Row, "C").Value) = "yes" _
        And LCase(Cells(cell.Row, "D").Value) <> "send" Then

         Set OutMail = OutApp.CreateItem(0)

         On Error Resume Next
         With OutMail
            .To = cell.Value
            .Subject = "Reminder"
            .Body = "Dear " & Cells(cell.Row, "A").Value _
            & vbNewLine & vbNewLine & _
                  "Please contact us to discuss bringing " & _
                  "your account up to date."
            'You can add files also like this
            '.Attachments.Add ("C:\test.txt")
            .Send  'Or use Display
         End With
         On Error GoTo 0
         Cells(cell.Row, "D").Value = "send"
         Set OutMail = Nothing
      End If
   Next cell

cleanup:
   Set OutApp = Nothing
   Application.ScreenUpdating = True
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

Dim OutApp As Object

```
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail every worksheet with address in A1 in the body

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

This procedure will mail every Worksheet with an address in cell A1. It does this by
cycling through each worksheet in the workbook and checking cell A1 for the @ character.
If found it will create a mail and send it to the address in cell A1.
Don't forget to copy the function RangetoHTML in the same module.

Be sure that your UsedRange is not to big on each worksheet.
See this site : http://www.contextures.on.ca/xlfaqApp.html#Unused

```
Sub Outlook_Mail_Every_Worksheet_Body()
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim ws As Worksheet

    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    Set OutApp = CreateObject("Outlook.Application")
```

```
    OutApp.Session.Logon

    For Each ws In ActiveWorkbook.Worksheets
        If ws.Range("A1").Value Like "?*@?*.?*" Then
            Set OutMail = OutApp.CreateItem(0)

            On Error Resume Next
            With OutMail
                .To = ws.Range("A1").Value
                .CC = ""
                .BCC = ""
                .Subject = "This is the Subject line"
                .HTMLBody = RangetoHTML(ws.UsedRange)
                'You can add a file like this
                '.Attachments.Add ("C:\test.txt")
                .Send    'or use .Display
            End With
            On Error GoTo 0

            Set OutMail = Nothing
        End If
    Next ws

    Set OutApp = Nothing
    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
Function RangetoHTML(rng As Range)
' Changed by Ron de Bruin 28-Oct-2006
' Working in Office 2000-2007
    Dim fso As Object
    Dim ts As Object
    Dim TempFile As String
    Dim TempWB As Workbook

    TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

    'Copy the range and create a new workbook to past the data in
    rng.Copy
    Set TempWB = Workbooks.Add(1)
    With TempWB.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
```

```
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
        .Cells(1).Select
        Application.CutCopyMode = False
        On Error Resume Next
        .DrawingObjects.Visible = True
        .DrawingObjects.Delete
        On Error GoTo 0
    End With

    'Publish the sheet to a htm file
    With TempWB.PublishObjects.Add( _
         SourceType:=xlSourceRange, _
         Filename:=TempFile, _
         Sheet:=TempWB.Sheets(1).Name, _
         Source:=TempWB.Sheets(1).UsedRange.Address, _
         HtmlType:=xlHtmlStatic)
        .Publish (True)
    End With

    'Read all data from the htm file into RangetoHTML
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)
    RangetoHTML = ts.ReadAll
    ts.Close
    RangetoHTML = Replace(RangetoHTML, "align=center x:publishsource=", _
                          "align=left x:publishsource=")

    'Close TempWB
    TempWB.Close savechanges:=False

    'Delete the htm file we used in this function
    Kill TempFile

    Set ts = Nothing
    Set fso = Nothing
    Set TempWB = Nothing
End Function
```

Tips

If you want to add a few text lines above the HTML body you can add this to the macro.
Note: This is not working if Word is your mail editor in Outlook 2000-2003, you can change this setting in Outlook: Tools>Options>…Mail Format tab

Add this Dim line

```
Dim StrBody As String
```

Build the string you want to add

```
StrBody = "This is line 1" & "<br>" & _
          "This is line 2" & "<br>" & _
          "This is line 3" & "<br><br><br>"
```

Or use this for cell values

```
StrBody = Sheets("Sheet2").Range("A1").Value & "<br>" & _
          Sheets("Sheet2").Range("A2").Value & "<br>" & _
          Sheets("Sheet2").Range("A3").Value & "<br><br><br>"
```

And change the HTMLBody line to this

```
.HTMLBody = StrBody & RangetoHTML(ws.UsedRange)
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Insert Outlook Signature in mail

Ron de Bruin (last update 27-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Information

If you create a signature in Outlook it will save three files (HTM, TXT and RTF) into

```
SigString = "C:\Documents and Settings\" & Environ("username") & _
        "\Application Data\Microsoft\Signatures\Mysig.txt"
```

In Vista use this

```
SigString = "C:\Users\" & Environ("username") & _
        "\AppData\Roaming\Microsoft\Signatures\Mysig.txt"
```

Note: "Application Data" and "AppData" are hidden folders (Use Tools>Folder Options to change it)

In the two examples on this page we use the HTML and TXT file.
You must change the file name of the signature to your signature name in the code;
I use the name Mysig in the examples.

Important : This will not work if Word is your mail editor, you can turn that of in the options in Outlook

Example 1 : Add signature within an plain message
Example 2 : Add HTML signature within an HTML message

Both examples use this Function

```
Function GetBoiler(ByVal sFile As String) As String
'Dick Kusleika
    Dim fso As Object
    Dim ts As Object
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(sFile).OpenAsTextStream(1, -2)
    GetBoiler = ts.readall
    ts.Close
End Function
```

Example 1

This example add a signature to a mail with a small plain message.
Change the mail address and the name of the signature file in the code before you run it.

```
Sub Mail_Outlook_With_Signature_Plain()
' Don't forget to copy the function GetBoiler in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strbody As String
    Dim SigString As String
    Dim Signature As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strbody = "Hi there" & vbNewLine & vbNewLine & _
            "This is line 1" & vbNewLine & _
            "This is line 2" & vbNewLine & _
            "This is line 3" & vbNewLine & _
            "This is line 4"
```

```
    'Use the second SigString if you use Vista as operating system

    SigString = "C:\Documents and Settings\" & Environ("username") & _
                "\Application Data\Microsoft\Signatures\Mysig.txt"

    'SigString = "C:\Users\" & Environ("username") & _
     "\AppData\Roaming\Microsoft\Signatures\Mysig.txt"

    If Dir(SigString) <> "" Then
       Signature = GetBoiler(SigString)
    Else
       Signature = ""
    End If

    On Error Resume Next
    With OutMail
       .To = "ron@debruin.nl"
       .CC = ""
       .BCC = ""
       .Subject = "This is the Subject line"
       .Body = strbody & vbNewLine & vbNewLine & Signature
       'You can add files also like this
       '.Attachments.Add ("C:\test.txt")
       .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Example 2

This example add a html signature to a html mail.
Change the mail address and the name of the signature file in the code before you run it.

```
Sub Mail_Outlook_With_Signature_Html()
' Don't forget to copy the function GetBoiler in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strbody As String
```

```
    Dim SigString As String
    Dim Signature As String

    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strbody = "<H3><B>Dear Customer</B></H3>" & _
              "Please visit this website to download the new version.<br>" & _
              "Let me know if you have problems.<br>" & _
              "<A HREF=""http://www.rondebruin.nl/tips.htm"">Ron's Excel Page</A>" & _
              "<br><br><B>Thank you</B>"

    'Use the second SigString if you use Vista as operating system

    SigString = "C:\Documents and Settings\" & Environ("username") & _
                "\Application Data\Microsoft\Signatures\Mysig.htm"

    'SigString = "C:\Users\" & Environ("username") & _
     "\AppData\Roaming\Microsoft\Signatures\Mysig.htm"

    If Dir(SigString) <> "" Then
       Signature = GetBoiler(SigString)
    Else
       Signature = ""
    End If

    On Error Resume Next
    With OutMail
       .To = "ron@debruin.nl"
       .CC = ""
       .BCC = ""
       .Subject = "This is the Subject line"
       .HTMLBody = strbody & "<br><br>" & Signature
       'You can add files also like this
       '.Attachments.Add ("C:\test.txt")
       .Send   'or use .Display
    End With

    On Error GoTo 0
    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation
http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail a row to each person in a range (HTML)

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In this example I use the range A1:J100
In column A : Names of the students
In column B : E-mail addresses
In column C : yes or no (if the value is yes it will create a mail)
In column D:J : Grades or other info for the student

How do I Change filter range and filter column? :
In this example I use the filter range A1:J100
You can change the filter range and filter column in this code line in the macro.

Ash.Range("A1:J100").AutoFilter Field:=2, Criteria1:=cell.Value
Field = 2 'Filter column = B because the filter range start in A

Tip : For testing I use .Display, change it to .Send if it is working OK.

Note: This example use the function RangetoHTML, copy this function together
with the macro in a Standard module of your workbook. You can find the
RangetoHTML function below the example macro on this page.

If you have more rows with information for a student see this example
http://www.rondebruin.nl/mail/folder3/row2.htm

```
Sub Send_Row()
' Don't forget to copy the function RangetoHTML in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
```

```
Dim OutMail As Object
Dim cell As Range
Dim rng As Range
Dim Ash As Worksheet

Set Ash = ActiveSheet
On Error GoTo cleanup
Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon

With Application
   .EnableEvents = False
   .ScreenUpdating = False
End With

For Each cell In Ash.Columns("B").Cells.SpecialCells(xlCellTypeConstants)
   If cell.Value Like "?*@?*.?*" _
     And LCase(cell.Offset(0, 1).Value) = "yes" Then

      'Change the filter range and filter Field if needed
      'It will filter on Column B now (mail addresses)
      Ash.Range("A1:J100").AutoFilter Field:=2, Criteria1:=cell.Value

      With Ash.AutoFilter.Range
         On Error Resume Next
         Set rng = .SpecialCells(xlCellTypeVisible)
         On Error GoTo 0
      End With

      Set OutMail = OutApp.CreateItem(0)

      On Error Resume Next
      With OutMail
         .To = cell.Value
         .Subject = "Grades Aug"
         .HTMLBody = RangetoHTML(rng)
         .Display  'Or use .Send
      End With
      On Error GoTo 0

      Set OutMail = Nothing
      Ash.AutoFilterMode = False
   End If
Next cell
```

```
cleanup:
    Set OutApp = Nothing
    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
Function RangetoHTML(rng As Range)
' Changed by Ron de Bruin 28-Oct-2006
' Working in Office 2000-2007
    Dim fso As Object
    Dim ts As Object
    Dim TempFile As String
    Dim TempWB As Workbook

    TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

    'Copy the range and create a new workbook to past the data in
    rng.Copy
    Set TempWB = Workbooks.Add(1)
    With TempWB.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
        .Cells(1).Select
        Application.CutCopyMode = False
        On Error Resume Next
        .DrawingObjects.Visible = True
        .DrawingObjects.Delete
        On Error GoTo 0
    End With

    'Publish the sheet to a htm file
    With TempWB.PublishObjects.Add( _
         SourceType:=xlSourceRange, _
         Filename:=TempFile, _
         Sheet:=TempWB.Sheets(1).Name, _
         Source:=TempWB.Sheets(1).UsedRange.Address, _
         HtmlType:=xlHtmlStatic)
        .Publish (True)
    End With

    'Read all data from the htm file into RangetoHTML
```

```
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)
    RangetoHTML = ts.ReadAll
    ts.Close
    RangetoHTML = Replace(RangetoHTML, "align=center x:publishsource=", _
                          "align=left x:publishsource=")

    'Close TempWB
    TempWB.Close savechanges:=False

    'Delete the htm file we used in this function
    Kill TempFile

    Set ts = Nothing
    Set fso = Nothing
    Set TempWB = Nothing
End Function
```

Tips

If you want to add a few text lines above the HTML body you can add this to the macro.
Note: This is not working if Word is your mail editor in Outlook 2000-2003, you can change this setting in Outlook: Tools>Options>…Mail Format tab

Add this Dim line

```
    Dim StrBody As String
```

Build the string you want to add

```
    StrBody = "This is line 1" & "<br>" & _
              "This is line 2" & "<br>" & _
              "This is line 3" & "<br><br><br>"
```

Or use this for cell values

```
    StrBody = Sheets("Sheet2").Range("A1").Value & "<br>" & _
              Sheets("Sheet2").Range("A2").Value & "<br>" & _
              Sheets("Sheet2").Range("A3").Value & "<br><br><br>"
```

And change the HTMLBody line to this

```
        .HTMLBody = StrBody & RangetoHTML(rng)
```

Early Binding

If you want to use the the Intellisense help showing you the properties and methods of the objects as you type you can use Early binding. (bit faster but have problems when you distribute your workbooks)

See Dick's site for a explanation

http://www.dicks-clicks.com/excel/olBinding.htm

Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Then replace this three lines in the code

```
Dim OutApp As Object
Dim OutMail As Object

Set OutMail = OutApp.CreateItem(0)
```

With this three

```
Dim OutApp As Outlook.Application
Dim OutMail As Outlook.MailItem

Set OutMail = OutApp.CreateItem(olMailItem)
```

Mail a row or rows to each person in a range (HTML)

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.
If you not use Outlook see the examples in the first section on my mail index page.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Check out this page for Tips If you want to change the code on this page.
http://www.rondebruin.nl/mail/tips2.htm

Example 1

Important :
1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code

4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have:

In column A : Names of the students or ?
In column B:H : Information about the student or ?

We filter the range A1:H? for every unique name in the name column (column A in this example)
For every unique name we create a new mail with only the data of that person and send it to the
mail address it find with the VLookup function in the worksheet "Mailinfo".

Important: You must create this worksheet manual and add the names and mail addresses one time.
Add a worksheet to your workbook with the name "Mailinfo" with in column A the names
and in column B the mail addresses of every possible person in your Name column..

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 1 'Filter column = A because the filter range start in A

Tip : For testing I use .Display, change it to .Send if it is working OK.

Note: This example use the function RangetoHTML, copy this function together
with the macro in a Standard module of your workbook. You can find the
RangetoHTML function below the second example on this page.

```
Sub Send_Row_Or_Rows_1()
' Don't forget to copy the function RangetoHTML in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
    Dim Rnum As Long
    Dim FilterRange As Range
    Dim FieldNum As Integer
    Dim mailAddress As String
```

```
On Error GoTo cleanup
Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon

With Application
   .EnableEvents = False
   .ScreenUpdating = False
End With

'Set filter sheet, you can also use Sheets("MySheet")
Set Ash = ActiveSheet

'Set filter range and filter column (Column with names)
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 1    'Filter column = A because the filter range start in A

'Add a worksheet for the unique list and copy the unique list in A1
Set Cws = Worksheets.Add
FilterRange.Columns(FieldNum).AdvancedFilter _
       Action:=xlFilterCopy, _
       CopyToRange:=Cws.Range("A1"), _
       CriteriaRange:="", Unique:=True

'Count of the unique values + the header cell
Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

'If there are unique values start the loop
If Rcount >= 2 Then
   For Rnum = 2 To Rcount

       'Filter the FilterRange on the FieldNum column
       FilterRange.AutoFilter Field:=FieldNum, _
                         Criteria1:=Cws.Cells(Rnum, 1).Value

       'Look for the mail address in the MailInfo worksheet
       mailAddress = ""
       On Error Resume Next
       mailAddress = Application.WorksheetFunction. _
                 VLookup(Cws.Cells(Rnum, 1).Value, _
                     Worksheets("Mailinfo").Range("A1:B" & _
                     Worksheets("Mailinfo").Rows.Count), 2, False)
       On Error GoTo 0
```

```
            If mailAddress <> "" Then
                With Ash.AutoFilter.Range
                    On Error Resume Next
                    Set rng = .SpecialCells(xlCellTypeVisible)
                    On Error GoTo 0
                End With

                Set OutMail = OutApp.CreateItem(0)

                On Error Resume Next
                With OutMail
                    .to = mailAddress
                    .Subject = "Test mail"
                    .HTMLBody = RangetoHTML(rng)
                    .Display  'Or use Send
                End With
                On Error GoTo 0

                Set OutMail = Nothing
            End If

            'Close AutoFilter
            Ash.AutoFilterMode = False

        Next Rnum
    End If

cleanup:
    Set OutApp = Nothing
    Application.DisplayAlerts = False
    Cws.Delete
    Application.DisplayAlerts = True

    With Application
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
```

Example 2

Important :

1) The code is not working if your data is a List(Excel 2003) or Table(Excel2007)
2) The first row in the range must have Headers
3) Turn off AutoFilter before you use the code
4) Be sure that the sheet with the data is the active worksheet

In your worksheet you must have:

In column A : Names of the students or ?
In column B : E-mail addresses
In column C:H : Information about the student or ?

Note: Every row must have a mail address in column B

We filter the range A1:H? for every unique mail address in column B.
For every unique mail address we create a new mail with only the records
with that mail address and send it to that mail address.

How do I Change filter range and filter column? :
In this example I use the filter range A1:H? (we use all the rows on the sheet)
You can change the filter range and filter column in this two code lines in the macro.

Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 2 'Filter column = B because the filter range start in A

Tip : For testing I use .Display, change it to .Send if it is working OK.

Note: This example use the function RangetoHTML, copy this function together
with the macro in a Standard module of your workbook. You can find the
RangetoHTML function below the second example on this page.

```
Sub Send_Row_Or_Rows_2()
' Don't forget to copy the function RangetoHTML in the module.
' Working in Office 2000-2007
    Dim OutApp As Object
    Dim OutMail As Object
    Dim rng As Range
    Dim Ash As Worksheet
    Dim Cws As Worksheet
    Dim Rcount As Long
    Dim Rnum As Long
    Dim FilterRange As Range
    Dim FieldNum As Integer
```

```
On Error GoTo cleanup
Set OutApp = CreateObject("Outlook.Application")
OutApp.Session.Logon

With Application
   .EnableEvents = False
   .ScreenUpdating = False
End With

'Set filter sheet, you can also use Sheets("MySheet")
Set Ash = ActiveSheet

'Set filter range and filter column (column with e-mail addresses)
Set FilterRange = Ash.Range("A1:H" & Ash.Rows.Count)
FieldNum = 2    'Filter column = B because the filter range start in column A

'Add a worksheet for the unique list and copy the unique list in A1
Set Cws = Worksheets.Add
FilterRange.Columns(FieldNum).AdvancedFilter _
       Action:=xlFilterCopy, _
       CopyToRange:=Cws.Range("A1"), _
       CriteriaRange:="", Unique:=True

'Count of the unique values + the header cell
Rcount = Application.WorksheetFunction.CountA(Cws.Columns(1))

'If there are unique values start the loop
If Rcount >= 2 Then
   For Rnum = 2 To Rcount

      'Filter the FilterRange on the FieldNum column
      FilterRange.AutoFilter Field:=FieldNum, _
                        Criteria1:=Cws.Cells(Rnum, 1).Value

      'If the unique value is a mail addres create a mail
      If Cws.Cells(Rnum, 1).Value Like "?*@?*.?*" Then

         With Ash.AutoFilter.Range
            On Error Resume Next
            Set rng = .SpecialCells(xlCellTypeVisible)
            On Error GoTo 0
         End With
```

```
            Set OutMail = OutApp.CreateItem(0)

            On Error Resume Next
            With OutMail
                .to = Cws.Cells(Rnum, 1).Value
                .Subject = "Test mail"
                .HTMLBody = RangetoHTML(rng)
                .Display  'Or use Send
            End With
            On Error GoTo 0

            Set OutMail = Nothing
        End If

        'Close AutoFilter
        Ash.AutoFilterMode = False

    Next Rnum
  End If

cleanup:
  Set OutApp = Nothing
  Application.DisplayAlerts = False
  Cws.Delete
  Application.DisplayAlerts = True

  With Application
    .EnableEvents = True
    .ScreenUpdating = True
  End With
End Sub
```

Tips

If you want to add a few text lines above the HTML body you can add this to the macro.
Note: This is not working if Word is your mail editor in Outlook 2000-2003, you can change this setting in Outlook: Tools>Options>…Mail Format tab

Add this Dim line

```
  Dim StrBody As String
```

Build the string you want to add

```
  StrBody = "This is line 1" & "<br>" & _
```

```
        "This is line 2" & "<br>" & _
        "This is line 3" & "<br><br><br>"
```

Or use this for cell values

```
    StrBody = Sheets("Sheet2").Range("A1").Value & "<br>" & _
        Sheets("Sheet2").Range("A2").Value & "<br>" & _
        Sheets("Sheet2").Range("A3").Value & "<br><br><br>"
```

And change the HTMLBody line to this

```
    .HTMLBody = StrBody & RangetoHTML(rng)
```

The RangetoHTML function

Copy this function also in a standard module in your workbook.

```
Function RangetoHTML(rng As Range)
' Changed by Ron de Bruin 28-Oct-2006
' Working in Office 2000-2007
    Dim fso As Object
    Dim ts As Object
    Dim TempFile As String
    Dim TempWB As Workbook

    TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

    'Copy the range and create a new workbook to past the data in
    rng.Copy
    Set TempWB = Workbooks.Add(1)
    With TempWB.Sheets(1)
        .Cells(1).PasteSpecial Paste:=8
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
        .Cells(1).Select
        Application.CutCopyMode = False
        On Error Resume Next
        .DrawingObjects.Visible = True
        .DrawingObjects.Delete
        On Error GoTo 0
    End With

    'Publish the sheet to a htm file
    With TempWB.PublishObjects.Add( _
        SourceType:=xlSourceRange, _
```

```
        Filename:=TempFile, _
        Sheet:=TempWB.Sheets(1).Name, _
        Source:=TempWB.Sheets(1).UsedRange.Address, _
        HtmlType:=xlHtmlStatic)
       .Publish (True)
    End With

    'Read all data from the htm file into RangetoHTML
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)
    RangetoHTML = ts.ReadAll
    ts.Close
    RangetoHTML = Replace(RangetoHTML, "align=center x:publishsource=", _
                          "align=left x:publishsource=")

    'Close TempWB
    TempWB.Close savechanges:=False

    'Delete the htm file we used in this function
    Kill TempFile
    Set ts = Nothing
    Set fso = Nothing
    Set TempWB = Nothing
End Function
```

- OTHER TIPS -

Use the Account you want in mail macro in Excel/Outlook 2007

Ron de Bruin (last update 26-Jan-2009)
Go back to the Mail index page
Important read this :

The code on this page is only working with Outlook 2007 and not with Outlook Express or Windows Mail.
Copy the code in a Standard module of your workbook, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Then Add a reference to the Microsoft outlook Library

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook ? Object Library
   ? is the Excel version number

Example

If you want to mail from another account then your default mail account in Outlook 2007 then you can use SendUsingAccount, this is new in Outlook 2007.

First we must know the account number that we want to use in the macro.
Run the macro below so you know the number that you must use in the mail macro.

```
Sub Which_Account_Number()
   Dim OutApp As Outlook.Application
   Dim I As Long

   Set OutApp = CreateObject("Outlook.Application")

   For I = 1 To OutApp.Session.Accounts.Count
      MsgBox OutApp.Session.Accounts.Item(I) & " : This is account number " & I
   Next I

End Sub
```

Now you know the number of the account that you want to use in your mail macro.
The following test subroutine sends a small text in an e-mail message.
Change the mail address and the Item number "Item(1)" in the macro before you run it.

```
Sub Mail_small_Text_Change_Account()
'Is only working in Office 2007
'You must add a reference to the Microsoft outlook Library
   Dim OutApp As Outlook.Application
   Dim OutMail As Outlook.MailItem
   Dim strbody As String

   Set OutApp = CreateObject("Outlook.Application")
   OutApp.Session.Logon
   Set OutMail = OutApp.CreateItem(olMailItem)
```

```
    strbody = "Hi there" & vbNewLine & vbNewLine & _
            "This is line 1" & vbNewLine & _
            "This is line 2" & vbNewLine & _
            "This is line 3" & vbNewLine & _
            "This is line 4"

    On Error Resume Next
    With OutMail
        .To = "ron@debruin.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = strbody

        'SendUsingAccount is new in Outlook 2007
        'Change Item(1)to another number to use another account
        .SendUsingAccount = OutApp.Session.Accounts.Item(1)

        .Send   'or use .Display
    End With
    On Error GoTo 0

    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

If it is working Ok for you can use it in all the Outlook examples on my mail page.
http://www.rondebruin.nl/sendmail.htm

If you have problems with this example let me know

Save E-mail attachments to folder (For Outlook)

Ron de Bruin (last update 28-Jan-2009)
Go back to the Mail index page
Important read this :

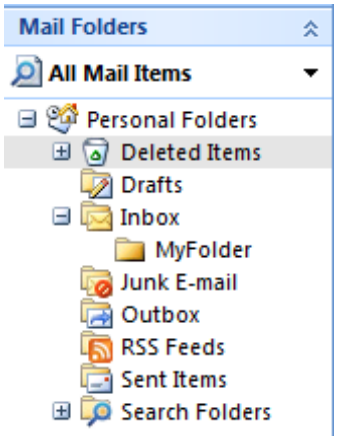
The code on this page is only working with Outlook and not with Outlook Express or Windows Mail.

Copy the code in a Standard module, if you just started with VBA see this page.
http://www.rondebruin.nl/code.htm

Example

If you receive a lot of mail with attachments and you want to save the files in a folder
on your hard disk then you can use the code on this page to save the files in the folder you want.
After you save the attachments see this page if you want to merge data from the files.
http://www.rondebruin.nl/copy3.htm

First right click on the Inbox and choose New Folder, in the code example I use the name “MyFolder”.



Tip: Create a mail rule in Outlook (Tools>Rules and Alerts) and move the mail from ? or with the
subject ? to the folder in your Inbox named “MyFolder” when the mail arrived.
You can also move the files from your Inbox to the folder “MyFolder” manual.

To test the code you can copy a few mails from your Inbox in the new folder.

There are two macros on this page but we only run the macro named Test with one code line.
There are three arguments in this macro

Arg 1 = Folder name in your Inbox
Arg 2 = File extension, "" is every file
Arg 3 = Save folder, "C:\Users\Ron\test" or ""
If you use "" it will create a date/time stamped folder for you in your "Documents" folder
Note: If you use this "C:\Users\Ron\test" the folder must exist.

This will copy all files from “MyFolder” to a new folder in My Documents (Documents in Vista)
SaveEmailAttachmentsToFolder "MyFolder", "", ""

This will copy all xls files from “MyFolder” to "C:\Users\Ron\test"

SaveEmailAttachmentsToFolder "MyFolder", "xls", "C:\Users\Ron\test"

This will copy all xlsx files from “MyFolder” to "C:\Users\Ron\test"
SaveEmailAttachmentsToFolder "MyFolder", "xlsx", "C:\Users\Ron\test"

The code

Set a reference to Outlook and copy/paste the code in a standard module

1) Go to the VBA editor, Alt -F11
2) Tools>References in the Menu bar
3) Place a Checkmark before Microsoft Outlook **?** Object Library
   ? is the Excel version number
4) Insert>Module
5) Paste the code in this module
6) Alt q to close the editor
7) Save the file

```
Sub Test()
'Arg 1 = Folder name in your Inbox
'Arg 2 = File extension, "" is every file
'Arg 3 = Save folder, "C:\Users\Ron\test" or ""
'If you use "" it will create a date/time stamped
'folder for you in the "My Documents" folder.
'Note: If you use this "C:\Users\Ron\test" the folder must exist

    SaveEmailAttachmentsToFolder "MyFolder", "xls", ""

End Sub
```

Do not change code in the macro below

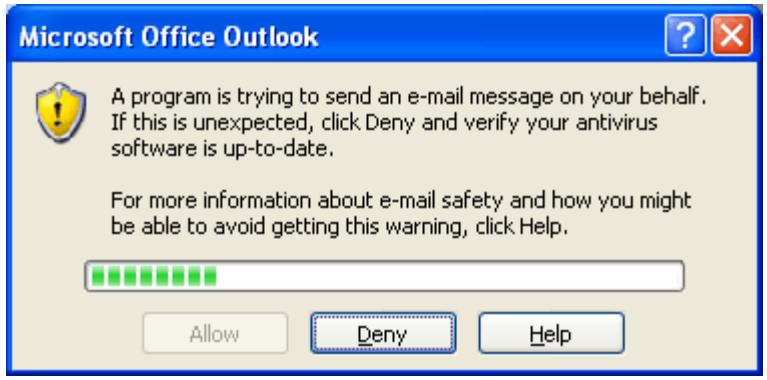
```
Sub SaveEmailAttachmentsToFolder(OutlookFolderInInbox As String, _
                                 ExtString As String, DestFolder As String)
    Dim ns As Namespace
    Dim Inbox As MAPIFolder
    Dim SubFolder As MAPIFolder
    Dim Item As Object
    Dim Atmt As Attachment
    Dim FileName As String
    Dim MyDocPath As String
    Dim I As Integer
    Dim wsh As Object
    Dim fs As Object

    On Error GoTo ThisMacro_err

    Set ns = GetNamespace("MAPI")
```

```
Set Inbox = ns.GetDefaultFolder(olFolderInbox)
Set SubFolder = Inbox.Folders(OutlookFolderInInbox)

I = 0
' Check subfolder for messages and exit of none found
If SubFolder.Items.Count = 0 Then
   MsgBox "There are no messages in this folder : " & OutlookFolderInInbox, _
         vbInformation, "Nothing Found"
   Set SubFolder = Nothing
   Set Inbox = Nothing
   Set ns = Nothing
   Exit Sub
End If

'Create DestFolder if DestFolder = ""
If DestFolder = "" Then
   Set wsh = CreateObject("WScript.Shell")
   Set fs = CreateObject("Scripting.FileSystemObject")
   MyDocPath = wsh.SpecialFolders.Item("mydocuments")
   DestFolder = MyDocPath & "\" & Format(Now, "dd-mmm-yyyy hh-mm-ss")
   If Not fs.FolderExists(DestFolder) Then
      fs.CreateFolder DestFolder
   End If
End If

If Right(DestFolder, 1) <> "\" Then
   DestFolder = DestFolder & "\"
End If

' Check each message for attachments and extensions
For Each Item In SubFolder.Items
   For Each Atmt In Item.Attachments
      If LCase(Right(Atmt.FileName, Len(ExtString))) = LCase(ExtString) Then
         FileName = DestFolder & Item.SenderName & " " & Atmt.FileName
         Atmt.SaveAsFile FileName
         I = I + 1
      End If
   Next Atmt
Next Item

' Show this message when Finished
If I > 0 Then
   MsgBox "You can find the files here : " _
       & DestFolder, vbInformation, "Finished!"
```

```
    Else
        MsgBox "No attached files in your mail.", vbInformation, "Finished!"
    End If

    ' Clear memory
ThisMacro_exit:
    Set SubFolder = Nothing
    Set Inbox = Nothing
    Set ns = Nothing
    Set fs = Nothing
    Set wsh = Nothing
    Exit Sub

    ' Error information
ThisMacro_err:
    MsgBox "An unexpected error has occurred." _
        & vbCrLf & "Please note and report the following information." _
        & vbCrLf & "Macro Name: SaveEmailAttachmentsToFolder" _
        & vbCrLf & "Error Number: " & Err.Number _
        & vbCrLf & "Error Description: " & Err.Description _
        , vbCritical, "Error!"
    Resume ThisMacro_exit

End Sub
```

Warnungsfenster umgehen (Prevent displaying the dialog to Send or not Send)

Ron de Bruin (last update 22-June-2008)
Go to the Mail tips page
When you use the code examples from my mail page it is possible you get this security warning



If you use Outlook Express or Windows Mail as your default mail program you can turn off this warning when you use the code examples in the first section on my mail page.

1: Start Outlook Express or Windows Mail, and then on the Tools menu, click Options.
2: Click the Security tab, and then click to remove the check mark from the
"Warn me when other applications try to send mail as me" check box.
3: Click OK to close the Options dialog box.

But if you use Outlook it is not so easy.

If you use Outlook 2007 you not see this warning with a correct installed system.
http://msdn2.microsoft.com/en-us/library/ms778202.aspx

But if you use Outlook 2000-2003 you must use a third party program to avoid the warning.

Express ClickYes
http://www.contextmagic.com/express-clickyes/

Outlook Redemption

http://www.dimastr.com/redemption/

Or try to use this SendKeys example if you use one of the Outlook object model examples
from my mail page. Instead of .Send in the code examples you can use this three lines.

```
.Display
Application.Wait (Now + TimeValue("0:00:02"))
Application.SendKeys "%S"
```

Note: SendKeys is not always reliable and this will not work on every computer
The S is from Send, if you not use a English version you must change this letter.
This tip is not working for the SendMail examples in the first section on my mail page.

Use CDO

Tip (my favorite way to send mail)

A much better option is to check out this page and see if CDO is working on
your machine. Working with every mail program (It will not use a mail program)
There are no security warnings when you use CDO to send mail.
http://www.rondebruin.nl/cdo.htm

Problems with sending mail from Excel

Ron de Bruin (last update 22-June-2008)
Go back to the mail tips page

Excel can't find or use the wrong mail program

Close Excel first and check out if Outlook or Outlook Express/Windows Mail is your mail program for Office.

Start>Settings>Control Panel....Internet options (Program Tab)
In Vista : Start>Default programs

Controls to send mail are missing or disabled or there are errors when you try to mail.

Excel 2007
When you get a "General mail failure" error or have other problems then first run Office Diagnostics.
Office Button>Excel Options...Resources

The "E-mail" command is missing or is unavailable, see
http://support.microsoft.com/kb/918792/en-us

Where is the Send to Mail Recipient option in Excel 2007 ?
You can add this option to the QAT like this:

1) Office Button > Excel Options
2) Customize
3) Choose "Commands Not in the Ribbon" in the "Choose Commands from" list
4) Select the command named "Send to mail recipient"
5) Add
6) OK

Excel 2003
For Office 2003 see this Kb first to check if you registry values are correct.
http://support.microsoft.com/kb/834008
Office 2003 only uses the registry for MAPI information.

Excel 97-2002
This behavior may occur when the following entry is missing from the [Mail] section of
the Win.ini system file: MAPIX=1

How do I add the MAPIX=1 entry to the Win.ini file ?

1) Click Start>Run.
2) In the Open box, type Sysedit, and then click OK.
3) In System Configuration Editor, click the title bar of the drive:\WINDOWS\WIN.INI window.
4) Under [Mail], type a new line that reads MAPIX=1, and then press ENTER.
5) On the File menu, click Exit and save the win.ini file.
6) Open Excel and see if it is working now.

Excel 2000-2002 in Vista
In Vista it is not possible to edit/save the win.ini file in the sysedit window because of the security settings.
Browse to C:\Windows and copy the win.ini file on your desktop and edit/save it there and copy it back to C:\Windows

If your register/win.ini file is OK and controls to send mail in Excel are still missing or disabled then try to rename your .xlb file.
Reason: Maybe you have a corrupt or bloated xlb file *normal* size is < 30 kb.
The .xlb file has all Toolbar customization in it.

1) Close Excel
2) Do a search for .xlb in Windows (Use: search hidden files and folders)
3) Rename or delete the .xlb file or files (In 2002 the name = Excel10.xlb)
4) Start Excel

Deleting the file or renaming will do no harm to your system
Excel will create a new file for you. (You lost your customization remember that)
If you make your own toolbars or add buttons to the others
this file is important (backup it so you can restore it)

Excel could not start the E-mail program error

Error <Excel could not start the E-mail program>
http://support.microsoft.com/kb/828509

If you use for example Excel 2000 or 2002.
And use File>Send To...Mail recipient or use the E-Mail button on the Standard toolbar
and have Outlook 2003 as your default mail program then Excel will tell you
<Excel could not start the E-mail program>

The Outlook version and the Office version must be the same to send in the body of the mail.

Note: as a workaround you can use my Body Examples

http://www.rondebruin.nl/mail/folder3/mail2.htm
Or
http://www.rondebruin.nl/mail/folder3/mail4.htm

Or use my SendMail add-in for Outlook
http://www.rondebruin.nl/mail/add-in.htm

Other problems

Search in the MS Knowledge Base for solutions of other problems (Click here)

Convert Excel data to Outlook Contacts

Ron de Bruin (last update 22-Jan-2009)
Go back to the mail tips page
For example your data in Excel looks like this:

	A	B	C
1	**FullName**	**GivenName**	**Mail address**
2	Estefan, Andrew	Andrew	andrew@test.com
3	Mcclure, Jeff	Jeff	jeff@test.com
4	Watermasysk, Peter	Peter	peter@test.com
5	Kimmel, Denise	Denise	denise@test.com
6			

You can add more columns with information but I use three columns in my example.

Import data to Outlook contacts

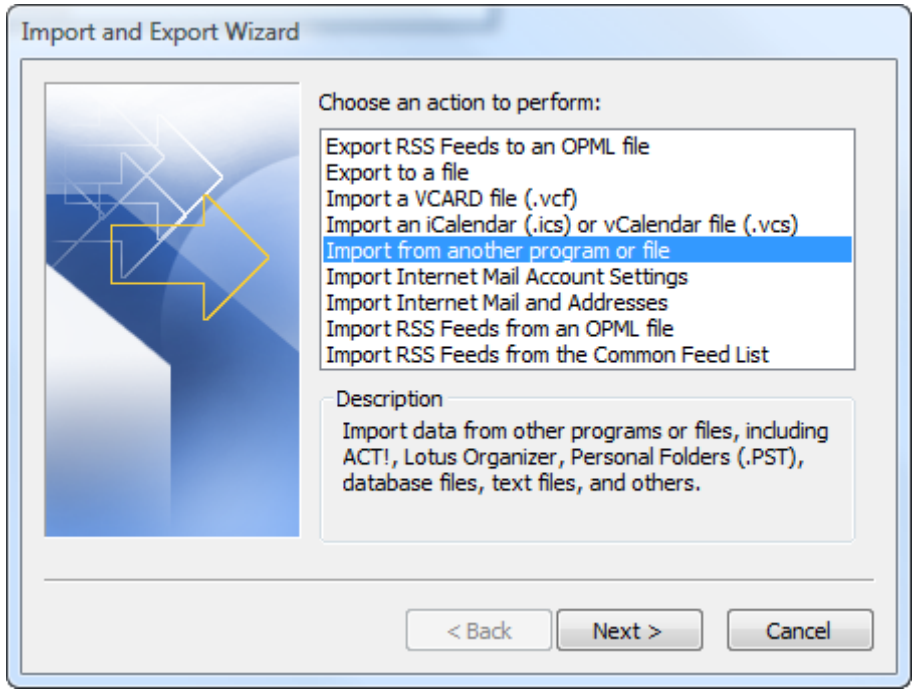
Read this first:
In the example below I show you how to import from a Excel file but it is also possible to save your file as CSV (comma delimited). There is no need to give the range a name if you do that. Select "Comma Separated Values (Windows)" then in the second screenshot below.

How do I Import data ?

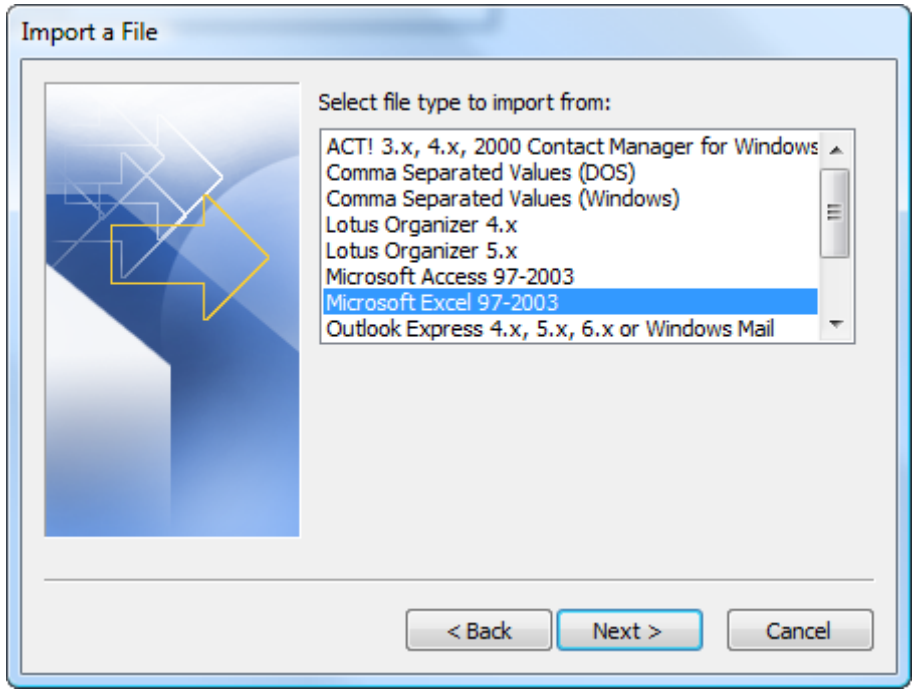
Outlook can only import data from an Excel file from a named range so we select A1:C5 in this example (with the header row) and use the shortcut Ctrl F3 to open the Name dialog to give the range a name. I use the name "ContactsData" . Then we save this file and close it.

Note: Outlook 2007 cannot import Excel 2007 files, save the file in 2007 as Excel 97-2003 file.

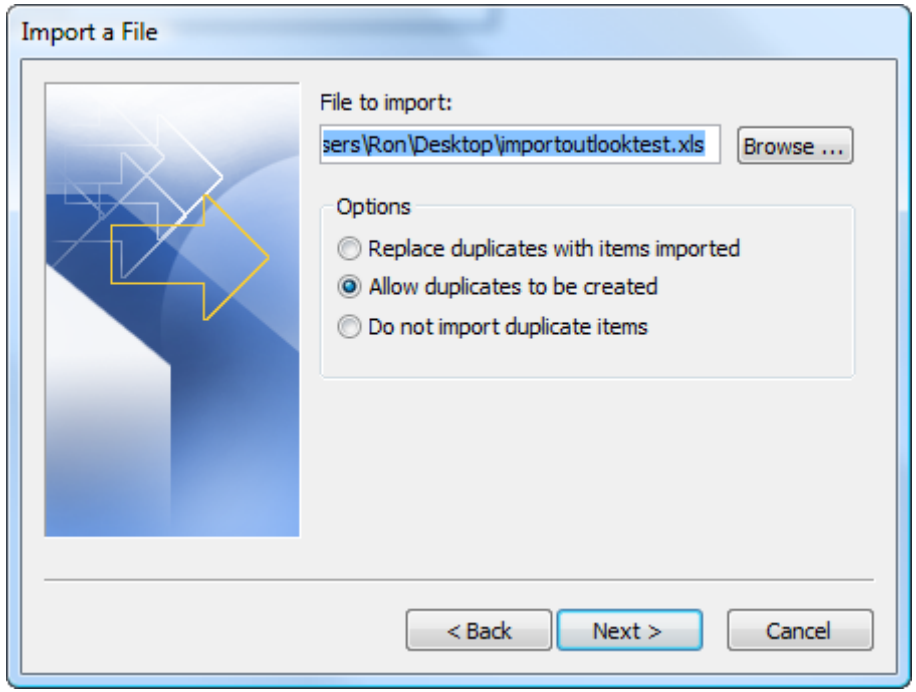
Open Outlook and go to File>Import and Export
(Screenshots are from Outlook 2007 but almost the same in Outlook 2003)



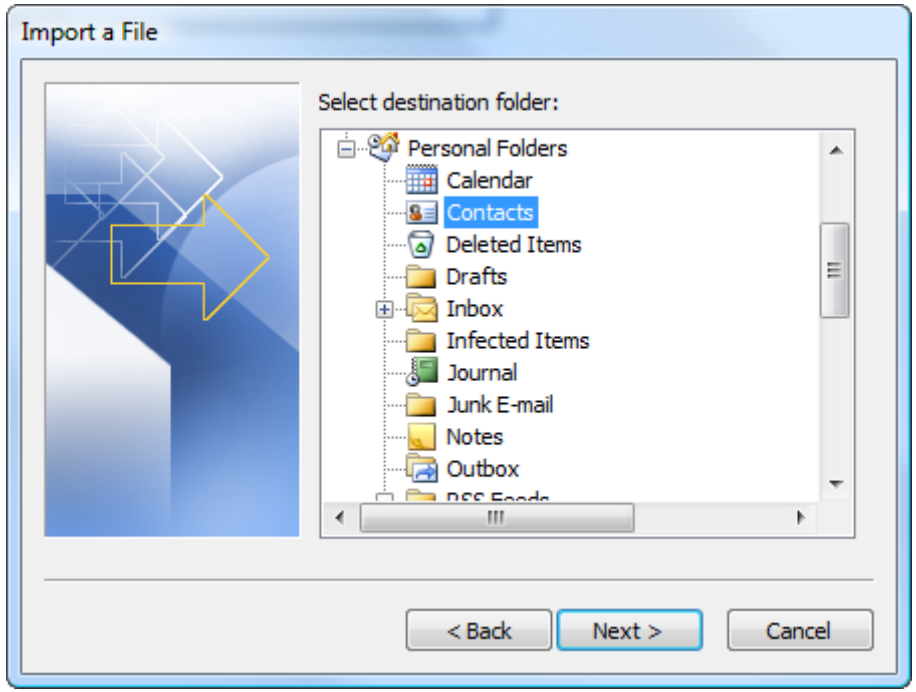
Select "Import from another program or file" and press "Next >"



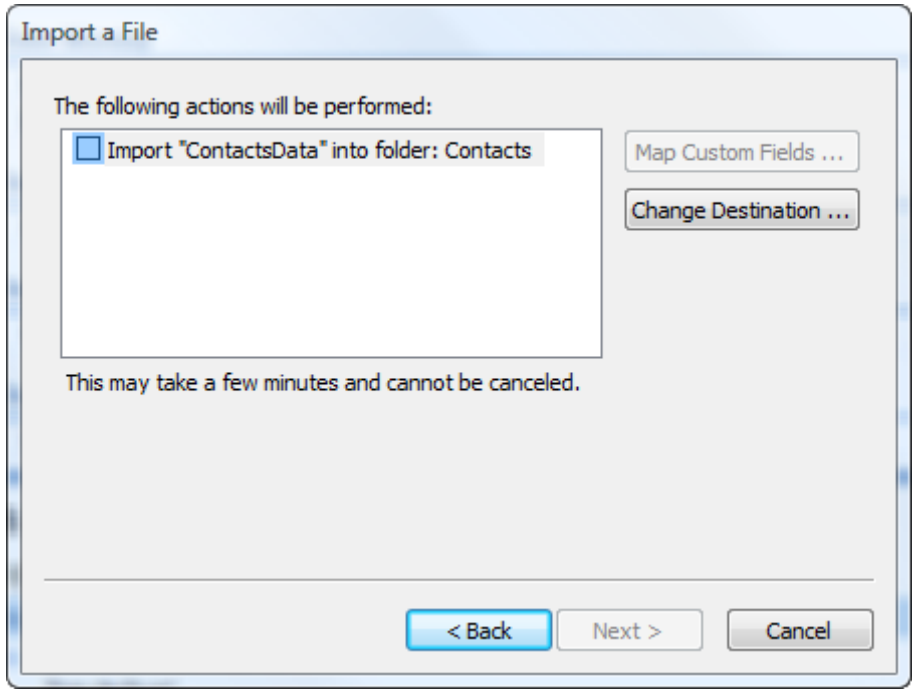
Select "Microsoft Excel 97-2003" or "Microsoft Excel" and press "Next >"
Note: Outlook 2007 cannot import Excel 2007 files, save the file in 2007 as Excel 97-2003 file.



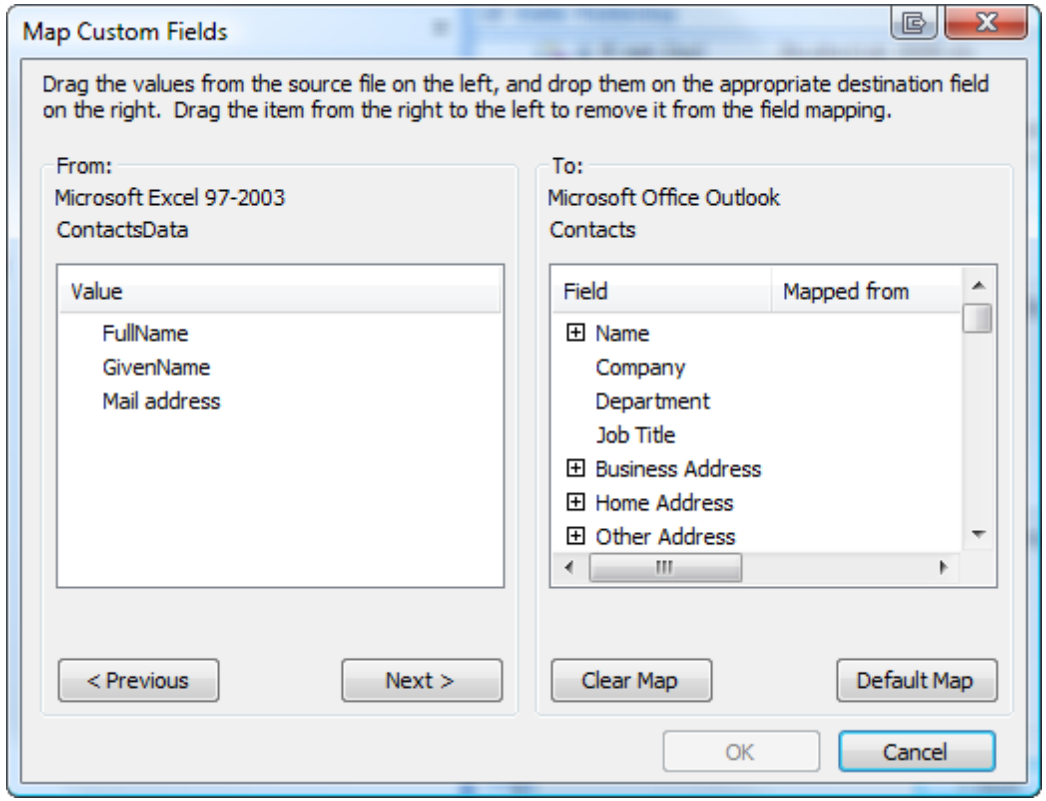
Browse to the file and press "Next >"



Select "Contacts" and press "Next >"



You see your named range now in this dialog. If there are more named ranges you see them also in this dialog.When you mark the checkbox before your named range the dialog below popup and you can map your fields.



After you are ready press OK in the "Map Custom Fields" dialog
and press Finish in the "Import file" dialog.

Send a mail when a cell reaches a certain value

(Outlook, Outlook Express and CDO)
Ron de Bruin (Last update 28 Oct 2006 **)**

To run a macro automatic when a specific cell reaches a certain value you can use
the Change event in the worksheet module.
The examples on this page use Cell A1 and will run the macro if the cell value > 200

Change Event

1) Right click on a sheet tab and choose view code
2) Paste one of the events in this module.

3) Alt-Q to go back to Excel

If you change the cell manual you can use this example for A1.
Change YourMacroName **to the name of your macro.**

```
Private Sub Worksheet_Change(ByVal Target As Range)
   If Not Application.Intersect(Range("A1"), Target) Is Nothing Then
      If IsNumeric(Target.Value) And Target.Value > 200 Then
         YourMacroName
      End If
   End If
End Sub
```

If A1 is a formula you can use this example

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim rng As Range
   If Target.Cells.Count > 1 Then Exit Sub
   On Error GoTo EndMacro
   If Not Target.HasFormula Then
```

```
    Set rng = Target.Dependents
    If Not Intersect(Range("A1"), rng) Is Nothing Then
        If Range("A1").Value > 200 Then YourMacroName
    End If
  End If
EndMacro:
End Sub
```

Example macro's

Note : copy the macro's in a normal module

Example 1

This is the best way if you can use it?
Read the information on this page first.
http://www.rondebruin.nl/cdo.htm

```
Sub Mail_CDO()
  Dim iMsg As Object
  Dim iConf As Object
```

```
    '    Dim Flds As Variant

    Set iMsg = CreateObject("CDO.Message")

    Set iConf = CreateObject("CDO.Configuration")

    '    iConf.Load -1    ' CDO Source Defaults

    '    Set Flds = iConf.Fields

    '    With Flds

    '        .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2

    '            .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = "Fill in your
SMTP server here"

    '        .Item("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 25

    '        .Update

    '    End With

    With iMsg

        Set .Configuration = iConf

        .To = "ron@debruin.nl"

        .CC = ""

        .BCC = ""
```

```
    .From = """Ron"" <ron@something.nl>"
    .Subject = "Important message"
    .TextBody = "Hi there" & vbNewLine & vbNewLine & _
                "Cell A1 is changed"
    .Send
  End With
  Set iMsg = Nothing
  Set iConf = Nothing
End Sub
```

Example 2

You can use this example only if you use Outlook.

If you don't like the security warning in Outlook see this page
http://www.rondebruin.nl/mail/prevent.htm

```
Sub Mail_with_outlook()
  Dim OutApp As Object
  Dim OutMail As Object
  Dim strto As String, strcc As String, strbcc As String
  Dim strsub As String, strbody As String
```

```
    Set OutApp = CreateObject("Outlook.Application")
    OutApp.Session.Logon
    Set OutMail = OutApp.CreateItem(0)

    strto = "ron@debruin.nl"
    strcc = ""
    strbcc = ""
    strsub = "Important message"
    strbody = "Hi there" & vbNewLine & vbNewLine & _
              "Cell A1 is changed"

    With OutMail
        .To = strto
        .CC = strcc
        .BCC = strbcc
        .Subject = strsub
        .Body = strbody
        .Send
    End With

    Set OutMail = Nothing

    Set OutApp = Nothing

End Sub
```

Example 3

This example is for Outlook Express.

If you don't like the security warning in Outlook Express see this page
http://www.rondebruin.nl/mail/prevent.htm

```
Sub Mail_Outlook_Express()


    Dim Recipient As String, Subj As String, HLink As String
```

```
Dim Recipientcc As String, Recipientbcc As String

Dim msg As String

Recipient = "ron@debruin.nl"

Recipientcc = ""

Recipientbcc = ""

Subj = "Important message"

msg = "Hi there" & vbNewLine & vbNewLine & _

    "Cell A1 is changed"

msg = WorksheetFunction.Substitute(msg, vbNewLine, "%0D%0A")

HLink = "mailto:" & Recipient & "?" & "cc=" & Recipientcc _

    & "&" & "bcc=" & Recipientbcc & "&"

HLink = HLink & "subject=" & Subj & "&"

HLink = HLink & "body=" & msg

ActiveWorkbook.FollowHyperlink (HLink)

Application.Wait (Now + TimeValue("0:00:02"))

Application.SendKeys "%s"
```

```
End Sub
```

Send text in the body of the mail with Outlook Express

Ron de Bruin (Last update 20 June 2004 **)**

It is only possible with OE when you use SendKeys to press the Send button. Also the amount of characters is limited.

Below there are two examples to send a range in the body with Outlook Express.
The first send Sheets("mysheet").Range("C1:C20") and the second one Sheets("mysheet").Range("C1:C60").
The second sub can send more characters then the first one.

Example 1

```
Sub Mail_Text_in_Body()
'Example for Outlook Express
'In Excel 2002 I can use around 600-700 characters
    Dim msg As String, cell As Range
    Dim Recipient As String, Subj As String, HLink As String
    Dim Recipientcc As String, Recipientbcc As String
    Recipient = "ron@debruin.nl"
    Recipientcc = ""
    Recipientbcc = ""
    'You can use a cell value also like this
    'Recipient = Sheets("mysheet").Range("A1").Value
    Subj = "Testbodymail"
    'Subj = Sheets("mysheet").Range("A2").Value
    msg = "Dear customer" & vbNewLine & vbNewLine
    For Each cell In Sheets("mysheet").Range("C1:C20")
        msg = msg & vbNewLine & cell
    Next cell
    msg = WorksheetFunction.Substitute(msg, vbNewLine, "%0D%0A")
    'If you have hard returns in one of your cells you also need this line (Tip from Keepitcool)
    msg = WorksheetFunction.Substitute(msg, vbLf, "%0D%0A")
    HLink = "mailto:" & Recipient & "?" & "cc=" & Recipientcc & "&" & "bcc=" & Recipientbcc & "&"
    HLink = HLink & "subject=" & Subj & "&"
    HLink = HLink & "body=" & msg
```

```
    ActiveWorkbook.FollowHyperlink (HLink)
    Application.Wait (Now + TimeValue("0:00:03"))
    Application.SendKeys "%s"
End Sub
```

Example 2

Example 2 : copy the code below in a normal module

```
Private Declare Function ShellExecute Lib "Shell32.dll" _
Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, _
ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, _
ByVal nShowCmd As Long) As Long
Sub Mail_Text_in_Body_2()
'Example for Outlook Express with API call
'In Excel 2002 I can use around 1800 characters
    Dim msg As String, URL As String
    Dim Recipient As String, Subj As String
    Dim Recipientcc As String, Recipientbcc As String
    Dim cell As Range
    Recipient = "ron@debruin.nl"
    Recipientcc = ""
    Recipientbcc = ""
    'You can use a cell value also like this
    'Recipient = Sheets("mysheet").Range("A1").Value
    Subj = "Testbodymail"
    'Subj = Sheets("mysheet").Range("A2").Value
    msg = "Dear customer" & vbNewLine & vbNewLine
    For Each cell In Sheets("mysheet").Range("C1:C60")
        msg = msg & vbNewLine & cell
    Next cell
    msg = WorksheetFunction.Substitute(msg, vbNewLine, "%0D%0A")
    'If you have hard returns in one of your cells you also need this line (Tip from Keepitcool)
    msg = WorksheetFunction.Substitute(msg, vbLf, "%0D%0A")
    URL = "mailto:" & Recipient & "?cc=" & Recipientcc & "&bcc=" & Recipientbcc _
        & "&subject=" & Subj & "&body=" & msg
    ShellExecute 0&, vbNullString, URL, vbNullString, vbNullString, vbNormalFocus
    Application.Wait (Now + TimeValue("0:00:03"))
    Application.SendKeys "%s"
```

```
End Sub
```

Send personalized email (with Outlook Express)

See John Walkenbach his website.
http://www.j-walk.com/ss/excel/tips/tip86.htm

This sub is doing the same as John's Example (see Webpage)
Only I use **FollowHyperlink** in the code.
See the website for more information.

```
Sub SendEmail2()
  ' Ron de Bruin
  For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
    If cell.Value Like "*@*" Then
      Recipient = cell.Value
      Subj = "Your Annual Bonus"
      Msg = "Dear " & cell.Offset(0, -1).Value & "%0A"
      Msg = Msg & "%0A" & "I am pleased to inform you that your annual bonus is "
      Msg = Msg & cell.Offset(0, 1).Value & "%0A"
      Msg = Msg & "%0A" & "William Rose"
      Msg = Msg & "%0A" & "President"
      HLink = "mailto:" & Recipient & "?"
      HLink = HLink & "subject=" & Subj & "&"
      HLink = HLink & "body=" & Msg
      ActiveWorkbook.FollowHyperlink (HLink)
      Application.Wait (Now + TimeValue("0:00:02"))
      SendKeys "%s", True
    End If
  Next
```

```
End Sub
```

- OFFICIAL MSDN-Beiträge
Different Ways to Take Advantage of the E-mail Features of Excel

Ron de Bruin
Microsoft Excel MVP
Frank C. Rice
Microsoft Corporation
August 2003
Applies to:
Microsoft® Excel 97 and later versions
Summary: Microsoft Excel MVP Ron de Bruin provides a number of samples and a handy add-in that enhances your experience when working with e-mail from Excel. From sending individual worksheets in a workbook to sending e-mail to multiple recipients, the code samples and add-in should become a part of your reference library. The SendMail 2.0 add-in and a user guide is available from http://www.rondebruin.nl. (13 printed pages)
Contents

Introduction
This article features code samples, a wizard, and an add-in utility that you can use to perform various e-mail functions from Microsoft Excel. Ron de Bruin, an Excel Most Valuable Professional (MVP) and a frequent contributor to the newsgroups provided the samples and add-in.
Note You should be aware of the two main security features for Outlook 2002 and later versions (and with a security patch for Outlook 2000):
Blocking of a customizable list of file attachments considered unsafe because they can be used to propagate viruses.
Pop-up confirmation dialogs that occur whenever a program accesses address-related properties or attempts to send a message.
These constraints can affect the way you interact with Outlook in the following procedures.
Sending E-mail in Excel
One way to send e-mail from Excel is to the use the Mail Merge Wizard which is only available in Excel 2000 and later versions (called the Mail Merge Helper in Excel 2000). The Mail Merge Wizard simplifies the batch creation of letters, sending of e-mail messages, creating postal mailing labels, labeling envelopes for postal mail, and creating directory lists. You can find more detailed information about the Mail Merge Wizard by reading the Excel help topic titled *Create a Word mail merge with Excel data*.

Another way to use the e-mail features of Excel are with the code samples discussed in this article. In addition to augmenting your own code, you can use the samples provided here as a starting point to better help you understand how you can send as e-mail the various objects and data in your own worksheets and workbooks. Several samples also include alternate ways of modifying the code to send just the piece of information you want.
Yet another way to send mail is to use the SendMail 2.0 add-in utility created by Ron de Bruin. The SendMail 2.0 utility gives you the ability to send all or part of a workbook either in working format or with just the values of the current data and formulas. Once you install SendMail 2.0, to access these features, click **Tools**, and then click **SendMail**.
Before You Send E-mail
Before you use the e-mail features of Excel, either by feeding data to the Mail Merge Wizard or through the Microsoft Visual Basic® for Applications (VBA) code sample in this article, it is a good idea to ensure your data is structured so that there are no surprises. The following criteria are applicable for using the wizard but also improve predictability when using the code samples:
Add column headers above each column in the first row such as Title, First Name, Last Name, Address1, and so forth.
Ensure that individual pieces of information you want to work with are in separate fields. For example, separating the recipient's name into separate first and last name fields allows you to use just the parts you want for a specific task such as using the last name in a salutation and then combining the first and last fields in the address block in a letter.
Ensure that each field name is unique to avoid ambiguity.
Ensure each row of data refers to a single entity. For example, each row refers to one recipient.
Avoid blank rows in your data.
Note The VBA code in the following sections was tested in Microsoft Outlook 2002 and Microsoft Outlook Express 6.0. It may or may not work with other e-mail clients.
Send the Active Workbook by E-mail
The following subroutine sends the active workbook where the workbook that contains the code is not the active workbook:

Copy Code
```
Sub Mail_workbook()
    ActiveWorkbook.SendMail "someone@microsoft.com","Subject_line"
End Sub
```
To e-mail the workbook that contains the code, then use the following line instead:

Copy Code
```
ThisWorkbook.SendMail "someone@microsoft.com","Subject_line"
```
Note The workbook doesn't have to be the active workbook in use at the time that the code is executed.
Send a Single Sheet by E-mail
The following example illustrates how to send a single sheet in e-mail with the following options:
By creating a workbook with just the active sheet
By saving the workbook before sending it with a date/time stamp
By sending the active workbook
By deleting the file from your hard disk after you send it

Copy Code
```
Sub Mail_ActiveSheet()
    Dim strDate As String
    ActiveSheet.Copy
    strDate = Format(Date, "dd-mm-yy") & " " & Format(Time, "h-mm-ss")
    ActiveWorkbook.SaveAs "Part of " & ThisWorkbook.Name _
```

```
      & " " & strDate & ".xls"
   ActiveWorkbook.SendMail "someone@microsoft.com", _
      "Subject_line"
   ActiveWorkbook.ChangeFileAccess xlReadOnly
   Kill ActiveWorkbook.FullName
   ActiveWorkbook.Close False
End Sub
```

You can use the following line if you know which sheet you want to send:

Copy Code

```
...
Sheets("Sheet5").Copy
...
```

Note The code doesn't have to be in the active sheet when it is executed.

Send an Array of Worksheets by E-mail

You can use the following sample code to send multiple worksheets in e-mail with the following options:

By creating a workbook with two worksheets

By saving the workbook before sending it with a date/time stamp

By deleting the file from your hard disk after you send it

Copy Code

```
Sub Mail_SheetsArray()
   Dim strDate As String
   Sheets(Array("Sheet1", "Sheet3")).Copy
   strDate = Format(Date, "dd-mm-yy") & " " & Format(Time, "h-mm-ss")
   ActiveWorkbook.SaveAs "Part of " & ThisWorkbook.Name _
      & " " & strDate & ".xls"
   ActiveWorkbook.SendMail "someone@microsoft.com", _
      "Subject_line"
   ActiveWorkbook.ChangeFileAccess xlReadOnly
   Kill ActiveWorkbook.FullName
   ActiveWorkbook.Close False
End Sub
```

Send the Selection by E-mail

This subroutine sends a newly created workbook with just the visible cells in the selection. The cells are added as values using **Paste Special** in the workbook you send.

The routine saves the workbook before sending it by e-mail with a date/time stamp.

After the file is sent, the workbook is deleted from your hard disk.

Because it sends only the visible cells, the subroutine also works if you want to send a range with hidden rows or columns in it.

The normal **Copy** and **Paste** commands make the hidden rows and columns visible!

Copy Code

```
Sub Mail_Selection()
   Dim source As Range
```

```
    Dim ColumnCount As Long
    Dim FirstColumn As Long
    Dim ColumnWidthArray() As Double
    Dim lIndex As Long
    Dim lCount As Long
    Dim dest As Workbook
    Dim i As Long
    Dim strdate As String

    Set source = Nothing
    On Error Resume Next
    Set source = Selection.SpecialCells(xlCellTypeVisible)
    On Error GoTo 0
    If source Is Nothing Then
        MsgBox "The selection is not a range or the sheet is protect, please correct and try again.", vbOKOnly
        Exit Sub
    End If
If ActiveWindow.SelectedSheets.Count > 1 Or _
       Selection.Cells.Count = 1 Or _
       Selection.Areas.Count > 1 Then
        MsgBox "An Error occurred :" & vbNewLine & vbNewLine & _
               "You have more than one sheet selected." & vbNewLine & _
               "You only selected one cell." & vbNewLine & _
               "You selected more than one area." & vbNewLine & vbNewLine & _
               "Please correct and try again.", vbOKOnly
        Exit Sub
    End If

    Application.ScreenUpdating = False
    ColumnCount = Selection.Columns.Count
    FirstColumn = Selection.Cells(1).Column - 1
    ReDim ColumnWidthArray(1 To ColumnCount)
    lIndex = 0
    For lCount = 1 To ColumnCount
        If Columns(FirstColumn + lCount).Hidden = False Then
            lIndex = lIndex + 1
            ColumnWidthArray(lIndex) = Columns(FirstColumn + lCount).ColumnWidth
        End If
    Next lCount
    Set dest = Workbooks.Add(xlWBATWorksheet)
    source.Copy
    With dest.Sheets(1)
        .Cells(1).PasteSpecial xlPasteValues, , False, False
        .Cells(1).PasteSpecial xlPasteFormats, , False, False
```

```
        .Cells(1).Select
        Application.CutCopyMode = False
        For i = 1 To lIndex
            .Columns(i).ColumnWidth = ColumnWidthArray(i)
        Next
    End With
    strdate = Format(Now, "dd-mm-yy h-mm-ss")
    With dest
        .SaveAs "Selection of " & ThisWorkbook.Name _
             & " " & strdate & ".xls"
        .SendMail "ron@debruin.nl", _
                 "This is the Subject line"
        .ChangeFileAccess xlReadOnly
        Kill .FullName
        .Close False
    End With
    Application.ScreenUpdating = True
End Sub
```

You can use this if you know the range you want to send:

Copy Code

```
Set source = Range("A1:E50").SpecialCells(xlCellTypeVisible)
Range("A1:E50").Select
```

Then you can remove this section:

Copy Code

```
If ActiveWindow.SelectedSheets.Count > 1 Or _
       Selection.Cells.Count = 1 Or _
       Selection.Areas.Count > 1 Then
        MsgBox "An Error occurred :" & vbNewLine & vbNewLine & _
               "You have more than one sheet selected." & vbNewLine & _
               "You only selected one cell." & vbNewLine & _
               "You selected more than one area." & vbNewLine & vbNewLine & _
               "Please correct and try again.", vbOKOnly
        Exit Sub
    End If
```

Sending Specific Worksheets by E-mail

This procedure illustrates how to send every worksheet with an address in cell A1 by e-mail. This way you can send each sheet to another person. It does this by cycling through each worksheet in the workbook and checking cell A1 for the @ character. If found, the code makes a copy of the worksheet and then sends the copy by e-mail to the address in cell A1. Finally, the code deletes the file copy from your hard disk.

Copy Code

```
Sub Mail_every_Worksheet()
```

```
    Dim sh As Worksheet
    Application.ScreenUpdating = False
    For Each sh In ThisWorkbook.Worksheets
        If sh.Range("a1").Value Like "*@*" Then
            sh.Copy
            ActiveWorkbook.SaveAs "Sheet " & sh.Name & " of " _
                & ThisWorkbook.Name & ".xls"
            ActiveWorkbook.SendMail ActiveSheet.Range("a1").Value, _
                "Subject_line"
            ActiveWorkbook.ChangeFileAccess xlReadOnly
            Kill ActiveWorkbook.FullName
            ActiveWorkbook.Close False
        End If
    Next sh
    Application.ScreenUpdating = True
End Sub
```

Send Worksheets to One or More People by E-mail

To do this, add a sheet with the name **mail** to your workbook. Then, for every e-mail you want to send, create three columns that include:

The sheet or sheets you want to send

The e-mail address or addresses

The subject of the e-mail

Columns **A:C** include the information for the first e-mail and **D:F** for the second one (see Figure 1). You can send 85 different e-mails this way (85*3 = 255 columns).

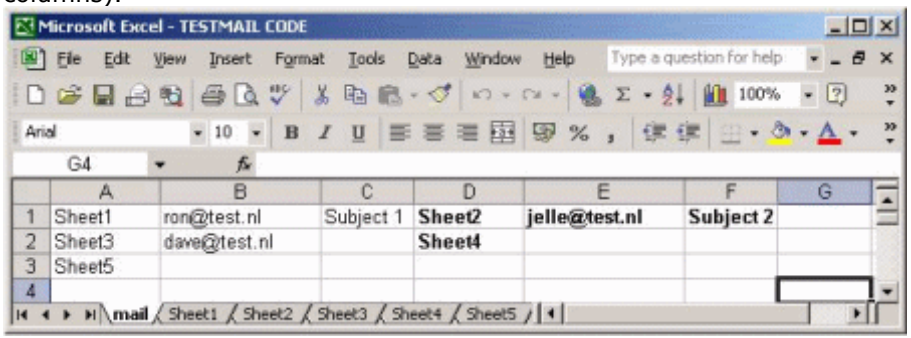


Figure 1. Send e-mail to multiple people

In this example, the following results:

Ron and Dave receive sheet1, sheet3 and sheet5 (in one workbook) and the subject line of "Subject 1"

Jelle receives sheet2 and sheet4 (in one workbook) and the subject line of "Subject 2"

The following code accomplishes this:

Copy Code

```
Sub Mail_sheets()
    Dim MyArr As Variant
```

```
    Dim last As Long
    Dim shname As Long
    Dim a As Integer
    Dim Arr() As String
    Dim N As Integer
    Dim strdate As String
    For a = 1 To 253 Step 3
       If ThisWorkbook.Sheets("mail").Cells(1, a).Value = "" Then
          Exit Sub
       End
       Application.ScreenUpdating = False
       last = ThisWorkbook.Sheets("mail").Cells(Rows.Count, _
          a).End(xlUp).Row
       N = 0
       For shname = 1 To last
          N = N + 1
          ReDim Preserve Arr(1 To N)
          Arr(N) = ThisWorkbook.Sheets("mail").Cells(shname, a).Value
       Next shname
       ThisWorkbook.Sheets(Arr).Copy
       strdate = Format(Date, "dd-mm-yy") & " " & _
          Format(Time, "h-mm-ss")
       ActiveWorkbook.SaveAs "Part of " & ThisWorkbook.Name _
          & " " & strdate & ".xls"
       With ThisWorkbook.Sheets("mail")
          MyArr = .Range(.Cells(1, a + 1), .Cells(Rows.Count, _
             a + 1).End(xlUp))
       End With
       ActiveWorkbook.SendMail MyArr, ThisWorkbook.Sheets("mail").Cells(1, a + 2).Value
       ActiveWorkbook.ChangeFileAccess xlReadOnly
       Kill ActiveWorkbook.FullName
       ActiveWorkbook.Close False
       Application.ScreenUpdating = True
    Next a
End Sub
```

Tips for Changing the Examples

The preceding macros are just some examples you can use to send e-mail from Excel. The following examples show ways you can change the code to suit your needs.

Keep the File

Delete these lines to keep the file you sent:

Copy Code

```
...
ActiveWorkbook.ChangeFileAccess xlReadOnly
```

```
Kill ActiveWorkbook.FullName
...
```

Don't Save the File
Delete these four lines if you don't want to save the workbook:

Copy Code

```
...
strDate = Format(Date, "dd-mm-yy") & " " & Format(Time, "h-mm-ss")
ActiveWorkbook.SaveAs

ActiveWorkbook.ChangeFileAccess xlReadOnly
Kill ActiveWorkbook.FullName
...
```

Change the Recipient Line
Use a cell (A1) containing an e-mail address as follows:

Copy Code

```
...
ActiveWorkbook.SendMail Sheets("mysheet").Range("a1").Value, _
    "Subject_line"
...
```

Use also a cell for the subject like this.

Copy Code

```
...
ActiveWorkbook.SendMail Sheets("mysheet").Range("a1").Value, _
    Sheets("mysheet").Range("b1").Value
...
```

To send e-mail to more people use this line:

Copy Code

```
...
ActiveWorkbook.SendMail Array("someone@microsoft.com", "someone@microsoft.com"), _
    "Subject_line"
...
```

To send e-mail to all addresses in a range:

Copy Code

```
...
Dim MyArr As Variant
MyArr = Sheets("mysheet").Range("c1:c10")
ActiveWorkbook.SendMail MyArr, _
    "Subject_line"
...
```

If you use this line, you can choose an address in the address book or type one yourself and put some text in the body:

Copy Code
```
...
ActiveWorkbook.SendMail "", "Subject_line"
...
```
Change the Save Line
You can change the save line to this, for example:

Copy Code
```
...
ActiveWorkbook.SaveAs Sheets("mysheet").Range("d1").Value
ActiveWorkbook.SaveAs "myfilename"
...
```
Important Use error checking to verify that a file with that name doesn't already exist or isn't already open. In the examples above, the file name includes the date and time so that the chance the file name already exists is very small.
Copy the Cells as Values
To paste cells as values you cannot protect the sheet. Alternatively, you can use unprotect and then protect the worksheet in the subroutine.
Use one of these lines to copy a single sheet:

Copy Code
```
...
Sh.copy
...

...
Activesheet.copy
...
```
Use this code to paste as values:

Copy Code
```
...
Cells.Copy
Cells.PasteSpecial xlPasteValues
Cells(1).Select
Application.CutCopyMode = False
...
```
If you copy more sheets in the newly created workbook then use this:

Copy Code
```
...
Worksheets.Select
Cells.Copy
Cells.PasteSpecial xlPasteValues
```

```
Cells(1).Select
Worksheets(1).Select
Application.CutCopyMode = False
...
```

The SendMail 2.0 Utility

The SendMail 2.0 utility is an add-in for sending customized Excel workbooks and worksheets by e-mail. In addition, SendMail allows significant customizing of what you send. As soon as you have the workbook open in Excel, you have the ability to send all or part of workbook either with the formatting or with just the values of the current data and formulas. Once you install SendMail 2.0, to access it, click **Tools**, and then click **SendMail**.

Note The SendMail 2.0 utility was tested in Outlook and Outlook Express. It will not work with other e-mail clients.

To install the Send Mail 2.0 utility for Excel 97 or later

Download and extract the SendMail utility to a local directory.

Copy SendMail 2.0.xla to the following directory:

local_drive:\Program Files\Microsoft Office\Office*Number*\Library

Note Depending on the version of Excel, the OfficeNumber directory may be only Office or may include a number. For example:

local_drive:\Program Files\Microsoft Office\Office\Library

-OR-

local_drive:\Program Files\Microsoft Office\Office11\Library

Open Excel.

Note You must have a workbook open.

Click the **Tools** menu and then click **Add-ins**.

Click **Browse. . .** to browse to *local_drive*:\Program Files\Microsoft Office\Office*number*\Library.

Click SendMail 2.0.xla and then click Open

Verify SendMail 2.0 is checked and then click **OK**.

For instructions on using SendMail 2.0, see the User Guide included in the download.

Conclusion

In this article, we looked at several code samples you can use to make mailing from Excel much easier. We also introduced the SendMail add-in that you can use to assist you in sending customized Excel workbooks and worksheets by e-mail. Exploring and implementing these tools in your own applications can help make your job as a developer much easier and make your solutions more versatile.

Working with Excel Workbooks and Worksheets in E-Mail

Summary: Microsoft Excel MVP Ron de Bruin provides a number of samples and a handy add-in that enhances your experience when working with e-mail from Excel. From sending individual worksheets in a workbook to sending e-mail to multiple recipients, the code samples and add-in should become a part of your reference library. (12 printed pages)

Ron de Bruin , Microsoft Excel MVP
Frank C. Rice, Microsoft Corporation

Published: January 2007

Applies to: Microsoft Office Excel 2007, Microsoft Office Excel 2003, Microsoft Excel 2002, Microsoft Excel 2000, Microsoft Excel 97, Microsoft Office Outlook 2007, Microsoft Office Outlook 2003, Microsoft Office Outlook 2002, Microsoft Office Outlook 2000, Microsoft Office Outlook Express, Microsoft Windows Mail

Download the SendMail tool add-in and user guide .

Note:

This article is an updated version of the Excel 2003 article, Different Ways to Take Advantage of the E-mail Features of Excel.

Contents

Background

This article features code samples and an add-in tool that you can use to perform various e-mail functions from Microsoft Office Excel. Ron de Bruin, an Excel Most Valuable Professional (MVP) and a frequent contributor to the newsgroups, provided the samples and add-in. You can find many Microsoft Outlook code samples and sample workbooks on Ron's Excel page.

This article covers the following e-mail programs:

Microsoft Office Outlook (included as part of the Microsoft Office system)
Microsoft Outlook Express (included in Microsoft Windows XP)
Microsoft Windows Mail (included in Microsoft Windows Vista; replaces Outlook Express)

Microsoft Office Outlook is a part of the Microsoft Office System. Outlook Express and Windows Mail are a part of the Windows operating system.

Note:

The code in this article is compatible with these three programs and uses the SendMail method in each program.

Of these programs, the object model for Office Outlook gives you many more options when you also have an account in Outlook. For example, you can:

Programmatically insert text in the body of an e-mail.
Add contacts into the CC or BCC fields.
Add attachments to the e-mail.

You should be aware of the two main security features for Outlook 2002, Outlook 2003, and Outlook 2007. Note that these features are implemented through a security patch for Outlook 2000:

Blocking of a customizable list of file attachments considered unsafe because they can be used to propagate viruses.
Pop-up confirmation dialogs that occur whenever a program accesses address-related properties or attempts to send a message.

For more information about Outlook 2007, see this Code Security Changes in Outlook 2007.

These constraints can affect the way you interact with Outlook in the procedures that appear in this article.

Note:

If you use Outlook Express or Windows Mail, you can turn off the confirmation dialogs. On the Tools menu, click Options, and then click the Security tab.

An alternative is to use the samples that use CDO for Microsoft Windows 2000, to download, see Sending Mail from Excel with CDO.

Sending E-Mail in Excel

There are many ways to send Excel data through e-mail. One way to send Excel data in e-mail is to use the Microsoft Office Word Mail Merge Wizard. This feature is only available in Microsoft Excel 2000, Microsoft Excel 2002, Microsoft Office Excel 2003, and Microsoft Office Excel 2007. In Microsoft Excel 2000, this feature is called the Mail Merge Helper. The Word Mail Merge Wizard simplifies many e-mail related tasks, including creating batch mailings, sending e-mail messages, creating postal mailing labels, labeling envelopes for postal mail, and creating directory lists.

Another way to use the e-mail features of Excel is with the code samples discussed in this article. You can use the samples as a starting point to help you understand how you can send the various objects and data in your own worksheets and workbooks as e-mail. Several samples also include alternate ways of modifying the code to send just the piece of information you want.

Yet another way to send mail is to use the SendMail add-in tool created by Ron de Bruin. To download, see Mail Add-ins for Excel. The SendMail tool gives you the ability to send all or part of a workbook either in working format or with just the values of the current data and formulas. After you install SendMail, to access these features, in previous versions of Excel, click Tools, and then click SendMail. In Excel 2007, click the Microsoft Office Button and then click Send.

Before Sending E-Mail

Before you use the e-mail features of Excel, either by feeding data to the Mail Merge Wizard or through the Microsoft Visual Basic for Applications (VBA) code sample in this article, it is a good idea to ensure your data is structured so that there are no surprises. The following criteria are applicable for using the wizard but also improve predictability when using the code samples:

Add column headers above each column in the first row such as Title, First Name, Last Name, Address1, and so forth.

Ensure that individual pieces of information you want to work with are in separate fields. For example, separating the recipient's name into separate first and last name fields allows you to use just the parts you want for a specific task such as using the last name in a salutation and then combining the first and last fields in the address block in a letter.

Ensure that each field name is unique to avoid ambiguity.

Ensure each row of data refers to a single entity. For example, each row refers to one recipient.

Avoid blank rows in your data.

Sending the Active Workbook by E-Mail

The following subroutine sends the active workbook in an e-mail message:

Visual Basic

Copy Code

```
Sub Mail_Workbook()
' Works with Excel 97-2007.
Dim wb As Workbook
Set wb = ActiveWorkbook

' Check to see if this is Excel 2007 and is not a macro-enabled file.
If Val(Application.Version) >= 12 Then
   ' The code 51 represents the enumeration for a macro-free Excel 2007 Workbook (.xlsx).
   If wb.FileFormat = 51 And wb.HasVBProject = True Then
      MsgBox "There is VBA code in this xlsx file that will be removed if you try to send this file." & vbNewLine & _
      "Save the file first as xlsm and then try the macro again.", vbInformation
      Exit Sub
   End If
End If
```

```
On Error Resume Next
  wb.SendMail "someone@example.com", _
     "This is the Subject line"
On Error GoTo 0
End Sub
```

Note:

To send a workbook other than the active workbook, change the assignment to the wb variable.

Sending a Single Worksheet by E-Mail

The following example illustrates how to send a single worksheet in e-mail by:
Creating a workbook with just the active worksheet.
Saving the workbook before sending it with a date/time stamp.
Sending the active workbook.
Deleting the file from your hard disk after you send it.
Visual Basic

Copy Code

```
Sub Mail_ActiveSheet()
' Works in Excel 97 through Excel 2007.
Dim FileExtStr As String
Dim FileFormatNum As Long
Dim Sourcewb As Workbook
Dim Destwb As Workbook
Dim TempFilePath As String
Dim TempFileName As String

With Application
   .ScreenUpdating = False
   .EnableEvents = False
End With

Set Sourcewb = ActiveWorkbook
' Using ActiveSheet.Copy creates a new workbook with
' the sheet and the file format is the same as the
' original workbook.
' Copy the worksheet to a new workbook.
ActiveSheet.Copy
Set Destwb = ActiveWorkbook

' Determine the Excel version and file extension/format.
With Destwb
```

```
  If Val(Application.Version) < 12 Then
    ' You are using Excel 97-2003.
    FileExtStr = ".xls": FileFormatNum = -4143
  Else
    ' You are using Excel 2007.
    ' When you use ActiveSheet.Copy to create a workbook,
    ' you are prompted with a security dialog. If you click No
    ' in the dialog, then the name of Sourcewb is the same
    ' as Destwb and you exit the subroutine. You only see this
    ' dialog when you attempt to copy a worksheet from an .xlsm file with macros disabled.
    If Sourcewb.Name = .Name Then
      With Application
        .ScreenUpdating = True
        .EnableEvents = True
      End With
      MsgBox "Your answer is No in the security dialog."
      Exit Sub
    Else
      Select Case Sourcewb.FileFormat
        ' Code 51 represents the enumeration for a macro-free
        ' Excel 2007 Workbook (.xlsx).
        Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
        ' Code 52 represents the enumeration for a
        ' macro-enabled Excel 2007 Workbook (.xlsm).
        Case 52:
          If .HasVBProject Then
            FileExtStr = ".xlsm": FileFormatNum = 52
          Else
            FileExtStr = ".xlsx": FileFormatNum = 51
          End If
        ' Code 56 represents the enumeration for a
        ' a legacy Excel 97-2003 Workbook (.xls).
        Case 56: FileExtStr = ".xls": FileFormatNum = 56
        ' Code 50 represents the enumeration for a
        ' binary Excel 2007 Workbook (.xlsb).
         Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
      End Select
    End If
  End If
End With

' Change all cells in the worksheet to values, if desired.
''   With Destwb.Sheets(1).UsedRange
''     .Cells.Copy
```

```
''      .Cells.PasteSpecial xlPasteValues
''      .Cells(1).Select
''   End With
''Application.CutCopyMode = False

'Save the new workbook and then mail it.
   TempFilePath = Environ$("temp") & "\"
   TempFileName = "Part of " & Sourcewb.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")

With Destwb
   .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
      On Error Resume Next
   .SendMail "someone@example.com", _
      "This is the Subject line."
      On Error GoTo 0
   .Close SaveChanges:=False
End With

' Delete the file you just sent.
Kill TempFilePath & TempFileName & FileExtStr

With Application
   .ScreenUpdating = True
   .EnableEvents = True
End With
End Sub
```

Note:

To send a worksheet other than the active worksheet, change ActiveSheet.Copy to copy the worksheet you wish to send. For instance, Sheets("Sheet 5").Copy.

Looking at the macro, first, we point the workbook variable Sourcewb to the active workbook, and then make a copy of the worksheet, which makes a new workbook and copies the worksheet to that workbook. Next, we make the new workbook the active workbook. This is the workbook that you send in the e-mail message.

Visual Basic

Copy Code

```
Set Sourcewb = ActiveWorkbook

' Copy the worksheet to a new workbook.
ActiveSheet.Copy
Set Destwb = ActiveWorkbook
```

Then we test to see if the application version is less than "12". If so, we use the file extension ".xls" and the file format enumeration of "-4143." This is the

Excel binary workbook format in Excel 97 through Excel 2003.
If you run this code in Excel 2007 it identifies the file format of the parent workbook and saves the new file in that format. If the parent workbook is a macro-enabled (.xlsm) file and there is no code in the new workbook, the new file is saved as a macro-free file (.xlsx). This way, the recipient of the e-mail knows that this is a macro-free file. However, if the parent workbook is not .xlsx, .xlsm, or .xls then it is saved as an Excel binary (.xlsb) file.
The principle file format enumerations in Excel 2007 are:
51 = xlOpenXMLWorkbook (macro-free Excel 2007 workbook, .xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (Excel 2007 workbook with or without macros, .xlsm)
50 = xlExcel12 (Excel 2007 binary formatted workbook, with or without macros, .xlsb)
56 = xlExcel8 (Excel 97 through Excel 2003 formatted files used in Excel 2007, .xls)
If you always want to save in a certain format, you can replace the Select Case block in the previous macro with one of the lines from this list:
Visual Basic

Copy Code
```
FileExtStr = ".xlsb": FileFormatNum = 50
FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsm": FileFormatNum = 52
FileExtStr = ".xls": FileFormatNum = 56
```
In addition, you may want to save the file in an alternative format such as Comma Separated Values file (.csv), a text file (.txt), or printer file (.prn). In that case, replace the Select Case block with one of the following lines:
Visual Basic

Copy Code
```
FileExtStr = ".csv": FileFormatNum = 6
FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
```
Getting back to the code, we said that Destwb is the one worksheet workbook that we created. We copy all cells in the first worksheet by using the UsedRange method. The worksheet index is used because we do not know the name of the worksheet. Then we use the PasteSpecial method to paste just the values (and not any formulas) and select just the first cell of the used range because when you use PasteSpecial, the entire range is selected. Next, the CutCopyMode method clears the clipboard.
Visual Basic

Copy Code
```
''   With Destwb.Sheets(1).UsedRange
''      .Cells.Copy
''      .Cells.PasteSpecial xlPasteValues
''      .Cells(1).Select
''   End With
''Application.CutCopyMode = False
```
Sending an Array of Worksheets by E-Mail
You can use the following sample code to send multiple worksheets in e-mail. It does this by:
Creating a workbook with just the worksheets in the array.
Saving the workbook before sending it with a date/time stamp.
Sending the active workbook.
Deleting the file from your hard disk after you send it.

Visual Basic

Copy Code

```
Sub Mail_Sheets_Array()
' Works in Excel 97 through Excel 2007.
Dim FileExtStr As String
Dim FileFormatNum As Long
Dim Sourcewb As Workbook
Dim Destwb As Workbook
Dim TempFilePath As String
Dim TempFileName As String
Dim sh As Worksheet

With Application
  .ScreenUpdating = False
  .EnableEvents = False
End With

Set Sourcewb = ActiveWorkbook

' Copy the worksheets to a new workbook.
Sourcewb.Sheets(Array("Sheet1", "Sheet3")).Copy
Set Destwb = ActiveWorkbook

' Determine the Excel version, file extension, and format.
With Destwb
  If Val(Application.Version) < 12 Then
    ' You are using Excel 97 through Excel 2003.
    FileExtStr = ".xls": FileFormatNum = -4143
  Else
    ' You are using Excel 2007.
    ' When you use ActiveSheet.Copy to create a new workbook,
    ' you are prompted with a security dialog. If you click No
    ' in the dialog, then the name of Sourcewb is the same
...   ' as Destwb and you exit the subroutine. You only see this
    ' dialog when you attempt to copy a worksheet from an .xlsm
    ' file with macros disabled.
    If Sourcewb.Name = .Name Then
      With Application
        .ScreenUpdating = True
        .EnableEvents = True
      End With
      MsgBox "Your answer is No in the security dialog."
      Exit Sub
```

```
      Else
        Select Case Sourcewb.FileFormat
          Case 51: FileExtStr = ".xlsx": FileFormatNum = 51
          Case 52:
            If .HasVBProject Then
              FileExtStr = ".xlsm": FileFormatNum = 52
            Else
              FileExtStr = ".xlsx": FileFormatNum = 51
            End If
          Case 56: FileExtStr = ".xls": FileFormatNum = 56
          Case Else: FileExtStr = ".xlsb": FileFormatNum = 50
        End Select
      End If
    End If
End With

' Change all cells in the worksheets to values if desired.
''For Each sh In Destwb.Worksheets
''   sh.Select
''   With sh.UsedRange
''      .Cells.Copy
''      .Cells.PasteSpecial xlPasteValues
''      .Cells(1).Select
''   End With
''   Application.CutCopyMode = False
''   Destwb.Worksheets(1).Select
''Next sh

' Save the new workbook and then mail it.
TempFilePath = Environ$("temp") & "\"
TempFileName = "Part of " & Sourcewb.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")

With Destwb
   .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
      On Error Resume Next
   .SendMail "someone@example.com", _
      "This is the Subject line"
      On Error GoTo 0
    .Close SaveChanges:=False
End With

' Delete the file you just sent.
Kill TempFilePath & TempFileName & FileExtStr
```

```
With Application
   .ScreenUpdating = True
   .EnableEvents = True
End With
End Sub
```

You can also use this line if you want to send all of the selected worksheets:

Visual Basic

Copy Code

```
ActiveWindow.SelectedSheets.Copy
```

This code is similar to the procedure to send a single worksheet, with the exception of the statement that copies the array of worksheets:

Visual Basic

Copy Code

```
Sourcewb.Sheets(Array("Sheet1", "Sheet3")).Copy
```

The commented code to change all cells in the worksheets to values is also different in this macro.

Sending a Range or Selection by E-Mail

The following subroutine sends a newly created workbook with just the visible cells in the selection. It does this by doing the following:

Creates a workbook with just the visible cells in the range ("A1:K50") from the source workbook.
Adds the cells as values using the Paste Special command into the workbook you send.
Saves the workbook with a date/time stamp before sending it.
Sends the active workbook.
Deletes the file from your hard disk after you send it.

Because it sends only the visible cells, the subroutine also works if you want to send a range with hidden rows or columns in it. The standard Copy and Paste commands make the hidden rows and columns visible.

Visual Basic

Copy Code

```
Sub Mail_Range()
' Works in Excel 2000 through Excel 2007.
Dim Source As Range
Dim Destwb As Workbook
Dim wb As Workbook
Dim TempFilePath As String
Dim TempFileName As String
Dim FileExtStr As String
Dim FileFormatNum As Long

Set Source = Nothing
On Error Resume Next
Set Source = Range("A1:K50").SpecialCells(xlCellTypeVisible)
On Error GoTo 0

If Source Is Nothing Then
```

```
    MsgBox "The source is not a range or the worksheet is protected. Please correct the problem and try again.", vbOKOnly
    Exit Sub
End If

With Application
    .ScreenUpdating = False
    .EnableEvents = False
End With

Set wb = ActiveWorkbook
Set Destwb = Workbooks.Add(xlWBATWorksheet)

Source.Copy
With Destwb.Sheets(1)
    ' The number 8 pastes the column width. Because of
    ' of a bug in Excel 2000, you must use the number
    ' instead of "xlPasteColumnWidths".
    .Cells(1).PasteSpecial Paste:=8
    .Cells(1).PasteSpecial Paste:=xlPasteValues
    .Cells(1).PasteSpecial Paste:=xlPasteFormats
    .Cells(1).Select
    Application.CutCopyMode = False
End With

TempFilePath = Environ$("temp") & "\"
TempFileName = "Selection of " & wb.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")

If Val(Application.Version) < 12 Then
    ' You are using Excel 2000 through Excel 2003.
    FileExtStr = ".xls": FileFormatNum = -4143
Else
    ' You are using Excel 2007.
    FileExtStr = ".xlsx": FileFormatNum = 51
End If

With Destwb
    .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
    On Error Resume Next
    .SendMail "someone@example.com", _
    "This is the Subject line"
    On Error GoTo 0
    .Close SaveChanges:=False
End With
```

```
' Delete the file you just sent.
Kill TempFilePath & TempFileName & FileExtStr

With Application
   .ScreenUpdating = True
   .EnableEvents = True
End With
End Sub
```

If you run the code in the macro in Excel 2000 through Excel 2003 (versions less than 12), the code saves the file in the legacy Excel binary workbook format:
Visual Basic

Copy Code

```
FileExtStr = ".xls": FileFormatNum = -4143
```

And if you are use Excel 2007 it saves as an .xlsx file:
Visual Basic

Copy Code

```
FileExtStr = ".xlsx": FileFormatNum = 51
```

Note For code samples that send the selection, see Example Code for Sending Mail from Excel.

Sending Specific Worksheets by E-Mail

The following procedure illustrates how to send any worksheet with an address in cell A1 by e-mail. This way you can send each worksheet to a different person.

Note:

Use this macro in a module in the workbook with the worksheets you want to send, not in your Personal Macro Workbook (personal.xls(b)).

Visual Basic

Copy Code

```
Sub Mail_Every_Worksheet()
' Works in Excel 97 through Excel 2007.
Dim sh As Worksheet
Dim wb As Workbook
Dim FileExtStr As String
Dim FileFormatNum As Long
Dim TempFilePath As String
Dim TempFileName As String

TempFilePath = Environ$("temp") & "\"

If Val(Application.Version) < 12 Then
   ' You are using Excel 97 through Excel 2003.
   FileExtStr = ".xls": FileFormatNum = -4143
```

```
Else
  'You use Excel 2007
  FileExtStr = ".xlsm": FileFormatNum = 52
End If

With Application
  .ScreenUpdating = False
  .EnableEvents = False
End With

For Each sh In ThisWorkbook.Worksheets
  If sh.Range("A1").Value Like "?*@?*.?*" Then
    sh.Copy
    Set wb = ActiveWorkbook
    TempFileName = "Sheet " & sh.Name & " of " _
      & ThisWorkbook.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")

    With wb
      .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
      On Error Resume Next
      .SendMail sh.Range("A1").Value, _
        "This is the Subject line"
      On Error GoTo 0
      .Close SaveChanges:=False
    End With

    Kill TempFilePath & TempFileName & FileExtStr

  End If
Next sh

With Application
  .ScreenUpdating = True
  .EnableEvents = True
End With
End Sub
```

The procedure works by cycling through each worksheet in the workbook and checking cell A1 for the @ character. If found, the code makes a copy of the worksheet and then sends the copy to the address specified in cell A1. Finally, the code deletes the file copy from the hard disk.

Tips for Modifying the Examples

The preceding macros are examples that you can use to send e-mail from Excel. You can modify these procedures to suit your own needs. For example:

Changing the Address Line

Use the contents of a cell for the To line:

Visual Basic

Copy Code
```
.SendMail Sheets("mysheet").Range("A1").Value, "This is the Subject line"
```
You can also use the contents of a cell for the subject:
Visual Basic

Copy Code
```
.SendMail Sheets("mysheet").Range("A1").Value, Sheets("mysheet").Range("B1").Value
```
To send e-mail to more than one person, you can use an array of addresses as follows:
Visual Basic

Copy Code
```
.SendMail Array("someone@example.com", "someone@example.com"), "Subject_line"
```
To send e-mail to all addresses within a range, use an array as follows:
Visual Basic

Copy Code
```
Dim MyArr As Variant
MyArr = Sheets("mysheet").Range("C1:C10")
.SendMail MyArr, _
    "Subject_line"
```
You can also leave the address blank, and then manually insert one, either by choosing an address in the address book or by typing one yourself. The following statement leaves the address blank but inserts a subject line in the e-mail message:
Visual Basic

Copy Code
```
.SendMail "", "Subject_line"
```
Changing the Save Line
You can change the TempFileName string in the preceding code sample to change the SaveAs line. You can also save the new workbook with the name of the source workbook and a date/time stamp:
Visual Basic

Copy Code
```
"Part of " & Sourcewb.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")
```
You can also save the file with an arbitrary string with:
Visual Basic

Copy Code
```
"New project" & " " & Format(Now, "dd-mmm-yy h-mm-ss")
```
Or with the value from cell A1 of the worksheet we send:
Visual Basic

Copy Code
```
Destwb.Sheets(1).Range("A1").Value & " " & Format(Now, "dd-mmm-yy h-mm-ss")
```
Or with the value from a cell in another worksheet from the original workbook:

Visual Basic

Copy Code
```
Sourcewb.Sheets("YourSheet").Range("A1").Value & " " & Format(Now, "dd-mmm-yy h-mm-ss")
```

Important:

Use error checking to verify that a file with that name does not already exist or is not already open. In the examples in this article, the file name includes the date and time so that the chance of using a file name that already exists is very small.

The SendMail Tool

To download the SendMail tool, see Mail Add-ins for Excel.

This tool is an add-in for sending customized Excel workbooks and worksheets by e-mail. In addition, the SendMail tool allows significant customization of the information you send. For example, after you have a workbook open in Excel, you have the ability to send all or part of the workbook, either with the formatting or with just the values of the current data and formulas.

To use the SendMail tool:

Download and extract it to a local directory.

Copy SendMail.xla to the following directory:

local_drive :\Program Files\Microsoft Office\ **OfficeNumber** \Library

Note:

Depending on the version of Excel, the OfficeNumber directory may be named Office or may include a version number. For example:
local_drive :\Program Files\Microsoft Office\Office\Library
-OR-
local_drive :\Program Files\Microsoft Office\Office11\Library

After you install the add-in, to access it:

Start Excel and open a workbook.

In Excel 97, Excel 2000, Excel 2003, and Excel 2003, click Tools, click Add-Ins, select SendMail, and then OK.

In Excel 2007, click the Microsoft Office button, click Excel Options, click the Add-Ins tab. In the Manage drop-down, click Excel Add-ins, and click Go. Verify SendMail is checked in this list and then click OK.

Note:

The SendMail tool was tested in Microsoft Office Outlook, Outlook Express, and Window Mail. It will not work with other e-mail clients.

For additional instructions on using the SendMail tool, see the user guide included in the download.

Conclusion

In this article, we looked at several code samples that you can use to make sending e-mail from Excel much easier. The SendMail add-in can assist you in sending customized Excel workbooks and worksheets by e-mail. Exploring and implementing these tools in your own applications can help make your job as a developer easier and make your solutions more versatile.

--- Weitere Outlook-Funktionen ---

Zwei höchst ungleiche Programme,... und doch eignen sie sich hervorragend um die Möglichkeiten der Zusammenarbeit von VBA zu zeigen und zu testen.
Auch wenn auf den ersten Blick eine sinnvolle Zusammenarbeit nicht möglich ist, so ergeben sich doch sinnvolle Synergien.

http://www.office.gmxhome.de/_excel_outlook.htm

Einen Termin aus einer EXCEL Tabelle nach Outlook übertragen

Absteigend von A2 werden alle Zellen durchgegangen bis eine Zelle leer ist.
Zu diesen Daten werden 7 Tage addiert und in Outlook ein Erinnerungstermin mit Hinweis
auf die aktive Arbeitsmappe um 8:00 Uhr eingetragen.

```
Sub Excel_Control_Termin_nach_Outlook()
'E 2000
'Dim OutApp As Outlook.Application
Dim OutApp As Object, apptOutApp As Object
'Hier beginnen die Termine
Range("A2").Select
Do Until ActiveCell.Value = ""
    Set OutApp = CreateObject("Outlook.Application")
    Set apptOutApp = OutApp.CreateItem(1) 'olAppointmentItem)
    With apptOutApp
        'Datum und Uhrzeit
        'Hier werden zum aktuellen Tag 7 Tage addiert
        .Start = Format(Now()+7, "dd.mm.yyyy") & " 08:00"
        'Alternativ werden die Termine aus der Zelle genommen
        '.Start = Format(ActiveCell.Value, "dd.mm.yyyy") & " 08:00"
        'Dauer. Angabe ist jeweils in ganzen Minuten zu setzen
        'Termininfo
        .Subject = "Rechnung: " & ActiveWorkbook.Name & " kontrollieren"
        'oder der Betreff steht in der Spalte rechts von den Terminen
        .Subject = ActiveCell.Offset(0,1)
        'Zusätzlicher Text
        .Body = ""
        'ort
        .Location = ""
        .Duration = "5"
        'Erinnerung
        .ReminderMinutesBeforeStart = 10
        'mit Sound :-)
        .ReminderPlaySound = True
        'Erinnerung wiederholen
```

```
        .ReminderSet = True
        'Termin speichern
        .Save
    End With
    'Nächste Zelle auswählen
    ActiveCell.Offset(1, 0).Select
    'Variablen leeren,... sonst "kotzt" Outlook irgendwann mal
    Set apptOutApp = Nothing
    Set OutApp = Nothing
Loop
MsgBox "Termine an Outlook übertragen!"
End Sub
```

Termin aus Outlook in EXCEL einlesen

```
Option Explicit

Sub Read_Control_Termin_to_Excel()
'by Ramses
'Datumsabfage über Inputbox
'Verweis auf Outlook 11 Library im VB-Editor muss gesetzt sein
'Early Binding ab Outlook 2003 nicht möglich
'weil die Rückgabewerte der ITEM-Indexes zufällig ist und von der
'Installation abhängt !!
Dim myR As Integer, i As Integer
Dim startDate As Date, endDate As Date, recDate As Date, extDate As Date
Dim myOlApp As Object, myOlSpace As Object, myOlFolder As Object
Dim myOlDateRange As Object, sAppoint As Object
Dim extRecurr As Object
Dim strRecurr As String
'Datum vorschlagen
Select Case Weekday(Now + 1, vbMonday)
    Case Is > 5
        recDate = Now + 3
    Case Else
        recDate = Now + 1
End Select
'Datum abfragen
startDate = Format(DateValue(InputBox("Welches Datum soll abgefragt werden ?" & Chr$(13) & _
    "Datum muss im Format ""01.01.2004"" eingeben werden", "Terminsuche", Format(recDate, "dd.mm.yyyy"))))
endDate = startDate
'Deklaration
```

```
Set myOlApp = CreateObject("Outlook.Application")
Set myOlSpace = myOlApp.GetNamespace("MAPI")
Set myOlFolder = myOlSpace.GetDefaultFolder(olFolderCalendar)
'Einträge ab Zeile 2
myR = 2
'Löscht alle zellen in der aktiven Tabelle
Cells.ClearContents
Cells.Interior.ColorIndex = xlNone
Cells(1, 1) = "Termin"
Cells(1, 2) = "Dauer"
Cells(1, 3) = "Ende"
Cells(1, 4) = "Ort"
Cells(1, 5) = "Betreff"
Cells(1, 6) = "Textinfo"
Set myOlDateRange = myOlFolder.Items.Restrict("[Start] >= '" & startDate & "' And [End] < & '" & endDate + 1 & "'")
For Each sAppoint In myOlDateRange
   With sAppoint
      'Termindaten eintragen
      Cells(myR, 1) = Format(.Start, "dd.mm.yyyy hh:mm")
      Cells(myR, 2) = Format((((1 / 24) / 60) * .Duration), "hh:mm")
      Set extRecurr = .GetRecurrencePattern
      'OlRecurrenceType sein:
      'olRecursDaily = 1
      'olRecursMonthly = 2
      'olRecursMonthNth = 3
      'olRecursWeekly = 4
      'olRecursYearly = 5
      'olRecursYearNth = 6
      Select Case extRecurr.RecurrenceType
         Case 1
            strRecurr = "Täglich für "
         Case 2
            strRecurr = "Monatlich für "
         Case 3
            strRecurr = "Monatlich jeden "
         Case 4
            strRecurr = "Wöchentlich für "
         Case 5
            strRecurr = "Jährlich für "
         Case 6
            strRecurr = "Jährlich jeden "
      End Select
      If Format(extRecurr.PatternEndDate, "dd.mm.yyyy") <> Format(DateValue("31.12.4500"), "dd.mm.yyyy") Then
         Cells(myR, 3) = Format(DateValue(extRecurr.PatternEndDate), "dd.mm.yyyy")
```

```
            Cells(myR, 3).Interior.ColorIndex = 3
            Cells(myR, 7) = strRecurr & DateValue(Format(extRecurr.PatternEndDate, "dd.mm.yyyy")) - startDate + 1 & " Tage"
        Else
            Cells(myR, 3) = Format(.Start + (((1 / 24) / 60) * .Duration), "hh:mm")
        End If
        Cells(myR, 4) = .Location
        Cells(myR, 5) = .Subject
        Cells(myR, 6) = .Body
        myR = myR + 1
    End With
Next
'Variablen leeren
Set myOlApp = Nothing
Set myOlSpace = Nothing
Set myOlFolder = Nothing
MsgBox "Alle Termine für den " & startDate & " eingelesen!"
End Sub
```

Der gleiche Code, nur dass hier das Start- und Enddatum in einer Zelle stehen

```
Sub Read_Control_Terminrange_to_Excel()
'by Ramses
'Zeitraumabfrage über Startdatum = A1 und Enddatum = B1
'Verweis auf Outlook 11 Library im VB-Editor muss gesetzt sein
'Early Binding ab Outlook 2003 nicht möglich
'weil die Rückgabewerte der ITEM-Indexes zufällig ist und von der
'Installation abhängt !!
Dim myR As Integer, i As Integer
Dim startDate As Date, endDate As Date, recDate As Date, extDate As Date
Dim myOlApp As Object, myOlSpace As Object, myOlFolder As Object
Dim myOlDateRange As Object, sAppoint As Object
Dim extRecurr As Object
Dim strRecurr As String
'Datum abfragen
startDate = Range("A1")
endDate = Range("B1")
'Deklaration
Set myOlApp = CreateObject("Outlook.Application")
Set myOlSpace = myOlApp.GetNamespace("MAPI")
Set myOlFolder = myOlSpace.GetDefaultFolder(olFolderCalendar)
'Einträge ab Zeile 3
myR = 3
'Löscht alle zellen in der aktiven Tabelle
Cells.ClearContents
```

```
Cells.Interior.ColorIndex = xlNone
'oder alle Zellen im Bereich bis Spalte G
'Range("A:G").ClearContents
'Range("A:G").Interior.ColorIndex = xlNone
Cells(1, 1) = startDate
Cells(1, 2) = endDate
Cells(myR - 1, 1) = "Termin"
Cells(myR - 1, 2) = "Dauer"
Cells(myR - 1, 3) = "Ende"
Cells(myR - 1, 4) = "Ort"
Cells(myR - 1, 5) = "Betreff"
Cells(myR - 1, 6) = "Textinfo"
'Datenbereich abfragen
Set myOlDateRange = myOlFolder.Items.Restrict("[Start] >= '" & startDate & "' And [End] < & '" & endDate + 1 & "'")
For Each sAppoint In myOlDateRange
   With sAppoint
      'Termindaten eintragen
      Cells(myR, 1) = Format(.Start, "dd.mm.yyyy hh:mm")
      Cells(myR, 2) = Format((((1 / 24) / 60) * .Duration), "hh:mm")
      Set extRecurr = .GetRecurrencePattern
      'OlRecurrenceType sein:
      'olRecursDaily = 1
      'olRecursMonthly = 2
      'olRecursMonthNth = 3
      'olRecursWeekly = 4
      'olRecursYearly = 5
      'olRecursYearNth = 6
      Select Case extRecurr.RecurrenceType
         Case 1
            strRecurr = "Täglich für "
         Case 2
            strRecurr = "Monatlich für "
         Case 3
            strRecurr = "Monatlich jeden "
         Case 4
            strRecurr = "Wöchentlich für "
         Case 5
            strRecurr = "Jährlich für "
         Case 6
            strRecurr = "Jährlich jeden "
      End Select
      If Format(extRecurr.PatternEndDate, "dd.mm.yyyy") <> Format(DateValue("31.12.4500"), "dd.mm.yyyy") Then
         Cells(myR, 3) = Format(DateValue(extRecurr.PatternEndDate), "dd.mm.yyyy")
         Cells(myR, 3).Interior.ColorIndex = 3
```

```
            Cells(myR, 7) = strRecurr & DateValue(Format(extRecurr.PatternEndDate, "dd.mm.yyyy")) - startDate + 1 & " Tage"
        Else
            Cells(myR, 3) = Format(.Start + (((1 / 24) / 60) * .Duration), "hh:mm")
        End If
        Cells(myR, 4) = .Location
        Cells(myR, 5) = .Subject
        Cells(myR, 6) = .Body
        myR = myR + 1
    End With
Next
'Variablen leeren
Set myOlApp = Nothing
Set myOlSpace = Nothing
Set myOlFolder = Nothing
MsgBox "Alle Termine  im Zeitraum vom " & startDate & " bis " & endDate & " eingelesen!"
End Sub
```

Excel Arbeitsmappe mit Outlook versenden

Die aktive Arbeitsmappe wird direkt als Mail Attachment mit Outlook gesendet

```
Sub Excel_Workbook_via_Outlook_Senden()
Dim Nachricht As Object, OutApp As Object
Set OutApp = CreateObject("Outlook.Application")
Dim AWS As String
'Aktive Arbeitsmappe wird als Mail gesendet
AWS = ThisWorkbook.FullName
Set Nachricht = OutApp.CreateItem(0)
With Nachricht
        .To = "irgendwer@provider"
        .Subject = "Testmeldung von Excel2000 " & Date & Time
        .attachments.Add AWS
        .Body = "Das ist ein Test." & vbCrLf & "Bitte ignorieren."
        'Hier wird die Mail nochmals angezeigt
        .Display
        'Hier wird die Mail gleich in den Postausgang gelegt
        'Mail.Send
End With
'Outlook schliessen
OutApp.Quit
Set OutApp = Nothing
```

```
Set Nachricht = Nothing
End Sub
```

Bestimmten Bereich einer Arbeitsmappe mit Outlook senden

Mit diesem Code können Sie entweder einen vorher kopierten Bereich,
oder einen bestimmten Bereich in ihrer Arbeitsmappe, mit Outlook versenden
ohne die ganze Arbeitsmappe an den Empfänger zu senden.

```
Sub Excel_Range_via_Outlook_Senden()
    Dim OutApp As Object, Mail As Object, i
    Dim Nachricht
    'Verweis auf "Microsoft Forms 2.0 Object Library" aktivieren !!
    'sonst geht es nicht
    'Dataobject wird gebraucht wegen der Zwischenablage
    Dim ClpObj As DataObject
    For i = 1 To 10
        Set ClpObj = New DataObject
        Set OutApp = CreateObject("Outlook.Application")
        Set Nachricht = OutApp.CreateItem(0)
        'Excelbereich der versendet werden soll.
'Wenn kein Bereich versendet werden soll sondern
'der Bereich bereits kopiert wurde, können sie die
'nächsten beiden Zeilen auskommentieren
        Range("A1:A5").Select
        'Bereich wird in die Zwischenablage kopiert
        Selection.copy
        With Nachricht
            .Subject = "Betreffzeile Header"
            'Zwischenablage wird eingefügt
            ClpObj.GetFromClipboard
            .Body = ClpObj.GetText(1)
            .To = "irgendwer@irgendein-provider.de"
            'Hier wird die Mail angezeigt
            '.Display
            'Hier wird die Mail gleich in den Postausgang gelegt
            .Send
        End With
        Set OutApp = Nothing
        Set Nachricht = Nothing
        'Auf Outlook warten. Ist nicht schnell genug :-))
        Application.Wait (Now + TimeValue("0:00:05"))
    Next i
End Sub
```

Ab EXCEL XP

```
Sub Send_Chart_from_Excel()
'Es geht nur wenn ds Chart aktiviert und SELEKTIERT ist
'Ohne Select geht es nicht :-))
ActiveSheet.ChartObjects("Diagramm 2").Activate
ActiveChart.ChartArea.Select
'Das anzeigen der Envelope Commandbar ist unabdingbar
ActiveWorkbook.EnvelopeVisible = True
'Nun werden die Adressen vergeben
With ActiveSheet.MailEnvelope
     .Introduction = "Das ist der Einleitungstext." & vbCrLf & "mit einer zweiten Zeile"
     .Item.To = "irgendwer@irgendwo.de"
     .Item.Subject = "Das aktuelle Diagramm"
     .Item.Send
  End With
End Sub



Sub Send_OriginalRange_from_Excel()
'Getestet unter Office XP
'Ohne Select geht es nicht :-))
Range("A1:C9").Select
'Das anzeigen der Envelope Commandbar ist unabdingbar
ActiveWorkbook.EnvelopeVisible = True
'Nun werden die Adressen vergeben
With ActiveSheet.MailEnvelope
     .Introduction = "Das ist der Einleitungstext." & vbCrLf & "mit einer zweiten Zeile"
     .Item.To = "irgendwer@irgendwo.de"
     .Item.Subject = "Die aktuellen Daten"
     .Item.Send
  End With
End Sub
```

Aufgabe mit Dateilink an Outlook senden

In diesem Beispiel erstellen Sie eine Aufgabe in Outlook und erstellen gleichzeitig
einen Hyperlink in der Aufgabe auf die aktuelle Arbeitsmappe.
Sie müssen dann, wenn Sie die Aufgabe öffnen, nur noch auf den Link klicken
um die zu kontrollierende Datei anzusehen.

```
Sub Excel_an_Outlook_Aufgabe()
On Error GoTo ErrorAufgabe
Dim MyError As Integer
Dim Faellig As Date
Dim Link As String
Dim myolApp As Object, myitem As Object
'Eigene Fehleroutine/Nummer eröffnen
MyError = 1
'Fälligkeit ist übermorgen
Faellig = Date + 1
MyError = 2
Set myolApp = CreateObject("Outlook.Application")
Set myitem = myolApp.CreateItem(3)
   myitem.Subject = "Datei ERFASSEN !" ' Text der Aufgabe
   myitem.DueDate = Faellig
   myitem.ReminderTime = Faellig
   'Ein Link kann nur erstellt werden, wenn die Pfadangabe eine
   'Angabe mit einem ShareName ist
   myitem.Body = "\\Computername\Freigabename\dateiname.xls"
   myitem.Save
Set myitem = Nothing
ErrorExit:
Exit Sub

ErrorAufgabe:
Select Case MyError
   Case 1
      MsgBox "Die Datei wurde noch nicht gespeichert"
   Case 2
      MsgBox "Outlook kann nicht gestartet werden" & Chr$(13) &_
 "Aufgabe wurde nicht erstellt !"
End Select
Resume ErrorExit
End Sub
```

Serienmail mit E-Mail Adressen aus einer Tabelle mit Outlook senden

In diesem Beispiel stehen in der aktiven Tabelle in A1 bis A10 E-Mail-Adressen (deshalb die Schleife von 1 bis 10)

```
Sub Excel_Serienmail_via_Outlook_Senden()
   Dim OutApp As Object, Mail As Object
   Dim i As Integer
   Dim Nachricht
   For i = 1 To 10
```

```
    'Variablen müssen bei jeder Schleife neu initalisiert werden
        Set OutApp = CreateObject("Outlook.Application")
        Set Nachricht = OutApp.CreateItem(0)
        With Nachricht
            .To = Cells(i, 1)'Adresse
            .Subject = Cells(i, 2) 'Betreffzeile
            .Body = Cells(i, 3) 'Sendetext
            'Hier wird die Mail gleich in den Postausgang gelegt
            'und die Sicherheitsabfrage muss jedesmall bestätigt werden
            '.Send
            'Hier wird die Mail "angezeigt"
            'aber gleich versendet,... OHNE Sicherheitsabrage
            .Display
            SendKeys "%s",True
        End With
        'Variablen zurücksetzen sonst geht es nicht
        Set OutApp = Nothing 'CreateObject("Outlook.Application")
        Set Nachricht = Nothing 'OutApp.CreateItem(0)
        Application.Wait (Now + TimeValue("0:00:05"))
    Next i
End Sub
```

Serienmail mit verschiedenen Attachments aus einer Tabelle mit Outlook senden

Ein ähnliches Beispiel wie oben mit dem Unterschied, dass die Empfänger in den Zellen stehen und
die jeweiligen Attachments (in diesem Fall 10) stehen inclusive Pfad in den Zellen F2:F10
die jeweiligen Attachments mit den Pfadangaben in den Nachbarzellen.
In diesem Beispiel wird das FileSystemObject zu Hilfe genommen um die Ordner bzw. die Dateien auf Existenz zu testen.
Das ganze könnte auch etwas einfacher gelöst werden, aber so kann das FS-Object wunderbar gezeigt werden.

```
Sub Excel_Serienmail_mit_mehreren_Anlagen_via_Outlook_Senden()
    'Variablendefinition
    Dim fs As Object, F As Object
    Dim OutApp As Object, Mail As Object
    Dim i As Integer, y As Integer, Msg As Integer
    Dim Nachricht As Variant
    Dim AWS As String
    Dim AnzEmpfänger As Integer
    'Variablen füllen
    'Filesystemobjekt erstellen
    Set fs = CreateObject("Scripting.FileSystemObject")
    'Hier die Anzahl Empfänger definieren
    'Kann auch ein Range auf der Tabelle sein
```

```
AnzEmpfänger = 10
'1. Fehlerprüfung
'Prüfen ob alle Inhalte vorhanden sind
'Wenn nicht wird das Makro abgebrochen
'In Spalte A steht der Name
'In Spalte B steht der Betreff
'In Spalte C steht der Text
For i = 1 To AnzEmpfänger
   If Cells(i, 1) = "" Or Cells(i, 2) = "" Or Cells(i, 3) = "" Then
      Msg = MsgBox("Unvollständige Angaben beim Empfänger in Zeile "_
               & i, vbCritical + vbOKOnly, "Abbruch")
      Exit Sub
   End If
Next i
'2. Fehlerprüfung
'Mit dem FilesystemObjekt wird zuerst die Existenz
'der Dateien geprüft. Wenn eine nicht existiert
'wird das Makro abgebrochen
'Die Links auf deine Anlagen liegen im
'Bereich F2 : F10
For y = 2 To 10
   'Wenn eine Zelle leer ist, wird aus der Schleife ausgestiegen
   'ohne weitere Fehlerprüfung
   If Cells(y, 6) = "" Then Exit For
   If fs.fileexists(Cells(y, 6)) = False Then
      Msg = MsgBox("Die Datei: " & Cells(y, 6) & " in F" & y & " exitstiert nicht !"_
               & vbCrLf & "Der Sendevorgang an; " & Cells(i, 1) & " wird abgebrochen!",_
                vbCritical + vbOKOnly, "Dateifehler")
      Exit Sub
   End If
Next y
'Sendevorgang einleiten
For i = 1 To AnzEmpfänger
   Set OutApp = CreateObject("Outlook.Application")
   Set Nachricht = OutApp.CreateItem(0)
   With Nachricht
      .To = Cells(i, 1) 'irgendwer@irgendein-provider.de
      .Subject = Cells(i, 2) 'Betreffzeile
      .Body = Cells(i, 3) 'Sendetext"
      For y = 2 To 10
         AWS = Cells(y, 6)
         'Wenn die Zelle / Variable leer ist
         'wird diese Schleife für die Attachments abgebrochen
         If AWS = "" Then Exit For
```

```
            .attachments.Add AWS
        Next y
        'Hier wird die Mail zuerst angezeigt
        .Display
        'Hier wird die Mail gleich in den Postausgang gelegt
        '.Send
    End With
    'Variablen zurücksetzen
    Set OutApp = Nothing
    Set Nachricht = Nothing
    'Warten auf Outlook :-))
    Application.Wait (Now + TimeValue("0:00:05"))
  Next i
End Sub
```

Bei überschreiten eines Wertes eine Mail mit Outlook senden

Diese Funktion ist in EXCEL und von MS natürlich nicht vorgesehen.
Es geht trotzdem,... wenn auch über einen kleinen Umweg :-).
Aus einer Formel können sie kein Makro direkt starten oder aufrufen,... aber das Ergebnis einer Funktion kann ein Makro ausführen. Eine benutzerdefinierte Funktion können Sie jedoch selbst erstellen,.... et voilà:
Aufgabe:
Sie haben eine Tabelle die in regelmässigen Abständen von aussen (Internet oder andere Tabellen) mit neuen Werten versorgt wird. Nun möchten Sie, z.B. am Wochenende oder wenn sich nicht im Büro sind, über E-Mail beim überschreiten eines Wertes darüber informiert werden.
In A1 steht der externe Wert, oder die Formel welche den Wert liefert.
Nur bis Office 2000 ohne Sicherheitsupdate ServicePack2
Fügen Sie in B1 diese Formel ein...

```
=WENN(A1>150;Active_Mail();"")
```

...und diese beiden Makros in ein Modul Ihrer Arbeitsmappe

```
Function Active_Mail()
'Über eine Function ein Mail auslösen
Call Active_Mail_Senden
End Function

Sub Active_Mail_Senden()
ActiveWorkbook.SendMail "dein.name@dein.provider", "Wert überschritten"
End Sub
```

Wenn der Wert in A1 150 übersteigt wird das Makro über die Funktion aufgerufen, und die Mail versandt.
Coole Variante,.... oder ?

Ab Office 2000 für alle Outlook-Versionen
Verschiedene EXCEL Interne Funktionen werden ab dem Sicherheitsupdate nicht mehr unterstützt, daher ist man nun gezwungen einen Umweg zu machen.
Anstelle der Formel verwende ich hier das CHANGE Ereignis bzw. das CALCULATE Ereignis der Mappe/Tabelle.
Das CHANGE Ereignis funktioniert bei direkter Eingabe, aber nicht bei DDE-Daten Update (z.B. verknüpfte Zellen)
Das CALCULATE Ereignis funktioniert NUR, wenn auch tatsächlich das Sheet neu berechnet wird.
Bei direkter Eingabe in einer Zelle OHNE Formeln in der Tabelle wird dieses nicht aktualisiert.
Sie müssen daher entscheiden, ob Sie bei Bedarf 2 Mails erhalten wollen, oder ob eine der beiden Varianten für Sie ausreichend ist
Dies gehört in das Klassenmodul der Tabelle, in der Sie einen Wert überwachen wollen

```
Option Explicit

Private Sub Worksheet_Calculate()
If Range("A1") > 25 Then Send_Excel_Message
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
If Range("A1") > 25 Then Send_Excel_Message
End Sub
```

Und dieses Makro gehört in ein Modul ihrer Mappe

```
Sub Send_Excel_Message()
Dim MyMessage As Object, MyOutApp As Object
'InitializeOutlook = True
Set MyOutApp = CreateObject("Outlook.Application")
'Nachrichtenobject erstellen
Set MyMessage = MyOutApp.CreateItem(0)
With MyMessage
    .To = "irgendwer@Irgenwo.de"
    .Subject = "Testmeldung von Excel2000 " & Date & Time
    'Hier wird eine normale Text Mail erstellt
    '.body = "Das ist ein Test" & vbCrLf & "Bitte ignorieren"
    'Hier wird die HTML Mail erstellt
    .HTMLBody = "Das ist ein Test." & vbCrLf & "Bitte ignorieren."
    'Hier wird die Mail nochmals angezeigt
    .Display
    'Nicht ganz offiziell :-)
    .Save
    SendKeys "%S"
End With
MyOutApp.Quit
Set MyOutApp = Nothing
Set MyMessage = Nothing
End Sub
```

Mit ".Display" wird die Mail angezeigt, mit ".Save" gespeichert (da wir Sie geändert haben), ... und mit " SendKeys "%S" " OHNE Sicherheitsabfrage mit

Outlook versandt :-)
Die SendKeys Variante ist zwar nicht die eleganteste, lässt sich dafür aber ohne Installation von Zusatztools verwenden, um die Sicherheitsabfrage zu umgehen.

Wenn es ohne Outlook gehen soll

Nicht immer hat man das mächtige Outlook installiert, ... Mail senden kann man aber auch:

```
Sub Mini_Mail()
Shell "C:\Programme\Outlook Express\msimn.exe"
ActiveWorkbook.SendMail Recipients:="irgendwer@irgendwo.com", Subject:="Test"
End Sub
```

Allerdings werden die Funktionen und Möglichkeiten von Outlook natürlich nicht unterstützt.

Einlesen von Outlook Kontakten in eine Listbox

```
Option Explicit

Sub ListBox_Fill_With_Outlook_Contacts()
'modified by Ramses
'Voraussetzung:
'1 Userform mit einer Listbox die den Namen "Listbox1" hat
'ansonsten bitte entsprechend anpassen
'--------------------
'Variablen Deklaration
Dim myOutlook  As Object
Dim conId     As Integer
Dim conFolder  As Object
Dim conItem    As Object
Dim Qe As Integer
Dim ErrMsg As String
'Bildschirmaktualisierung ausschalten
'Application.DisplayAlerts = False
'... und Statusbar-Info ausgeben
Application.StatusBar = " . . .  die Adressen werden aus Outlook eingelesen"
'Object Deklaration
Set myOutlook = CreateObject("Outlook.Application")
Set conFolder = myOutlook.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)
'Zuweisen der Anzahl Spalten in der Listbox
Me.ListBox1.ColumnCount = 7
'Zuweisen der Spaltenbreite in Pt
'1 cm ~ 28,3 Pt
Me.ListBox1.ColumnWidths = "70; 70; 28; 70; 28; 70; 70"
```

```
'Einlesen der Daten
For conId = 1 To conFolder.Items.Count
   'Zuweisen des Object für jeden Contact
   Set conItem = conFolder.Items(conId)
   'Einlesen des Contacts beginnen
   With conItem
      'Neuen Eintrag in Listbox einfügen
      Me.ListBox1.AddItem " "
      'iIndx - 1 um auf das vorher erzeugte Item zuzugreifen
      On Error GoTo conError
      Me.ListBox1.List(conId - 1, 0) = .FirstName & " " & .LastName
      'Statusbar Info
      Application.StatusBar = "Datensatz " & conId & " von " & conFolder.Items.Count & " wird gelesen: " & .FirstName
      If .BusinessAddressPostOfficeBox = "" Then
         UserForm1.ListBox1.List(conId - 1, 1) = .BusinessAddressStreet
      Else
         UserForm1.ListBox1.List(conId - 1, 1) = .BusinessAddressPostOfficeBox
      End If
      Me.ListBox1.List(conId - 1, 2) = .BusinessAddressPostalCode
      Me.ListBox1.List(conId - 1, 3) = .BusinessAddressCity
      Me.ListBox1.List(conId - 1, 4) = .CustomerID
      Me.ListBox1.List(conId - 1, 5) = .AssistantName
      Me.ListBox1.List(conId - 1, 6) = .MiddleName
errorStepin:
   End With
Next conId

ErrorExit:
'Object Variablen leeren
Set conItem = Nothing
Set conFolder = Nothing
Set myOutlook = Nothing
'Bildschirm einschalten
Application.DisplayAlerts = True
'Statusbar zurücksetzen
Application.StatusBar = False
Exit Sub

conError:
Select Case Err
   Case 438
      Set conItem = conFolder.Items(conId)
      ErrMsg = "Datensatz " & conId & " ist korrupt, oder untestützt die Abfrage nicht."
      ErrMsg = ErrMsg & vbCrLf & "Datensatzkennung:"
```

```
        ErrMsg = ErrMsg & vbCrLf & "Erstelldatum: " & conItem.CreationTime
        ErrMsg = ErrMsg & vbCrLf & "ObjectID" & conItem.EntryID
        ErrMsg = ErrMsg & vbCrLf
        ErrMsg = ErrMsg & vbCrLf & "Löschen ? "
        Qe = MsgBox(ErrMsg, vbYesNo + vbCritical + vbDefaultButton2, "Datenfehler")
        If Qe = vbYes Then
           conItem.Delete
           MsgBox ("Datensatz " & conId & " wurde gelöscht")
           Resume errorStepin
        Else
           MsgBox "Datenimport wegen Datenfehler bei Datensatz " & conId & " abgebrochen"
           Resume ErrorExit
        End If
    Case Else
        MsgBox Err & ": " & Err.Description
        Resume ErrorExit
End Select
End Sub
```

Einlesen von Kontakten in das aktuelle Tabellenblatt

```
Sub Read_Contact_from_Outlook()
'by Ramses
'Liest alle Kontakte aus Outlook in das aktuelle Tabellenblatt
Dim myOlk As Object
Dim myOlkContact As Object
Set myOlk = CreateObject("outlook.application")
Set myOlkContact = myOlk.CreateItem(olContactItem)
Range("A2").Select
For Each myOlkContact In myOlk.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts).Items
    With myOlkContact
        ActiveCell.Value = .LastName
        ActiveCell.Offset(0, 1).Value = .FirstName
        ActiveCell.Offset(0, 2).Value = .BusinessAddressStreet
        ActiveCell.Offset(0, 4).Value = .BusinessAddressCity
        ActiveCell.Offset(0, 3).Value = .BusinessAddressPostalCode
        ActiveCell.Offset(0, 5).Value = .BusinessAddressCountry
        ActiveCell.Offset(0, 6).Value = .BusinessAddressState
        ActiveCell.Offset(0, 7).Value = .Email1Address
        'Alle verfügbaren Eigenschaften eines Kontaktes
        '---Outlook 2003
        '.AutoResolvedWinner
```

```
'.HasPicture
'.AddPicture
'.RemovePicture
'---Outlook 2000 / 2002
'.Actions
'.Anniversary
'.AssistantName
'.AssistantTelephoneNumber
'.Birthday
'.Business2TelephoneNumber
'.BusinessAddress
'.BusinessAddressCity
'.BusinessAddressCountry
'.BusinessAddressPostalCode
'.BusinessAddressPostOfficeBox
'.BusinessAddressState
'.BusinessAddressStreet
'.BusinessFaxNumber
'.BusinessHomePage
'.BusinessTelephoneNumber
'.CallbackTelephoneNumber
'.CarTelephoneNumber
'.Categories
'.Children
'.Companies
'.CompanyAndFullName
'.CompanyMainTelephoneNumber
'.CompanyName
'.CreationTime
'.CustomerID
'.Department
'.DownloadState
'.Email1Address
'.Email1AddressType
'.Email1DisplayName
'.Email1EntryID
'.Email2Address
'.Email2AddressType
'.Email2DisplayName
'.Email2EntryID
'.Email3Address
'.Email3AddressType
'.Email3DisplayName
'.Email3EntryID
```

```
'.EntryID
'.FirstName
'.FTPSite
'.FullName
'.FullNameAndCompany
'.Gender    'Geschlecht
'.GovernmentIDNumber    'Passnummer
'.Hobby
'.Home2TelephoneNumber
'.HomeAddress
'.HomeAddressCity
'.HomeAddressCountry
'.HomeAddressPostalCode
'.HomeAddressPostOfficeBox
'.HomeAddressState
'.HomeAddressStreet
'.HomeFaxNumber
'.HomeTelephoneNumber
'.IMAddress    'Microsoft Instant Messenger Adresse
'.Importance    'Wichtigkeitsstufe des Kontakt
'.Initials
'.InternetFreeBusyAddress    'Frei/Gebucht-Informationen
'.ISDNNumber
'.JobTitle
'.Language
'-----Wird automatisch generiert
'.LastFirstAndSuffix    'Vor und Zuname und Suffix zusammen
'.LastFirstNoSpace    'Vor und Zuname ohne Leerzeichen
'.LastFirstNoSpaceAndSuffix
'.LastFirstNoSpaceCompany
'.LastFirstSpaceOnly
'.LastFirstSpaceOnlyCompany
'.LastNameAndFirstName
'.LastModificationTime
'---
'.LastName
'.MailingAddress
'.MailingAddressCity
'.MailingAddressCountry
'.MailingAddressPostalCode
'.MailingAddressPostOfficeBox
'.MailingAddressState
'.MailingAddressStreet
'.ManagerName
```

```
        '.MiddleName
        '.MobileTelephoneNumber
        '.NetMeetingAlias
        '.NetMeetingServer
        '.NickName
        '.NoAging
        '.OfficeLocation
        '.OrganizationalIDNumber
        '.OtherAddress
        '.OtherAddressCity
        '.OtherAddressCountry
        '.OtherAddressPostalCode
        '.OtherAddressPostOfficeBox
        '.OtherAddressState
        '.OtherAddressStreet
        '.OtherFaxNumber
        '.OtherTelephoneNumber
        '.PagerNumber
        '.PersonalHomePage
        '.PrimaryTelephoneNumber
        '.Profession
        '.RadioTelephoneNumber
        '.ReferredBy    'Kontakt empfohlen von
        '.Saved
        '.SelectedMailingAddress
        '.Sensitivity    'Vertraulichkeitsstatus des Elements
        '.Size    'Grösse in Byte der Kontaktdaten
        '.Spouse    'Partnername des Kontakt
        '.Suffix
        '.TelexNumber
        '.Title
        '.Delete
        '.Display
        '.ForwardAsVcard
        '.Move
        '.PrintOut
        '.Save
        '.SaveAs
        '.ShowCategoriesDialog
        '.AttachmentAdd
    End With
    ActiveCell.Offset(1, 0).Select
Next
Set myOlkContact = Nothing
```

```
Set myOlk = Nothing
End Sub
```

EXCEL Chart mit Outlook als HTML senden

```
Sub Send_Chart_from_Excel()
'by Ramses
'Erst ab EXCEL XP
'Es geht nur wenn ds Chart aktiviert und SELEKTIERT ist
'Ohne Select geht es nicht :-))
ActiveSheet.ChartObjects("Diagramm 2").Activate
ActiveChart.ChartArea.Select
'Das anzeigen der Envelope Commandbar ist unabdingbar
ActiveWorkbook.EnvelopeVisible = True
'Nun werden die Adressen vergeben
With ActiveSheet.MailEnvelope
      .Introduction = "Das ist der Einleitungstext." & vbCrLf & "mit einer zweiten Zeile"
      .Item.To = "irgendwer@irgendwo.de"
      .Item.Subject = "Das aktuelle Diagramm"
      .Item.Send
   End With
End Sub
```

Importiert alle Mails aus dem Posteingang in die aktuelle Tabelle als Text in Zeilen

```
  Sub Get_EMAIL_from_Outlook_in_workbook_from_Inbox_Folder_E2000()
'(C) Ramses
'Erstellt für jede Mail im Posteingang eine Tabelle und schreibt dort
'den Text der Mail hinein
Dim objOutlook As Object
Dim objnSpace As Object
Dim objFolder As Object
Dim objMsg As Object
Dim intCounter As Integer, intCount As Integer, iRow As Integer
Dim sTxt As String
Application.ScreenUpdating = False
Set objOutlook = CreateObject("Outlook.Application")
Set objnSpace = objOutlook.GetNamespace("MAPI")
Set objFolder = objnSpace.folders("Persönlicher Ordner").folders("Posteingang")
intCount = objFolder.Items.count
If intCount > 0 Then
For intCounter = 1 To intCount
```

```
    Set objMsg = objFolder.Items(intCounter)
    Worksheets.Add after:=Worksheets(Worksheets.count)
    objMsg.SaveAs ThisWorkbook.path & "\temp.txt"
    Close
    iRow = 0
    Open ThisWorkbook.path & "\temp.txt" For Input As #1
    Do Until EOF(1)
       iRow = iRow + 1
       Line Input #1, sTxt
       Cells(iRow, 1).Value = "'" & sTxt
    Loop
    Close
Next intCounter
Kill ThisWorkbook.path & "\temp.txt"
End If
Set objnSpace = Nothing
Set objFolder = Nothing
Set objMsg = Nothing
Set objOutlook = Nothing
End Sub


Sub Get_EMAIL_from_Outlook_in_workbook_from_Inbox_Folder_EXCEL_XP()
'(C) Ramses
'Erstellt für jede Mail im Posteingang eine Tabelle und schreibt dort
'den Text der Mail hinein
Dim objOutlook As Object
Dim objnSpace As Object
Dim objFolder As Object
Dim objMsg As Object
Dim intCounter As Integer, intCount As Integer, iRow As Integer
Dim sTxt As String
Application.ScreenUpdating = False
Set objOutlook = CreateObject("Outlook.Application")
Set objnSpace = objOutlook.GetNamespace("MAPI")
Set objFolder = objnSpace.folders(1).folders(2)
intCount = objFolder.Items.count
If intCount > 0 Then
    For intCounter = 1 To intCount
       Set objMsg = objFolder.Items(intCounter)
       Worksheets.Add after:=Worksheets(Worksheets.count)
       'Temporäre Datei anlegen zum einlesen
       objMsg.SaveAs ThisWorkbook.path & "\temp.txt"
       Close
```

```
        iRow = 0
        Open ThisWorkbook.path & "\temp.txt" For Input As #1
        'Mail einlesen
        Do Until EOF(1)
           iRow = iRow + 1
           Line Input #1, sTxt
           Cells(iRow, 1).Value = "'" & sTxt
        Loop
        Close
    Next intCounter
    Kill ThisWorkbook.path & "\temp.txt"
End If
Set objnSpace = Nothing
Set objFolder = Nothing
Set objMsg = Nothing
Set objOutlook = Nothing
End Sub
```

Alle Kalenderdaten aus Outlook in die aktuelle Mappe einlesen

```
Beide Codeteile gehören zusammen
Option Explicit

Sub Kalenderdaten_auf_Terminbereich_einlesen()
'(C) Ramses
'Zunächst Verweis auf OL-Bibliothek erstellen
'Early Binding ab Outlook 2003 nicht möglich
'weil die Rückgabewerte der ITEM-Indexes zufällig ist und von der
'Installation abhängt !!
'-------------
'Version Office 2000 (nicht getestet sollte aber tun)
'Dim olApp As Outlook.Application
'Dim Termin As Outlook.AppointmentItem
'Dim myTerminPatt As Outlook.RecurrencePattern
'-------------
'Set olApp = New outlook.Application
'Set Termin = olApp.CreateItem(olAppointmentItem)
'-------------
'Version XP
Dim olApp As Object
Dim Termin As Object
```

```
Set olApp = CreateObject("Outlook.Application")
'Allgemein gültig
Dim i As Long, j As Long, myErr As Integer
Dim startInput As String, startDate As Date
Dim endInput As String, endDate As Date
Dim myTerminPatt As Object
On Error GoTo myErrorhandler
'Erst mal alles löschen
Cells.ClearContents
Cells.Interior.ColorIndex = xlNone
'Startdatum abfragen
startInput = InputBox("Bitte Datum eingeben im Format ""01.01.2004""", "Datum für Terminsuche", Format(Now, "dd.mm.yyyy"))
myErr = 1
If startInput = "" Then
    MsgBox "Abbruch des Makros durch Benutzer"
    Exit Sub
ElseIf Not IsDate(DateValue(startInput)) Then
    MsgBox "Falsches Datum eingegeben"
    Exit Sub
End If
myErr = 2
endInput = InputBox("Bitte Datum eingeben im Format ""01.01.2004""", "Datum für Terminsuche", Format(DateValue(startInput) + 7, "dd.mm.yyyy"))
If endInput = "" Then
    MsgBox "Abbruch des Makros durch Benutzer"
    Exit Sub
ElseIf Not IsDate(DateValue(endInput)) Then
    MsgBox "Falsches Datum eingegeben"
    Exit Sub
End If
myErr = 0
'Variable definitiv zuweisen
startDate = DateValue(startInput)
endDate = DateValue(endInput)
'Variable fü¨r Termin neu setzen
'Set Termin = olApp.CreateItem(Appointment)
Cells(1, 1) = "Termine vom " & Format(startDate, "dd.mm.yyyy") & " bis " & Format(endDate, "dd.mm.yyyy")
i = 3
Application.ScreenUpdating = False
Cells(i, 1) = "Termin Betreff"
Cells(i, 2) = "Inhalt/Body"
Cells(i, 3) = "Start"
Cells(i, 4) = "Ende"
Cells(i, 5) = "Erinnerung Minuten"
Cells(i, 6) = "Anzeigen als"
```

```
Cells(i, 7) = "Kategorien"
Cells(i, 8) = "Erstellt am"
Range(Cells(i, 1), Cells(i, 8)).Select
Selection.Interior.ColorIndex = 15

'Durchlaufe alle Termine des aktuellen Standardkalenders
i = i + 1
For Each Termin In olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar).Items
   Set myTerminPatt = Termin.GetRecurrencePattern
   If Format(Termin.Start, "dd.mm.yyyy") >= startDate And Format(Termin.End, "dd.mm.yyyy") <= endDate Then
      If Not Termin.AllDayEvent Then Trag_ein Termin, i, False
   End If
   If myTerminPatt.RecurrenceType = olRecursDaily Then
      If Format(myTerminPatt.PatternEndDate, "dd.mm.yyyy") >= startDate Then
            Trag_ein_Recurr Termin, i, False, startDate, endDate
      End If
   End If
Next
Range("C1").Select
Range("A1:H" & Range("A1").CurrentRegion.Rows.Count).Sort Key1:=Range("C1"), Order1:=xlAscending, Header:=xlGuess

'Jetzt die Ereignisse
i = i + 1
j = i
Cells(i, 1) = "Ganzer Tag Betreff"
Cells(i, 2) = "Ereignis am"
Cells(i, 3) = "Erinnerung Minuten"
Cells(i, 4) = "Anzeigen als"
Cells(i, 5) = "Kategorien"
Cells(i, 6) = "Erstellt am"
Range(Cells(i, 1), Cells(i, 6)).Select
Selection.Interior.ColorIndex = 15
i = i + 1
For Each Termin In olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar).Items
   Debug.Print Termin.Start
   If DateSerial(Year(startDate), Month(Termin.Start), Day(Termin.Start)) >= startDate And DateSerial(Year(startDate), Month(Termin.Start), Day(Termin.Sta
rt)) <= endDate Then
      If Termin.AllDayEvent And Not Termin.IsRecurring Then Trag_ein Termin, i, True
   End If
Next
Range("C" & j).Select
Range("A1:F" & Range("A" & j).CurrentRegion.Rows.Count).Sort Key1:=Range("C1"), Order1:=xlAscending, Header:=xlGuess
'und noch die jährlichen Ereignisse
i = i + 2
```

```
j = i
Cells(i, 1) = "Betreff ""Jährliches Ereignis"""
Cells(i, 2) = "jährliches Ereignis am"
Cells(i, 3) = "Erinnerung Minuten"
Cells(i, 4) = "Anzeigen als"
Cells(i, 5) = "Kategorien"
Cells(i, 6) = "Erstellt am"
Range(Cells(i, 1), Cells(i, 6)).Select
Selection.Interior.ColorIndex = 15
i = i + 1
For Each Termin In olApp.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar).Items
    If DateSerial(Year(startDate), Month(Termin.Start), Day(Termin.Start)) >= startDate And DateSerial(Year(startDate), Month(Termin.Start), Day(Termin.Sta
rt)) <= endDate Then
        If Termin.AllDayEvent And Termin.IsRecurring Then Trag_ein Termin, i, True
    End If
Next
Range("C" & j).Select
Range("A1:F" & Range("A" & j).CurrentRegion.Rows.Count).Sort Key1:=Range("C1"), Order1:=xlAscending, Header:=xlGuess
'Variablen leeren
Set Termin = Nothing
Set olApp = Nothing
Columns("A:H").Select
Columns("A:H").EntireColumn.AutoFit
Range("A1").Select
Cells.RowHeight = "12.75"

'Ausstieg
ErrorExit:
Application.ScreenUpdating = True
If myErr = 0 And Err.Number = 0 Then
    MsgBox "Kalenderdaten eingelesen"
End If
Exit Sub

myErrorhandler:
    Select Case myErr
        Case 1
            MsgBox "Ungültiges Startdatum"
            Resume ErrorExit
        Case 2
            MsgBox "Ungültiges Enddatum"
            Resume ErrorExit
    End Select
    MsgBox Err.Number & " " & Err.Description
```

```
    Resume ErrorExit
End Sub


Sub Trag_ein(Termin, i As Long, Ereignis As Boolean)
Dim Anzeigen_als As String
Dim Erinnerung As String
Select Case Termin.BusyStatus
    Case olFree
        Anzeigen_als = "Frei"
    Case olTentative
        Anzeigen_als = "Unter Vorbehalt"
    Case olBusy
        Anzeigen_als = "Gebucht"
    Case olOutOfOffice
        Anzeigen_als = "Abwesend"
End Select
Cells(i, 1) = Termin.Subject
If Not Ereignis Then
    Cells(i, 2) = Termin.Body
    Cells(i, 3) = Termin.Start
    Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
    Cells(i, 4) = Termin.End
    Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
    Cells(i, 5) = Termin.ReminderMinutesBeforeStart
    Cells(i, 6) = Anzeigen_als
    Cells(i, 7) = Termin.Categories
    Cells(i, 8) = Termin.CreationTime
Else
    Cells(i, 2) = Termin.Start
    Cells(i, 2).NumberFormat = "dd/mm/yyyy hh:mm"
    If Termin.ReminderMinutesBeforeStart <= 60 Then
        Erinnerung = Termin.ReminderMinutesBeforeStart & " Minuten"
    ElseIf Termin.ReminderMinutesBeforeStart / 60 < 24 Then
        Erinnerung = Termin.ReminderMinutesBeforeStart / 60 & " Stunden"
    Else
        Erinnerung = Termin.ReminderMinutesBeforeStart / 60 / 24 & " Tage"
    End If
    Cells(i, 3) = Erinnerung
    Cells(i, 3).NumberFormat = "General"
    Cells(i, 4) = Anzeigen_als
    Cells(i, 5) = Termin.Categories
    Cells(i, 6) = Termin.CreationTime
End If
```

```
i = i + 1
End Sub

Sub Trag_ein_Recurr(Termin, i As Long, Ereignis As Boolean, startDate As Date, endDate As Date)
Dim Anzeigen_als As String
Dim Erinnerung As String
Dim n As Integer
Dim myReccTermin As Object
Select Case Termin.BusyStatus
   Case olFree
      Anzeigen_als = "Frei"
   Case olTentative
      Anzeigen_als = "Unter Vorbehalt"
   Case olBusy
      Anzeigen_als = "Gebucht"
   Case olOutOfOffice
      Anzeigen_als = "Abwesend"
End Select
Set myReccTermin = Termin.GetRecurrencePattern
If startDate = endDate Then
   Cells(i, 1) = Termin.Subject
   Cells(i, 3) = startDate + (i - 1)
   Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
   Cells(i, 4) = startDate + (i - 1)
   Cells(i, 4).Interior.ColorIndex = 3
   Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
   Cells(i, 5) = Termin.ReminderMinutesBeforeStart
   Cells(i, 6) = Anzeigen_als
   Cells(i, 7) = Termin.Categories
   Cells(i, 8) = Termin.CreationTime
   i = i + 1
   Set myReccTermin = Nothing
   Exit Sub
End If
If myReccTermin.PatternEndDate < endDate Then
   Debug.Print myReccTermin.PatternEndDate
   If myReccTermin.PatternStartDate > startDate Then
      For n = 1 To endDate - myReccTermin.PatternEndDate '(myReccTermin.PatternStartDate - startDate)
         Cells(i, 1) = Termin.Subject
         Cells(i, 3) = myReccTermin.PatternStartDate + n
         Cells(i, 3).Interior.ColorIndex = 3
         Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
         Cells(i, 4) = myReccTermin.PatternStartDate + n
         Cells(i, 4).Interior.ColorIndex = 3
```

```
            Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
            Cells(i, 5) = Termin.ReminderMinutesBeforeStart
            Cells(i, 6) = Anzeigen_als
            Select Case myReccTermin.RecurrenceType
               Case 1
                  Cells(i, 7) = "Täglich"
               Case 2, 3
                  Cells(i, 7) = "Monatlich"
               Case 4
                  Cells(i, 7) = "Wöchentlich"
               Case 5, 6
                  Cells(i, 7) = "Jährlich"
               Case Else
                  Cells(i, 7) = "Serie"
            End Select
            Cells(i, 8) = Termin.CreationTime
            i = i + 1
         Next n
      Else
         For n = 1 To myReccTermin.PatternEndDate - startDate
            Cells(i, 1) = Termin.Subject
            Cells(i, 3) = startDate + n
            Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
            Cells(i, 4) = startDate + n
            Cells(i, 4).Interior.ColorIndex = 3
            Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
            Cells(i, 5) = Termin.ReminderMinutesBeforeStart
            Cells(i, 6) = Anzeigen_als
            Select Case myReccTermin.RecurrenceType
               Case 1
                  Cells(i, 7) = "Täglich"
               Case 2, 3
                  Cells(i, 7) = "Monatlich"
               Case 4
                  Cells(i, 7) = "Wöchentlich"
               Case 5, 6
                  Cells(i, 7) = "Jährlich"
               Case Else
                  Cells(i, 7) = "Serie"
            End Select
            Cells(i, 8) = Termin.CreationTime
            i = i + 1
         Next n
      End If
```

```
End If

If myReccTermin.PatternEndDate > endDate Then
   If myReccTermin.PatternStartDate > startDate Then
      For n = 1 To endDate - myReccTermin.PatternStartDate
         Cells(i, 1) = Termin.Subject
         Cells(i, 3) = myReccTermin.PatternStartDate + n
         Cells(i, 3).Interior.ColorIndex = 3
         Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
         Cells(i, 4) = myReccTermin.PatternStartDate + n
         Cells(i, 4).Interior.ColorIndex = 3
         Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
         Cells(i, 5) = Termin.ReminderMinutesBeforeStart
         Cells(i, 6) = Anzeigen_als
         Select Case myReccTermin.RecurrenceType
            Case 1
               Cells(i, 7) = "Täglich"
            Case 2, 3
               Cells(i, 7) = "Monatlich"
            Case 4
               Cells(i, 7) = "Wöchentlich"
            Case 5, 6
               Cells(i, 7) = "Jährlich"
            Case Else
               Cells(i, 7) = "Serie"
         End Select
         Cells(i, 8) = Termin.CreationTime
         i = i + 1
      Next n
   Else
      For n = 1 To myReccTermin.PatternEndDate - startDate
         Cells(i, 1) = Termin.Subject
         Cells(i, 3) = startDate + n
         Cells(i, 3).Interior.ColorIndex = 3
         Cells(i, 3).NumberFormat = "dd/mm/yyyy hh:mm"
         Cells(i, 4) = startDate + n
         Cells(i, 4).Interior.ColorIndex = 3
         Cells(i, 4).NumberFormat = "dd/mm/yyyy hh:mm"
         Cells(i, 5) = Termin.ReminderMinutesBeforeStart
         Cells(i, 6) = Anzeigen_als
         Select Case myReccTermin.RecurrenceType
            Case 1
               Cells(i, 7) = "Täglich"
            Case 2, 3
```

```
                Cells(i, 7) = "Monatlich"
            Case 4
                Cells(i, 7) = "Wöchentlich"
            Case 5, 6
                Cells(i, 7) = "Jährlich"
            Case Else
                Cells(i, 7) = "Serie"
        End Select
        Cells(i, 8) = Termin.CreationTime
        i = i + 1
    Next n
  End If
End If
Set myReccTermin = Nothing
End Sub
```

Hier die Tabelle zum Download: outkaldaten.xls

Füllt eine Listbox in einer Userform mit Outlook Kontakten

Der Code füllt eine Listbox in einer Userform. Der Benefit dabei, es werden korrupte Datensätze gefunden, die bei einer Synchronisation mit einem PDA z.B. nicht synchronisiert werden können, und zum Löschen angeboten.

```
Sub ListBox_Fill_With_Outlook_Contacts()
'(C) by Ramses
'Variablen Deklaration
Dim outId     As Integer
Dim outFolder  As Object
Dim myOutlook  As Object
Dim outItem    As Object
Dim Qe As Integer
Dim ErrMsg As String
'Bildschirmaktualisierung ausschalten
'Application.DisplayAlerts = False
'... und Statusbar-Info ausgeben
Application.StatusBar = "   die Adressen werden aus Outlook geholt " _
   & " - das kann einen Moment dauern."
'Object Deklaration
Set myOutlook = CreateObject("Outlook.Application")
Set outFolder = myOutlook.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)
'Zuweisen der Anzahl Spalten in der Listbox
UserForm1.ListBox1.ColumnCount = 7
'Zuweisen der Spaltenbreite in Pt
'1 cm ~ 28,3 Pt
```

```
UserForm1.ListBox1.ColumnWidths = "70; 70; 28; 70; 28; 70; 70"
'Einlesen der Daten
For outId = 1 To outFolder.Items.count
    'Zuweisen des Object für jeden Contact
    Set outItem = outFolder.Items(outId)
    'Einlesen des Contacts beginnen
    With outItem
        'Neuen Eintrag in Listbox einfügen
        UserForm1.ListBox1.AddItem " "
        'iIndx - 1 um auf das vorher erzeugte Item zuzugreifen
        On Error GoTo outError
        UserForm1.ListBox1.List(outId - 1, 0) = .FirstName & " " & .LastName
        'Statusbar Infor
        Application.StatusBar = "Datensatz " & outId & " von " & outFolder.Items.count & " wird gelesen: " & .FirstName
        If .BusinessAddressPostOfficeBox = "" Then
            UserForm1.ListBox1.List(outId - 1, 1) = .BusinessAddressStreet
        Else
            UserForm1.ListBox1.List(outId - 1, 1) = .BusinessAddressPostOfficeBox
        End If
        UserForm1.ListBox1.List(outId - 1, 2) = .BusinessAddressPostalCode
        UserForm1.ListBox1.List(outId - 1, 3) = .BusinessAddressCity
        UserForm1.ListBox1.List(outId - 1, 4) = .CustomerID
        UserForm1.ListBox1.List(outId - 1, 5) = .AssistantName
        UserForm1.ListBox1.List(outId - 1, 6) = .MiddleName
errorStepin:
    End With
Next outId

ErrorExit:
'Object Variablen leeren
Set outItem = Nothing
Set outFolder = Nothing
Set myOutlook = Nothing
'Bildschirm einschalten
Application.DisplayAlerts = True
'Statusbar zurücksetzen
Application.StatusBar = False
Exit Sub

outError:
Select Case Err
    Case 438
        Set conItem = outFolder.Items(outId)
        ErrMsg = "Datensatz " & outId & " ist korrupt, oder untestützt die Abfrage nicht."
```

```
        ErrMsg = ErrMsg & vbCrLf & "Datensatzkennung:"
        ErrMsg = ErrMsg & vbCrLf & "Erstelldatum: " & conItem.CreationTime
        ErrMsg = ErrMsg & vbCrLf & "ObjectID" & conItem.EntryID
        ErrMsg = ErrMsg & vbCrLf
        ErrMsg = ErrMsg & vbCrLf & "Löschen ? "
        Qe = MsgBox(ErrMsg, vbYesNo + vbCritical + vbDefaultButton2, "Datenfehler")
        If Qe = vbYes Then
           outItem.Delete
           MsgBox ("Datensatz " & outId & " wurde gelöscht")
           Resume errorStepin
        Else
           MsgBox "Datenimport wegen Datenfehler bei Datensatz " & outId & " abgebrochen"
           Resume ErrorExit
        End If
    Case Else
        MsgBox Err & ": " & Err.Description
        Resume ErrorExit
End Select
End Sub
```

EXCEL Tabelle mit Outlook senden

Es ist grundsätzlich nicht möglich, eine einzelne Tabelle aus einer Mappe als Anlage zu senden, weil die Tabelle ein integrierter Bestandteil einer Mappe ist.
Eine Variante ist daher, die Tabelle in eine Arbeitsmappe zu exportieren, und diese Mappe, nur mit der Tabelle als Inhalt, als Attachment zu versenden.

```
 Sub Excel_Sheet_via_Outlook_Senden()
   Dim Nachricht As Object, OutApp As Object
   Dim SavePath As String
   Dim AWS As String
   SavePath = "E:\Eigene Dateien"
   Set OutApp = CreateObject("Outlook.Application")
   'Kopiert aktuelles Sheet in eine neue Mappe
   'welche nur diese Tabelle enthält
   ActiveSheet.Copy
   'Speichert die Datei unter dem Tabellennamen und dem Namen in A1
   ActiveWorkbook.SaveAs SavePath & "\" & ActiveSheet.Name & " " & ActiveSheet.Range("A1")
   'Aktive Arbeitsmappe wird als mail gesendet
   AWS = ActiveWorkbook.FullName
   'InitializeOutlook = True
   Set Nachricht = OutApp.CreateItem(0)
   With Nachricht
      .To = "irgendwer@Irgenwo.de"
      .Subject = "Testmeldung von Excel2000 " & Date & Time
      .Attachments.Add AWS
```

```
        'Hier wird eine normale Text Mail erstellt
        '.body = "Das ist ein Test" & vbCrLf & "Bitte ignorieren"
        'Hier wird die HTML Mail erstellt
        .HTMLBody = "Das ist ein Test." & vbCrLf & "Bitte ignorieren."
        'Hier wird die Mail nochmals angezeigt
        .Display
        'Hier wird die Mail gleich in den Postausgang gelegt
        '.Send
        'Hier könnte die Datei wieder gelöscht werden
        'Kill AWS
    End With
    OutApp.Quit
    Set OutApp = Nothing
    Set Nachricht = Nothing
End Sub
```

EXCEL-APPLIKATION

-→ Grundlagen EXCEL OBJEKTE

There are many categories (classes) of Excel objects that can be controlled in VBA. In fact, nearly all objects can be controlled in VBA that users manipulate in the Excel interface. VBA can also control more than the Excel interface provides which is one of the key reasons why '*power users*' use VBA!

The Object hierarchy provides the levels of various key objects ranging from the cell ranges (the lowest level) through to the application itself (the highest level).

This section focuses on the **Application**, **WorkBook(s), Worksheet(s)** and **ActiveSheet/Workbook** objects (see Range & Selection objects for more extended information).

Application object

The word **Application** refers to the host (in this case Excel) and is deemed the top level object.

(Note: VBA can communicate beyond Excel and technically this is not the top level as you have the ability to code to Microsoft Office (Word, PowerPoint etc) and to other applications including the operating system).

Use this object as the entry point (the gateway) to the Excel object model and is implicit which means that you can omit this keyword in your code as it's the default. The following two VBA commands do the same thing:

```
Application.ActiveSheet.Name = "January"
```

```
ActiveSheet.Name = "January"
```

The first example included the **Application** object keyword (as explicit) and the second one excluded (as implicit) it but produced the same result.

You only need to use this keyword if you are coding with other applications (that is not Excel) or wish to communicate to Excel from another application's environment (i.e. Microsoft Word). You will need to learn about object variables and set application objects to Excel.

The following code snippet creates an Excel object from outside of Excel (which uses VBA too) and opens a workbook called "Sales.xlsx":

```
Sub OpenExcelWorkbook()
      Dim xl As Object
      Set xl = CreateObject("Excel.Sheet")
      xl.Application.WorkBooks.Open("Sales.xlsx")
      'executed code continues...
```

```
End Sub
```

ActiveWorkbook and Workbooks objects

This object appears below the **Application** object along with other key objects including **Chart** and **Pivot Table** and control the tasks for any workbook from creating, opening, printing to saving and closing documents.

The singular keyword **Workbook** refers to the current or a single file you wish to control compared with the plural keyword **Workbooks** which is the collection of one or more documents you wish to control

Use the **Workbook** object referred in code as **ActiveWorkbook** to open, save, print, close and manipulate the documents attributes as required.

```
Sub WorkBookNameExample()

      MsgBox "Current workbook is " & ActiveWorkbook.Name

End Sub
```

Save a copy of the current workbook:

```
Sub SaveAsWorkBookExample1()

      ActiveWorkbook.SaveAs "VBA Workbook.xlsx"

End Sub
```

The above can also be expressed as follows:

```
Sub SaveAsWorkBookExample2()

      Workbooks(1).SaveAs "VBA Workbook.xlsx"

End Sub
```

Using the **Workbooks** keyword which is a collection of current workbooks, you can provide an index number (starting at 1 for the first document and incrementing by 1 for each open document) to execute code using the same identifiers as **ActiveWorkbook** object.

How many workbooks are currently open?

```
Sub WorkBookCount()

      MsgBox "There are currently " & Workbooks.Count & _
                                          " workbook(s) open"

End Sub
```

The **Workbooks** object doesn't have any parenthesis and an index number reference when dealing with a collection of many documents.

(Note: the above will also count all open and hidden documents).

ActiveSheet and Worksheets objects

Most of the time, you will work with this object along the range object as the normal practice is worksheet management in a workbook when working with the Excel interface.

Again, the singular **Worksheet** object referred as **ActiveWorkSheet** controls the current or single worksheet objects including its name. The plural keyword **Worksheets** refers to one or more worksheets in a workbook which allows you to manipulate a collection of worksheets in one go.

Name a worksheet:

```
Sub RenameWorksheetExample1()

      ActiveWorkSheet.Name = "January"

End Sub
```

or use

```
Sub RenameWorksheetExample2()

      WorkSheets(1).Name = "January"

End Sub
```

assuming the first worksheet is to be renamed.

Insert a new worksheet and place it at the end of the current worksheets:

```
Sub InsertWorksheet1()

      Worksheets.Add After:=Worksheets(Worksheets.Count)

End Sub
```

or it can shortened using the **Sheets** keyword instead:

```
Sub InsertWorksheet2()

      Sheets.Add After:=Sheets(Sheets.Count)

End Sub
```

(Note: Have you noticed when adding a new worksheet via Excel interface how it always inserts it to the left of the active sheet!).

'Active' objects

Within the **Application** object you have other properties which act as shortcuts (*Globals*) to the main objects directly below it. These include **ActiveCell**, **ActiveChart**, **ActivePrinter**, **ActiveSheet**, **ActiveWindow** and **ActiveWorkbook**.

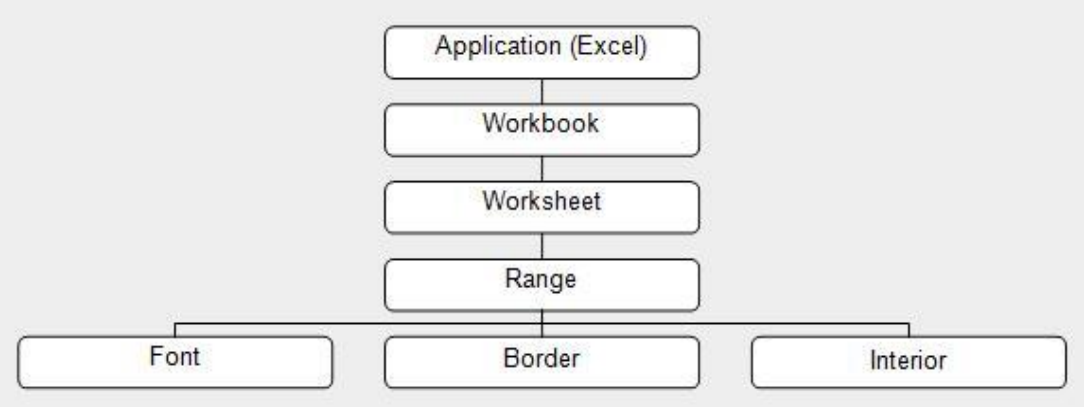
You use the above keywords as a direct implicit reference to the singular active object in the same way (as in the above already illustrated).

Remember, you can only have one active object when working in the Excel interface and therefore the VBA code is emulating the way users are conditioned to work. Even when a range of cells is selected (Selection object) only on cell is active (the white cell).

```
Sub PrinterName()

      MsgBox "Printer currently set is " & ActivePrinter

End Sub
```

OBJECT HIERARCHY

Most applications consist of many objects arranged in a hierarchy.



Objects, Methods, Properties and Variables

Each line of code generally has the same structure (which is also known as **Syntax**). VBA is loosely based around the concept of **Object Orientated Programming** (OOP) and the following syntax is used to define how you write code using any of the libraries that are loaded.

OBJECT.*Identifier*[.*sub_Identifier*]

The square brackets wrapped around the *sub_Identifier* is the convention meaning it is optional and therefore not always required.

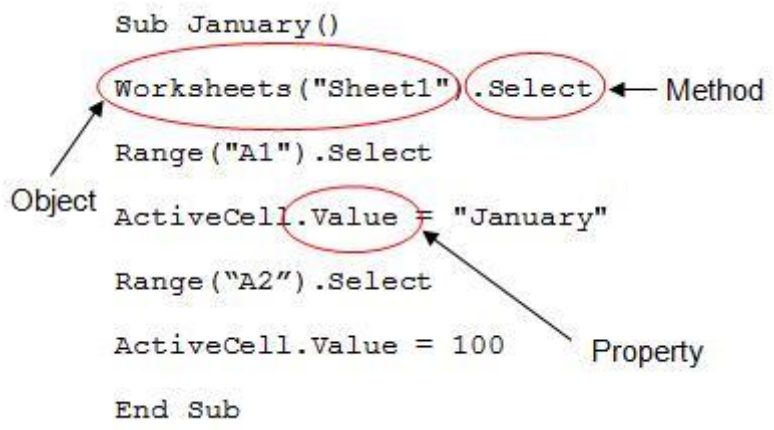
An *Identifier* and *sub_Identifier* can be one of three types:

1. Property
2. Method
3. Event

Similar to physical objects such as a car or a chair, the application objects, as listed above, have **Properties** and **Methods** (*as well as* ***Events***)

Object	**Property**	**Method**
`Car`	`Colour`	`Accelerate`
`ActiveCell`	`Value`	
`Worksheets("Sheet1")`		`Select`

Identifying the Objects, Methods and Properties from the previous example.



Examples

1. Create a new blank workbook.
2. Click on the **Tools** menu, select **Macro** and choose Visual Basic Editor.
3. Click on the **Insert** menu and select **Module**.

Properties

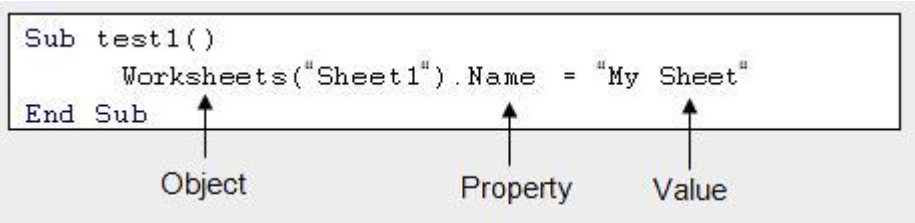
A **Property** is an attribute of an object, e.g. the colour of a car, the name of a worksheet.

Object.Property = Value

```
Car.Colour = Red
```

```
Worksheets("Sheet1").Name = "My Sheet"
```

The following example will change the name of "Sheet1" to "My Sheet".



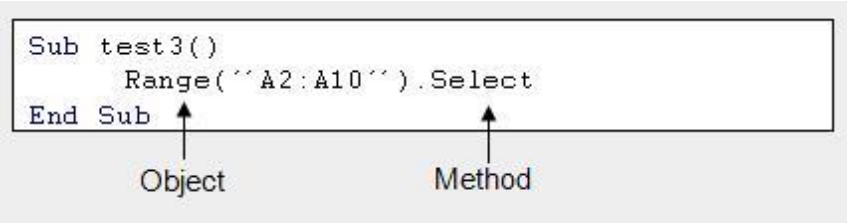
Methods

A Method is an activity that an object can be told to do, e.g. accelerate the car, select a cell, insert a worksheet, delete a worksheet.

Object.Method

```
Car.Accelerate
Range("A2:A10").Select
```

The following example will select a range of cells (A2 to A10) in the current worksheet.



Methods that contain arguments

There are methods that contain many arguments, for example inserting a worksheet(s). The numerous arguments contain information about how many worksheets you would like to insert, the position of the worksheet(s) and the type of the worksheet(s).

Object.Method Argument1,Argument2,...

Example:

Worksheets.Add Before, After, Count, Type

Add	Add a new worksheet.
Before/After	Before which worksheet? After which worksheet?
Count	How many worksheets
Type	What type of worksheet ie worksheet, chart sheet etc

The following example will place 2 new sheets after Sheet 2.

```
Worksheets.Add, Sheets("Sheet2"), 2
```

- The comma after **Add** represents the "**Before**" argument.
- If "Type" is omitted, then it will assume the Default Type. The Default Type is **xlworksheet**.

```
Sub Insert2Sheets()

      Worksheets.Add Sheets("Sheet2"), 2

End Sub
```

Events

Events are like Methods the only difference being when and who/what calls this action. It is an action like a method but the system will trigger it for you instead of the user invoking the action.

This is very useful when a system changes state and needs to automate a procedure on behalf of the user.

In fact, there are many events being triggered all the time; users are simply not aware of this unless there is a procedure to trigger it. The system constantly listens for the event to take place.

When you use standard Excel features like **Data Validation** or **Conditional Formatting**, the system automatically creates code for an event so when users enter a value in a cell it will automatically trigger the feature and validate and format the active cell without a user calling the a macro manually. This type of event is normally known as '**Enter**' for a Worksheet.

There are many predefines events which have no code just a starting and ending signature and users need to add code in between these signatures. Now take a look at Event Handling for more information:

EVENT HANDLING

An Event is something that happens in a program such as:

- Opening or closing a workbook
- Saving a workbook
- Activating or deactivating a worksheet or window
- Pressing a key or key combinations
- Entering/Editing data in the worksheet
- Clicking the mouse on a control/object
- Double clicking on a cell
- Data in a chart is updated
- Recalculating the worksheet
- A particular time of day occurs

You can therefore run a procedure automatically when a certain Event in Excel occurs.

There are different objects (and therefore different levels) when Excel automatically triggers a procedure as the system is constantly *listening* for the event to occur.

Workbook Events

Open Event

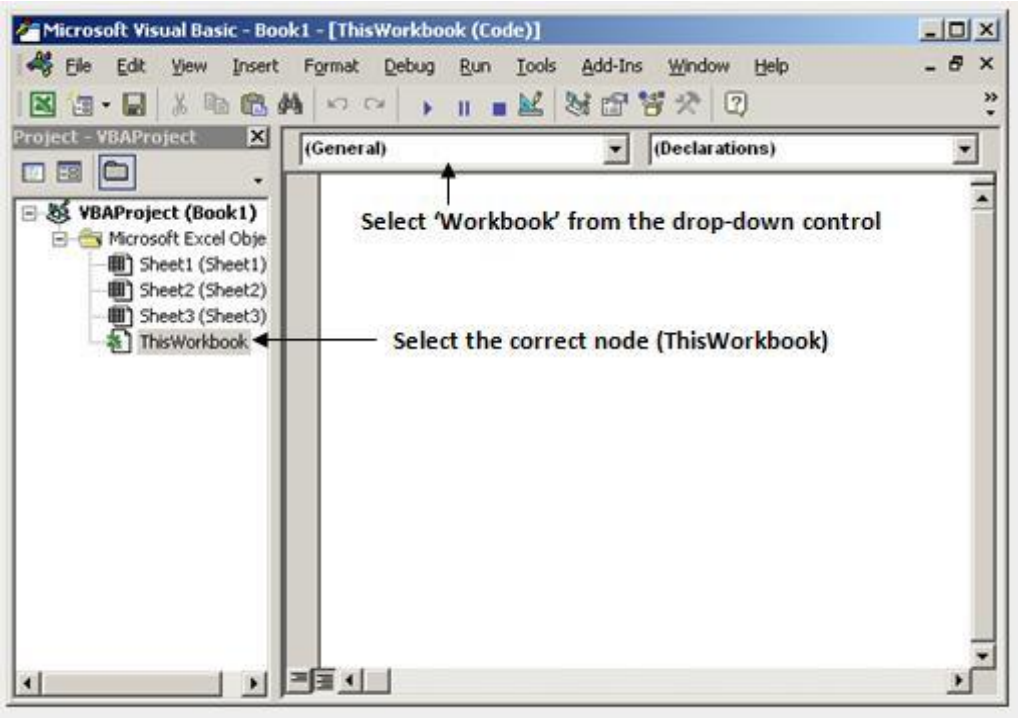
The most common type of Open Event is **`Workbook_Open`**. This procedure is executed when the workbook is opened and is often used for the tasks such as:

- Displaying a welcome message
- Opening other workbooks
- Setting up custom menus and toolbars
- Activating a particular sheet or cell

Example:

Every time the user opens the workbook, they are greeted with a message box displaying the day of the week. If it is a Friday, a message box will remind the user to submit their timesheet.

1. Open the required workbook.
2. Switch to the Visual Basic Editor.
3. Double click on **ThisWorkbook** from within the Project Explorer.



4. Click on the **Object** drop down list and select **Workbook**

5. Enter the following between the signature `Private Sub Workbook_Open()` and `End Sub` keywords:

```
Private Sub Workbook_Open()
    MsgBox "Today is " & WeekdayName(Weekday(Now), False, vbSunday)
    If Weekday(Now) = vbFriday Then
        MsgBox "Don't forget to submit your timesheet"
    End If
End Sub
```

Note: **Private** means that the procedure won't appear in the Run Procedure dialog box (i.e. Macros dialog). See Scope & Visibility in Variables & Constants for information.

Workbook Activate Event

The procedure is executed whenever the workbook is activated (gets the focus).

Example:

Call the signature **Private Sub Workbook_Activate()** using the same methods as previously explained above.

Enter the following code:

```
Private Sub Workbook_Activate()
    ActiveWindow.WindowState = xlMaximized
End Sub
```

Now the window will always maximise when the workbook gets the focus.

Note: Deleting an event (the signature) will not harm the system as it is re-generated each time you call one of the pre-defined signatures.

Example:

Using the **Private Sub Workbook_SheetActivate(ByVal Sh As Object)** signature is triggered across any worksheet in the active workbook.

Enter the following code:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    Range("A1").Value = Date 'Enters the current date in A1
    Range("A2").Select 'Position the cursor in A2
End Sub
```

The **'Sh'** argument can also be used to refer to which worksheet is being called should you wish to control the index or name of a particular worksheet or group of worksheets.

By including a code line: **If Sh.Name = "Sheet3" Then...** it will handle the logic and control flow for 'Sheet3'.

Worksheet Events

Worksheet Activate Event

Within a workbook you also have separate nodes for each added worksheet chart sheet which contain a private (local) module over an above standard modules in a VBA project.

Example:

Every time the user clicks on 'Sheet1' if the first cell (A1) is empty then prompt the user with an InputBox function to enter a title.

```
Private Sub Worksheet_Activate()
    If Trim(Range("A1").value) = Empty Then
        Range("A1").Value = Trim(InputBox("Enter title:"))
        Range("A1").EntireColumn.AutoFit
    End If
End Sub
```

Note: If there are events at both the worksheet and workbook level which point to the same object (worksheet), then it's the worksheet level will run first followed by the workbook event.

Other Events

There are other ways to get Excel to trigger a macro using other events from other objects or controls.

It is possible to attach procedures to the ActiveX Controls so that whenever the user clicks onto a control, the procedure will run.

Example:

When the user clicks on the **Command Button**, a message box will appear.

1. From Excel, click on the **Developer** tab (Ribbon Bar), select **Insert** icon and choose **Button** icon from the Form Control section.
2. Draw the Command Button onto the spreadsheet.
3. The **Assign Macro** dialog box appears, Click the **New...** button.
4. Enter the following code:

```
Sub Button37_Click()
    MsgBox "Button click event!"
End Sub
```

Any control drawn on a worksheet or user form will have pre-defined events that can be coded to respond by the system.

How do you think features like *conditional formatting* and *data validation* work in a worksheet when set in Excel? When the user enters a value in a cell, the **Change** event is triggered:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Target = Range("A2") Then Range("A2").Font.Bold = True
```

```
End Sub
```

Target is the argument to test which cell address is being changed.

ERROR HANDLING

No matter how thorough you are when writing code, errors can and will happen.

There are steps that developers can take to help reduce unwanted errors and this is considered just as important as the actual process of the procedure.

Before understanding and applying error-handling routines, planning to avoid errors should be undertaken.

- Design the procedure's process electronically or on paper – flow chart and paper test.
- Creating smaller portions of code – snippets to be called and re-used
- Using the Option Explicit statement – declaring your variables officially.
- Syntax checking – user defined commands and functions.
- Comments – remarking your code at various points.
- Testing application – functional and usability.

Note: Some of the above points are methodologies which are outside the scope of this reference guide.

There are three different types of errors:

1. Design Time Errors
2. Run Time Errors

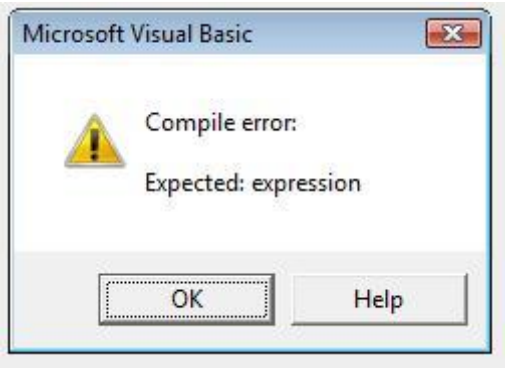
3. Logical Errors

The order of the above progressively is harder to find leaving the last item the most challenging!

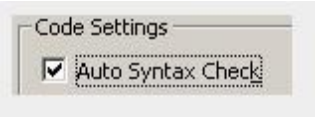
Design Time Errors

The simplest form of error and often caused by typing (typo's) mistakes.

When typing a known keyword or statement, VBA will turn the text to red (*default colour*) and if the option is enabled, provide a prompt:



To switch off the above prompt, go to **Tools** select **Options...** and deselect **Auto Syntax Check** option.



The routine will instantly cause a run time error if not corrected at the design time and must but resolved before macros can run.

Run Time Errors

When executing code, no matter how thorough the debugging process has been, code may encounter errors while running.

There is only one way of doing this - **On Error GoTo** instruction. It is not a very sophisticated function, but it allows the flow of the code to continue and also where applicable, prevent infinite loops (*when the computer keeps on calculating never coming to an end*).

Three variations are available:

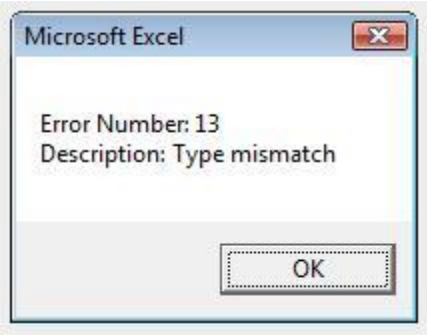
1. **On Error GoTo** *LabelName*
2. **On Error Resume Next**
3. **On Error GoTo 0**

On Error GoTo *LabelName* branches to the portion of the code with the label *LabelName* (*'LabelName' must be a text string and not a value*).

These commands are usually placed at the beginning of the procedure and when the error occurs, the macro will branch to another part of the procedure and continue executing code or end, depending on the instruction given.

```
'Simple Error handler with Err Object
Sub ErrorTestOne()
    On Error GoTo myHandler

    Dim intDay As Integer
    intDay = "Monday"
    MsgBox intDay
    Exit Sub

myHandler:
    MsgBox "Error Number: " & Err.Number & vbNewLine _
                            & "Description: " & Err.Description
End Sub
```

The above procedure will cause an error when executed and users will see:



myHandler is a user defined label (*must not use known keywords*) which listens for any errors that may occur. When an error is detected, the procedure jumps to a bookmark of the same label with a colon (**:**) (**myHandler:**) and executes from that point forward.

Using the **Err** object, developers can return two common properties '**Number**' and '**Description**'. The example message box concatenates these two properties into a user-friendly message (*see above*).

It is important to include the **Exit Sub** statement prior to the bookmark label otherwise the procedure will execute to the very end of the sub routine and should only be executed when an error has genuinely occurred.

The error above was due to a type mismatch. In other words I declared a variable **intDay** as an **integer** and assigned a string value to it.

Another example:

```
'Error to handle incorrect InpuBox value.
Sub ErrorTestTwo()
    On Error GoTo myHandler

    Dim intInput As Integer
    Dim strResponse As String
    Dim blnErr As Boolean
    intInput = CInt(InputBox("Enter your age:"))
    blnErr = False
    If Not blnErr Then
```

```
        If intInput > 64 Then
            strResponse = "You are at the retirement age!"
        Else
            strResponse = "You have " & (65 - intInput) & _
                " year(s) remaining until retirement."
        End If
    Else
        strResponse = "Unknown error entered!"
    End If
    MsgBox strResponse
    Exit Sub

myHandler:
    intInput = 0
    blnErr = True
    Resume Next
End Sub
```

The above example illustrates how to gracefully handle incorrect (*type mismatched)* values and then resume the next line of execution using `Resume Next` statement.

The variable `blnErr` is flagged as true if an error occurs which is then tested with an **`If`** statement.

If the `Resume Next` is replaced with just the `Resume` statement, you will find the input box will loop itself until the correct data is entered. Be careful before testing this out due to infinite loops that may occur (*if you edit the wrong part of the procedure*).

The statement **`On Error GoTo 0`** (*zero*) simply disables the error command during the procedure.

Should users wish to switch off this feature? To switch it back on, just introduce a new statement line of either:

1. **On Error Goto** *myLabel*
2. **On Error Resume**
3. **On Error Resume Next**

Any code can be written to handle errors gracefully which can include **If** and **Case** statements. It is common to have a **Case** statement to test which error was fired and deal with it in a separate calling procedure (*branch out another procedure*).

Logical Errors

This type of error is the most difficult to trace as its syntax is correct and it runs without any run time errors.

A **logical error** is one that does not give users any indication that an error has occurred due to the fact that a logical error is the *process of logic* and not the code itself.

Performing a calculation in a spreadsheet using a function will return an answer but is it the correct answer?

Example:

```
'Logical Error Test Example
Sub LogicalErrorTest()
    Dim lngQty As Long
    Dim dblNet As Double
    Dim sngDiscount As Single

    lngQty = 10
    dblUPrice = 250
    sngDiscount = 0.15

    'Calculate gross (inc VAT @ 17.5%)
    'Logically INCORRECT!
    MsgBox Round(lngQty * dblUPrice * 1 - sngDiscount * _ 1.175, 2)

    'Logically CORRECT!
    MsgBox Round(((lngQty * dblUPrice) * (1 - sngDiscount))
End Sub
```

The above procedure showed a quantity (**intQty**) of goods, with a unit price (**dblUPrice**), a discount (**sngDiscount**) at a fixed vat rate of 17.5%.

To calculate the correct gross value, there is an order of which operands are calculated (see Formulas) first and without the care of using brackets, the system follows the rules of mathematics and looks at the operator's precedence automatically.

The first message box shows:

WRONG!

Followed by the second message box:

CORRECT!

Both calculations worked but the first was illogical to the objective of the process (workflow).

How we find such errors? ***Debugging:***

DEBUGGING

Debugging is the process of stepping through the code line by line and checking the reaction of each line to help trace errors that may be difficult to find at run time especially logical errors.

The **Debug** toolbar allows users to step in, out, over or watch certain variables change state in a controlled manner and can be switched on or off in the Visual Basic Editor window.

1 **Design Mode**.

2 **Run Sub/User Form** starts the macros where the insertion point is or displays a Macro Dialog Box.

3 **Break** pauses the macro while it's running and switches to break mode.

4 **Reset** current macro clearing all breaks, step into/over procedures and variables.

5 **Toggle Breakpoint** allows marking a line of code at which point a macro will stop.

6 **Step Into** a macro one line at a time.

7 **Step Over** a macro one line at a time ignoring any other sub routines.

8 **Step Out** over a macro and continue running the rest of that macro.

9 **Locals Window** is displayed showing all variables and expressions with values for the procedure currently running.

10 **Immediate Window** is displayed allowing pasting of code to the window and testing the code by using the ENTER key (*cannot save contents*).

11 **Watch Window** is displayed allowing drag 'n' drop of expressions into it to

monitor their values.

12 **Quick Watch** displays a Dialog Box showing the current line of codes value.

13 **Call Stack** displays a Dialog Box listing all active calls statement to the current procedure. This option is used when using a step procedure.

The most effective way to debug a procedure is to learn some keystrokes and mark breakpoints in the code.

To add breakpoints, place the mouse pointer to the left grey margin at the point where you wish to pause the procedure and click once with the left mouse button, click button 5 (*as above*) or press **F9** function key (toggles on/off).

```
Sub CalcPay()
    On Error GoTo HandleError
    Dim hours
    Dim hourlyPay
    Dim payPerWeek
    hours = InputBox("Please enter number of hours worked", "Hours Worked")
    hourlyPay = InputBox("Please enter hourly pay", "Pay Rate")
    payPerWeek = CCur(hours * hourlyPay)
    MsgBox "Pay is:    " & Format(payPerWeek, "£##,##0.00"), , "Total Pay"
HandleError: 'any error - gracefully end
End Sub
```

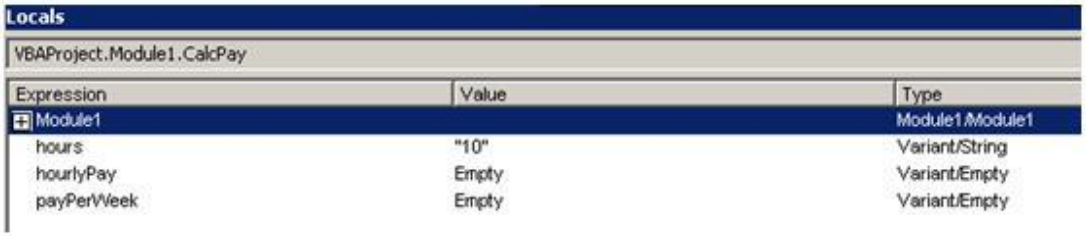
When you run the procedure or press the **F5** key, the procedure will pause at the first highlighted break:

```
Sub CalcPay()
    On Error GoTo HandleError
    Dim hours
    Dim hourlyPay
    Dim payPerWeek
    hours = InputBox("Please enter number of hours worked", "Hours Worked")
    hourlyPay = InputBox("Please enter hourly pay", "Pay Rate")
    payPerWeek = CCur(hours * hourlyPay)
    MsgBox "Pay is: hours = Empty mat(payPerWeek, "£##,##0.00"), , "Total Pay"
HandleError: 'any error - gracefully end
End Sub
```

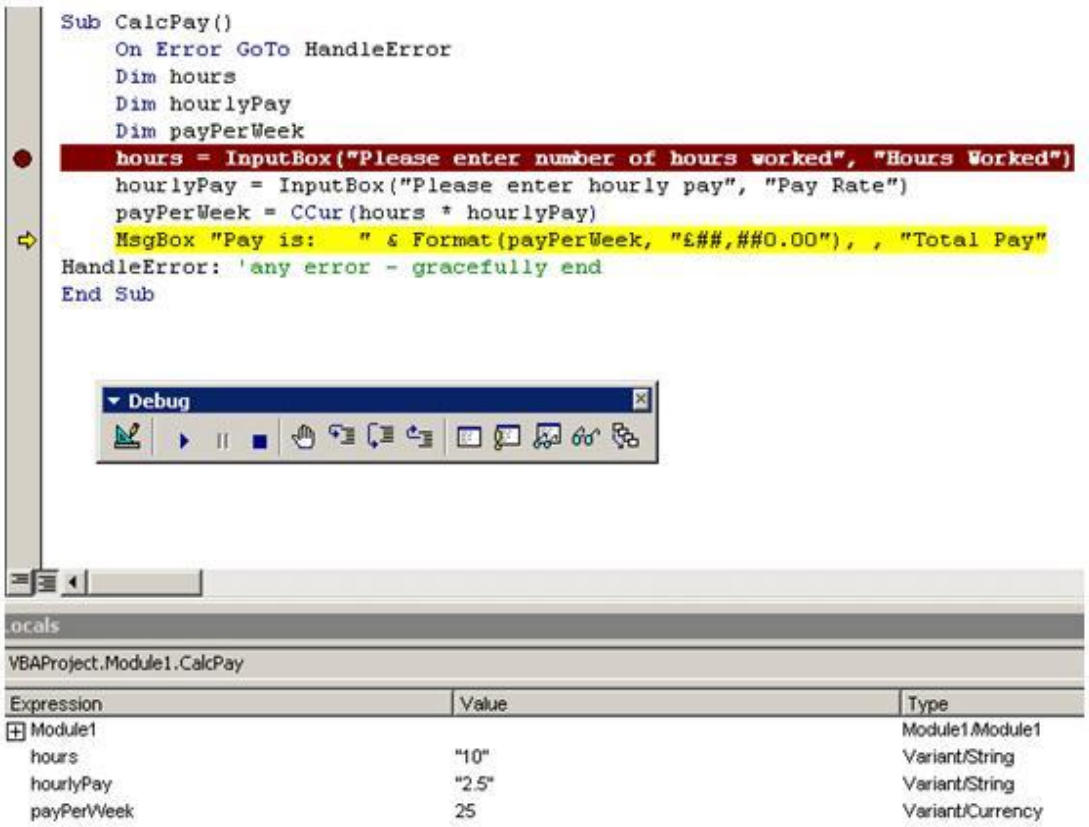
At this point, users can either continue to run the remaining procedure (*press F5 key*) or step through line by line by pressing the **F8** key.

By placing the mouse pointer over any variable or object property, the user will, after a few seconds, see the current value assigned.

Alternatively, by revealing the **Locals Window** (*button 9 above*), users can see all variables and property's values:



After a few steps (**F8** key):



Debugging between calling procedures can be controlled as the **F8** key steps in order line by line across more than one procedure.

To step out of a sub procedure and carry on with the main procedure, press the **SHIFT + F8** keys.

Debug.Print Command

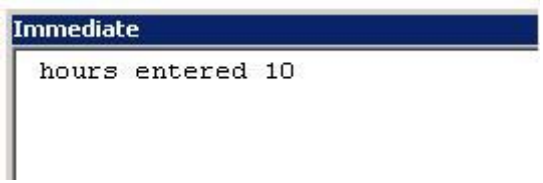
A return value will be printed to the **Immediate Window** (*button 10 above or CTRL + G*).

Two ways to print an output value in the immediate window:

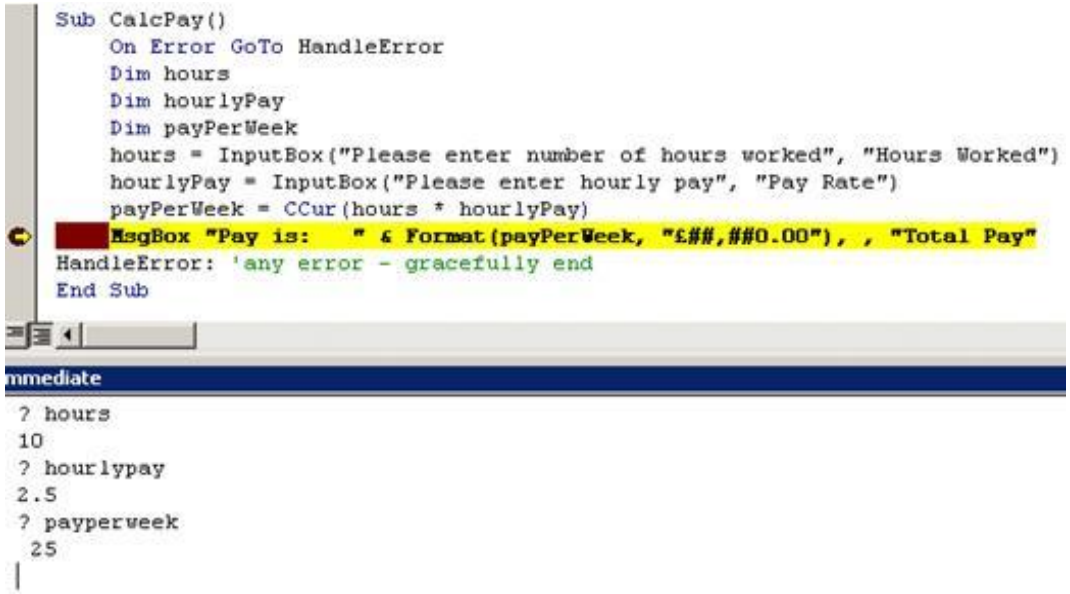
1. **Debug.Print** *Expression*
2. **?** *Expression* (*within the Immediate Window*)

```
Sub CalcPay()
    On Error GoTo HandleError

    Dim hours
    Dim hourlyPay
    Dim payPerWeek
    hours = InputBox("Please enter number of hours worked", _
                                        "Hours Worked")

    Debug.Print "hours entered " & hours

    hourlyPay = InputBox("Please enter hourly pay", "Pay Rate")
    payPerWeek = CCur(hours * hourlyPay)
    MsgBox "Pay is: " & Format(payPerWeek, "£##,##0.00"), _
                                                    , "Total Pay"
HandleError: 'any error - gracefully end

End Sub
```

The above will print the 'hours' variable to the immediate window:



If you set a breakpoint and have the **Immediate Window** visible, you can use a different method to reveal the current values of any known variable or property:



```
Sub CalcPay()
    On Error GoTo HandleError
    Dim hours
    Dim hourlyPay
    Dim payPerWeek
    hours = InputBox("Please enter number of hours worked", "Hours Worked")
    hourlyPay = InputBox("Please enter hourly pay", "Pay Rate")
    payPerWeek = CCur(hours * hourlyPay)
    MsgBox "Pay is:    " & Format(payPerWeek, "£###,##0.00"), , "Total Pay"
HandleError: 'any error - gracefully end
End Sub
```

```
mmediate
? hours
10
? hourlypay
2.5
? payperweek
 25
```

Type a question mark (**?**) followed by a space and then the variable or property and press the enter key to reveal the output value.

Abfragen ob Arbeitsmappe nur lesend geöffnet wurde vom 2. User

Wenn jemand eine Exceldatei bereits geöffnet hat, kann der andere sie nur zum Lesen öffnen.

Möchte man diesen Status per VBA abrufen

1. Lösung:

```
Sub Example2()
```

```
    ' Check to see if the active workbook was
    ' opened as read-only within Microsoft Excel.

    If ActiveWorkbook.ReadOnly Then
        MsgBox "File was opened as read-only"
    Else
        MsgBox "File was not opened as read-only"
    End If

End Sub
```

2.Lösung

```
1.  If Not ActiveWorkbook.ReadOnly = True Then
2.      MsgBox "Current file must be opened READ ONLY to use this feature"
3.      Exit Sub
```

End If

3. LÖsung für noch nicht geöffnete Dateien:

The use of Activeworkbook mean the file is open.
Sub TestReadonly()

Dim strFullFilename As String

strFullFilename = "C:\temp\readonly.xls"

If (GetAttr(strFullFilename) And vbReadOnly) = 1 Then
MsgBox strFullFilename & " Is readonly"
Else

```
MsgBox strFullFilename & " Is NOT readonly"
End If
End Sub
```

4.Lösung

Beschreibung: Recently, I have developed an Excel VBA for my internal users. In this VBA, user can press a button to save the workbook immediatedly.
Now, I have placed this Excel file onto our SharePoint website for sharing to the public. However, I have realized when the user open this Excel file,
it will be in Readonly status. User can open it but they can't save data in this Excel (through marco).
I would like to ask that are there any coding can change the Excel workbook from readonly to read-write status, so the user can save the data?
Otherwise, any other codes can help me to save the Excel without any notice for the user in SharePoint website?

```
 ' Determine if workbook can be checked out.
        On Error GoTo ErrorHandler
        'Fails if User not Logged into Sharepoint
        If Workbooks.CanCheckOut(Filename:=savename) = True Then
        Workbooks.CheckOut savename
        MsgBox "Checked out file from SharePoint: " & file
        End If

        'I actually don't need to open file
        'Workbooks.Open (savename)

        'Instead save over it.
        ActiveWorkbook.SaveAs Filename:=savename,
FileFormat:=xlWorkbookNormal
        'Check it in -- works if I checked it Out

        If Workbooks(ActiveWorkbook.Name).CanCheckIn = True Then
           Workbooks(ActiveWorkbook.Name).CheckIn
savechanges:=SaveDuringClose, Comments:="", MakePublic:=True
           MsgBox file & " has been checked in."
        Else
           MsgBox "This file cannot be checked in " & _
              "at this time.  Please try again later, " & _
```

```
            "someone else who has write access could have it Checked
Out."

        End If
```

Hier noch Code um den Schreibschutz-Lesestatus zu ändern:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
SetAttr Me.FullName, vbReadOnly
End Sub
```

oder

You can use the ChangeFileAccess method. Here's an example:

```
Sub SetAsReadOnly()
'** Test for PC User Name
Dim strUser As String
strUser = Environ("USERNAME")

'**Set Read only File Access for each Office's specific version
Select Case strUser
'** Full Workbook Access
Case Is = "YourUserName", "AnotherUser"
    If ActiveWorkbook.ReadOnly Then ActiveWorkbook.ChangeFileAccess Mode:=xlReadWrite, WritePassword:="admin"
    '** Limit Access
    Case Is <> "YourUserName"
    If Not ActiveWorkbook.ReadOnly Then ActiveWorkbook.ChangeFileAccess Mode:=xlReadOnly, WritePassword:="admin"
End Select

End Sub
```

oder

You can use this code to set the ReadOnly property

SetAttr *pathname, attributes*

To set ReadOnly=True
SetAttr "C:\Blank.xls", vbReadOnly

To set ReadOnly=False
SetAttr "C:\Blank.xls", vbNormal

Alle Windows-Fenster minimieren

```
' Alle Programmfenster minimieren
    Set objShell = CreateObject("Shell.Application")
    objShell.ToggleDesktop
```

Arbeitsmappe minimieren und / oder schließen

```
Application.WindowState = xlMinimized
ActiveWorkbook.Close (False) ' False = Schließen ohne Speichern - True = wahrscheinlich kommt Abfrage
```

Arbeitsmappe nach bestimmter Zeit sperren

Step Down Timer = Misst in Sekundenabständen die abgelaufene Zeit und sperrt nach abgelaufener Zeit die Arbeitsmappe vor weiteren Eingaben .

Dieser Code wurde für eine Prüfmappe entwickelt.
In A1 steht die gesamte Zeitdauer, in A2 wird die Restzeit geschrieben.
Der Ablauf kann mit einem Button angehalten werden (Zuweisung: Sub StopStepDownTimer()) . Bei erneutem Klick auf den Button wird der Ablauf erneut gestartet und bei der vorher abgelaufenen Zeit fortgesetzt.

```
'In das Klassenmodul "Diese Arbeitsmappe"
Sub Workbook_Open()
Worksheets("Test").Range("A1") = "00:10:00"
StopStepDown = False
End Sub

'In ein Modul
Option Explicit
Public StopStepDown As Boolean

Sub StopStepDownTimer()
If StopStepDown = True Then
        StopStepDown = False
Else
        StopStepDown = True
        StepDownTimer
End Sub


Sub StepDownTimer()
Dim i As Integer
Dim Start, Step, Waittime, TimerTest
'Variable für Timerlauf definieren
StopStepDown = True
'Startzeit
Start = Range("a1")
'Sekunde zum Rückwärtszählen berechnen
'Das muss ich so rechnen, bei 1/(24*3600) erhalte ich einen Überlauffehler
'Keine Ahnung warum
Step = 1 / 24
Step = Step / 60
```

```
Step = Step / 60
'Prüfen ob der Timer vorher gestoppt wurde
TimerTest = Range("A2").Value
'Beim ersten Start die PrüfSequenz überspringen
If TimerTest = vbNullString Then
        GoTo ErsterStart
End If
'Variablenformat ändern für Prüfung
TimerTest = TimeValue(Format(Range("A2").Value, "hh:mm:ss"))
If TimerTest > TimeValue("00:00:00") Then
    'Variable Start mit Restzeit füllen
    Start = Range("a2")
    For i = 1 To Start / Step
        'Aktionen erlauben
        DoEvents
        'Zwischenstop abfragen
        If StopStepDown = False Then
            Exit Sub
        End If
       'Wartezeit immer wieder neu berechnen
       Waittime = TimeSerial(Hour(Now()), Minute(Now()), Second(Now()) + 1)
       'Application anhalten
       Application.Wait Waittime
       'neuen Wert in Zelle schreiben
       If Range("a2").Value - Step = Step Or Range("a2").Value = 0 Then
            Range("a2").Value = 0
            Exit Sub
       Else
            Range("a2").Value = Range("a2").Value - Step
       End If
    Next i
```

```
    Exit Sub
End If
ErsterStart:
'1. Sekunde abziehen
Range("a2").Value = Start - Step
'Anzahl der Schleifen berechnen
For i = 1 To Start / Step - 1
        'Aktionen erlauben
        DoEvents
        'Zwischenstop abfragen
        If StopStepDown = False Then
                Exit Sub
        End If
        'Wartezeit immer wieder neu berechnen
        Waittime = TimeSerial(Hour(Now()), Minute(Now()), Second(Now()) + 1)
         'Application anhalten
         Application.Wait Waittime
         'neuen Wert in Zelle schreiben
          If Range("a2").Value - Step = Step Or Range("a2").Value = 0 Then
                  Range("a2").Value = 0
                  Worksheets("Test").Protect Password:="Schutz"
                  Exit Sub
          Else
                  Range("a2").Value = Range("a2").Value - Step
          End If
Next i
End Sub
```

Ausschalten Kalkulation, Schirmupdate und Warnungen

WICHTIG: wenn man das Screenupdating deaktiviert, führt eine etwaige vorhandene SENDKEYS-Anweisung* zu Fehlern / Crash ; (Beim

Formatieren von Zellen sende ich gerne F2 und ENTER mittels Sendkeys in die betreffende Zelle, damit Excel die Zelle neu kalkuliert)

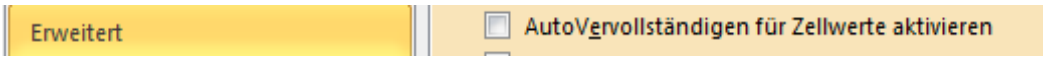
Da das Ein- und Ausschalten so häufig gebraucht wird, legt man sich am besten 2 Prozeduren an, die dann in der gesamten Arbeitsmappe zur Verfügung stehen

```
Public Sub AKTUALISIERUNGAUS

' ABSCHALTEN FEHLERMELDUNGEN u. SEITENAKTUALISIERUNG

Application.ScreenUpdating = False:  ' Bild nicht aktualisieren
' Calculation Manual macht Sinn, wenn die Tabelle sehr viele Formeln mit Zellbezügen hat
Application.Calculation = xlCalculationManual: ' Kalkulation ausschalten
Application.DisplayAlerts = False:  ' Fehlermeldungen werden unterdrückt

End sub

Public Sub AKTUALISIERUNGEIN

' EINSCHALTEN FEHLERMELDUNGEN u. SEITENAKTUALISIERUNG

Application.ScreenUpdating = True
Application.Calculation = xlCalculationAutomatic
Application.DisplayAlerts = True

End Sub
```

Anmerkung zu Display Alerts: Der Standardwert ist True. Legen Sie diese Eigenschaft auf False fest, wenn Sie während der Ausführung eines Makros nicht durch Eingabeaufforderungen und Warnmeldungen unterbrochen werden wollen und stattdessen Microsoft Excel die Standardantwort auswählen soll.

Autovervollständigen in Zellen deaktivieren

Es gibt ja in den Exceloptionen folgende Einstellung, damit Zellen in der selben Spalte nicht die Werte von anderen Zellen zum Vervollständigen vorgeschlagen bekommen. Leider kann man das nicht fix mit einer Arbeitsmappe speichern.



Dies kann auch per VBA gemacht werden und direkt eingefügt werden in die Arbeitsmappe beim Öffnen und Schließen:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.EnableAutoComplete = True
End Sub

Private Sub Workbook_Open()
   Application.EnableAutoComplete = False
End Sub
```

DEBUG PRINT

2.) Eine Liste von Werten in das Direktfenster von VBA reinschreiben (mit STRG-G kann man es im VBA-Fenster öffnen oder im Menü ANSICHT)

```
Debug.print "Schreib war rein"
```

DoEvents

Sie kennen das sicherlich: Während eines längeren Vorgangs soll dem Benutzer die Möglichkeit gegeben werden, den Vorgang bspw. durch Klick auf einen Button abbrechen zu können.

Während der Vorgang läuft reagiert das Programm jedoch nicht auf Benutzer-Aktionen. Überall ist zu lesen, dass hier **DoEvents** das Allheilmittel ist!

In gewisser Weise stimmt das auch! Allerdings sollte mit die DoEvents-Anweisung "sparsam" eingesetzt werden, da diese den gesamten Vorgang enorm verlangsamt.

Beispiel: Es werden sehr viele Datensätze innerhalb einer Schleife verarbeitet. Der Vorgang soll jederzeit vom Benutzer angehalten / abgebrochen werden können. Wir setzen also ein DoEvents in die Schleife, damit wir auf einen Button-Klick (Abbruch) reagieren können.

```
bAbort = False
```

```
Do
  ' Aktion ...
  ' ...

  DoEvents  ' Damit man auf einen Button-Klick oder ähnliches reagieren kann
  If bAbort Then Exit Do
Loop Until ...
```

Im Click-Event des Abbrechen-Buttons steht:

```
Private Sub btnAbort_Click()
  If MsgBox("Wollen Sie den Vorgang wirklich abbrechen?", vbQuestion Or vbYesNo) = vbYes Then
    ' Abbrechen
    bAbort = True
  End If
End Sub
```

Mit obigen Code wird bei jeder einzelnen Aktion ein DoEvents aufgerufen. Dies verlangsamt den Prozess bei sehr vielen Schleifendurchläufen gewaltig!

Besser ist es, wenn man DoEvents bspw. nur alle x - Schleifendurchläufe aufruft:

```
Dim nCount As Long

bAbort = False
nCount = 0
Do
  ' Aktion ...
  ' ...

  nCount = nCount + 1

  ' DoEvents nur alle 10 Schleifendurchläufe aufrufen
  If nCount Mod 10 = 0 Then DoEvents
  If bAbort Then Exit Do
Loop Until ...
```

Das Aufrufen von DoEvents alle x-Schleifendruchläufe verbessert die Performance schon enorm.

Noch besser wäre es aber, wenn man DoEvents nur dann aufruft, wenn der User auch tatsächlich eine Taste oder einen Mausklick getätigt hat - oder?

Genau für diesen Fall gibt es die **GetInputState** API-Funktion. Diese Funktion liefert einen Wert ungleich Null zurück, wenn ein Tastatur- oder Mausklick-Ereignis erfolgte.

```
Option Explicit

' benötigte API-Deklarationen
Public Declare Function GetInputState Lib "user32" () As Long
```

Wir ändern den Code der Schleife also wie folgt ab, um die beste Performance zu erzielen:

```
Dim nCount As Long

bAbort = False
Do
  ' Aktion ...
  ' ...

  ' Falls Tastatur-/Mausklick-Ereignis vorliegt
  If GetInputState() <> 0 Then
    DoEvents
    If bAbort Then Exit Do
  End If
Loop Until ...
```

Enable-Events (Ereignisse deaktivieren)

Ich habe früher das rekursive Aufrufen der Selection_Change bzw. Worksheet_Change Prozeduren immer mit einer Variable VBA-CODE ausgetrickst, wenn mein VBA-Code selbst eine Auswahl oder Zelle änderte, aber ich nicht wollte, dass dadurch die betreffenden Selection_Change bzw. Worksheet_Change Prozeduren aufgerufen wurden, indem ich an deren Beginn immer gleich abrief: If VBA-CODE=1 then exit sub.

Das manuelle Aussteigen aus Worksheet-Events ist gar nicht nötig, denn man kann das Ausführen aller automatischen vorübergehend Worksheet-Events deaktivieren mit

Application.EnableEvents = False
Danach bitte wieder Aktiv schalten mit
Application.EnableEvents = True

Wichtig: der EnableEvents-Parameter bleibt auch nach Ende des VBA-Codes erhalten ! Daher immer nach dem Ausschalten wieder ans Einschalten denken - vor allem auch wenn innerhalb der Prozedur ein END oder EXIT SUB-Befehl kommt !

Eventuell möchte man die Excel-Ereignisse deaktivieren - also, dass etwa das BEFORE_SAVE oder das WORKSHEET_CHANGE oder andere Ereignisse NICHT erfolgen (also der betreffende VBA-Code dieser Ereignisse nicht ausgeführt wird)

```
Application.EnableEvents = True ' aktiviert
Application.EnableEvents = False ' deaktiviert
```

In diesem Beispiel werden Ereignisse vor dem Speichern einer Datei deaktiviert, sodass das Ereignis **BeforeSave** nicht auftritt.
*Application.**EnableEvents** = False*
ActiveWorkbook.Save
*Application.**EnableEvents** = True*

EXCEL 97 Beschränkungen

1.)
Es gibt keine ungebundenen freien Userforms - wie ich sie bei der Royal Kassa Bank verwendet habe - sprich Userforms, die etwas anzeigen, ohne die durchführung des Programmcodes auf sich zu fixieren - diese sind aber wichtig um zB den Statusbalken wachsen zu lassen, während im Hintergrund der Programmcode des Moduls 1 läuft.

Ist der Befehl
Userform1.Show vbmodeless
im Code von "DIESERARBEITSMAPPE" - also der obersten Ebene - dann kommt es unter Excel97 zu einem Fehler:

Es reicht jedoch das Einblenden der Userform als eine SUB-Routine zu definieren und diese in das Modul zu legen. Vor dem Aufrufen der Subroutine braucht man nur die Excelversion auslesen und wenn Sie höher als 8 (97) ist, dann ruft man die Subroutine auf.

Excel prüft wie es scheint automatisch den Code von "DIESEARBEITSMAPPE" ob die Parameter richtig sind - selbst, wenn man mit einer "If Excelversion > 8" eigentlich verhindern würde, dass er diesen Code je auszuführen hätte unter Excel 97.

Eine andere Lösung, die nicht getestet ist:

```
Public Sub ShowUserFormModeless()
  #If VBA6 Then
    frmUserFormModeless.Show vbModeless
  #Else
    frmUserFormModeless.Show
  #End If
End Sub
```

2.)
Wenn ein Kontrollkästchen (das mit dem Häkchen) sich mit seinem Ergebnis auf eine Zelle bezieht (sprich den WAHR / FALSCH-Wert auf eine Zelle überträgt), darf diese nicht mit anderen Zellen verbunden sein - sonst geht das Häkchen nicht !

3.)
Der Befehl ROUND klappt nicht in Excel 97 - dafür habe ich folgende Funktion im Einsatz:

```
Public Function Runden(vZahl As Variant, iStellen As Integer)
   Runden = CVar(Format(vZahl * (10 ^ iStellen), "0") / (10 ^ iStellen))
End Function

Sub Test

Zahl= 3.1415926
msgbox Runden(Zahl, 2)
```

End sub

Excel aktivieren

wenn man nur zwischen zwei Excelmappen hin und her-aktiviert, geht dies mit:

Workbooks("Arbeitsmappe1.xls").Activate

wenn man jedoch aus Excel heraus eine andere Anwendung wie Word, Powerpoint aktiviert hat, muss man die gesamte Excel-Anwendung mit aktivieren. Dazu verwendet man dann diesen Befehl

AppActivate ("Arbeitsmappe1.xls")

Code zum Aktivieren und Anzeigen von Word, Excel etc

```
' ------------------------------------------------------------
' bringing Excel to front
' ------------------------------------------------------------

        WordApp.Selection.HomeKey Unit:=wdStory
        Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second
        'WordApp.Application.WindowState = wdWindowStateMinimize
        AppActivate (File_PsWin)
        Application.WindowState = xlMinimized
        Application.WindowState = xlMaximized
        Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second

        Workbooks(File_PsWin).Sheets("Wordreport").Activate

' ------------------------------------------------------------
' bringing Word to front
' ------------------------------------------------------------

'    ' --- 1. possibility --- minimizing Excel (secure)
'    '   Application.WindowState = xlMinimized
'
'    ' --- 2. possibility --- activating Word (insecure)
```

```
'   '      AppActivate (ActiveDocument)
'
'   ' --- 3. possibility --- minimize and maximize (or normalize) Word (secure)
'      If WordApp.Application.WindowState <> wdWindowStateNormal Then
'           WordApp.Application.WindowState = wdWindowStateMinimize
'           WordApp.Application.WindowState = wdWindowStateMaximize
'         Else
'           WordApp.Application.WindowState = wdWindowStateMinimize
'           WordApp.Application.WindowState = wdWindowStateMaximize
'      End If
```

Wird durch VBA eine andere Anwendung aktiviert - zB Word und man möchte zurückkehren zu Excel, muss man dieses wieder aktivieren:

```
      AppActivate "Microsoft Excel"
```

Ein reines Word ohne Vorlage kann man aktivieren mit

```
      AppActivate "Microsoft Word"
```

Ist dort jedoch ein Dokument offen, geht es so:

```
      AppActivate "Dokument2 - Microsoft Word"
```

Für die IAEA habe ich immer mit diesem Code gearbeitet:

```
      Dim File_PsWin As String ' name of the active Excelfile
      File_PsWin = ActiveWorkbook.Name

      ' code which changes focus to Powerpoint or Word
      …

      ' returning to original Excel-File
      AppActivate (File_PsWin)
```

Achtung: dieser Code funktioniert nur, wenn die XLS-Dateierweiterungen eingeblendet sind. denn nur dann heißt auch das Excelfenster inklusive XLS-Dateiendung XLS

Man kann das natürlich mit einer ON ERROR-Meldung abfangen und im Falle eines Errors vom ActiveWorkbook.Name die Dateiendung wegschneiden und dann das Aktivieren wiederholen.

Excel im abgesicherten Modus starten

für den Fall der Fälle dass irgendein AddIn korrupt ist, oder EXCEL nicht mehr starten will, hier eine letzte Möglichkeit
"C:\Dokumente und Einstellungen\All Users\Startmenü\Programme\Office XP\Excel.exe /s"

Excel ohne Arbeitsmappe starten

Erstellen Sie einen Shortcut, oder ändern Sie den bestehenden aus dem Startmenü, und fügen Sie einen Schrägstrich und ein e an:
"C:\Dokumente und Einstellungen\All Users\Startmenü\Programme\Office XP\Excel.exe /e"
und Excel startet künftig ohne die obligatorische "Mappe1"

Excel schließen

Mein EXIT-Button in Excel:

```
Sub SAVE_EXIT()

' Speichert die aktuelle Datei und schließt sie
Dim ANZAHL As Integer

' Zuerst auslesen wieviele unsichtbare Dateien (Powertools und persönliche Makroarbeitsmappe PERSONAL.XLSB) es gibt
' wenn es beide Dateien gibt, dann ist die Anzahl 2
If UCase(Left(Workbooks(1).Name, 8)) = "PERSONAL" Then ANZAHL = ANZAHL + 1
If UCase(Left(Workbooks(2).Name, 8)) = "PERSONAL" Then ANZAHL = ANZAHL + 1
If UCase(Left(Workbooks(1).Name, 10)) = "POWERTOOLS" Then ANZAHL = ANZAHL + 1
If UCase(Left(Workbooks(2).Name, 10)) = "POWERTOOLS" Then ANZAHL = ANZAHL + 1

On Error Resume Next
' wenn nur noch die unsichtbaren Hilfsdateien offen sind=>Beenden von Excel
If Workbooks.Count = ANZAHL Then Application.Quit ' wenn nur noch 2 (Powertools und Personal.XLSM)
' sonst speichern und schließen der aktuellen Datei
```

```
ActiveWorkbook.Save
ActiveWorkbook.Close

' wenn danach nur noch die unsichtbaren Hilfsdateien offen sind=>Beenden von Excel
If Workbooks.Count = ANZAHL Then Application.Quit ' wenn nur noch 2 (Powertools und Personal.XLSM)

End Sub
```

1.) Sanfte Variante mit Nachfragen

```
Application.quit
```

2.) Harte Variante ohne Fragen:

```
Application.Displayalerts= false
Application.Quit
```

3.) Variante Hart mit Sicherheitscheck

Eventuell puffert man die harte Variante ab, indem man zuvor überprüft, dass eh nur 1 Arbeitsmappe offen ist

```
If Workbooks.count < 2 then
        Application.Displayalerts= false
        Application.Quit
  else
        Application.Quit
End if
```

4.) Variante mit F4

Du kennst bestimmt die Tastenkombination Alt+F4, das bedeutet, dass das gerade geöffnete Programm geschlossen wird. Das kann man in VBA einbinden. Und zwar so:

```
Sub excel_schließen()
SendKeys "%{F4}", True
```

End Sub

Excel VBA-Fehlermeldungen manuell auslösen

um ZB den Fehler mit der Nr. 53 auszulösen:

Err.Raise 53

Excelfenster positionieren

```
Sub InTheMiddle()

Dim dWidth As Double, dHeight As Double

With Application
.WindowState = xlMaximized
dWidth = .Width
dHeight = .Height

.WindowState = xlNormal
.Top = dHeight / 4
.Height = dHeight / 2

.Left = dWidth / 4
.Width = dWidth / 2
End With

End Sub
```

Excelversion auslesen

BEST

```
  MsgBox "Willkommen zur Microsoft Excel Version " & _
    Application.Version & ", sie läuft hier unter " & _
    Application.OperatingSystem & "!", _
    64, "   die aktuelle Excel Version."
```

1.) Von Simplefibu

```
   MsgBox Val(Application.Version)

         MsgBox Application.OperatingSystem
```


```
If Win64 Then
     MsgBox "64Bit"
  Else
     MsgBox "32Bit"
End If
```

```
    EXCELVERSION = Val(Application.Version)

    If EXCELVERSION < 7 Then Msgbox "This is before Excel 95"
    If EXCELVERSION = 7 Then Msgbox "This is Excel 95"
    If EXCELVERSION = 8 Then Msgbox "This is Excel 97"
    If EXCELVERSION = 9 Then Msgbox "This is Excel 2000"
    If EXCELVERSION = 10 Then Msgbox "This is Excel 2002/XP"
    If EXCELVERSION = 11 Then Msgbox "This is Excel 2003"
    If EXCELVERSION = 12 Then Msgbox "This is Excel 2007"
          ' Version 13 was skipped by Microsoft because of the fear of Nr. 13
    If EXCELVERSION = 14 Then Msgbox "This is Excel 2010"
    If EXCELVERSION = 15 Then Msgbox "This is Excel 2012/2013"
    If EXCELVERSION = 16 Then Msgbox "This is Excel 2015"
```

2.) Excelversion + BIT-System

```
MsgBox "Willkommen zur Microsoft Excel Version " & _
   Application.Version & ", sie läuft hier unter " & _
   Application.OperatingSystem & "!", _
   64, "   die aktuelle Excel Version."

MsgBox "Application: " & Application.Application
MsgBox "Version: " & Application.Version
MsgBox "Build: " & Application.Build
MsgBox "CalculationVersion: " & Application.CalculationVersion
```

3.)

```
Function fGetExcelVer() As Integer
If Application.Version Like "*5*" Then
fGetExcelVer = 5
ElseIf Application.Version Like "*7*" Then
fGetExcelVer = 7
Else
fGetExcelVer = 8
End If
End Function
Sub PerVersion()
MsgBox Application.Version
Select Case Left(Application.Version, 1)
Case "5"
MsgBox "Sie verwenden Excel 5"
Case "7"
MsgBox "Sie verwenden Excel 7/95"
Case "8"
MsgBox "Sie verwenden Excel 8/97"
Case Else
MsgBox "Sie verwenden eine unbekannte Excel- Version"
End Select
ThisWorkbook.Activate
End Sub
```

Excelsprachversion Deutsch + Englisch

```
Msgbox Application.LanguageSettings.LanguageID(msoLanguageIDUI)

MsgBox "User Interface Language - " & _
          Application.LanguageSettings.LanguageID(msoLanguageIDUI) & Chr(10) & _
          "Help Language - " & _
          Application.LanguageSettings.LanguageID(msoLanguageIDHelp)
```

1031:deutsch
1033:englisch

Results:

1025: Arabic
1052: Albanian
1067: Armenian
1093: Bengali (India)
5146: Bosnian
1026: Bulgarian
1028: Chinese (Taiwan)
2052: Chinese (China)
1050: Croatia
1088: Cyrillic
1029: Czech
1030: Danish
1043: Dutch
1033: English
1061: Estonian
1065: Farsi (Persian)
1124: Filipino
1035: Finnish
1036: French
1084: Gaelic (Scotland)
2108: Gaelic (Ireland)
1079: Georgian
1031: German
1032: Greek
1037: Hebrew
1081: Hindi
1038: Hungarian
1057: Indonesian
1039: Icelandic
1040: Italian
1041: Japanese
1099: Kannada
1087: Kazakh
1042: Korean
1108: Lao
1062: Latvian
1063: Lithuanian
1086: Malay
1104: Mongolian
1121: Nepali
1044: Norwegian
1045: Polish
1046: Portuguese
1048: Romanian
2072: Romanian (Moldova)
2073: Russian (Moldova)
1049: Russian
1103: Sanskrit
2074: Serbian (Latin)
1051: Slovak
1060: Slovenian
1070: Sorbian
1034: Spanish
1089: Swahili
1053: Swedish
1054: Thai
1055: Turkish
1058: Ukrainian
1066: Vietnamese

Excelsprachversion allgemein

On this page you will find examples of how to avoid problems when your workbooks are used in different language versions of Excel. If you have a suggestion for this page, please let me know.
The first thing I suggest you should do is read the ‘International Issues’ chapter from the following link:

Excel 2002 VBA Programmer's Reference
Written by John Green, Stephen Bullen, Rob Bovey and Robert Rosenberg
http://www.oaltd.co.uk/ExcelProgRef/Ch22/default.htm

As you can see, Stephen Bullen has done a great job and I do not want to duplicate this page, so I will only add tips to this page that are not in this chapter or do it in a different way.

Index

Excel version and Office language settings

It can be useful to know what the Excel version and the Excel language is of the Excel application that opens your workbook so your code can do different things depending of the version/language.

Excel Version Number

You can use this to get the version number of Excel :
Application.Version

But this will display 12.0 for Excel 2007 English and 12,0 for Excel 2007 Dutch. To avoid problems use Val().
Val(Application.Version)

You can use this in your code in any local:

```
Sub Test()
   If Val(Application.Version) < 12 Then
      'You use Excel 97-2003
   Else
      'You use Excel 2007 or higher
   End If
End Sub
```

Excel 97 = 8
Excel 2000 = 9
Excel 2002 = 10
Excel 2003 = 11
Excel 2007 = 12
Excel 2010 = 14

Country code

You can use this to get the language chosen in Windows Regional Settings :
Application.International(xlCountrySetting)

This will give you a Country code, for example 31 for Dutch and 7 for Russian.
For a list of country codes see
http://support.microsoft.com/kb/213833/en-us

You can use Select case to run the code you want like this

```
Sub Test1()
   Select Case Application.International(xlCountryCode)
   Case 31: MsgBox "Run code for Dutch"
   Case 7: MsgBox "Run code for Russion"
   Case Else: MsgBox "Run code for English (default)"
   End Select
End Sub
```

See also this page for examples for the Ribbon in Excel 2007
http://www.rondebruin.nl/dynamic.htm

Language ID of Excel

If you want to know the exact language of the userinterface of Excel (Because you can install many different language packs) you can use this to return the language ID number:
Application.LanguageSettings.LanguageID(msoLanguageIDUI)

To know what language belong the each ID press F2 in the VBA editor to open the object browser and enter msoLanguageID in the search field and press the search button. You see a long list now with language Id's.
If you select one you can see the number on the bottom of the object browser. For example if I select "msoLanguageIDDutch" I see this on the bottom of the object browser :

Const msoLanguageIDDutch = 1043 (&H413)

You can use Select Case now in your code to run different code for a few languages, for example to display the captions of your buttons in the correct language or anything else.

```
Sub Test2()
```

```
    Select Case Application.LanguageSettings.LanguageID(msoLanguageIDUI)
    Case 1043: MsgBox "Run code for Dutch"
    Case 1049: MsgBox "Run code for Russion"
    Case Else: MsgBox "Run code for English (default)"
    End Select
End Sub
```

Application.International (VBA)

In the section above named "Excel version and Office language settings" I use
Application.International(xlCountryCode) to get the language chosen in Windows Regional Settings.
But you can use it to get a lot more information about the users Excel application with this in your VBA code.
If you search for Application.international in the VBA editor the Help will give a you a table with all
the XlApplicationInternational constants that you can use in your code.

For example this code line will give you the local Year, Month and Day character

```
    MsgBox "The Year, Month and Day Characters are " & _
        Application.International(xlYearCode) & " " & _
        Application.International(xlMonthCode) & " " & _
        Application.International(xlDayCode)
```

If you not want to use VBA to get this kind of information check out Get.workspace example workbook
in the section "Using Strings as worksheet function arguments"

Week Numbering Systems and Date/Time Representations in Excel

Learn about the four different week-numbering systems available in Microsoft Office Excel and how to use them. Also learn why it is important to implement the date and time representation of the ISO8601:2000 standard in your Excel applications.

MSDN article about the two pages below on my own site
http://msdn.microsoft.com/en-us/library/bb277364.aspx

Week numbers in Excel
http://www.rondebruin.nl/weeknumber.htm

ISO Date Representatation and Week Numbering
http://www.rondebruin.nl/isodate.htm

Analysis ToolPak add-in worksheet functions

Excel 97-2003 :

ATP Formulas will not be translated if you open your workbook in another language version
The best thing to do is to avoid them, see Dick's blog for formulas that do not use the ATP add-in.
These are all default Excel functions that always translate correct.

http://www.dicks-blog.com/archives/2004/12/18/replacing-the-analysis-toolpak-addin-part-1/

http://www.dicks-blog.com/archives/2004/12/19/replacing-the-analysis-toolpak-addin-part-2/

http://www.dicks-blog.com/archives/2004/12/20/replacing-the-analysis-toolpak-addin-part-3/

http://www.dicks-blog.com/archives/2004/12/22/replacing-the-analysis-toolpak-addin-part-4/

Excel 2007 and up :

In Excel 2007 the Analysis ToolPak add-in with the extra worksheet functions does not exist anymore.
The old ATP functions are now standard worksheet functions in Excel 2007.

This a great change because:
1) No problem if a user does not have the add-in installed
2) The formulas will be translated if you open the workbook in another language version of Excel 2007.

But there is a problem when you use a non English version of Excel 2007 and save the workbook in the 97-2003 format so that other users can use the workbook in the Excel versions 97-2003.

See this page
http://www.rondebruin.nl/atp.htm

This problem is Fixed in Office 2007 SP2

Using Strings as worksheet function arguments

Read Carefully :
In the example from Kirill Lapin below about the Text() function we use the Get.workspace function to get the Year, Month and Day symbol. If you understand this example you can use a similar trick for Row, Column, Date and Decimal separator symbols and many more so you can use them in other worksheet functions.

Download this zip-file with a workbook containing the information about every item in the Get.workspace array.
Download workspace.zip

TEXT()

Overview: TEXT() uses number format strings in its 2nd argument.

Issue: If you use English date, time and decimal number formats as well as 1000 separators, your formulas might not work in

other locales and vice versa. Unlike number formats applied to cells, the number format strings used as function’s argument are not translated automatically from one locale to another.

If you use for example a formula like this in an English version of Excel

="Today is " & TEXT(TODAY(),"yyyy-mm-dd")

The result is : Today is 2009-09-16

But if I open the workbook with this formula for a example in a Dutch version of Excel
the result is : Today is yyyy-09-16

The reason is that we use a J (Year = Jaar) instead of the y in the Netherlands.

You can download a zip file here with two examples of how to avoid problems like this.
One trick from Kirill Lapin (KL) and one from Stephen Bullen.

Download the two example workbooks

INDIRECT()

Overview: INDIRECT() allows "R1C1" notation in its 1st argument if the 2nd argument is equal to FALSE

Example:
EN =INDIRECT("R1C1",0)

Issue: R1C1 notation may vary depending on the locale (both letters and parenthesis),
so the English symbols won’t always work.

Example:
NL =INDIRECT("R1K1";0) - the English string won’t work
ES =INDIRECTO("F1C1",0) - the English string won’t work
RU =ДВССЫЛ("R1C1";0) - the English string will work

Solution 1: The function ADDRESS() allows you to get string-reference in local style.
So you could use it inside INDIRECT() (Trick from Hector Miguel Orozco Diaz)

Example:
EN =INDIRECT(ADDRESS(1,1,1,0),0) - for absolute reference
EN =INDIRECT(ADDRESS(1,1,3,0),0) - for relative reference

You could also extract local style symbols.

Example:
EN =LEFT(ADDRESS(1,1,1,0)) - letter for rows
EN =MID (ADDRESS(1,1,1,0),3,1) - letter for columns
EN =MID(ADDRESS(1,1,3,0),2,1) - left relativity symbol (parenthesis, bracket, etc.)
EN =MID(ADDRESS(1,1,3,0),4,1) - right relativity symbol (parenthesis, bracket, etc.)

Solution 2: Use defined names with the Excel4 macro-function GET.WORKSPACE()

Define two names in your workbook

IR =INDEX(GET.WORKSPACE(37),6)
IC =INDEX(GET.WORKSPACE(37),7)

You can use this then to get the value of A1 in any local
=INDIRECT(IR &1 & IC & 1,0)

Overview: INDIRECT() allows defined names as strings in the 1st argument if the name refers to a non-calculated range.

Example:
EN =INDIRECT(“MyRange”)

Issue: If you have named a range “Database”, its name will be automatically translated (!) into local language depending on the locale. Moreover, Excel will sometimes use space, which is an illegal character in names, in the translated name (!). Thus the formula =INDIRECT(“database”) will fail in a non-English locale as there wouldn’t be a range with such a name.

Example:
EN =INDIRECT(“database”)
NL =INDIRECT("database") - No problem with a named range Database in a Dutch version
ES =INDIRECTO("base de datos") - Excel uses space character here.
RU =ДВССЫЛ("база_данных")

Solution: Avoid using the name “Database” as a string argument of INDIRECT()

CELL() and INFO()

Overview: CELL() in its 1st argument and INFO() in its only argument use predefined keywords. These keywords can be expressed both in English and local language.

Example:
EN =CELL("filename",A1)

Issue: Local versions of the keywords won’t work in the English locale

Example:
NL =CEL("bestandsnaam";A1)
ES =CELDA("nombrearchivo";A1)
RU =ЯЧЕЙКА("имяфайла";A1)

Solution: Use the English version of the keywords in all locales

SUMIF() and COUNTIF()

Overview: In their 2nd argument, SUMIF() and COUNTIF() allow error values
or strings representing error values.

Example:
EN =COUNTIF(A1:A10,#VALUE!)
EN =COUNTIF(A1:A10,"#VALUE!")

Note : In the first example the COUNTIF function and the Error value will be automatically translated
when you open your workbook in another language version of Excel.

Issue: If you need to use <> (unequal) operator with an error value, you must use error strings for concatenation (the error value itself won't work) and the error string must be in the local language,
thus English strings won’t work in other locales and vice versa.

Example:
NL =AANTAL.ALS(A1:A10;"<>#Waarde!")
ES =CONTAR.SI(A1:A10;"<>#¡VALOR!")
RU =СЧЁТЕСЛИ(A1:A10;"<>#ЗНАЧ!")

Solution 1:

This example counts all cells that do not have the #VALUE! error.
EN =ROWS(A1:A10)-COUNTIF(A1:A10,#VALUE!)
Since there are no strings involved, the functions and error values will translate correctly
in another language version of Excel.

Solution 2 : Use defined names with Excel4 macro-function GET.CELL() with
intermediate cells as in the attached workbook. Trick from Hector Miguel Orozco Diaz.

Download GetCell.zip

MATCH(), VLOOKUP(), HLOOKUP() and LOOKUP()

Overview: Many people use lookup-type functions to find the last number or the last string in an array.
When using approximate search, those functions return the last value in an array if the searched value
is impossibly high (for text strings – alphabetically high)

Example:
EN =MATCH("zzzzz",A1:A10)
EN =MATCH(REPT("z",5),A1:A10)

Issue: In the non-Latin-based languages a string like “zzzzz” might not work.

Example:
RU =ПОИСКПОЗ("яяяяя";A1:A10) - will work for both Russian and English strings
RU =ПОИСКПОЗ("zzzzz";A1:A10) - won't work for Russian strings

Solution: Use the following formula instead:
EN =MATCH("*",A1:A10,-1)

DATEDIF()

Read this webpage first :

Introduction To The DATEDIF Function (Chip Pearson)
http://www.cpearson.com/Excel/datedif.aspx

Note: This function is not supported by Microsoft and in Excel 2007 SP2 there seems to be a new bug.
=DATEDIF(DATE(2009,6,27),DATE(2012,1,5),"md") gives a wrong result there (122 instead of 9)

Overview: in its 3rd argument, DATEDIF() uses predefined strings that indicate the date unit to be used for the result: “y”, “ym”, “m”, “md” and “d”

Example:
EN =DATEDIF(DATE(1968,9,13),TODAY(),"ym")

Issue: One might be tempted to replace the English date symbols by the local ones, which would lead to an error.

Example:
NL =DATUMVERSCHIL(DATUM(1968;9;13);VANDAAG();"jm")
ES =SIFECHA(FECHA(1968;9;13);HOY();"am")
RU =РАЗНДАТ(ДАТА(1968;9;13);СЕГОДНЯ();"гм")

Solution: Use only “y”, “ym”, “m”, “md” and “d” as DATEDIF() uses English symbols in all locales

DATEVALUE(),VALUE(), etc.

Overview: You can coerce date-strings into numeric values by using the functions DATEVALUE() and VALUE(), binary negation or any basic math operation (*,/,+,-,^)

Example:
EN =DATEVALUE("01/03/2009") for 3-Jan-2009
EN =--"01-Aug-09"

Issue: Month literals, date separators as well as the order of year, month and day vary depending on the localization.

EN =DATEVALUE("01/03/2009")
NL =DATUMWAARDE("03/01/2009")
ES =FECHANUMERO("03/01/2009")
RU =ДАТАЗНАЧ("03.01.2009")

NL =--"01-mei-09" – for May
ES =--"01-ago-09" – for Aug
RU =--"01-янв-09" – for Jan

Solution: Never use month literals. If you absolutely have to use date-strings, then use the ISO-format strings: "YYYY-MM-DD". Otherwise use the function DATE()

Example:
EN =DATEVALUE("2009-03-01")
NL =DATUMWAARDE("2009-03-01")
ES =FECHANUMERO("2009-03-01")

RU =ДАТАЗНАЧ("2009-03-01")

Command bars and controls in Excel 97-2003

If you want to add a menu item or disable a Command bar or Menu/Control you must always use the English name of the Command bar in the code. If you use the local name of the Command bar it is not working.
But for a Menu/Control you must use the local name (to make it easy <g>).

To avoid problems use the Menu/Control Id's instead of the captions of the Menu/Controls

This is not working in a non English version to disable the File menu
Application.CommandBars("Worksheet Menu Bar").Controls("File").Enabled = False

This is always working :
Application.CommandBars("Worksheet Menu Bar").FindControl(ID:=30002).Enabled = False

Because there is only one control with the ID 30002 you can use this line also because
FindControl will find the first occurrence of the ID.
Application.CommandBars.FindControl(ID:=30002).Enabled = False

See this page for more examples and how you can find all the Id's
http://www.rondebruin.nl/menuid.htm

Useful links

Nice Add-ins for formula translations

See this add-in: TranslateIT from KeepItCool (Jurgen Volkerink)
http://members.chello.nl/jvolk/keepitcool/download.html

Or the Analysis ToolPak Translator add-in from Eric Desart
http://www.rondebruin.nl/atptranslator.htm

F9-Taste Werte aktualisieren

Application.Calculate

Fehlerbehandlung ON ERROR zweistufig machen

Soll eine Fehlerroutine einen zweiten Versuch machen einen fehlerhaften Vorgang doch noch korrekt abzuschließen (zB eine Exceldatei öffnen und zuerst versucht der Code eine XLSM-Datei zu öffnen und wenn das nicht klappt, soll er versuchen eine XLS-Datei zu öffnen) und man möchte auch in der Fehlerroutine einen Fehler abfangen und mit einer Fehler-Hinweismeldung ausweisen, dann geht das nur so

```
Public Dateiname ' Variablen, die in den Fehlerroutinen verfügbar sein sollen, müssen ALLGEMEIN definiert werden

Sub Datei_Oeffnen()

On Error GoTo Fehler
Workbooks.Open Filename:=Dateiname & "\" & ActiveCell.Value & ".xlsm", Password:=Cells(ActiveCell.Row, ActiveCell.Column + 1)

End

' ---------------------------------------------------------------------------
' Fehlerbehebungsroutine
' ---------------------------------------------------------------------------

' 1.Fehlerebene: - wenn die Datei mit Endung XLSM nicht gefunden wurde, versuche es als XLS für die 2003-Version
```

```
Fehler:

On Error GoTo 0

Datei_Oeffnen_XLS ' Subroutine, die den Vorgang in leichter Variation erneut versuchen soll

End Sub

Sub Datei_Oeffnen_XLS()

' Hilfsprozedur für die Hauptprozedur Datei_Oeffnen, die von der Fehlerroutine der Hauptprozedur aufgerufen wird

On Error GoTo Fehler2

Workbooks.Open Filename:=Dateiname & "\" & ActiveCell.Value & ".xls", Password:=Cells(ActiveCell.Row, ActiveCell.Column + 1)

End

' --- 2. Fehlerebene ---

Fehler2:

MsgBox "Sorry, ein Fehler: Entweder wurde die Datei " & ActiveCell.Value & " nicht gefunden oder das Passwort war falsch" & vbCrLf & vbCrLf &
_
"Bitte stellen Sie sicher, " & vbCrLf & vbCrLf & _
" - dass der Dateiname " & ActiveCell.Value & " korrekt ist" & vbCrLf & vbCrLf & _
" - die Dateiendung dieser Datei auch wirklch '.xlsm' ist" & vbCrLf & vbCrLf & _
" - sich die Datei in diesem Ordner befindet: " & vbCrLf & vbCrLf & "    " & Dateiname & vbCrLf & vbCrLf & _
" - und dass das Passwort hier richtig eingetragen ist. Danke"

End Sub
```

Fehlermeldungen anzeigen und zurückspringen

```
' Verzweigen bei Fehler zu Fehlersubroutine

On Error Resume FEHLERBEHANDLUNG
```

```
...
On Error Goto 0 ' Fehlermeldungen aktivieren

Exit Sub ' damit nicht irrtümlich die Fehlerroutine abläuft

FEHLERBEHANDLUNG:

    MsgBox "Es ist folgender Fehler aufgetreten :" & vbCrLf & vbCrLf & "' Error " & Err.Number & " - " & Err.Description & " "' & vbCrLf & vbCrLf & _
    "Bitte überprüfen Sie, ob …"
    ' Falls Bildschirmaktualisierung deaktiviert wurde, müsste man diese hier wieder aktivieren, falls Code hier abbricht und nicht mit Resume zurückkehrt
    Resume

End Sub
```

Im Fehlerbehandlungsteil kann man mit der Resume-Anweisung zurückspringen zum Fehlerauslösenden Code.

Gibt man nur Resume ein, springt man direkt zur Fehlerauslösenden Code-Zeile zurück und versucht sie erneut.
Mit Resume Next springt man zur nächsten Zeile NACH der fehlerauslösenden Zeile.
Mit Resume ZEILE springt man zu einer Sprungmarkierung (Zeilenmarke)

Fehlermeldungen auswerten

Hier steht in A2 eine Datei (inklusive Pfad - also zB C:\Test.xls) und in B2 ihr neuer Name - mit NAME wird die Datei umbenannt.

Nun können verschiedene Fehler auftreten und die sollen je nach Fehlerart verschieden behandelt werden:

```
Sub DateiUmbenennen()
  Const csMsg As String = "Datei existiert, überschreiben?"
  On Error GoTo ERRORHANDLER
  Name Range("A2").Value As Range("B2").Value
  Exit Sub
ERRORHANDLER:
  If Err = 58 Then
    Beep
    If MsgBox(csMsg, vbQuestion + vbYesNo) = vbNo Then Exit Sub
    Kill Range("B2").Value
    Resume
  ElseIf Err = 53 Then
    Beep
    MsgBox "Datei nicht gefunden!"
  End If
End Sub
```

Fehlermeldungen deaktivieren

' Ausschalten von Fehlermeldungen
On Error Resume Next ' Fehlermeldung deaktivieren
[BEFEHLSCODE]
On Error Goto 0 ' Fehlermeldungen aktivieren

ACHTUNG1: wenn On Error Resume Next in einer Prozedur gesetzt wird und dann eine Unterprozedur aufgerufen wird, so wirkt sich dies so aus, dass bei einem Fehler die Unterprozedur komplett abgebrochen wird - ohne Fehlermeldung und ohne die nachfolgenden Zeilen in der Unterprozedur abzuarbeiten. Wir hatten das bei der IAEA, wo ganze Unterprozedur-Teile komplett ausgefallen sind, weil in der aufrufenden Hauptprozedur On Error Resume Next gesetzt war. Lösung: On Error Resume next immer nur vor einer oder ganz wenigen Zeilen setzen und danach gleich wieder mit On Error Goto 0 zurücksetzen. Niemals sollte bei aktivem On Error Resume Next eine Unterprozedur aufgerufen werden. Denn auch ein "On Error Goto 0" in der Unterprozedur behebt den Fehler nicht.

Testen kann man diesen Umstand mit diesen beiden Prozeduren - wobei Test1 gestartet wird

```
Sub test1()
MsgBox 0
On Error Resume Next
test3
MsgBox 4
End Sub

Sub test3()
MsgBox 1
MsgBox 1 / 0
MsgBox 2
End Sub
```

ACHTUNG2: wir hatten mal den Fall, dass eine ON ERROR RESUME NEXT Anweisung nicht zurecht kam mit einer

```
For Each wkbook.charts. ....
Next
```

– Dataserien-Umbenennung-Schleife. Es wurde ein Fehler ausgewiesen in der Schleife, den die On error resume next Anweisung offensichtlich nicht abfangen konnten.

Als wir eine Schleife draus machten durch alle Charts mit

```
For ii = 1 To ActiveWorkbook.Charts.Count

    ActiveWorkbook.Charts(ii).Visible = True  ' genauer war es natürlich kein visible, sondern ein Dataseries-umbenennen
```

Next ii

Dann konnte On Error Resume next den Fehler übergehen.

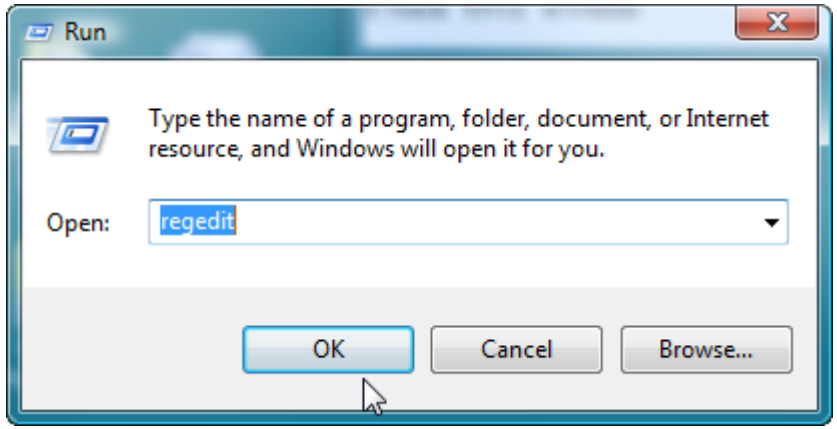
Macros in passwortgeschützen Mappen 2007 aktivieren

When a password protected workbook is opened in Excel 2007 macros are automatically disabled, presumably because the macros cannot be scanned due to the file being encrypted.
You have this problem when you use a File Open password or if use the normal workbook protection.
This is independent of the usual macro security setting which controls whether macros are enabled, not scanned. Note: It is possible that you not have this problem.

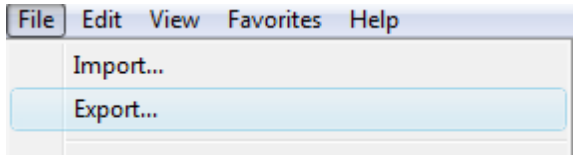
Jim Rech posted this solution if you want to use the code in your protected workbooks:

Before you try Jim's solution you may want to make a back up of the registry before you start.

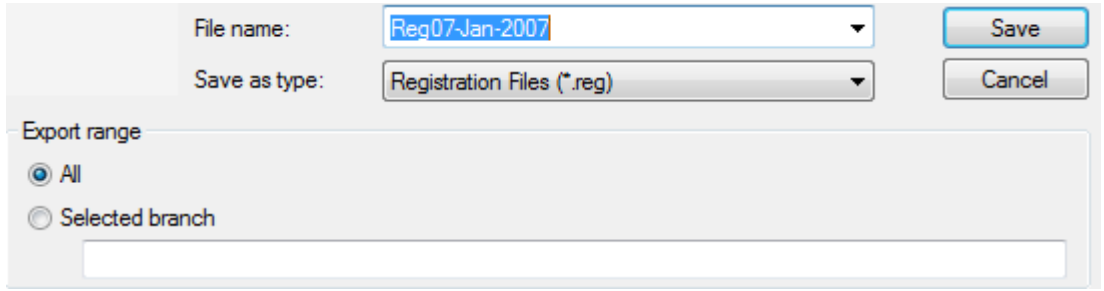
1: Click on the Windows start button
2: Click on Run
3: Type regedit
4: OK



Then we use File Export in the Registry Editor



Enter a name for the backup file and be sure that All under the "Export Range" option is selected and click on Save.

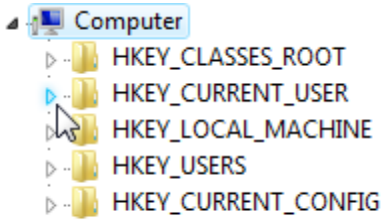


Now we follow Jim's steps to add a DWORD registry key.

You can also download this PassWord.reg file to do the same, easy if you want to do it on more then one computer. Double click the file to open it and it will add the DWORD in the register for you.

Follow the steps below if you want to do it manual.

Click before HKEY_CURRENT_USER in the editor to expand it.



Click on each of these in order:

\Software
\Microsoft
\Office
\12.0
\Excel
\Security

1: Right click on Security
2: Choose New>DWORD
3: Copy/Paste this in the Name box ExcelBypassEncryptedMacroScan

4: Press Enter
5: Double click on the name
6: Change the 0 to 1
7: OK
8: File>Exit to close the Registry Editor

If you follow the steps correct you can use the code now in your protected workbooks.

See also this Microsoft article
http://support.microsoft.com/kb/927150

Makro nach jeder Eingabe ausführen

```
Application.OnEntry = "MeinMakro"
Application.OnEntry = ""
```

Dieser Code arbeitet global, d.h. in allen geöffneten Mappen und Tabellen.

Makrocode dynamisch per VBA erstellen / löschen (s.a. VBA-Code löschen / löscht sich selbst)

```
Sub OpenProzedurAnlegen()
Dim nWB As Workbook
Dim mdlWB As Object
Set nWB = Workbooks.Add
Set mdlWB = nWB.VBProject.VBComponents("DieseArbeitsmappe")
With mdlWB.CodeModule
.InsertLines 3, "Private Sub Workbook_Open()"
.InsertLines 4, " Msgbox ""Bin jetzt da!"""
.InsertLines 5, "End Sub"
End With
End Sub
Sub Loeschen()
With Workbooks("test.xls").VBProject
.VBComponents.Remove .VBComponents("Modul1")
```

```
End With
End Sub

Sub Loeschen()
With Workbooks("test.xls").VBProject
.VBComponents.Remove .VBComponents("Modul1")
End With
End Sub
```

Maximieren des Anwendungsfensters

Möchte man sichergehen, dass eine Anwendung maximiert läuft, dann stell in die Open-Procedur des Workbooks:

Application.WindowState = xlMaximized

MSGBOX für bestimmte Zeit einblenden und dann wieder ausblenden

```
Sub MsgBox3Sekunden()
'   Blendet eine Msgbox nach 3 Sekunden automatisch wieder aus
    ' von Franz W Herber.de
    ' Verweis auf Microsoft Scripting Runtime
    Dim WsShell
    Dim intText As Integer
    Set WsShell = CreateObject("WScript.Shell")
    intText = WsShell.Popup("Diese Meldung wird nach 3 Sekunden geschlossen. Variante 1", 3, "Automatisch...")
    ' Die 3 in der letzten Zeile gibt die Dauer der Öffnung an.
End Sub

Sub MsgZeit()
'   Blendet eine Msgbox nach 3 Sekunden automatisch wieder aus
'   von K.Rola
'   kein Verweis notwendig
    Const bytZeit As Byte = 3
    Dim objWSH As Object, intMSG As Integer
    Set objWSH = CreateObject("WScript.Shell")
    intMSG = objWSH.Popup("Ich bin in " & bytZeit & " Sekunden verschwunden! Variante 2" & Space(10), bytZeit, "gebe bekannt...")
    Set objWSH = Nothing
End Sub
```

Neuberechnen / Neu-Kalkulation erzwingen

```
SendKeys "^%{F9}"
```

Ontime-Event (Zeitsteuerung)

1. Grundlagenkurs

Applicaton.OnTime - Zeitgesteuerte Makros

Autor: Peter Haserodt - Erstellt: -- - Letzte Revision: --

Application.OnTime

Haben Sie etwas Zeit oder müssen Sie OnTime (~Zu einer bestimmten Zeit) weg?
Im letzeren Fall kommen Sie einfach nochmal später wieder, denn unser OnTime Ausflug dauert schon etwas.

Grundlage ist der Wunsch, ein Makro zu einer bestimmten Zeit oder nach Ablauf einer bestimmten Zeit zu starten.

Hierzu bietet Excel die Prozedur **Application.OnTime** an.
So einfach diese auch auf den ersten Blick aussieht, soviele Tücken hat diese auch.

Ich möchte sie Ihnen etwas genauer vorstellen - aber auf die Tücken reinfallen müssen Sie dann schon selber

Wichtige Hinweise:

1. Starten Sie die Makros nur **einmal**, wenn ich schreibe starten - außer ich will es ausdrücklich anders.
Denn diese sind mehrfach hintereinander ausführbar und können dabei am Anfang Verwirrung stiften.

2. Lesen Sie diesen Artikel **bis zum Ende -** oder gar nicht. Denn Application.OnTime hat es in sich und wenn Sie damit arbeiten wollen, sollten Sie sich genau damit beschäftigen.

Das Grundgerüst:

Die Excelhilfe bietet dazu folgendes an:
Ausdruck.OnTime(*EarliestTime*, *Procedure*, *LatestTime*, *Schedule*)

Wobei Ausdruck für ein Application Objekt steht - in den meisten Fällen also unsere Application - und die Klammern für die Füße sind. (Außer ich setze ein Call davor, dann bitte mit Klammern)

Wir wollen uns dies mal übersetzen - so frei Schnauze - so halt wie ich's mir denke - einfach so halt:

Application.OnTime *Wann, WelchesMakro ,[BisWannSollIchsVersuchen] ,[SetzenOderLöschen]*

Damit ist es hoffentlich etwas besser zu verstehen. (Wir werden später noch sehen, dass unser Wann nicht ein so glücklicher Begriff ist)

Die Argumente in den eckigen Klammern sind optional anzugeben.
Und wenn etwas optional ist, dann fangen wir doch mal mit den Zwingenden an.

Schauen wir zuerst mal auf **Wann** und **WelchesMakro**

Dazu bauen wir uns eine Testumgebung auf.
Bitte **seien Sie sehr sorgfältig** mit den nachfolgenden Anweisungen, sonst funktioniert es nicht so, wie es soll.

Der erste Code

Erstellen Sie nachfolgenden Code in einem allgemeinen Modul

```
Public Sub StartZeitGeber()
 Application.OnTime Now + TimeValue("0:0:3"), "Ausfuehrende"
End Sub

Private Sub Ausfuehrende()
 MsgBox Format(Now, "hh:nn:ss")
End Sub
```

Führen Sie nun die Prozedur StartZeitGeber aus und warten Sie 3 Sekunden.
Und Hurra - wenn Sie schön brav waren, hat auch alles geklappt.

Dann schauen wir mal:

Unsre **Wann** ist hier der Ausdruck : Application.OnTime Now + TimeValue("00:00:03"),
Wann -*EarliestTime*- will einen genauen Zeitpunkt haben, an welchem etwas getan werden soll.

Tatsächlich hätte ich auch schreiben können: Application.OnTime TimeValue("17:30:00"), "Ausfuehrende" aber dies wäre für Sie fatal, wenn Sie diesen Artikel um 12:30 Uhr lesen (oder gar noch früher) denn dann hätte dies seine Zeit gedauert, bis die MsgBox erschienen wäre und Sie hätten mich wahrscheinlich GeOnTimed.

Also haben wir den auszuführenden Zeitpunkt einfach um 3 Sekunden nach der aktuellen Zeit festgelegt, durch den Ausdruck Now + TimeValue("00:00:03") - Merken Sie es ? In der Prozedur habe ich nur 0:0:3 geschrieben!
Frisst diese auch.
Wer will kann auch mal ausprobieren:
Application.OnTime Now + 3 / 86400, "Ausfuehrende"
und sich Gedanklich damit auseinandersetzen.

Bevor ich noch näher auf das **Wann** eingehe, kurz zum **WelchesMakro.**
Dies ist sicherlich einfach zu verstehen. Hier schreibe ich die Prozedur hin, welche ausgeführt werden soll. Und zwar in doppelten Anführungszeichen. Wichtig ist natürlich, dass die Prozedur erreichbar sein muss. Hier habe ich sie als Private deklariert und das ist ok so, da sie ja im selben Modul steht. Nur so als Hinweis.

Wann - EarliestTime-unter der Lupe

Dies ist so wichtig zu verstehen, dass es einen eigenen Kasten bekommt.

Tatsächlich habe ich EarliestTime mit Wann sehr schlecht übersetzt - es sollte ungefähr heißen: *FrühesterZeitpunkt*
Aber ich habe dies bewusst so gewählt, damit das Grundverständnis erweckt wird.

Tatsächlich ist unser **Wann** auch ersteinmal der Zeitpunkt, an welchem die Ausführung erfolgen soll.
Und wird auch in der Regel so sein.
Nur gibt es viele Situationen, in welchen dies für Excel nicht möglich ist.

Ein Beispiel:
Starten Sie die StartZeitgeber und klicken Sie dann ganz schnell in eine Zelle, damit Sie in der Zelle sind (also der Cursor blinkt) und warten Sie ca. 10 Sekunden.
Sie merken, während dieser Zeit passiert nichts.
Verlassen Sie nun die Zelle und die MsgBox erscheint sofort.
(Wenn dies nicht so richtig mit Ihrer Grundschnelligkeit des Zellenauswählens funktioniert, erhöhen Sie auf 10 Sekunden, also Now + TimeValue("00:00:10") - aber das haben Sie auch selber rausgefunden - oder)

Sie merken, dass der Begriff EarliestTime gar nicht so dämlich ist.
Denn ExcelVBA versucht die Anweisung zum angegebenen Zeitpunkt auszuführen, kann dies aus irgendwelchen Gründen nicht geschehen, wartet ExcelVBA so lange mit der Ausführung, bis Excel wieder für Makrooperationen bereit ist.

Und damit kommen wir zum:

BisWannSollIchsVersuchen - LatestTime

Aus dem zuvor Erlernten ist dies nun einfach zu erschließen:

Wir haben gesehen, dass ExcelVBA praktisch den nächsten freien Termin zu/nach unserem Wann wahrnimmt, um die Prozedur zu starten.
Dies ist wie der Flug von Timbuktu auf den Mond. Sagen wir der ist für 12:54:30 angesetzt, dann wird er auch zu diesem Zeitpunkt starten, wenn nichts dazwischen kommt. Ansonsten wartet der Pilot solange, bis er die Startfreigabe erhält.

Jetzt kann es aber passieren, dass man sagt: Ne, wenn der Flug nicht nach 3 Stunden gestartet ist, will ich gar nicht mehr (Weil der Pilot mittlerweile soviele Whiskeys in sich hat, dass selbst die mondsüchtige Stewardess Angst bekommt) .
Genau dies stellt uns unsere OnTime Anweisung mit dem optionalen Parameter **LatestTime** zur Verfügung.

Hier kann ich angeben, wann der Versuch abgebrochen werden soll.
Beispiel:
Wir ändern unsere Prozedur StartZeitGeber wie folgt:

```
Application.OnTime Now + TimeValue("00:00:03"), "Ausfuehrende", Now + TimeValue("00:00:30")
```

Nun probieren Sie wieder die Tests mit der Zelle und warten mal länger als 30 Sekunden.
Sie werden feststellen, dass das Makro dann nicht ausgeführt wird.

Und wer jetzt glaubt, dass ich zum vierten Argument übergehe, der irrt sich gewaltig.
Jetzt schauen wir ersteinmal was das mit dem Application auf sich hat.

Wo Application draufsteht ist auch Application drinne

Bitte ändern Sie wieder die StartZeitGeber damit dort die Zeile so heißt:

Application.OnTime Now + TimeValue("00:00:30"), "Ausfuehrende"

Genau so und nicht anders.
Ach, weil Sie es sind und bis jetzt durchgehalten haben, nochmal der ganze Code schön buntig:

```
Public Sub StartZeitGeber()
 Application.OnTime Now + TimeValue("00:00:30"), "Ausfuehrende"
End Sub

Private Sub Ausfuehrende()
 MsgBox Format(Now, "hh:nn:ss")
End Sub
```

Und jetzt speichern Sie die Mappe bitte - machen Sie ja sowieso immer - oder?
Aber diesmal wirklich!

Der Ablauf ist jetzt wie folgt, lesen Sie erst bist zu Ende:
Starten der StartZeitGeber
Dann die Mappe schließen, aber Excel unbedingt geöffnet lassen.
Dann genüsslich am Kaffee nuckeln.
Dann überrascht sein.

Also jetzt dürfen Sie loslegen.

Ja was war den dies nun. Spinnt Excel jetzt komplett oder wie oder was?
Nein, überhaupt nicht!
Es heißt doch **Application.OnTime** und nicht Workbook.OnTime!
Das ist an die Anwendung gekoppelt und nicht an den Mond.

Selbstverständlich haben Sie sofort ausprobiert, wie das ist, wenn Sie Excel schließen.
Klar, da wird der Prozess beendet!

Aber nun Schluss mit lustig. Können Sie denn nicht ernst bleiben. Das ist nicht zum Lachen. Das ist bittere Wahrheit und nichts als die Wahrheit so wie mir meine Oma das immer gesagt hat (mütterlicherseits).

Dies ist der Moment für:

SetzenOderLöschen - Schedule oder Kill the biest

Der Zauberlehrling: Walle Walle Walle - bis ich in die Falle falle
OK - Goethe würde es vielleicht nicht gefallen - aber ich bin ja nur ein armer P[o]eter

Hier kommen wir nun zum schwierigsten Teil des Ganzen. (Haben Sie wirklich geglaubt es bliebe so einfach wie bisher?)
Etwas starten ist ja ganz lustig, aber etwas aufzuhalten dann ..

Das vierte Argument - klingt fast nach Science Fiction - ist für das Stoppen einer angestossenen zeitgesteuerten Prozedur.
Lasse ich es weg - oder setze es auf **True** - dann bedeutet es: Starten.
Setze ich es hingegen auf **False**, dann bedeutet es: Ne, wird nicht mehr gebraucht.

Dies ist aber viel komplizierter als man denkt. Denn ExcelVBA muss ja auch erkennen, welches Ereignis es weghauen soll.

Nehmen wir an, Sie starten unsere StartZeitGeber zweimal.
Jetzt gibt es für Excel zwei Aufträge. Und zwar jeweils den Auftrag die Prozedur Ausführende durchzuführen, nur zu verschiedenen Zeitpunkten.
Um einen solchen Auftrag zu stoppen, brauchen wir also zwei Angaben, nämlich welche Prozedurausführung soll gestoppt werden und zu welchem Zeitpunkt sollte sie gestartet werden.
Wir brauchen also Informationen, um zu stoppen.
Dies ist bei einer festverdrahteten Ausführung nicht so schwierig - also wenn wir den Zeitpunkt auf z.B. 17:00:00 festlegen aber bei Zeitpunkten, die wir abhängig von der aktuellen Zeit - wie bisher benutzt - brauchen wir diese Information.
Sehr verwirrend - dann schauen wir es uns besser an einem Beispiel an:
Bitte nehmen Sie sich ein neues allgemeines Modul und :

```
Option Explicit
Dim iTimerSet As Double

Public Sub StartEs()
 iTimerSet = Now + TimeValue("00:00:15")
 Application.OnTime iTimerSet, "MachEs"
End Sub

Public Sub StopEs()
 Application.OnTime iTimerSet, "MachEs", , False
End Sub

Private Sub MachEs()
 MsgBox Format(Now, "hh:nn:ss")
End Sub
```

Verschiedene Experimente

Wir haben schon viele Worte verloren - beim Fundbüro abgegeben, deswegen nun einfach Experimente, an welchen Sie das ganze verstehen lernen.
Lesen Sie das Experiment immer erst durch, bevor Sie es ausführen.

1. Experiment:
Führen Sie StartEs aus und warten Sie 15 Sekunden.
Wie wir erwartet haben, erscheint unsere MsgBox

2.Experiment
Führen Sie StartEs aus und nach kurzer Zeit führen Sie auch StopEs aus.
Warten Sie nun - und Sie können warten wie Sie wollen, unsere MsgBox erscheint nicht.
Wir haben erfolgreich den Auftrag abgeschossen.

3.Experiment
Führen Sie StartEs aus und nach 2 drei Sekunden führen Sie **StartEs** nochmals aus.
Warten Sie nun und die MsgBox erscheint und nach Bestätigung der MsgBox erscheint die nächste MsgBox, denn wir haben ja zwei mal angekickt.

4.Experiment
Führen Sie StartEs aus und nach 2 drei Sekunden führen Sie **StartEs** nochmals aus.
Und gleich darauf führen Sie StopEs aus
Warten Sie nun und Überraschung: Die MsgBox erscheint - Einmal!

Dieses bedarf einer kurzen Erklärung:
In unserer Variablen iTimerSet speichern wir den Ausführungszeitpunkt.
Aber beim zweiten Starten der StartEs wird sie ja mit dem neuen Startwert überschrieben.
D.h. in unserer Sub StopEs wird nur die Aufgabe mit dem letzten Startwert gekillt.

5.Experiment
Führen Sie nur die StopEs aus.
Es erscheint eine Fehlermeldung. Es gibt nunmal keine Aufgabe mit den Angaben.

6.Experiment
Führen Sie **StartEs** aus und nach 2 drei Sekunden führen Sie **StartEs** nochmals aus.
Und gleich darauf führen Sie **StopEs** aus und gleich nochmals **StopEs**
Beim zweiten StopEs erscheint eine Fehlermeldung, denn in der Variablen iTimerSet haben wir ja nur den Wert vom zweiten Mal ausführen.
Beim Ausführen der StopEs wird dieser Auftrag gekillt.
(Und natürlich wird einmal die MsgBox angezeigt)

Und nun sind Sie dran

Mit diesen Experimenten sollten Sie ein Gefühl für die ganze Sache bekommen.
Wiederholen Sie diese, bis Sie es wirklich verstehen.
Denken Sie sich selbst neue Experimente aus. Versuchen Sie eine zweite Variable einzubauen.
Und und und..
Denn irgendwann muss ich nunmal Ihr Händchen loslassen und Sie müssen ganz alleine in die weite weite Welt hinaus.

Aber bevor dies geschieht, habe ich noch eine Kleinigkeit für Sie, in welchem wir das Ganze etwas zusammenfassen und auch ein Dauerfeuer uns anschauen.
Sie sind schon gespannt?
Sie dürfen aber erst weiterlesen, wenn Sie fleißig waren.

Wiederholende

Nehmen Sie sich eine neue leere Arbeitsmappe.
Beachten Sie die Codeplazierung.
Einmal im Modul diese Arbeitsmappe und einmal in einem allgemeinen Modul!

```
' ************************************************************
'  Modul:  DieseArbeitsmappe  Typ = Element der Mappe(Sheet, Workbook, ...)
' ************************************************************

Option Explicit

Private Sub Workbook_BeforeClose(Cancel As Boolean)
 EndeUhr
End Sub


' ************************************************************
'  Modul:  mdlSekunden  Typ = Allgemeines Modul
' ************************************************************

Option Explicit
Dim iTimerSet As Double

Public Sub SekundenZaehler()
 ThisWorkbook.Worksheets(1).Range("a1").Value = Format(Now, "hh:nn:ss")
 iTimerSet = Now + TimeValue("00:00:01")
 Application.OnTime iTimerSet, "SekundenZaehler" 'Selbstaufruf
End Sub

Public Sub EndeUhr()
 On Error Resume Next ' Sehr faul programmiert
 Application.OnTime iTimerSet, "SekundenZaehler", , False
End Sub
```

Diese kleine Spielerei soll das ganze mal zusammenfassen und auch die Möglichkeit zeigen, wie man eine Aktion immer wieder startet.
Gleichzeitig sorgen wir beim Schließen der Arbeitsmappe dafür, dass unsere Stop Prozedur ausgeführt wird.
Damit aber keine Fehlermeldung auftritt, wenn überhaupt nicht gestartet wurde, habe ich dort einfach ein OnError Resume Next reingehauen. Nicht schön - aber wirksam.

OnTime für mich zu gehen

Ich hoffe, dass ich Ihnen die OnTime etwas näher bringen konnte.
Aber die ganzen Fallstricke müssen Sie nun selbst herausfinden.

Gestatten Sie mir aber noch einen Hinweis:

So schön die OnTime auch sein mag, gibt es bessere Methoden - nämlich mit den entsprechenden API - Timerfunktionen.
Diese finden Sie sehr schön in folgendem Artikel demonstriert:

Stoppuhr in Excel

---

2. Scheduling Events With OnTime And Windows Timers

This page describes the Application.OnTime method and Windows Timers.

You may have the need to run some code periodically and automatically, such as code to pull data from an external data store. With some simple VBA code, you can call upon the the Application's OnTime method to automatically run a procedure. This pages describes how to do this. It also includes a section on using the timers provided via the Windows API functions.

Introduction

As parameters, the OnTime method takes a specific data and time at which it should run the procedure and the name of the procedure to run. It is important to remember that you provide the date and time to run the procedure, not an offset from the current time. If you need to cancel an OnTime, you must provide the *exact* time that the event was schedule to take place. There is no way to tell Excel to cancel the next OnTime event or to cancel all the pending OnTime events. Therefore, you need to store the time at which the procedure is to run in a Public variable and use that variable's value in calls to OnTime.

Note that if the workbook containing the procedure to be executed by OnTime is closed before the procedure is run, Excel will open the workbook before running the procedure and will leave the workbook open after the procedure is complete.

For example, declare Public variables in a standard code module, outside of and before any procedure (Sub or Function) declaration:

```
Public RunWhen As Double
Public Const cRunIntervalSeconds = 120 ' two minutes
Public Const cRunWhat = "TheSub"  ' the name of the procedure to run
```

Starting A Timer

To start a repeatable timer, create a procedure named StartTimer as shown below:

```
Sub StartTimer()
    RunWhen = Now + TimeSerial(0,0,cRunIntervalSeconds)
    Application.OnTime EarliestTime:=RunWhen, Procedure:=cRunWhat, _
        Schedule:=True
End Sub
```

This stores the time to run the procedure in the variable RunWhen, two minutes after the current time.

Next, you need to write the procedure that will be called by OnTime. For example,

```
Sub TheSub()
    ''''''''''''''''''''''''
    ' Your code here
    ''''''''''''''''''''''''
    StartTimer  ' Reschedule the procedure
End Sub
```

This procedure executes whatever code you include in it, and then at the end calls the StartTimer procedure to schedule another OnTime event. This is how the periodic calls are implemented. Note that if you close the workbook while an OnTime event is pending, Excel will re-open that workbook to execute the procedure and will not close the workbook after the OnTime event is finished.

Stopping A Timer

At some point, you or your code will need to terminate the OnTime schedule loop. To cancel a pending OnTime event, you must provide the *exact* time that it is scheduled to run. That is the reason we stored the time in the Public variable RunWhen. You can think of the RunWhen value as a unique key into the OnTime settings. (It is certainly possible to have multiple OnTime events pending. In this, you should store each procedure's scheduled time in a separate variable. Each OnTime event needs its own RunWhen value.) The code below illustrates how to stop a pending OnTime event.

```
Sub StopTimer()
```

```
    On Error Resume Next
    Application.OnTime EarliestTime:=RunWhen,Procedure:=cRunWhat, _
        Schedule:=False
End Sub
```

Using Windows Timers

In addition to Excel's OnTime method, you can use the Windows Timer functions provided via Window API. Windows Timers are automatically rescheduled and will continue to "pop" until you terminate the timer with the KillTimer API function. With a Windows Timer, you provide the interval, in milliseconds, that the timer will "pop". The timer will "pop" at that interval until terminated with KillTimer.

These procedures require Excel 2000 or later, since we use the AddressOf operator. The code will not work in Excel 97 or earlier versions.

A NOTE OF CAUTION: If the code executed by the timer changes a cell value, and you are presently in edit mode in Excel (e.g., entering data in a cell), Excel will likely crash completely and you will lose all unsaved work. Use Windows timers with caution.

To use Windows timers, paste the following code into a standard code module:

```
Public Declare Function SetTimer Lib "user32" ( _
    ByVal HWnd As Long, _
    ByVal nIDEvent As Long, _
    ByVal uElapse As Long, _
    ByVal lpTimerFunc As Long) As Long

Public Declare Function KillTimer Lib "user32" ( _
    ByVal HWnd As Long, _
    ByVal nIDEvent As Long) As Long

Public TimerID As Long
Public TimerSeconds As Single

Sub StartTimer()
    TimerSeconds = 1 ' how often to "pop" the timer.
    TimerID = SetTimer(0&, 0&, TimerSeconds * 1000&, AddressOf TimerProc)
End Sub

Sub EndTimer()
    On Error Resume Next
    KillTimer 0&, TimerID
End Sub
```

```
Sub TimerProc(ByVal HWnd As Long, ByVal uMsg As Long, _
        ByVal nIDEvent As Long, ByVal dwTimer As Long)

    ''''''
    ' This procedure is called by Windows. Put your
    ' code here.
    ''''''
End Sub
```

Run the procedure StartTimer to begin the periodic timer. The variable TimerSeconds indicates how often, in seconds, the timer is to "pop". The SetTimer function takes a value in milliseconds, so the code multiplies TimerSeconds by 1000.When the timer "pops" Windows will call the procedrue TimerProc. You may name this procedure anything you want, but it *must* be declared with the parameters exactly as shown above. If the parameters differ from what is shown above, Excel will crash. When Windows calls TimerProc, it passes the following parameters:

Parameter	**Meaning**
HWnd	This is the HWnd (Windows Handle) of the Excel application. You can ignore this parameter.
uMsg	The message ID of a timer message, value of 275. You can ignore this parameter.
nIDEvent	This value indicates which timer is being "pop" if you have more than one Windows timer in effict. This is the value that was returned from the SetTimer API function.
dwTimer	The number of milliseconds that Windows has been running. This is the same value that you would get from the GetTickCount Windows API function.

To terminate a Windows timer, use the EndTimer procedure shown above, which calls KillTimer to actually terminate the timer.

Pause für Anzahl Sekunden

siehe hier die Einträge "Warten / Pause für Anzahl von Sekunden" und "Zeitsteuerung von Excel"

Prozedur über eine Variable aurufen

Bei den Powertools möchte ich den Namen der letzten aufgerufenen Funktion in eine Variable speichern, sodass man dieletzte Funktion über eine eigene Prozedur wiederholen kann.

```
Public Sub TEST()
   MsgBox "Hallo"
End Sub
Sub TEST2()

Dim NAME As String

NAME = "TEST"

Application.Run NAME

End Sub
```

Achtung: die nachfolgende Variante mit CallByName Me klappt nur, wenn es sich um Prozeduren innerhalb einer Userform handelt - nicht bei normalen Modulen.

Beispiel:
In einer String-Variable steht der Name einer aufzurufenden Prozedur, z.B. "ShowMessage". Innerhalb des Programms existiert natürlich auch eine solche Prozedur. Der Inhalt der Variable wurde z.B. aus einer Textdatei ausgelesen. Wie aber schafft man es nun, dass die Prozedur auch aufgerufen wird?

Die Lösung hierfür bietet die **CallByName**-Funktion:

```
Dim sVariable As String

sVariable = "ShowMessage"
CallByName Me, sVariable, VbMethod
```

Erstellen Sie ein neues Projekt mit einer Form und einem CommandButton. Fügen Sie dann folgende Zeilen in den Codeteil der Form ein:

```
Option Explicit

Public Sub ShowMessage()
  MsgBox "abcd"
End Sub

Private Sub Command1_Click()
  Dim sVariable As String

  sVariable = "ShowMessage"
```

```
  CallByName Me, sVariable, VbMethod
End Sub
```

Sollte die aufzurufende Prozedur Parameter erwarten müssen diese nach **vbMethod** angegeben werden. Also erweitern wir unser kleines Beispiel...

```
Option Explicit

Public Sub ShowMessage(ByVal sText As String)
  MsgBox sText
End Sub

Private Sub Command1_Click()
  Dim sVariable As String
  Dim sText As String

  sVariable = "ShowMessage"
  sText = "Irgend ein Text"
  CallByName Me, sVariable, VbMethod, sText
End Sub
```

Schließen der Arbeitsmappe / Excel verhindern

Am einfachsten man stellt in die Arbeitsmappe folgenden Code

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
        Cancel = True
End Sub
```

Dann kommt man aber nie raus.

Ich habe vielmehr eine Public Variable ENDE definiert

und folgenden COde

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   If ENDE = 0 Then Cancel = True
   If ENDE = 0 Then UserForm1.Show
End Sub
```

Von Haus aus ist ENDE ohnedies immer Null => Cancel wird auf True gesetzt, wodurch das Schließen der Datei nicht mehr möglich ist und zugleich die Userform1 aufgerufen

Dort gibt es einen Button der eine Meldung "Bitte nicht schließen" nur mit OK bestätigt

Sein Code ist

```
Private Sub CommandButton1_Click()
   Unload UserForm1
End Sub
```

Es gibt in der Userform aber auch ein Bild, durch dessen Anklicken man die Datei dennoch schließen kann (wobei bei Änderungen noch gespeichert werden kann). Der Code des Bildes ist:

```
Private Sub Image1_Click()
   ENDE = 1
   ActiveWorkbook.Close
 End Sub
```

SPEICHERN der Arbeitsmappe / Excel verhindern

Der eleganteste Weg (wenn man sowieso schon VBA einsetzt) ist in Verwendung der folgenden zwei Ereignisse mit den entsprechenden Code-Zeilen.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

    ThisWorkbook.Saved = True

End Sub
```

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

    Cancel = True

End Sub
```

Beide Ereignisse müssen in das Modul “DieseArbeitsmappe” geschrieben werden.

Das Ereignis **Workbook_BeforeClose** wird immer ausgeführt, bevor die Arbeitsmappe geschlossen wird. Der verwendete Befehl suggeriert Excel, dass keine Änderungen vorgenommen wurden und damit wird die «Möchten Sie speichern» Abfrage unterdrückt.

Das Ereignis **Workbook_BeforeSave** wird immer dann ausgeführt, wenn der Benutzer bereits den Befehl zum speichern gegeben hat, aber noch bevor das Speichern ausgeführt wird. Mit der Anweisung «Cancel = True» wird der Vorgang ohne weitere Meldung abgebrochen.

Statusleiste - Zwischenergebnisse andrucken

Mit Messagebox muss man diese immer abklicken

1.) Besser ist es diese entweder in der Statusbar anzuzeigen

```
Sub Meldung()
 Application.StatusBar = "Hallo ich bin die Meldung in der Statuszeile"
End Sub
```

```
' Meldung deaktivieren
Sub Meldung_aus()
 Application.StatusBar = False
End Sub
```

2.) Eine Liste von Werten in das Direktfenster von VBA reinschreiben (mit STRG-G kann man es im VBA-Fenster öffnen oder im Menü ANSICHT)

Debug.print "Schreib war rein"

Statusleiste – Schleifen-Durchlauf-Prozent von 1-100

Wenn eine Schleife lange dauert, weil sie z.B. durch viele Zeilen durchwandert, kann man die Prozent der erledigten Schleifendurchläufe in der Statuszeile anzeigen. (Dabei ist Application.Screenupdating = True NICHT notwendig :o)

```
' ----------------------------------------
' ---  Such- Schleife durch alle Zeilen ---
```

```
' ----------------------------------------

Application.ScreenUpdating = False

LETZTEZEILE = LETZTEZELLE(ActiveSheet.Name).Row

For Z = 11 To LETZTEZEILE
' …. der Code

   ' Schleifenprozent
   SCHLEIFENPROZENT = (Z - 10) / (LETZTEZEILE - 10) * 100  ' Da unsere Schleife erst in Zeile 11 beginnt, müssen wir die ersten 10 Zeilen immer abziehen)
   If Round(SCHLEIFENPROZENT, 0) > PROZENT Then
      PROZENT = Round(SCHLEIFENPROZENT, 0)
      Application.StatusBar = PROZENT
   End If

Next Z

Application.ScreenUpdating = True
Application.StatusBar = ""
```

Stoppuhr in Excel

Autor: Max Kaffl (Nepumuk) - Erstellt: -- - Letzte Revision: --

Stoppuhr in Excel

Sie wollen einen Wettkampf stoppen, ihre Videos schneiden, oder was auch immer, und müssen dazu Zeiten, Millisekunden genau, in eine Tabelle eintragen. Kein Problem. Mit folgendem Code generieren sie eine eigene Symbolleiste, welche einer komfortablen Stoppuhr entspricht.

Die Funktion der Buttons:

- Start - Startet die Zeitmessung
- Stop - Beendet die Zeitmessung und gibt die gestoppte Zeit in der aktiven Zelle aus
- Pause - Unterbricht die Zeitmessung
- Lap - Gibt die Zwischenzeit in der aktiven Zelle aus
- Reset - Uhr zurücksetzen
- Preset - Zeit vorgeben, ab der die Zeitmessung beginnen soll

Die Zwischenzeit (Lap), kann auch mit der Tastenkombination Strg+y in die aktive, oder einem Doppelklick in eine beliebige Zelle, ausgegeben werden. Der Cursor spring, nach der Ausgabe der Zeit, automatisch eine Zelle nach unten.

Die benötigten Makros werden in drei Modulen untergebracht. Im Klassenmodul „DieseArbeitsmappe" befinden sich die Ereignisroutinen zum anlegen und löschen der Symbolleiste. Ein Standardmodul mit Namen „basCommandbar", in dem die Symbolleiste angelegt wird, sowie ein Standardmodul mit dem Namen „basClock", welches die Steuerung der Stoppuhr übernimmt.

```
' ************************************************************
' Modul: DieseArbeitsmappe Typ = Element der Mappe(Sheet, Workbook, ...)
' ************************************************************

Option Explicit

Private Sub Workbook_Activate()
 Call prc_CreateCommandBar
End Sub

Private Sub Workbook_Deactivate()
 Call prc_DeleteCommandBar
End Sub


' ************************************************************
' Modul: basCommandbar Typ = Allgemeines Modul
' ************************************************************

Option Explicit
```

```
Option Private Module

' Code Max Kaffl 2005

Public objCommandBar As CommandBar
Public objCommandBarButton(6) As CommandBarButton

Public Sub prc_CreateCommandBar()
 Call prc_DeleteCommandBar
 Set objCommandBar = CommandBars.Add(Name:="Stoppuhr", _
 Position:=msoBarFloating, Temporary:=True)
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(0), _
 enumType:=msoControlButton, varOnAction:="prc_Start", _
 varCaption:="Start", enumStyle:=msoButtonCaption, _
 varTipText:="starten")
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(1), _
 enumType:=msoControlButton, varOnAction:="prc_Stop", _
 varCaption:="Stop", enumStyle:=msoButtonCaption, _
 bolEnabled:=False, varTipText:="stoppen")
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(2), _
 enumType:=msoControlButton, varOnAction:="prc_Pause", _
 varCaption:="Pause", enumStyle:=msoButtonCaption, _
 bolEnabled:=False, varTipText:="anhalten")
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(3), _
 enumType:=msoControlButton, varOnAction:="prc_Lap", _
 varCaption:="Lap", enumStyle:=msoButtonCaption, _
 bolEnabled:=False, varTipText:="Zwischenzeit")
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(4), _
 enumType:=msoControlButton, varOnAction:="prc_Reset", _
 varCaption:="Reset", enumStyle:=msoButtonCaption, _
 bolEnabled:=False, varTipText:="zurücksetzen")
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(5), bolBeginGroup:=True, _
 enumType:=msoControlButton, varTipText:="Anzeige", _
 varCaption:="00:00:00,000", enumStyle:=msoButtonCaption)
 Call prcControlAdd(objCommandBar, _
 varControl:=objCommandBarButton(6), bolBeginGroup:=True, _
 enumType:=msoControlButton, varOnAction:="prc_Preset", _
 varCaption:="Preset", enumStyle:=msoButtonCaption, _
 varTipText:="Voreinstellung")
 With objCommandBar
 .Top = 150
 .Left = 100
 .Protection = msoBarNoChangeDock + msoBarNoChangeVisible _
 + msoBarNoCustomize + msoBarNoHorizontalDock _
 + msoBarNoResize + msoBarNoVerticalDock
 .Visible = True
 End With
```

```
End Sub

Public Sub prc_DeleteCommandBar()
 On Error Resume Next
 KillTimer lnghWnd, 0
 CommandBars("Stoppuhr").Delete
End Sub

Private Sub prcControlAdd( _
 ByRef objParent As Object, _
 Optional ByRef varControl As Variant, _
 Optional ByVal enumType As MsoControlType, _
 Optional ByVal varId As Variant, _
 Optional ByVal varBefore As Variant, _
 Optional ByVal varTemporary As Variant, _
 Optional ByVal bolBeginGroup As Boolean = False, _
 Optional ByVal varCaption As Variant, _
 Optional ByVal varFaceId As Variant, _
 Optional ByVal varOnAction As Variant, _
 Optional ByVal enumStyle As MsoButtonStyle, _
 Optional ByVal varTipText As Variant, _
 Optional ByVal enumState As MsoButtonState, _
 Optional ByVal varTag As Variant, _
 Optional ByVal enumLinkType As MsoCommandBarButtonHyperlinkType, _
 Optional ByVal bolEnabled As Boolean = True, _
 Optional ByVal bolVisible As Boolean = True, _
 Optional ByVal varWidth As Variant, _
 Optional ByVal varDropDownWidth As Variant, _
 Optional ByVal varDropDownLines As Variant)
 Dim cmbControl As CommandBarControl
 Select Case IIf(enumType, 1, 0) & IIf(IsMissing(varId), 0, 1) & _
 IIf(IsMissing(varBefore), 0, 1) & IIf(IsMissing(varTemporary), 0, 1)
 Case "0100": Set cmbControl = objParent.Controls.Add(ID:=varId)
 Case "0101": Set cmbControl = objParent.Controls.Add(ID:=varId, _
 Temporary:=varTemporary)
 Case "0110": Set cmbControl = objParent.Controls.Add(ID:=varId, _
 Before:=varBefore)
 Case "0111": Set cmbControl = objParent.Controls.Add(ID:=varId, _
 Before:=varBefore, Temporary:=varTemporary)
 Case "1000": Set cmbControl = objParent.Controls.Add(Type:=enumType)
 Case "1001": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 Temporary:=varTemporary)
 Case "1010": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 Before:=varBefore)
 Case "1011": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 Before:=varBefore, Temporary:=varTemporary)
 Case "1100": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 ID:=varId)
 Case "1101": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 ID:=varId, Temporary:=varTemporary)
 Case "1110": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
 ID:=varId, Before:=varBefore)
 Case "1111": Set cmbControl = objParent.Controls.Add(Type:=enumType, _
```

```
 ID:=varId, Before:=varBefore, Temporary:=varTemporary)
 End Select
 With cmbControl
 .BeginGroup = bolBeginGroup
 If Not IsMissing(varCaption) Then .Caption = varCaption
 If Not IsMissing(varFaceId) Then .FaceId = varFaceId
 If Not IsMissing(varOnAction) Then .OnAction = varOnAction
 If enumStyle Then .Style = enumStyle
 If Not IsMissing(varTipText) Then .TooltipText = varTipText
 If enumState Then .State = enumState
 If Not IsMissing(varTag) Then .Tag = varTag
 If enumLinkType Then .HyperlinkType = enumLinkType
 .Enabled = bolEnabled
 .Visible = bolVisible
 If Not IsMissing(varWidth) Then .Width = varWidth
 If Not IsMissing(varDropDownWidth) Then _
 .DropDownWidth = varDropDownWidth
 If Not IsMissing(varDropDownLines) Then _
 .DropDownLines = varDropDownLines
 End With
 If Not IsMissing(varControl) Then Set varControl = cmbControl
 Set cmbControl = Nothing
End Sub



' ************************************************************
' Modul: basClock Typ = Allgemeines Modul
' ************************************************************

Option Explicit
Option Private Module

' Code Max Kaffl 2005

Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" ( _
 ByVal lpClassName As String, _
 ByVal lpWindowName As String) As Long
Private Declare Function SetTimer Lib "user32" ( _
 ByVal hwnd As Long, _
 ByVal nIDEvent As Long, _
 ByVal uElapse As Long, _
 ByVal lpTimerFunc As Long) As Long
Public Declare Function KillTimer Lib "user32" ( _
 ByVal hwnd As Long, _
 ByVal nIDEvent As Long) As Long
Private Declare Function timeGetTime Lib "winmm.dll" () As Long

Public lnghWnd As Long

Private lngStartTime As Long, lngPauseTime As Long
Private lngPresetTime As Long
Private blnPause As Boolean
```

```
Private Sub prc_Start()
 Dim intIndex As Integer
 lngStartTime = timeGetTime - lngPresetTime
 With Application
 .MacroOptions Macro:="prc_Lap", _
 HasShortcutKey:=True, ShortcutKey:="y"
 .OnDoubleClick = "prc_Lap"
 End With
 blnPause = False
 lngPresetTime = 0
 objCommandBarButton(0).Enabled = False
 For intIndex = 1 To 4
 objCommandBarButton(intIndex).Enabled = True
 Next
 objCommandBarButton(6).Enabled = False
 lnghWnd = FindWindow("XLMAIN", Application.Caption)
 SetTimer lnghWnd, 0, 1, AddressOf prc_Display
End Sub

Public Sub prc_Lap()
 ActiveCell.Value = fnc_strTime(timeGetTime - lngStartTime)
 ActiveCell.Offset(1, 0).Select
End Sub

Private Sub prc_Pause()
 If blnPause Then
 lngStartTime = lngStartTime + (timeGetTime - lngPauseTime)
 objCommandBarButton(3).Enabled = True
 SetTimer lnghWnd, 0, 1, AddressOf prc_Display
 Else
 lngPauseTime = timeGetTime
 objCommandBarButton(3).Enabled = False
 KillTimer lnghWnd, 0
 objCommandBarButton(5).Caption = fnc_strTime(timeGetTime - lngStartTime)
 End If
 blnPause = Not blnPause
End Sub

Private Sub prc_Stop()
 Dim intIndex As Integer
 If blnPause Then _
 lngStartTime = lngStartTime + (timeGetTime - lngPauseTime)
 ActiveCell.Value = fnc_strTime(timeGetTime - lngStartTime)
 KillTimer lnghWnd, 0
 For intIndex = 1 To 3
 objCommandBarButton(intIndex).Enabled = False
 Next
 objCommandBarButton(5).Caption = ActiveCell.Text
 ActiveCell.Offset(1, 0).Select
End Sub

Private Sub prc_Reset()
```

```
 Dim intIndex As Integer
 KillTimer lnghWnd, 0
 lngStartTime = 0
 objCommandBarButton(0).Enabled = True
 For intIndex = 1 To 4
 objCommandBarButton(intIndex).Enabled = False
 Next
 objCommandBarButton(5).Caption = "00:00:00,000"
 objCommandBarButton(6).Enabled = True
 With Application
 .MacroOptions Macro:="prc_Lap", _
 HasShortcutKey:=True, ShortcutKey:=""
 .OnDoubleClick = ""
 End With
End Sub

Private Sub prc_Preset()
 Dim vntInput As Variant
 Do
 vntInput = InputBox("Vorgebezeit im Format hh:mm:ss eingeben.", _
 "Eingabe", "00:00:00")
 If StrPtr(vntInput) = 0 Then Exit Sub
 If vntInput Like "##:##:##" And IsDate(vntInput) Then Exit Do
 MsgBox "Fehlerhafte Eingabe.", 48, "Hinweis"
 Loop
 lngPresetTime = CDbl(CDate(vntInput)) * 86400000
 objCommandBarButton(5).Caption = fnc_strTime(lngPresetTime)
End Sub

Private Sub prc_Display(ByVal hwnd As Long, ByVal nIDEvent As Long, _
 ByVal uElapse As Long, ByVal lpTimerFunc As Long)
 objCommandBarButton(5).Caption = fnc_strTime(timeGetTime - lngStartTime)
End Sub

Private Function fnc_strTime(ByVal lngTime As Long) As String
 Dim lngHour As Long, lngMinute As Long, lngSecond As Long
 lngHour = lngTime \ 3600000
 lngMinute = (lngTime Mod 3600000) \ 60000
 lngSecond = (lngTime Mod 3600000 Mod 60000) \ 1000
 lngTime = lngTime Mod 3600000 Mod 60000 Mod 1000
 fnc_strTime = Format(CStr(lngHour), "00") & ":" & _
 Format(CStr(lngMinute), "00") & ":" & _
 Format(CStr(lngSecond), "00") & "," & _
 Format(CStr(lngTime), "000")
End Function
```

Titelleiste des Excelfensters ändern

```
Sub TitelWechseln()
    Application.Caption = "Veränderte Titelleiste"
End Sub
```

VBA-Code durch User Abbrechen

Siehe Kapitel TASTEN

VBA-Code löscht sich selbst

The code below can be used to delete the module which houses the code. In other words, it deletes itself after running once.
You will have to go to Tools>Macro>Security - Trusted Publishers and check Trust access to Visual Basic Editor before running the code. Change "Module1" to suit.

```
Sub DeleteThisModule()

Dim vbCom As Object

    MsgBox "Hi, I will delete myself "

    Set vbCom = Application.VBE.ActiveVBProject.VBComponents

    vbCom.Remove VBComponent:= _

    vbCom.Item("Module1")

End Sub
```

VBA-Code löschen in "Diese Arbeitsmappe"

Delete ThisWorkbook Event Code Macro. Delete VBA Code With Code
WorkbookEvent Code
Excel is rich in Event Code for both the Workbook Object and the Sheet Object. For example, the Workbook Open Event is possibly the most popular of Workbook Events (ThisWorkbook).

Delete Workbook Event Code With Code. See Also Delete Sheet Event Code With Code & Delete Module After Running VBA Code

You will have to go to Tools>Macro>Security - Trusted Publishers and check Trust access to Visual Basic Editor before running the code

```
Sub DeleteWorkbookEventCode()

' Needs Reference Set To _    "Microsoft Visual Basic For Applications Extensibility"

'Tools>References.

    With ThisWorkbook.VBProject.VBComponents("ThisWorkbook").CodeModule

        .DeleteLines 1, .CountOfLines

    End With
End Sub
```

Version 2

```
Function bRemoveAllCode(ByVal szBook As String) As Boolean

    Const lModule As Long = 1
    Const lOther As Long = 100

    Dim lCount As Long
    Dim objCode As Object
    Dim objComponents As Object
    Dim wkbBook As Workbook

    On Error GoTo bRemoveAllCodeError

    Set wkbBook = Workbooks(szBook)
    Set objComponents = wkbBook.VBProject.VBComponents
    lCount = wkbBook.VBProject.VBComponents.Count

    '''Remove all modules & code
    For Each objCode In objComponents
        If objCode.Type = lModule Then
            objComponents.Remove objCode
        ElseIf objCode.Type = lOther Then
            objCode.CodeModule.DeleteLines 1,
objCode.CodeModule.CountOfLines
        End If
    Next objCode

    bRemoveAllCode = True
    Exit Function

bRemoveAllCodeError:
```

```
    bRemoveAllCode = False
End Function
Sub PrepBook()
If Not bRemoveAllCode(ActiveWorkbook.Name) then MsgBox "An error _
occurred!", vbCritical,"bRemoveAllCode"
End sub
```

Vollbildmodus

Siehe auch Bereich MENÜLEISTEN - AUSBLENDEN MENÜPUNKTE

```
Sub VOLLBILD_MODUS()

    Application.DisplayFullScreen = True ' Aktiviert Vollbildmodus
    Application.DisplayStatusBar = False ' Statuszeile ausblenden
    Application.DisplayFormulaBar = False ' Zelladresse und Formelzeile ausblenden
    Application.ExecuteExcel4Macro "Show.Toolbar(""Ribbon"", False)" ' Ribbon-Menüleiste ausblenden

    With ActiveWindow
        .DisplayHorizontalScrollBar = False ' horizontale Bildlaufleiste ausblenden
        .DisplayVerticalScrollBar = False ' vertikale Leiste
        .DisplayWorkbookTabs = False ' Blattregisterkarten
        .DisplayGridlines = False ' Gitternetzlinien ausblenden
        .DisplayHeadings = False ' Spaltenbuchstaben und Zeilennummern ausblenden
        .DisplayOutline = False ' Gliederungssymbole ausblenden
        .DisplayZeros = False ' zeigt keine Nullwerte an
    End With

End Sub


Sub EDITOR_MODUS()

    Application.DisplayFullScreen = False ' Deaktiviert Vollbildmodus
    Application.DisplayStatusBar = True ' Statuszeile einblenden
    Application.DisplayFormulaBar = True ' Zelladresse und Formelzeile einblenden
    Application.ExecuteExcel4Macro "Show.Toolbar(""Ribbon"", True)" ' Ribbon-Menüleiste einblenden

    With ActiveWindow
        .DisplayHorizontalScrollBar = True ' horizontale Bildlaufleiste einblenden
        .DisplayVerticalScrollBar = True ' vertikale Leiste
        .DisplayWorkbookTabs = True ' Blattregisterkarten
```

```
    .DisplayGridlines = True ' Gitternetzlinien einblenden
    .DisplayHeadings = True ' Spaltenbuchstaben und Zeilennummern einblenden
    .DisplayOutline = True ' Gliederungssymbole einblenden
    .DisplayZeros = True ' zeigt keine Nullwerte an
  End With

End Sub
```

Warten / Pause für Anzahl Sekunden

Theorie: Einen großen Vorteil hat die Application.Wait Now – Variante (siehe Variante 0 hier) – sie blockiert nicht die CPU-Ressourcen im Gegensatz zur Variante 1, die den Rechner mit Milliarden Wurzelfunktionen beschäftigt. Wenn es nur darum geht, Excel kurz pausieren zu lassen, geht das mit beiden Varianten.

Möchte man aber eine Pause machen um zB auf eine ferngesteuerte Anwendung zu warten, bis sie mit einem Schritt fertig ist, dann sollte man nur mit der Variante Application.Wait – Variante 0 arbeiten, weil die die CPU freigibt, dass die andere Anwendung ihre Arbeit machen kann.

BSP: fügt man in Word ein Textfeld ein und möchte seine Markierung aufheben, geht dies nur durch Übersenden eines simulierten doppelten ESC-Tastendruck. Word braucht eine halbe Sekunde, um die Markierung des Textfeldes aufzuheben. Gibt man Word diese halbe Sekunde nur, indem man die CPU in Excel auf Trab hält mit Variante 1 (Wurzelrechnungen), dann steht Word zu wenig Prozessor-Leistung zur Verfügung, um seinen Fokus aufzuheben.

Variante 0 - verbraucht keine CPU-Power hat aber nur Sekunden-Taktung

```
Application.Wait Now + TimeSerial(0, 0, 2) ' 2 Sekunden warten
```

ACHTUNG: bei Werten kleiner 1 wird gar nicht gewartet !!! da Application.OnTime als kleinste Zeiteinheit nur die Sekunde verarbeiten kann.

Variante 1 - verbraucht volle CPU-Power, erlaubt aber kleinere Wartezeiten als 1 Sekunde (wie auch Variante 2)

```
' ---- Zentraler Code ---

Sub TESTE_CPU()
```

```
' Mit dieser Funktion wird anfangs einmal die Leistung der CPU getestet und damit die Funktion "WARTE" geeicht
' konkret: wie oft eine Mehrfach-Wurzelfunktion in der Sekunde durchlaufen werden kann (Variable ANZAHL)
' Da wir anschließend aber in Millisekunden Pausen machen wollen, wird nur
' ein Tausendstel der Anzahl der Schleifendurchgänge als Schleifenanzahl

' Variablendefinition

    Dim JETZT As Date                    ' die aktuelle Uhrzeit (inkl Datum)
    Dim ANZAHLJESEKUNDE As Single        ' dies ist der zentrale Stärkewert der CPU, der angibt, wie oft die Hauptschleife
durchlaufen werden kann
    Dim ANZAHLJEMILLISEKUNDE As Long     ' dies ist der für uns wichtige zu ermittelnde Zielwert, wie oft die Schleife für
eine Millisekunde brauchen wird
    Dim DUMMYWERT As Long                ' eine reine Hilfsvariable, die als Empfänger für die zentrale Bremsfunktion
(Mehrfache-Wurzelfunktion) dient


' Code

    JETZT = Now ' aktuelle Uhrzeit merken

    While Now = JETZT ' Schleife, die bis zum Ende der aktuellen Sekunde wartet
    Wend

    JETZT = Now ' Merken der nächsten Sekunde

    ' Hauptschleife für die CPU-Testung mit max. 9 Mrd. Durchgängen
    ' - 2011 konnte mein Intel i7/960  2 Mio. Durchgänge schaffen - die Schleife bietet also genug Luft nach oben,
    '   2 Mio. Durchgänge ergibt für die Millisekunde (kleinste Takteinheit meines Tools) einen Wert von 2.000,
    '   daher: selbst alte Rechner sollten einen ausreichend hohen Wert von zumindest einigen Dutzend erreichen

    For ANZAHLJESEKUNDE = 1 To 9000000000#

        DUMMYWERT = Sqr(Sqr(Sqr(Sqr(Sqr(2)))))

        If JETZT <> Now Then Exit For ' wenn die aktuelle Sekunde vorbei ist, Schleife verlassen

    Next ANZAHLJESEKUNDE

    ANZAHLJEMILLISEKUNDE = Int(ANZAHLJESEKUNDE / 1000)

    Range("A1") = ANZAHLJEMILLISEKUNDE ' Speichern des Ergebnisses in einer Zelle
```

```
    MsgBox "In 1 Sekunde konnte die Hauptschleife " & ANZAHLJESEKUNDE & " Mal durchlaufen werden." & vbCrLf & _
    "Der Wert für die Millisekundenbremse lautet daher: " & ANZAHLJEMILLISEKUNDE


End Sub

Public Sub WARTE(ByVal MILLISEKUNDE As Long)

' Dies ist die zentrale Wartefunktion, der man die Anzahl von Millisekunden übergibt, die sie warten soll

' Variablendefinition

    Dim ANZAHLJEMILLISEKUNDE As Long   ' dies ist der zuvor geeichte CPU-Wert, wie oft die Wurzelfunktion-Schleife
durchlaufen werden muss, um eine Millisekunde zu brauchen
    Dim LAUF1 As Long                  ' Variable für die Hauptschleife, die die Anzahl von Millisekunden durchläuft, die
man dieser Prozedur hier variabel übergeben kann
    Dim LAUF2 As Long                  ' Variable für die zentrale "Brems-Schleife", die bis zur Variable
ANZAHLJEMILLISEKUNDE hochzählt und genau 1 Millisekunde dauert
    Dim DUMMYWERT                      ' eine reine Hilfsvariable, die als Empfänger für die zentrale Bremsfunktion
(Mehrfache-Wurzelfunktion) dient

' Code

    ' Auslesen der "CPU-Power" (Anzahl Schleifendurchgänge der zentralen Wurzelfunktions-Schleife für eine Millisekunden)

    ANZAHLJEMILLISEKUNDE = Range("A1")

    ' Schleife, die die Anzahl von vorgegebenen Millisekunden durchwandert
    For LAUF1 = 1 To MILLISEKUNDE

        ' Schleife, deren Durchlauf genau 1 Millisekunde dauert
        For LAUF2 = 1 To ANZAHLJEMILLISEKUNDE
            DUMMYWERT = Sqr(Sqr(Sqr(Sqr(Sqr(2)))))
        Next LAUF2

    Next LAUF1

End Sub


' --- Demo-Code (nicht notwendig) ---
```

```
Sub Demo()

' Eine nicht benötigte Demoprozedur, der man zu Testzwecken in der Variable "Millisekunden" vorgeben kann
' was für ein Sekundenintervall sie zeigen soll

    ' Wieviele Millisekunden lange soll ein Takt dauern
    Millisekunden = 1000

    ' Demoschleife, die von 1 bis 10 hochzählen soll
    For T = 1 To 10

        ' Wartebefehl mit Anzahl von Millisekunden (der Wert 1000 ergibt also 1 Sekunde)
        WARTE (Millisekunden)

        ' Sichtbarmachung der Schleifendurchläufe
        Range("B1") = T

    Next T

End Sub
```

Variante 2 - Auch für Zeiten unter einer Sekunde

```
Private Declare Sub Sleep Lib "kernel32.dll" ( ByVal dwMilliseconds As Long)

Public Sub test()
     Sleep 5000
End Sub
```

Variante 3

```
Application.OnTime Now + TimeValue("00:00:05"), "Daily_Realtime_End"
```

Variante 4

```
' 3 Sekunde warten
STUNDE = Hour(Now())
MINUTEN = Minute(Now())
SEKUNDE = Second(Now()) + 3
WAITTIME = TimeSerial(STUNDE, MINUTEN, SEKUNDE)
Application.Wait WAITTIME
```

siehe auch nächstes Kapitel "Zeitsteuerung von Excel"

Warten mit SLEEP, WAIT und DOEVENTS

Idle time is often wasted time, however, sometimes you just need to wait for certain events to happen before you can continue code execution. VBA extends a couple of approaches to managing your idle time – the most popular approach is the Sleep procedure. The Sleep procedure pauses code execution for a certain amount of time putting the whole processes in a type of coma. It is one of the most popular approaches to pausing code execution, and at the same time simplest one. As I will try to prove – there are better, more productive approaches to pausing your code execution or utilizing potentially application idle time.

VBA Sleep function

Let us start by introducing the VBA Sleep function. The Sleep function pauses the entire process for a certain delay specified in milliseconds.

Definition and Syntax

The Sleep function is not available by default in VBA, and has to be imported from the *kernel32* library. For 64 Bit procedures we need to make sure to append the *PtrSafe* statement.

```
If VBA7 Then
    Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal milliseconds As LongPtr) 'MS Office 64 Bit
Else
    Public Declare Sub Sleep Lib "kernel32" (ByVal milliseconds as Long) 'MS Office 32 Bit
End If
```

Parameters:

milliseconds

The amount of milliseconds the process should be paused for.

VBA Sleep example

The Sleep function is pretty straightforward to use:

Sleep 1000 'Sleep for 1000 milliseconds = 1 sec

Other VBA Sleep examples

' Sleep for 10 seconds
Sleep 10000

' Sleep for 1 minute
Sleep 60000 '60 * 1000

Pros and Cons

The **pros** of using the Sleep function:

- Easy to use
- Precise sleep intervals (in milliseconds)

The **cons** of using the Sleep function:

- Requires importing Sleep function
- Freezes the entire process i.e. Excel does not respond
- There is no way to stop the Sleep function

The problem with the Sleep function is that it *freezes* your process entirely – preventing any input or interaction with your application (even breaking code execution). Why not use that time more productively? Or let the user cancel code execution?

VBA Application.Wait function

The VBA Application.Wait is a native VBA function that pauses code execution until a certain time is reached.

The syntax of the VBA Application.Wait function is:
Application.Wait(time)

Parameters

time

The time when the function should return e.g. 15:30:00 for 3:30 pm.

Beispielcode

Application.Wait example

The Application.Wait function is similarly pretty easy to use, although a little different than what you might expect with the Sleep function. This is because Application.Wait will wait until a time is reached instead of waiting for a precise interval of time.

```
Application.Wait "15:30:00"
```

This approach is of course less practical as usually you want to wait simply for a precise interval of time, like say 3 seconds. That is why we need to aid ourselves with the use of either the DateAdd function or the TimeValue function:

```
Application.Wait DateAdd("s", 1, Now) 'Wait for 1 second

'same as

Application.Wait Now + TimeValue("0:00:01) 'Wait for 1 second
```

What does the function above do? It adds 1 second to the current clock time and asks VBA to wait until that moment to return from the function. You can similarly wait for longer periods of time.

Other VBA Wait examples

```
' Wait for 10 seconds
Application.Wait DateAdd("s", 10, Now)

' Wait for 1 minute
Application.Wait DateAdd("n", 1, Now)

' Wait for 1 hour
Application.Wait DateAdd("h", 1, Now)

' Wait for 1 minute 30 seconds
Application.Wait DateAdd("s", 90, Now) '00:01:30 = 60 + 30
```

Pros and Cons

The **pros** of using the Application.Wait function:

- Fairly easy to use (a little less obvious than Sleep)
- You can break the function at any time (does not freeze the entire process)
- Available natively from VBA

The **cons** of using the Application.Wait function:

- Allows you to wait for intervals shorter no shorter than 1 second

The Application.Wait function is a little less obvious to use, but won't freeze your VBA Project allowing you to hit the ESC-Button at any time and resume control over your project.

Third option: VBA DoEvents

If you thought those were your only options – you were wrong. VBA thankfully allows you to also use another function called DoEvents. The function seemingly….does nothing. Seriously, it doesn't do anything more that handle all MS Office events. Now why would we want to use the DoEvents function you might ask? Well, remember that Application.Wait does not allow you to wait for intervals shorter than 1 second? On the other hand Sleep freezes your application entirely although being more granular.

Want an example of the VBA DoEvents function?

DoEvents 'That's it!

DoEvents is especially useful when you have a lot of computations going on but want to keep your VBA Project responsive WITHOUT introducing significant (1 second) delays.

VBA DoEvents example

Usually when running a macro you will want to turn off ScreenUpdating focusing your macro only on your computations. This however may seem to your user like the application has frozen.

DoEvents is to be used in pair with the ScreenUpdating property:

```
Application.ScreenUpdating = False 'Turn off screen updating
'...code here...
DoEvents 'Refresh the screen "on demand"
'...code here...
```

Usually you might want to use it like this:

```
Application.ScreenUpdating = False 'Turn off screen updating
'...code here...
Do Until someThingHappened = True
  'I am waiting for something to happen
  DoEvents
```

```
Loop
'...code here...
```

Conclusions

Let's summarize our findings:

Use Application.Wait for > 1 second intervals. Application.Wait can be used well to pause code execution without freezing the application as long as you need to pause for more that 1 second (integer values).

Use DoEvents to refresh the screen on demand. Don't want your user to get restless? Want to pause for a minimal interval just to refresh your application screen? Don't bother with Sleep and use DoEvents instead!

Use Sleep for precise intervals. Need to wait for 500 milliseconds – no more no less? Ok… seems like you are stuck with the Sleep function.

Zeitsteuerung von Excel

mit der Funktion OnTime und der Übergabe eines Zeitpunktes, kann das automtatische ausführen einer Prozedur zu diesem Zeitpunkt festgelegt werden.

Nachfolgender Code legt fest, dass 1 Minute nach Aufruf der Prozedur ZEITANZEIGE, die Prozedur UHRZEIT ausgeführt wird. Letztere zeigt für 5 Sekunden die aktuelle Uhrzeit und das Datum in der Statusleiste an. Wenn die letzte Zeile "Call Zeitanzeige" aktiviert wird (Call kann man auch weglassen), dann ruft die Uhrzeit zuletzt die Prozedur Zeitanzeige auf, um die nächste 1-Minutenfrist zu starten - so kommt es also zu einer Endlosschleife:

```
Sub Zeitanzeige()
  Application.OnTime Now + TimeValue("00:01:00"), _
  "Uhrzeit"
End Sub

Sub Uhrzeit()
    Application.DisplayStatusBar = True
    Application.StatusBar = Date & "," & Time
```

```
    Application.Wait (Now + TimeValue("0:00:05"))
    Application.StatusBar = False
    'Call Zeitanzeige
End Sub
```

Man kann auch Fixzeiten eingeben:

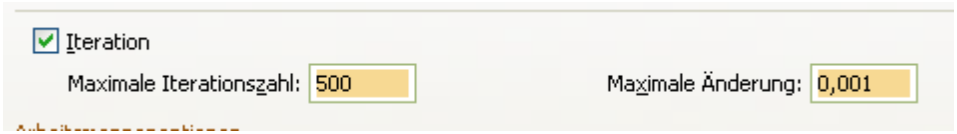
```
Sub ZeitSteuerung()
 Application.OnTime TimeValue("16:00:00"), "Verarbeitung"
End Sub
```

Zellsprungrichtung

```
Application.MoveAfterReturnDirection = xlDown
Application.MoveAfterReturnDirection = xlToRight
Application.MoveAfterReturn = False ' kein Sprung
```

Zirkelbezug

Zirkelbezug ist ja nichts falsches - man muss nur Excel die Möglichkeit geben mittels Iteration die Lösung zu finden (Extras - Optionen - Berechnen..



Einmal aktiviert bleibt diese Iteration solange aktiv, wie die Excelanwendung läuft. Schließt und öffnet man eine Vorlage mit Zirkelbezug, kommt keine Fehlermeldung über den Zirkelbezug und Excel rechnet brav. Schließt man aber Excel ganz und öffnet die Vorlage neu, dann kommt auch der Fehlerhinweis.

Diesen kann man auch nicht mit application.displayalerts in Workbook_Open-Bereich entfernen, weil der Fehlerhinweis schon kommt, bevor Excel diesen VBA-Teil abarbeitet. Man kann aber folgenden Code direkt in den Workbook-Open-Bereich kopieren - dann kommt zwar beim Öffnen ein Fehlerhinweis, aber sobald man diesen deaktiviert ist bei Excel für diese Sitzung der Excelanwendung die Iteration aktiviert und Excel löst automatisch den Zirkelbezug immer wieder auf.

```
With Application
    .Iteration = True
    .MaxIterations = 500
    .MaxChange = 0.001
End With
ActiveWorkbook.PrecisionAsDisplayed = False
```

Zwischenablage befüllen, auslesen, leeren

Inhalt der Zwischenablage löschen

```
Application.CutCopyMode= False
```

MEIN WEG 1)

a.) Code, wenn man mit COPY-Button in Userform ein Feld (Label) in die Zwischenablage kopieren will

```
Private Sub CommandButton4_Click()
        Dim MyData As DataObject
        Set MyData = New DataObject
        MyData.SetText Label26.Caption
        MyData.PutInClipboard
End Sub
```

b.) wenn man einen Zellenwert in der Tabelle in die Zwischenablage kopieren will

```
Sub TEST

        Dim MyData As DataObject
        Set MyData = New DataObject
        MyData.SetText Range("A2").Value
        MyData.PutInClipboard

End Sub
```

WEG 2) Dass man aus VBA auf die Zwischenablage (Clipboard) zugreifen kann, ist relativ unbekannt.
Dabei kann man dies mit dem DataObject zumindest für Texte relativ leicht lösen.

Sie benötigen einen Verweis auf die
Microsoft Forms 2.0 Objektlibrary
(Sollte Sie diese nicht so einfach finden, einfach eine Userform einfügen und dann wieder entfernen)

```
' ************************************************************
'  Modul:  mdlCopy  Typ = Allgemeines Modul
' ************************************************************

Option Explicit
' ************************************************************
```

```
' Benötigt einen Verweis auf die Microsoft Forms 2.0 Objektlibrary
' Peter Haserodt 2004
' ***********************************************************

Public Sub TesteEs()
        SchreibeTextInDieZwischenablage
        MsgBox HoleTextVonZwischenablage
        RausMitZwischenAblage
        MsgBox HoleTextVonZwischenablage
End Sub

Public Sub SchreibeTextInDieZwischenablage()
        Dim oData As New DataObject
        Dim sText As String
        sText = "Hallo Leute" & vbCrLf & "Neues von Online Excel"
        With oData
                .SetText sText
                .PutInClipboard
        End With
End Sub

Public Function HoleTextVonZwischenablage() As String
        Dim oData As New DataObject
        On Error Resume Next ' Brutal um falsche Formate abzuwürgen, gibt dann einen Leerstring
        oData.GetFromClipboard
        HoleTextVonZwischenablage = oData.GetText
End Function

Public Sub RausMitZwischenAblage()
        Dim oData As New DataObject
        oData.SetText ""
        oData.PutInClipboard
End Sub
```

Befüllen der Zwischenablage - WEG 3)

Um mit Visual Basic etwas in die Zwischenablage zu bekommen, sind ein paar Klimmzüge notwendig:

Kopiere den Text aus `strText` in die Zwischenablage.

```
Dim MyData As DataObject
Set MyData = New DataObject
MyData.SetText strText
```

```
MyData.PutInClipboard
```

Zwischenergebnisse andrucken

Mit Messagebox muss man diese immer abklicken

1.) Besser ist es diese entweder in der Statusbar anzuzeigen

```
Sub Meldung()
 Application.StatusBar = "Hallo ich bin die Meldung in der Statuszeile"
End Sub
```

```
' Meldung deaktivieren
Sub Meldung_aus()
 Application.StatusBar = False
End Sub
```

2.) Eine Liste von Werten in das Direktfenster von VBA reinschreiben (mit STRG-G kann man es im VBA-Fenster öffnen oder im Menü ANSICHT)

Debug.print "Schreib war rein" ’ dies ist die nützliche debugprint-funktion

FORMATIERUNG

Eine Zeile mit automatischer Umformatierung anbieten

Es gibt einen Button 

mit folgendem Code

```
Sub FORMATIEREN()

    ' Einblenden Zeile 9 mit Formatdropdown
    Tabelle2.Rows(9).Hidden = Not Tabelle2.Rows(9).Hidden

End Sub
```

Die Zeile selbst sieht so aus

Und mit folgendem Code im Tabellenblatt wird die Tabelle automatisch neu formatiert. Achtung: aktuell werden "nur" die obersten 50.000 Zellen formatiert

```
Private Sub Worksheet_Change(ByVal Target As Range)

    If Target.Row = 9 Then
        If Target.Value <> "" Then ' wenn Formatierungswunsch ausgefüllt ist
            S = Target.Column
            If Left(Target.Value, 4) = "DATU" Then Range(Cells(11, S), Cells(50000, S)).NumberFormat = "dd/mm/yyyy"
            If Left(Target.Value, 4) = "BETR" Then Range(Cells(11, S), Cells(50000, S)).NumberFormat = "#,##0.00"
            If Left(Target.Value, 4) = "GANZ" Then Range(Cells(11, S), Cells(50000, S)).NumberFormat = "0"
            If Left(Target.Value, 4) = "STAN" Then Range(Cells(11, S), Cells(50000, S)).NumberFormat = "General"
            If Left(Target.Value, 4) = "TEXT" Then Range(Cells(11, S), Cells(50000, S)).NumberFormat = "@"
        End If
    End If

End                                                                                                    Sub
```

FUNKTIONEN - PROZEDUREN ALLGEMEIN

Eine eigene Funktion mit Parameterübergabe

1. Beispiel

```
Public Function BETRAG_US(BETRAGSTEXT As String) As String
' aus dem seltsamen Paypalbetragsformat 1,234.56 soll 1234,56 werden

    BETRAGSTEXT = Replace(BETRAGSTEXT, ".", "Komma")
    BETRAGSTEXT = Replace(BETRAGSTEXT, ",", "")
    BETRAGSTEXT = Replace(BETRAGSTEXT, "Komma", ",")
    BETRAG_US = BETRAGSTEXT

End Function

Sub teste()

BETRAG = "1,234.56"
MsgBox BETRAG_US("1,234.56")

End Sub
```

2. Beispiel

Wir möchten eine Funktion "BMD_DATUM" erstellen, die ein an sie übergebenes Datum umwandelt in das BMD-Format

Sie soll nachfolgenden Code korrekt ausführen

```
Sub test()
        MsgBox BMD_DATUM("12122009")
End Sub
```

Uns so muss die Funktion aussehen

```
Function BMD_DATUM(DATUM As String) As String

   BMD_DATUM = Right(DATUM, 4) & Mid(DATUM, 3, 2) & Left(DATUM, 2)

End Function
```

Wichtig: man kann eine Funktion nicht alleinstehen aufrufen - also es kann keine Programmzeile geben

BMD_DATUM ("1234567")

Es muss immer jemand das Ergebnis der Funktion erhalten : Z=BMD_DATUM ("1234567").

Möchte man kein Ergebnis erhalten, sondern nur eine Sub-Routine haben, an die man Variablen übergibt und die soll selbständig arbeiten, ohne dass man von ihr einen Wert zurückerhält, dann kann man entweder die Variablen global stellen - oder sie einfach nur an die Prozedur selbst übergeben. Siehe auch hier im Bereich VARIABLEN - ÜBERGABE VON VARIABLEN AN EINE PROZEDUR.

UST-Funktion mit mehrfacher Parameterübergabe und mehrfachen Rückgabewerten

Wir wollen eine Funktion, die aus an sie übergebenen Werten – Brutto, Netto, Steuerbetrag und Steuerprozent – jene Werte errechnet, die nicht an sie übergeben wurden (also mit dem Wert 0). Konkret wollen wir – egal welche 2 Werte wir übergeben – dass die restlichen zwei Werte ermittelt werden.

```
Sub USTTESTER()

   BRUTTO = Range("F11")
   NETTO = Range("G11")
   STEUER = Range("I11")
   PROZENT = Range("H11")

   A = UST(BRUTTO, NETTO, STEUER, PROZENT)

   MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT & "%"
End Sub

Public Function UST(BRUTTO, NETTO, STEUER, PROZENT) As Boolean

' - diese Function rechnet von den an sie übergebenen Teilen die restlichen Teile aus
' - alle Variablen werden als Variant geführt, damit sie nicht definiert sein müssen
' - vor der Rückgabe werden sie alle auf 2 Stellen gerundet, die Prozent auf 0 Stellen (wobei auch die erste Kommastelle vor dem Runden geprüft wird,
' - sprich 20,04 % werden auf 20,0 gerunden und werden toleriert, 20,05 werden auf 20,1 aufgerundet und als Fehler ausgewiesen
' - Wenn kein eindeutiger Steuerprozentsatz ermittelt werden kann, wird der Prozentsatz 99 zurückgegeben
' - Beim Ermitteln von Beträgen (Brutto/Netto/Steuer) werden diese prinzipiell nur berechnet, wenn sie leer übergeben wurden
' - Die Function UST selbst ist als Boolean definiert, weil sie mit einer Hilfvariabel aufgerufen werden muss;
```

```
'   diese Hilfsvariable ist hier a, aber man kann jede beliebige Variable nehmen, da sie keine Relevanz hat.
' - der Aufruf dieser Funktion hier lautet:
'   a = UST(BRUTTO, NETTO, STEUER, PROZENT)
'   MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT


' ----------------------------
' --  Berechnen der Beträge --
' ----------------------------

' --- Wenn Bruttobetrag vorhanden ---

If BRUTTO <> 0 Then
   ' wenn Netto vorhanden => Steuer
   If NETTO <> 0 And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
      GoTo WEITER1
   End If
   ' wenn Steuer vorhanden => Netto
   If STEUER <> 0 And NETTO = 0 Then
      NETTO = BRUTTO - STEUER
      GoTo WEITER1
   End If
   ' wenn Prozent vorhanden => Netto + Steuer
   If PROZENT <> "" And NETTO = 0 Then
      NETTO = BRUTTO / (1 + PROZENT / 100)
   End If
   If PROZENT <> "" And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If

WEITER1:

' --- Wenn Nettobetrag vorhanden ---

If NETTO <> 0 Then
   ' wenn Brutto vorhanden => Steuer
   If BRUTTO <> 0 And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
      GoTo WEITER2
   End If
   ' wenn Steuer vorhanden => Netto
   If STEUER <> 0 And BRUTTO = 0 Then
```

```
      BRUTTO = NETTO + STEUER
      GoTo WEITER2
   End If
   ' wenn nur Prozent vorhanden sind => Brutto + Steuer
   If PROZENT <> "" And BRUTTO = 0 Then
      BRUTTO = NETTO * (1 + PROZENT / 100)
   End If
   If PROZENT <> "" And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If

WEITER2:

' --- Wenn Steuerbetrag vorhanden ---

If STEUER <> 0 Then
   ' wenn Brutto vorhanden => Netto
   If BRUTTO <> 0 And NETTO = 0 Then
      NETTO = BRUTTO - STEUER
      GoTo WEITER3
   End If
   ' wenn Netto vorhanden => Brutto
   If NETTO <> 0 And BRUTTO = 0 Then
      BRUTTO = NETTO + STEUER
      GoTo WEITER3
   End If
   ' wenn nur Prozent vorhanden sind => Brutto + Steuer
   If PROZENT <> "" And BRUTTO = 0 Then
      BRUTTO = STEUER * 100 / PROZENT + STEUER
   End If
   If PROZENT <> "" And NETTO = 0 Then
      NETTO = STEUER * 100 / PROZENT
   End If
End If

WEITER3:

' --- Von den Prozent ev. Fehlendes errechnen ---

If PROZENT <> "" Then
   ' Brutto ermitteln
   If BRUTTO = 0 Then
      If NETTO <> 0 Then
```

```
            BRUTTO = NETTO + NETTO * PROZENT / 100
         End If
         If STEUER <> 0 And BRUTTO = 0 Then ' nur wenn Brutto immer noch 0 ist
            BRUTTO = STEUER * 100 / PROZENT + STEUER
         End If
      End If
      ' Netto ermitteln
      If NETTO = 0 Then
         If BRUTTO <> 0 Then
            NETTO = BRUTTO / (1 + PROZENT / 100)
         End If
         If STEUER <> 0 And NETTO = 0 Then ' nur wenn Netto immer noch 0 ist
            NETTO = STEUER * 100 / PROZENT
         End If
      End If
      ' Steuer ermitteln
      If STEUER = 0 Then
         If BRUTTO <> 0 Then
            STEUER = BRUTTO * PROZENT / (100 + PROZENT)
         End If
         If NETTO <> 0 And STEUER = 0 Then ' nur wenn Steuer immer noch 0 ist
            STEUER = NETTO * PROZENT / 100
         End If
         If STEUER <> 0 And STEUER = 0 Then
            NETTO = STEUER * 100 / PROZENT
         End If
      End If

End If

If PROZENT = 0 Then
   If BRUTTO = 0 Then
      BRUTTO = NETTO + STEUER
   End If
   If NETTO = 0 Then
      NETTO = BRUTTO - STEUER
   End If
   If STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If


' --- die Prozent berechnen ---
```

```
If PROZENT = 0 Then

    ' default Prozent auf ERROR stellen 99
    PROZENT = 99

    If BRUTTO <> 0 Then
        If BRUTTO = NETTO Then
            PROZENT = 0
         Else
           If NETTO <> 0 Then ' Abfangen, falls Netto nicht bekannt
               PROZENT = Round((BRUTTO - NETTO) / NETTO * 100, 1) ' wir runden auf eine Kommastelle
           End If
        End If
      Else ' wenn brutto nicht bekannt ist
        PROZENT = Round(STEUER / NETTO * 100, 1)
    End If

End If

' --- zuletzt noch alles runden ---

BRUTTO = WorksheetFunction.Round(BRUTTO, 2) ' mit vorangestelltem WorksheetFunction rundet Round kaufmännisch korrekt - 1,25 wird 1,3 statt 1,2
NETTO = WorksheetFunction.Round(NETTO, 2)
STEUER = WorksheetFunction.Round(STEUER, 2)
PROZENT = WorksheetFunction.Round(PROZENT, 1)

' Wenn Prozentwert vorhanden, aber kein existierender echter => setze ihn auf 99
If PROZENT <> 20 And PROZENT <> 10 And PROZENT <> 13 And PROZENT <> 0 And PROZENT <> "" Then
    PROZENT = 99
End If



End Function

Sub TEST()

Dim BRUTTO
Dim NETTO
Dim STEUER
Dim PROZENT

NETTO = 24
```

```
STEUER = 4

a = UST(BRUTTO, NETTO, STEUER, PROZENT) ' A ist nur eine Hilfsvariable zum Aufrufen vom Typ Boolean

MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT


End Sub
```

Hier das Ergebnis der Funktion – oben die Eingaben für den UST-Rechner – und unten das Ergebnis – 99 bei Prozent steht für nicht korrekte Prozent.
Die Prozent werden relativ streng gerundet und geprüft ob sie 20, 13, 10 oder 0% sind – bei 100 € darf bei der Steuer eine Unschärfe von 5 Cent sein – bei 6 Cent gilt es nicht mehr als richtig

Angabe	4 Werte	3 Werte					2 Werte					1 Wert				Fehlangabe => 99 %				Unschärfen	
Netto	10,00	10,00	10,00	10,00		10,00	10,00		10,00				10,00			8,00	10,00	10,00		100,00	100,00
Prozent	20	20	20		20	20		20		20				20							
Steuer	2,00	2,00		2,00	2,00		2,00	2,00			2,00				2,00		3,00		3,00	20,06	20,05
Brutto	12,00		12,00	12,00	12,00				12,00	12,00	12,00	12,00				12,00		13,00	13,00		
Ergebnis																					
Netto	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	12,00	10,00	0,00	0,00	8,00	10,00	10,00	10,00	100,00	100,00
Prozent	20	20	20	20	20	20	20	20	20	20	20	0	0	20	99	99	99	99	99	99	20
Steuer	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	0,00	0,00	0,00	2,00	4,00	3,00	3,00	3,00	20,06	20,05
Brutto	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	10,00	0,00	2,00	12,00	13,00	13,00	13,00	120,06	120,05

Hilfs-Prozeduraufruf in anderer Prozedur ohne Parameterübergabe

Der einfachste Code wäre ja

```
Sub TEST()

        HILFSPROZEDUR

End Sub

Sub HILSPROZEDUR ()
```

```
        MSGBOX "HALLO"

End Sub
```

Daher: eine Hilfsprozedur wird nur direkt mit deren Name aufgerufen.

Drei Wichtigkeiten

1) Beide Prozeduren sind im gleichen Modul, bzw im Code des gleichen Tabellenblattes.

2) Ist der Code der Hilfsprozedur in einem anderen Modul, muss er dort mit Public Sub Hilfsprozedur angelegt sein

3) Eine Prozedur, die in einem Tabellenblatt angelegt ist, kann von anderen Prozeduren nur dann verwendet werden, wenn diese auch im Code desselben Tabellenblattes sind. Prozeduren in anderen Tabellenblättern oder in Modulen können Prozeduren eines Tabellenblates NICHT verwenden !

Beispiel einer Hilfsprozedur mit Parameterübergabe

1.) PROZEDUR

```
Sub TEST(a As String, b As String)
        MsgBox a & b
End Sub

Sub test2()
        Call TEST("123", "345")
        ' oder auch
        TEST "123", "345"
End Sub
```

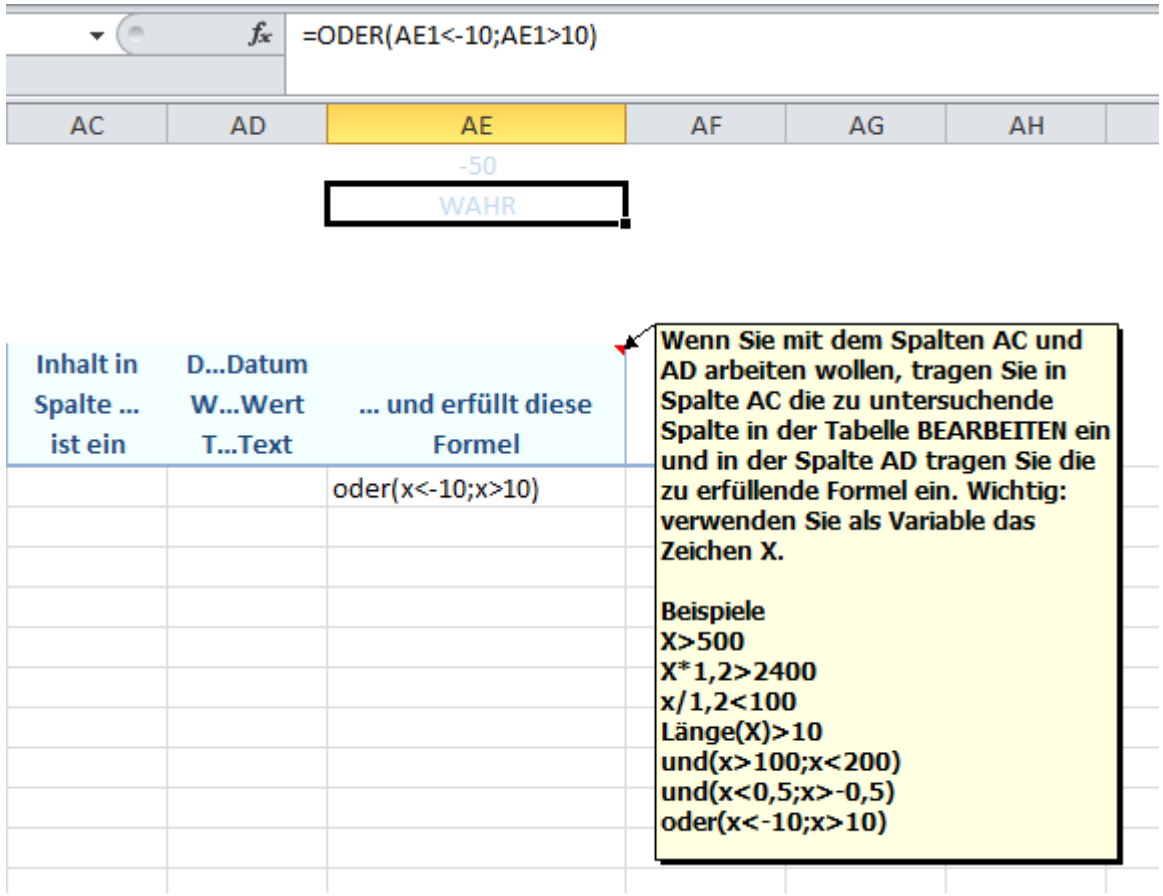
2.) FUNKTION

```
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

  Dim ExcelLastCell As Range
  Dim Row As Long
  Dim Col As Long
  Dim LastRowWithData As Long
  Dim LastColWithData As Long
  Dim TheSheet As Worksheet
  Dim MERKER
```

```
    Set TheSheet = Worksheets(TABELLENBLATT)

    MERKER = Application.ScreenUpdating
    Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

    Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

    ' letzte Zeile mit Daten herausfinden
    LastRowWithData = ExcelLastCell.Row
    Row = ExcelLastCell.Row
    Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
    Loop
    LastRowWithData = Row

    ' letzte Spalte mit Daten herausfinden
    LastColWithData = ExcelLastCell.Column
    Col = ExcelLastCell.Column
    Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
    Loop
    LastColWithData = Col

    Set LETZTEZELLE = TheSheet.Cells(Row, Col)
    Application.ScreenUpdating = MERKER

End Function
```

Funktion direkt in Zelle vom User eingeben lassen

Bisweilen gebe ich den Usern gern die Möglichkeit direkt eine Bedingung als Formel einzugeben. Dabei gibt es dann eine Zelle, in der ich den Wert / Text reinschreibe (in unserem Fall AE1), in eine zweiten Zelle schreibe ich die Formel hinein und werte sie aus (in unserem Fall AE2) und der User bekommt eine oder mehrere Zellen, wo er die Formeln hineinschreibt (in unserem Fall die Zellen AE11 und darunter):



Der Code sieht so aus:

```
Sub AUTOKONTIERUNG2()
' Dies ist der Code vom roten Button Auto2 in der Bearbeitentabelle
Dim FORMEL As String
' aktuell nur Democode für die Formelspalten AC/AD
' Range("AD1") = 100
```

```
FORMEL = Replace(UCase(Range("AE11")), "X", "AE1")
If Left(FORMEL, 1) <> "=" Then FORMEL = "=" & FORMEL

' --- Neue Version: Übergabe zuerst als Text und dann umwandeln in Formel mit F2
Range("AE2").NumberFormat = "@"
Range("AE2") = FORMEL
Range("AE2").Select
Selection.NumberFormat = "General"
SendKeys "{F2}", True
SendKeys "{ENTER}", True

End Sub
```

Prozedurenaufruf durch String

Gemeint ist, dass der Name der aufzurufenden Prozedur in einer Stringvariable ist. (Ich schreibe bei Tools mit vielen Prozeduren, gerne den Namen der zuletzt aufgerufenen Funktion in eine Zelle und kann so einfach eine Funktion dem User anbieten, die ihm die zuletzt benutzte Funktion wiederholt.

Es gibt dafür zwei Wege, abhängig davon, ob der Code in einem Modul ist (APPLICATION.RUN) oder in einem Sheet bzw einer Userform(CALLBYNAME).

CODE IN PROZEDUR

```
Private Sub test_Click()
    Dim i As String
    Dim pro As String

    i = 1
    pro = "sale_call" + i

    '~~> This will run sale_call1
    Application.Run pro

    i = 2
    pro = "sale_call" + i

    '~~> This will run sale_call2
    Application.Run pro
End Sub

Sub sale_call1()
    MsgBox "Hello"
```

```
End Sub

Sub sale_call2()
   MsgBox "goodbye"
End Sub
```

CODE IN USERFORM/SHEET

If your code is not in a module but in a Userform or Sheet Code area then `Application.Run` will not work till the time `sale_call1` or `sale_call2` is not placed in a module. If you do not wish to move them to a module then you will have to use `CallByName`. Check Excel's inbuilt help on this function. Here is an example which assumes that the code is in `Userform1`

```
Private Sub CommandButton1_Click()
    Dim i As String
    Dim pro As String

    i = 1
    pro = "sale_call" + i

    '~~> This will run sale_call1
    CallByName UserForm1, pro, VbMethod

    i = 2
    pro = "sale_call" + i

    '~~> This will run sale_call2
    CallByName UserForm1, pro, VbMethod
End Sub

Sub sale_call1()
    MsgBox "Hello"
End Sub

Sub sale_call2()
    MsgBox "goodbye"
End Sub
```

ANSATZ 1 CALLBYNAME

Vor kurzem wurde uns die Frage gestellt: "Wie lässt sich eine Prozedur durch eine Variable aufrufen?"

Beispiel:
In einer String-Variable steht der Name einer aufzurufenden Prozedur, z.B. "ShowMessage". Innerhalb des Programms existiert natürlich auch eine solche Prozedur. Der Inhalt der Variable wurde z.B. aus einer Textdatei ausgelesen. Wie aber schafft man es nun, dass die Prozedur auch aufgerufen wird?

Die Lösung hierfür bietet die **CallByName**-Funktion:

```
Dim sVariable As String

sVariable = "ShowMessage"
CallByName Me, sVariable, VbMethod
```

Erstellen Sie ein neues Projekt mit einer Form und einem CommandButton. Fügen Sie dann folgende Zeilen in den Codeteil der Form ein:

```
Option Explicit

Public Sub ShowMessage()
  MsgBox "abcd"
End Sub

Private Sub Command1_Click()
  Dim sVariable As String

  sVariable = "ShowMessage"
  CallByName Me, sVariable, VbMethod
End Sub
```

Sollte die aufzurufende Prozedur Parameter erwarten müssen diese nach **vbMethod** angegeben werden. Also erweitern wir unser kleines Beispiel...

```
Option Explicit

Public Sub ShowMessage(ByVal sText As String)
  MsgBox sText
End Sub

Private Sub Command1_Click()
  Dim sVariable As String
  Dim sText As String

  sVariable = "ShowMessage"
  sText = "Irgend ein Text"
```

```
  CallByName Me, sVariable, VbMethod, sText
End Sub
```

ANSATZ 2 APPLICATION.RUN

BSP PROZEDUR

```
Sub dummy(msgnumber as integer)
    msgbox msgnumber
End Sub

Sub Main()
    Dim SubName as String
    SubName = "dummy(1234)"

    Application.Run SubName

End Sub
```

BSP FÜR FUNKTION UND PROZEDUR

for a sub

```
Sub temp()
MsgBox 1
End Sub

Sub temp2()
Dim s$
s = "temp"
Application.Run s
End Sub
```

---
for a function

```
Function temp3$(ByVal vS$)
```

```
temp3 = UCase(vS)
End Function

Sub temp4()
Dim s$
s = "temp3"
MsgBox Application.Run("temp3", "abc")
End Sub
```

BSP2 FÜR FUNKTION UND PROZEDUR

```
Function MyFunction1()
MsgBox "Blah!"
End Function

Sub MySub1()
Dim s As String

s = "MyFunction1()"
Eval s
End Sub

Function MyFunction2(ByVal v_s As String) As String
MyFunction2 = StrReverse(v_s)
End Function
Sub MySub2()
Dim s As String
Dim p As String

p = "Blah!"
s = "MyFunction2()"
s = Replace(s, "()", "(""" & p & """)")

MsgBox Eval(s)
End Sub
```

BSP FÜR FUNKTION IN TABELLENCODE

I'm trying to call a function with a variable name that is generated at run time based upon a combo box value.

Answer: CallByName is what you'll need to accomplish the task.

Code in Sheet1

```
Option Explicit
Public Function Sum(ByVal x As Integer, ByVal y As Integer) As Long
    Sum = x + y
End Function
```

Code is Module1 (bas module)

```
Option Explicit

Sub testSum()
Dim methodToCall As String
methodToCall = "Sum"

MsgBox CallByName(Sheet1, methodToCall, VbMethod, 1, 2)
End Sub
```

Running the method testSum calls the method Sum using the name of the method given in a string variable, passing 2 parameters (1 and 2). The return value of the call to function is returned as output of CallByName

ZUSATZFRAGE: Hi.. what if I don't want the 'sum' function in Sheet1.. but say in Module1. How can I call Module1 'sum' function using CallByName. This doesn't seem to work. Please advise

ANTWORT: That is not possible. You may use Application.Run and pass the method name, parameters to it.

Jemand anderer antwortete: You should write a function that accepts the CB value as a parameter and then uses a select case to call the appropriate formatting function.

```
Function SelectFormatting(Name as String) As Boolean
Select Case CBinst.Value
Case "Text1":
  SelectFormatting = Text1FormattingFunction()
Case "Text2":
  .
  .
  .
End Select
End Function
```

Prüfen ob Modul und Prozedur existiert

```
'=============================================================================
'- CHECK IF A MODULE & SUBROUTINE EXISTS
'- VBA constant : vbext_pk_Proc = All procedures other than property procedures.
'- An error is generated if the Module or Sub() does not exist - so we trap them.
'-----------------------------------------------------------------------------
'- VB Editor : Tools/References - add reference TO ......
'-    .... "Microsoft Visual Basic For Applications Extensibility"
'-----------------------------------------------------------------------------
'- Brian Baulsom October 2007
'=============================================================================
Sub MacroExists()
    Dim MyModule As Object
    Dim MyModuleName As String
    Dim MySub As String
    Dim MyLine As Long
    '-------------------------------------------------------------------------
    '- test data
    MyModuleName = "TestModule"
    MySub = "Number2"
    '-------------------------------------------------------------------------
    On Error Resume Next
    '- MODULE
    Set MyModule = ActiveWorkbook.VBProject.vbComponents(MyModuleName).CodeModule
    If Err.Number <> 0 Then
    MsgBox ("Module : " & MyModuleName & vbCr & "does not exist.")
    Exit Sub
    End If
    '-------------------------------------------------------------------------
    '- SUBROUTINE
    '- find first line of subroutine (or error)
    MyLine = MyModule.ProcStartLine(MySub, vbext_pk_Proc)
    If Err.Number <> 0 Then
    MsgBox ("Module exists      : " & MyModuleName & vbCr _
           & "Sub " & MySub & "( )  : does not exist.")
    Else
    MsgBox ("Module : " & MyModuleName & vbCr _
        & "Subroutine   : " & MySub & vbCr _
```

```
        & "Line Number : " & MyLine)
    End If
End Sub
```

--- GRUNDLAGEN Teil 1

Die Aufruf-Syntax

Die Syntax der Aufrufe von VBA-Programmen und -Unterprogrammen mit oder ohne Übergabe von Parametern kann sehr unterschiedlich sein. Achten Sie bitte bei Ihren VBA-Programmierungen darauf, dass Sie Unterprogramme, die sich in der gleichen Arbeitsmappe wie die aufrufende Prozedur befinden, immer mit **Call** aufrufen:

```
Call Unterprogramm
```

Das vorangestellte **Call** ist optional, sollte aber im Interesse der Übersichtlichkeit des Codes dennoch verwendet werden.

Weichen Sie von dieser Regel nur dann ab, wenn Sie aus Ablaufgründen den Namen der aufzurufenden Unterprozedur variabel halten müssen. Weiter unten folgt hierfür ein Beispiel.

Befindet sich die aufzurufende Prozedur in einem Klassenmodul und der Aufruf erfolgt aus einem anderen Modul, so ist dem Aufruf die Klasse voranzustellen:

```
Call Tabelle1.Unterprogramm
```

Als **Private** deklarierte Funktionen können nicht aufgerufen werden.

Prozeduren in anderen Arbeitsmappen oder Anwendungen werden mit **Run** gestartet, wobei der Makroname zusammen mit dem Namen des Container-Dokuments als String übergeben wird:

```
Run "'Mappe1'!MeinMakro"
```

Hierbei ist zu beachten:

- Dateinamen mit Leerzeichen müssen im Run-Aufruf in Apostrophs gesetzt werden
- Die mit Run aufgerufene Arbeitsmappe wird - wenn nicht geöffnet - im aktuellen Verzeichnis (CurDir) gesucht. Nicht machbar ist:

```
Run "'c:\mappe1.xls'!Meldung"
```

Aufruf eines Makros in der aktuellen Arbeitsmappe ohne Parameterübergabe

Das aufzurufende Unterprogramm befindet sich in einem Standardmodul der aufrufenden Arbeitsmappe.

```
Sub Demo ()
      Call Test
End Sub

Sub Test()
   MsgBox "Ein normaler Aufruf!"
End Sub
```

Aufruf einer Funktion in der aktuellen Arbeitsmappe mit Parameterübergabe

```
Sub TEST(a As String, b As String)
        MsgBox a & b
End Sub

Sub test2()
        Call TEST("123", "345")
        ' oder auch
        TEST "123", "345"
End Sub
```

- Prozedur: CallFunction
- Art: Sub
- Modul: Standardmodul
- weck: Funktion mit Parameter aufrufen und Funktionsergebnis melden
- Ablaufbeschreibung:
    - Meldung eines von einer Funktion ermittelten Wertes

- Code:

```
Sub CallFunction()
   MsgBox "Anzahl der Punkte der Schaltfläche: " & vbLf & _
      CStr(GetPixel(ActiveSheet.Buttons(Application.Caller)))
End Sub
```

Aufruf eines Makros in einer anderen Arbeitsmappe ohne Parameterübergabe

- Prozedur: CallWkbA
- Art: Sub
- Modul: Standardmodul
- Zweck: Makro einer anderen Arbeitsmappe ohne Parameter aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Arbeitsmappenname an String-Variable übergeben
    - Fehlerroutine starten
    - Arbeitsmappe an Objektvariable übergeben
    - Fehlerroutine beenden
    - Wenn die Arbeitsmappe nicht geöffnet ist...
    - Negativmeldung
    - Sonst...
    - Makro in anderer Arbeitsmappe starten
- Code:

```
Sub CallWkbA()
   Dim sFile As String
   Dim wkb As Workbook
   sFile = "'vb07_test.xls'"
   On Error Resume Next
   Set wkb = Workbooks(sFile)
   On Error GoTo 0
   If wkb Is Nothing Then
      MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht gefunden!"
```

```
   Else
      Run sFile & "!Meldung"
   End If
End Sub
```

Aufruf einer Funktion in einer anderen Arbeitsmappe mit Parameterübergabe

- Prozedur: CallWkbB
- Art: Sub
- Modul: Standardmodul
- Zweck: Funktion einer anderen Arbeitsmappe mit Parameter aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Arbeitsmappenname an String-Variable übergeben
    - Fehlerroutine starten
    - Arbeitsmappe an Objektvariable übergeben
    - Fehlerroutine beenden
    - Wenn die Arbeitsmappe nicht geöffnet ist...
    - Negativmeldung
    - Sonst...
    - Funktion in anderer Arbeitsmappe aufrufen und Ergebnis melden
- Code:

```
Sub CallWkbB()
   Dim sFile As String
   Dim wkb As Workbook
   sFile = "'vb07_test.xls'"
   On Error Resume Next
   Set wkb = Workbooks(sFile)
   On Error GoTo 0
   If wkb Is Nothing Then
      MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht gefunden!"
   Else
      MsgBox Run(sFile & "!CallerName", Application.Caller)
```

```
   End If
End Sub
```

Auruf einer Prozedur in anderer Arbeitsmappe

Application.Run “AndereMappe.xls!Makroname”

Aufruf einer Prozedur in anderer Arbeitsmappe mit Übergabe von Variablen

Weg1: schreib die Werte in eine Tabelle der Zielmappe und lese sie dort vom Zielmakro aus

Weg 2 - direkt:

Code:

```
Sub ExternesMakroStarten()
    '28.05.2008, NoNet - www.excelei.de
    Dim extWB As Workbook, strExtName As String
    Set extWB = GetObject("c:\temp\extern.xls") 'Name der externen Mappe
    strExtName = extWB.Name
    Application.Run strExtName & "!externesMakro", "Hallo", "Welt" 'Makro mit Parameter-Übergabe
    extWB.Close
    Set extWB = Nothing
End Sub
```

Das Makro in der externen Mappe muss dazu in einem allgemeinen Modul (also z.B. "Modul1") stehen und darf m.E. nicht als **PRIVATE** deklariert sein ! Es könnte z.B. so aussehen :

Code:

```
Sub externesMakro(Wert1, Wert2)
    'Dieses Makro wird von extern aufgerufen
```

```
    MsgBox "Die Werte lauten :" & vbLf & Wert1 & vbLf & Wert2
End Sub
```

Aufruf eines Makros in einem Klassenmodul einer anderen Arbeitsmappe

- Prozedur: CallWkbC
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro im Klassenmodul einer anderen Arbeitsmappe aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Arbeitsmappenname an String-Variable übergeben
    - Fehlerroutine starten
    - Arbeitsmappe an Objektvariable übergeben
    - Fehlerroutine beenden
    - Wenn die Arbeitsmappe nicht geöffnet ist...
    - Negativmeldung
    - Sonst...
    - Makro in anderer Arbeitsmappe starten
- Code:

```
Sub CallWkbC()
   Dim sFile As String
   Dim wkb As Workbook
   sFile = "'vb07_test.xls'"
   On Error Resume Next
   Set wkb = Workbooks(sFile)
   On Error GoTo 0
   If wkb Is Nothing Then
```

```
      MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht gefunden!"
   Else
      Run sFile & "!Tabelle1.CallClassModule"
   End If
End Sub
```

Word-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallWord
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einem Word-Dokument aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Name des Worddokumentes an String-Variable übergeben
    - Wenn die Datei nicht existiert...
    - Negativmeldung
    - Sonst...
    - Word-Instanz bilden
    - Word-Dokument öffnen
    - Word-Makro aufrufen
    - Word-Instanz schließen
    - Objektvariable zurücksetzen
- Code:

```
Sub CallWord()
   Dim wdApp As Object
   Dim sFile As String
   sFile = ThisWorkbook.Path & "\vb07_WordTest.doc"
   If Dir$(sFile) = "" Then
      MsgBox "Test-Word-Dokument " & sFile & " wurde nicht gefunden!"
   Else
      With CreateObject("Word.Application")
```

```
         .documents.Open sFile
         .Run "Project.Modul1.WdMeldung"
         .Quit
      End With
   End If
End Sub
```

Access-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallAccess
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einer Access-Datenbank aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Name der Access-Datenbank an String-Variable übergeben
    - Wenn die Datei nicht existiert...
    - Negativmeldung
    - Sonst...
    - Acess-Instanz bilden
    - Access-Datenbank öffnen
    - Access-Makro aufrufen
    - Access-Instanz schließen
    - Objektvariable zurücksetzen
- Code:

```
Sub CallAccess()
   Dim accApp As Object
   Dim sFile As String
   ' Pfad, wenn die Access-MDB im gleichen Verzeichnis wie die XLS-Datei liegt
   sFile = ThisWorkbook.Path & "\vb07_AccessTest.mdb"
   If Dir(sFile) = "" Then
      Beep
```

```
      MsgBox "Access-Datenbank wurde nicht gefunden!"
   Else
      With CreateObject("Access.Application")
          .OpenCurrentDatabase sFile
          .Run "AcMeldung"
          .CloseCurrentDatabase
      End With
   End If
End Sub
```

Aufruf von Prozeduren in der aktuellen Arbeitsmappe mit variablen Makronamen

- Prozedur: CallMacros
- Art: Sub
- Modul: Standardmodul
- Zweck: Makros mit variablen Makronamen aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Das letzte 6 Zeichen des Namens der aufrufenden Schaltfläche an eine String-Variable übergeben
    - Meldung, dass jetzt zu dem Makro mit dem in der String-Variablen hinterlegten Namen verzweigt wird
    - Makro mit dem in der String-Variablen hinterlegten Namen aufrufen
- Code:

```
Sub CallMacros()
   Dim sMacro As String
   sMacro = Right(Application.Caller, 6)
   MsgBox "Ich verzweige jetzt zu " & sMacro
   Run sMacro
End Sub
```

---→ Grundlagen PROZEDUREN 2

There are three types of procedures:

1. **Sub** - Standard sub routine
2. **Function** - a routine that returns an answer
3. **Property** - reserved for *Class Modules*

The third item is not discussed in this topic as it is deemed advanced VBA.

Sub Procedure

This is the most commonly used procedure that a recorded and edited macro typically uses.

It executes code line by line in order, carrying out a series of actions and/or calculations.

The signature for this type of procedure is:

```
Sub NameOfProcedure([Arguments])
      1st line of executed code 'Comments
      2nd line of executed code 'Comments
      ..........
End Sub
```

The '*Arguments*' element is optional which can be **explicit** or **implicit**. This allows values and /or references to be passed into the calling procedure and handled as a variable.

When recording a macro, no arguments are used and the parenthesis for the named procedure remains empty.

If you create a procedure intended as a macro in Excel, users must not specify any arguments.

Sub procedures can be recursive meaning that branching to another procedure is permitted which then returns back to the main calling procedure.

Calling another procedure can include the **Call** statement followed by the name of the procedure with optional arguments. If arguments are used, users must use parenthesis around the argument list.

Example of the CALL statement

```
'Procedure to be called with a single argument explicitly 'declared as a string
Sub MyMessage(strText As String)

      MsgBox strText

End Sub
```

(Click here for an understanding of the **MsgBox** statement)

Correct

```
'Test the calling procedure
Sub TestMessage()

      Call MyMessage("It worked!")

End Sub
```

Incorrect - must use the parenthesis

```
'Test the calling procedure
Sub TestMessage()

      Call MyMessage "Did it work?"

End Sub
```

Correct (alternative) - No Call keyword used & no parenthesis therefore required.

```
'Test the calling procedure
Sub TestMessage()

      MyMessage "It worked!"

End Sub
```

A procedure can be prematurely terminated, placed before the '***End Sub***' statement by using the '***Exit Sub***' statement.

```
'This procedure will terminate after part A and never run part B.
Sub TerminateNow()

      Code part A here...

     Exit Sub

      Code part B here....

End Sub
```

Function Procedure

The main difference between a **Sub** and **Function** procedure is that a **Function** procedure carries out a procedure and will return an answer whereas a **Sub** procedure carries out the procedure without an answer.

A simple analogy of a **Function** procedure compared to that of a **Sub** procedure could be illustrated using two example features of Excel:

- *File*, *Save* is an action and does not return the answer – *Sub Procedure*.

- The *Sum* function calculates the range(s) and returns the answer – *Function Procedure*.

The signature for this type of procedure is:

```
Function NameOfProdedure([Arguments]) [As Type]
```

```
    Code is executed here

    NameOfProcedure = Answer of the above code executed

End Function
```

The **Arguments** element is optional which can be **explicit** or **implicit**. This allows values and /or references to be passed into the calling procedure and handled as a variable.

The optional **Type** attribute can be used to make the function explicit. Without a type declared, the function is implicit (*As Variant*).

The last line before the **End Function** signature uses the name of the procedure to return the expression (*or answer*) of the function.

Users cannot define a function inside another function, sub or even property procedures.

This type of procedure can be called in a module by a **Sub** procedure or executed as a user defined function on a worksheet in Excel.

A procedure can be prematurely terminated, placed before the ***End Function*** statement by using the **Exit Function** statement. This acts and responds in the same way as described in the previous section (*Sub Procedures*).

An example of a Function procedure:

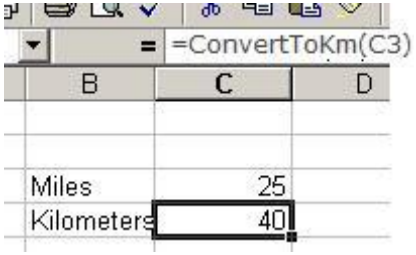
```
'This function calculates the distance of miles into kilometres.
Function ConvertToKm(dblMiles As Double) As Double

     ConvertToKm = dblMiles * 1.6

End Function
```

A **Sub** procedure that uses of the above function:

```
'Using the above function that must use parenthesis.
Sub CarDistance
```

```
    MsgBox ConvertToKm(25)

End Sub
```

In Excel, this function can also be used (known as a **User Defined Function - UDF**)



USER-DEFINED FUNCTIONS

A user defined function can be used when the built in Excel VBA functions do not meet the user's requirements. The user defined function can then be used in formulas in the same way as a built in Excel function is utilised. User defined functions are limited to doing just calculations that result in a single return of a value.

The syntax of a user-defined function is as follows:

```
Function NameofFunction([Optional] Argument1 [As Type], _
                        [Optional] Argument2 [As Type], _
                        ... [Optional] ArgumentN [As Type] ) [As Type]
   Statements here ...

   ....
   NameofFunction = Value being returned
```

```
End Function
```

NameofFunction	The name of the function.
Arguments	The arguments of the function. If an argument is to be optional, enter the word `Optional` before the name of the argument. The `As Type` option allows you to specify the data type for the return value.
Statements	The various lines of code.
NameofFunction=Value	Name is the name used in the first line of the function. Expression is the return value of the function.

Note: The square brackets wrapped around a keyword in the above syntax denotes as optional and can be left out altogether.

Creating a User Defined Function

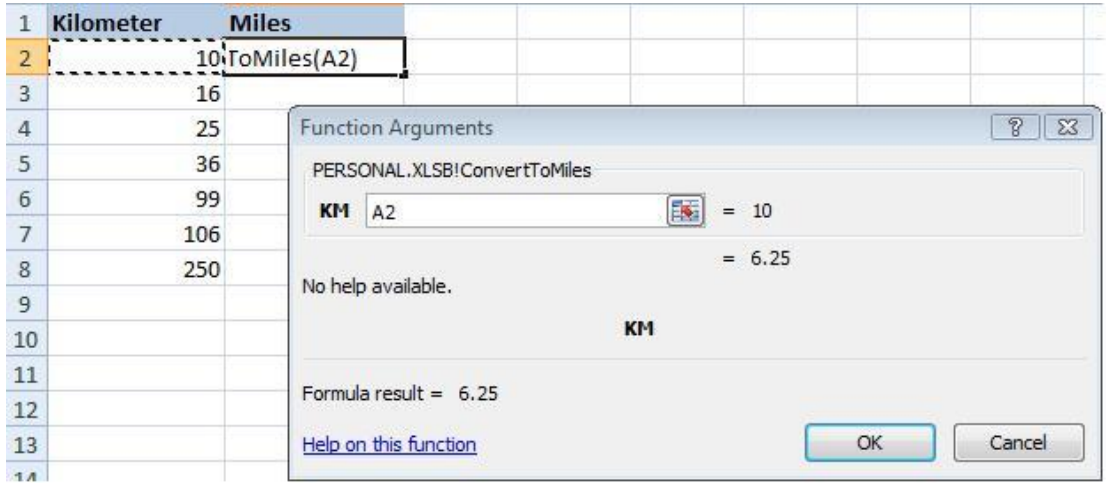
The following is a simple function example to convert *Kilometers* recorded into *Miles*.

1. From the **Developer** tab on the Ribbon Bar, click the **Visual Basic** icon.
2. Click on the **Insert** menu and select **Module**.
3. Enter the following code:

```
Function ConvertToMiles(KM)
      ConvertToMiles=KM / 1.6
End Function
```

4. Back in the Excel spreadsheet, click on the **Formula** tab on the Ribbon Bar and click the **Insert Function** icon.
5. From the list of **Categories**, select **User Defined**.
6. Select **ConvertToMiles** and click on **OK**.

7. Enter the cell reference of the Kilometer value you wish to convert into miles, into the **KM** field and click on **OK**.



Using built-in functions

It is possible to use built in Excel functions within a user defined function.

The syntax used for built in Excel Functions is as follows:

```
Application.NameofFunction(Arguments Required)
```

An example which incorporates the Excel **Round** function to the above user defined function (*ConvertToMiles*).

```
Function ConverToMiles(KM)
    ConvertToMiles = Applications.Round(KM / 1.6, 0)
```

```
End Function
```

The above amended code rounds the resulting returning value to zero decimal places using the standard Excel built-in **Round** function.

Using The Optional Argument

The **Optional** keyword preceding the argument name flags the argument as an optional parameter to the function call.

A lot of built-in Excel functions have optional arguments which always follow on from the *mandatory* arguments listed in a function and can therefore be omitted defaulting to a value the function procedure knows how to handle if left out.

This makes function more flexible and can give different returning values (answers) and/or change the behaviour of how the function will run. Think of the **VLOOKUP** function in Excel, see it's syntax below:

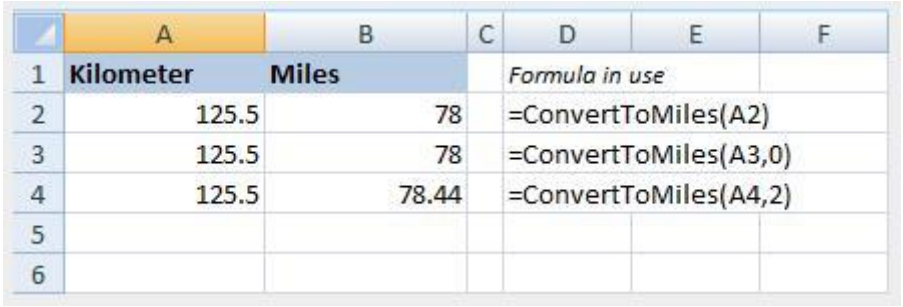
= VLookUp (Value , Range , Offset Column [, Type])

The last argument (wrapped in square brackets) is optional and always appears after all mandatory arguments (3 in this example) which can be omitted and still work. The optional argument is a value of either **True** or **False** which defaults to **True** if omitted and simply changes the way how this function will calculate.

An example - following on from the above code snippet above, I want a second argument (as optional) which allows the user to choose a positive whole number (**Byte** data type) as its value to represent the number of decimal places to pass into the calculation. If omitted, it defaults to 0 decimal places round to the nearest whole number:

```
Function ConvertToMiles(KM, Optional DecPlaces As Byte)
    If DecPlaces < 0 Then
        ConvertToMiles = KM / 1.6
    Else
        ConvertToMiles = Application.Round(KM / 1.6, DecPlaces)
```

```
    End If
End Function
```



	A	B	C	D	E	F
1	**Kilometer**	**Miles**		*Formula in use*		
2	125.5	78		=ConvertToMiles(A2)		
3	125.5	78		=ConvertToMiles(A3,0)		
4	125.5	78.44		=ConvertToMiles(A4,2)		
5						
6						

The user can now either omit the second argument (cell B2), add a value of 0 to represent no decimal places (cell B3) or add a positive number to pass into the Excel **Round** function (cell B4).

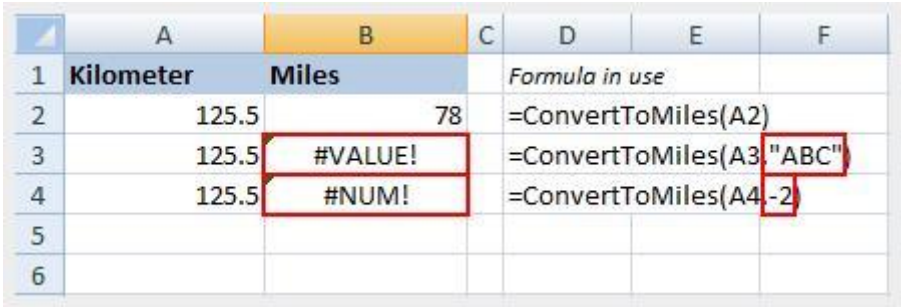
Using the As Type option

Optionally, the **`As Type`** keywords can be included to define a certain data type the argument and/or the function is controlled.

If omitted it will default to **`Variant`** (any data type it inherits) and can be open to abuse and more importantly errors.

You define a data type (see Variables & constants for more information) for each argument in the function and for the function's returning value too. If left out, you will need to add more code to handle different data input scenarios.

Let's take a look at what happens if the last above example function is abused.



	A	B	C	D	E	F
1	**Kilometer**	**Miles**		*Formula in use*		
2	125.5	78		=ConvertToMiles(A2)		
3	125.5	#VALUE!		=ConvertToMiles(A3,"ABC")		
4	125.5	#NUM!		=ConvertToMiles(A4,-2)		
5						
6						

In cell B3, setting the optional second argument to a **`String`** value "ABC" causes the **#VALUE!** error (a non numeric data input).

In cell B4, setting the optional second argument to a negative number causes another error **#NUM!** even though it's a number but the argument data type **`Byte`** only accepts positive numbers between 0 and 255 as its range.

The whole function is also expected to return a number which can be a larger than 255 and we therefore could apply the **`Integer`** as it's returning data type.

```
Function ConvertToMiles(KM, Optional DecPlaces As Byte) As Integer
    If DecPlaces < 0 Then
        ConvertToMiles = KM / 1.6
    Else
        ConvertToMiles = Application.Round(KM / 1.6, DecPlaces)
    End If
End Function
```

Notice I have left out the argument ***KM*** data type which defaults to **`Variant`**. Personally, I prefer to test for a data type in the code itself when the user or system passes a value to calculate.

An example:

```
Function ConvertToMiles(KM, Optional DecPlaces As Byte) As Integer
    If IsNumeric(KM) Then
```

```
        If DecPlaces < 0 Then
            ConvertToMiles = KM / 1.6
        Else
            ConvertToMiles = Application.Round(KM / 1.6, DecPlaces)
        End If
    Else
        ConvertToMiles = 0 'If it fails return a 0
    End If
End Function
```

I have tested to see if ***KM*** argument is a number by using the **`IsNumeric`** VB function.

All user defined functions can be called in Excel (as explained above) or into a calling Sub procedure like a VB or Excel function.

Benutzerdefinierte Werte an Prozedur übergeben

BSP1)

```
Sub TEST(a As String, b As String)
        MsgBox a & b
End Sub

Sub test2()
        Call TEST("123", "345")
        ' oder auch
        TEST "123", "345"
End Sub
```

BSP2.) Klassischer Fall ist meine SORTIERAUFSTEIGEND-FUNKTION

Aufpassen auf Public oder Private - nur Public - Prozeduren stehen in allen Modulen zur Verfügung

```
Public Sub SORTIERE_AUFSTEIGEND(SORTIERSPALTE As String, STARTZEILE As Integer, ENDZEILE As Integer)

' Aufruf über SORTIERE_AUFSTEIGEND "D",5,10  - oder über Variablen  SORTIERE_AUFSTEIGEND SPALTE, STARTZEILE, ENDZEILE

' 1.) Den zu sortierenden Bereich markieren
```

```
        Range("A" & STARTZEILE & ":" & "IV" & ENDZEILE).Select
        SORTIERSPALTE = SORTIERSPALTE & STARTZEILE ' Der VBA-Befehl SELECTION.SORT braucht die oberste Zelle der zu sortierenden Spalte

' 2.) die eigentliche Sortierung

    Selection.Sort Key1:=Range(SORTIERSPALTE), Order1:=xlAscending, Header:=xlGuess, OrderCustom:=1, _
                MatchCase:=False, Orientation:=xlTopToBottom, DataOption1:=xlSortTextAsNumbers

' 3.) Aufhebung der Markierung
    Range(SORTIERSPALTE).Select

End Sub
```

Aufgerufen werden solche Prozeduren NICHT wie Funktionen über Prozedur (Var1, Var2...) sondern über

Prozedurenname, Variable1, Variable2 ...

Daher obige Prozedur wird aufgerufen über

SORTIERE_AUFSTEIGEND "D",5,10

oder man verwendet Variablen:

SORTIERE_AUFSTEIGEND SPALTE, STARTZEILE, ENDZEILE

BSP.3)

```
Private Sub TestProzedur(Wert_A As String, Wert_B As String)

  MsgBox Wert_A & " " & Wert_B

End Sub

Public Sub Aufrufer()

  Dim a As String
  Dim b As String

  TestProzedur "Hallo", "Welt"

' Oder...

  a = "Hallo"
  b = "Welt"
```

```
 TestProzedur a, b

End Sub
```

Parameter - aber Achtung

Die Frage, warum der folgende Code einen Fehler bringt und wie man dies vermeiden kann

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 10
        MsgBox Quadrat(i)
End Sub

Private Function Quadrat(DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Wenn man die Sub TuEs startet, kommt eine Fehlermeldung:
Argumenttyp ByRef unverträglich.

Was bedeutet dies?

Wenn ich an eine Sub oder Function einen Parameter(Argument) übergebe, muss dies auch in der entsprechenden Routine(Sub,Function) deklariert sein:
In unserem Beispiel haben wir die Function Quadrat die das Argument DerWert haben will.
DerWert haben wir aber gesagt, dass dieser vom Typ Long sein soll. Wir übergeben aber einen Integer.
Dies ist aber tatsächlich nicht der wirkliche Grund zum Husten für VBA (bzw. nur bedingt)

Das Problem ist, dass ich ein Argument ByRef oder ByVal übergeben kann.
Dies bedeutet:
ByRef meint, dass wenn ich das Argument innerhalb meiner Routine verändere diesen Wert an die Variable zurückgebe.

ByVal meint, dass ich den Wert nur an die Funktion übergebe, aber keine Rückgabe einleite.
Schreibe ich nichts vor die Deklaration, ist dieses per Default ByRef.

Eine kleine Änderung würde VBA nicht mehr husten lassen:

```
Private Function Quadrat(ByVal DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Der Grund:

Jetzt wird das Argument nur mit seinem Wert übergeben. Und ein Integer passt ja wunderbar in einen Long.
Da kein Wert an die Aufrufende Variable zurückgegeben wird, kann auch nichts passieren.
Selbst wenn ich die Variable in der Routine verändere, also aus einem Integer ein Long würde.

Auch hier ein Beispiel:

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 1000
        MsgBox Quadrat(i)
        MsgBox i
End Sub

Private Function Quadrat(ByVal DerWert As Long) As Long
        DerWert = DerWert ^ 2
        Quadrat = DerWert
End Function
```

Und hier das Gegenbeispiel mit ByRef und gleicher Deklaration:

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Long
        i = 1000
        MsgBox Quadrat(i)
        MsgBox i
End Sub

Private Function Quadrat(ByRef DerWert As Long) As Long
```

```
        DerWert = DerWert ^ 2
        Quadrat = DerWert
End Function
```

Aber jetzt:

Kommen wir zurück auf unseren Ursprung:
Eine ganz kleine Änderung in unserem Ursprungscode macht es wieder möglich:

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 10
        MsgBox Quadrat((i))
End Sub

Private Function Quadrat(DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Durch die Klammerung der Variablen übergebe ich diese wieder ByVal

Benutzerdefinierte Funktionen FÜR VBA

BSP1.)

Wir möchten eine Funktion "BMD_DATUM" erstellen, die ein an sie übergebenes Datum umwandelt in das BMD-Format

Sie soll nachfolgenden Code korrekt ausführen

```
Sub test()
       MsgBox BMD_DATUM("12122009")
End Sub
```

Uns so muss die Funktion aussehen

```
Function BMD_DATUM(DATUM As String) As String

   BMD_DATUM = Right(DATUM, 4) & Mid(DATUM, 3, 2) & Left(DATUM, 2)

End Function
```

Wichtig: man kann eine Funktion nicht alleinstehen aufrufen - also es kann keine Programmzeile geben

BMD_DATUM ("1234567")

Es muss immer jemand das Ergebnis der Funktion erhalten : Z=BMD_DATUM ("1234567").

Möchte man kein Ergebnis erhalten, sondern nur eine Sub-Routine haben, an die man Variablen übergibt und die soll selbständig arbeiten, ohne dass man von ihr einen Wert zurückerhält, dann kann man entweder die Variablen global stellen - oder sie einfach nur an die Prozedur selbst übergeben. Siehe auch hier im Bereich VARIABLEN - ÜBERGABE VON VARIABLEN AN EINE PROZEDUR.

BSP2.)

Die Frage, warum der folgende Code einen Fehler bringt und wie man dies vermeiden kann

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 10
        MsgBox Quadrat(i)
End Sub

Private Function Quadrat(DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Wenn man die Sub TuEs startet, kommt eine Fehlermeldung:
Argumenttyp ByRef unverträglich.

Was bedeutet dies?
Wenn ich an eine Sub oder Function einen Parameter(Argument) übergebe, muss dies auch in der entsprechenden Routine(Sub,Function) deklariert sein:
In unserem Beispiel haben wir die Function Quadrat die das Argument DerWert haben will.
DerWert haben wir aber gesagt, dass dieser vom Typ Long sein soll. Wir übergeben aber einen Integer.
Dies ist aber tatsächlich nicht der wirkliche Grund zum Husten für VBA (bzw. nur bedingt)

Das Problem ist, dass ich ein Argument ByRef oder ByVal übergeben kann.

Dies bedeutet:
ByRef meint, dass wenn ich das Argument innerhalb meiner Routine verändere diesen Wert an die Variable zurückgebe.
ByVal meint, dass ich den Wert nur an die Funktion übergebe, aber keine Rückgabe einleite.
Schreibe ich nichts vor die Deklaration, ist dieses per Default ByRef.

Eine kleine Änderung würde VBA nicht mehr husten lassen:

```
Private Function Quadrat(ByVal DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Der Grund:
Jetzt wird das Argument nur mit seinem Wert übergeben. Und ein Integer passt ja wunderbar in einen Long.
Da kein Wert an die Aufrufende Variable zurückgegeben wird, kann auch nichts passieren.
Selbst wenn ich die Variable in der Routine verändere, also aus einem Integer ein Long würde.

Auch hier ein Beispiel:

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 1000
        MsgBox Quadrat(i)
        MsgBox i
End Sub

Private Function Quadrat(ByVal DerWert As Long) As Long
        DerWert = DerWert ^ 2
        Quadrat = DerWert
End Function
```

Und hier das Gegenbeispiel mit ByRef und gleicher Deklaration:

```
Option Explicit

Public Sub TueEs()
```

```
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Long
        i = 1000
        MsgBox Quadrat(i)
        MsgBox i
End Sub

Private Function Quadrat(ByRef DerWert As Long) As Long
        DerWert = DerWert ^ 2
        Quadrat = DerWert
End Function
```

Aber jetzt:

Kommen wir zurück auf unseren Ursprung:
Eine ganz kleine Änderung in unserem Ursprungscode macht es wieder möglich:

```
Option Explicit

Public Sub TueEs()
        'Die Prozedur (Das Makro) welches ausgeführt wird
        Dim i As Integer
        i = 10
        MsgBox Quadrat((i))
End Sub

Private Function Quadrat(DerWert As Long) As Long
        Quadrat = DerWert ^ 2
End Function
```

Durch die Klammerung der Variablen übergebe ich diese wieder ByVal 

Benutzerdefinierte Werte an Prozedur in anderer Arbeitsmappe übergeben

Lösung 1
Man kann auch Variablen übergeben an Prozedur in einer anderen Arbeitsmappe:

Dokument1:

Code:

```
Public Sub test()
DasIstEinTest = Application.Run("Dokument2.xls'!testSub", DasIstEinTest)
End Sub
```

Dokument2:

Code:

```
Public Function testSub(ByRef blabla As Integer)
testSub = blabla +5
End Function
```

Lösung 2:
wie rufe ich eine Prozedur mit Übergabeparameter in einer anderen Arbeitsmappe auf?

```
Die Prozedur muss sich in der *aufzurufenden* Mappe in einem Modul
befinden. In der *aufrufenden* Mappe benutzt Du dann die Run-Methode
des Application-Objects.

Peter


'/run1.xls - ruft auf/
'-----------------------------------------------
Sub test()
Application.Run "run2.xls!test_sub", 3, 2, "test"
MsgBox Application.Run("run2.xls!test_func", 3, 2)
End Sub


'/run2.xls - Modul1 - wird aufgerufen/
'-----------------------------------------------
Sub test_sub(zeile As Integer, spalte As Integer, text As String)
Cells(zeile, spalte) = text
End Sub

Function test_func(zeile As Integer, spalte As Integer)
```

```
test_func = Cells(zeile, spalte)
End Function
```
Reply

stefan onken mentioned hallo Reinhard,
Application.Run "mappe2.xls!makroname", variable

falls mappe2 nicht ge=F6ffnet ist, muss evtl noch der Pfad vor den
Dateinamen.
Makro muss so in einem Standardmodul sein, es ist aber auch m=F6glich,
ein Makro aus zB einem
Tabellenmodul aufzurufen.
Application.Run "mappe2.xls!tabelle1.makroname", variable

Gru=DF
stefan
Reply

Alor fan mentioned hallo Reinhard,

versuch's mal so:

```
Workbooks.Open Filename:=3DstrEigenerDateiPfad & "\" & strDateiName,
IgnoreReadOnlyRecommended:=3DTrue
varResult =3D Application.Run(Macro:=3D"'" & strDateiName & "'!
Fernsteuerung_Starten", arg1:=3DstrStopString,
arg2:=3DCStr(vareinreferat))
```

strEigenerDateiPfad & "\" & strDateiName ergibt zusammen den voll
qualifizierten Dateinamen des Workbooks (=3D der Arbeitsmappe), in der
Du die Prozedur aufrufen willst.
Der Name der Prozedur steht nach dem Dateinamen hinter dem "!", bei
mir "Fernsteuerung_Starten".
Die einzelnen Parameter, die Du der Prozedur =FCbergeben willst, h=E4ngtst
Du mit arg1:=3D..., arg2:=3D ... an. Ob man das Workbook vorab wirklich
=F6ffnen mu=DF kann ich Dir nicht sagen, in jedem Fall funktioniert es bei
mir so ohne Probleme.

HTH,

Andy

Benutzerdefinierte Funktionen FÜR DIE TABELLENBLÄTTER ERSTELLEN

Sie können sich Funktionen die Sie benötigen selbst erstellen. Sie stehen Ihnen dann genau so zur Verfügung wie Funktionen die mit Excel geliefert werden.

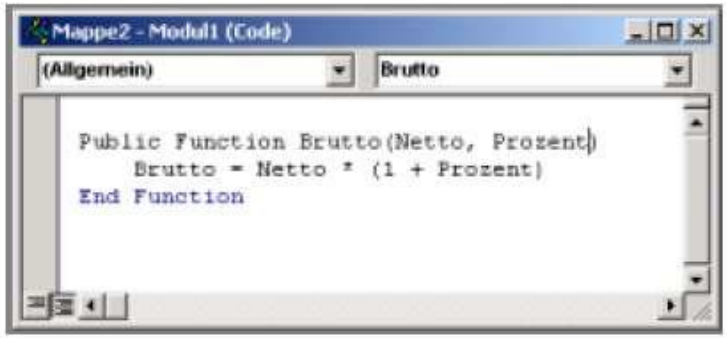
Diese werden im Editor in ein Modul geschrieben und stehen dann, wie in Excel, als eigene Funktionen im Funktionsassistent in der Kategorie **Benutzerdefiniert** zur Verfügung.

Rufen Sie in Excel den VB-Editor auf.

Im Editor wählen Sie Menü: ***Einfügen / Prozedur.***

Im Fenster ***Prozedur hinzufügen*** geben Sie den Namen der gewünschten Funktion ein und wählen als Typ ***Function***.

Im erscheinenden Modulfenster geben Sie dann den Code ein.



In den Klammern nach dem Modulnamen schreiben Sie die Namen der Argumente, die dann vom Funktionsassistenten zur Eingabe angezeigt werden.

In der Formel wird vor dem "=" Zeichen der Name der Funktion angegeben, in diesem Fall **Brutto.**

Um eine benutzerdefinierte Funktion zu verwenden, rufen Sie im Funktionsassistenten die Kategorie **Benutzerdefiniert** auf, dann werden Ihnen in der rechten Spalte alle Funktionen angezeigt, die Sie selbst erstellt haben.

Speicherort

Fortgeschrittene Anwender können entscheiden, wo die Funktion gespeichert wird, in der aktuellen oder der "persönlichen Arbeitsmappe".

Eigene Excel-Funktionen direkt in Excelzellen verfügbar machen

```
Function IsFormula(Check_Cell As Range)

IsFormula = Check_Cell.HasFormula

End Function
```

To use this UDF push Alt+F11 and go Insert>Module and paste in the code. Push Alt+Q and save. The Function will appear under "User Defined" in the Paste Function dialog box (Shift+F3). Use the Function in any cell as shown below.

=IsFormula(A1)
This will return TRUE if A1 houses a formula. It is very useful when used with Conditional Formatting as you can dynamically color all formulae cells.

Excel-Tabelle-Funktionen in VBA nutzen

Möchte man die Excelfunktionen, die in der eigentlichen Anwendung in den Zellen der Tabellenblätter zur Verfügung stehen, verwenden, so geht dies mit dem WorksheetFunction-Befehl oder mit dem Application-Befehl:

Es gibt in der Zelle die Funktion Tanh (Tangens Hyperbolicus) – aber nicht in VBA.

So geht's in VBA:

```
X = Application.WorksheetFunction.Tanh(1)
```

BSP2:

MsgBox Application.Sum(Range("C7:D7")) zeigt die Summe an

Man könnte auch WorksheetFunction.Sum verwenden - aber die Anweisung WorksheetFunction führt zu Programmabbruch, wenn das Ergebnis ein Fehlerwert ist, während die neue Application-Funktion den Fehlerwert übergibt und das Programm nicht beendet.

Bsp WorksheetFunction - hier Summenbildung des Bereiches C10 bis C9999

```
Dim Kostsumme as Integer
Kostsumme = Application.WorksheetFunction.Sum(Range("C10:C9999"))
```

In Visual Basic werden die Microsoft Excel-Tabellenfunktionen durch das **WorksheetFunction**-Objekt zur Verfügung gestellt. Benutzen Sie hier den englischen Namen der Tabellenfunktion und Ihre Argumente.
Die folgende **Sub**-Prozedur verwendet die Tabellenfunktion **Min**, um den niedrigsten Wert in einem Zellbereich zu bestimmen. Zunächst wird die Variable myRange als ein **Range**-Objekt deklariert und dann auf den Bereich A1:C10 in Tabelle1 festgelegt. Anschließend wird der Variablen answer das Ergebnis der auf myRange angewendeten Funktion **Min** zugewiesen. Der Wert von answer wird dann in einem Meldungsfeld angezeigt.

```
Sub UseFunction()
    Dim myRange As Range
    Set myRange = Worksheets("Sheet1").Range("A1:C10")
    answer = Application.WorksheetFunction.Min(myRange)
    MsgBox answer
End Sub
```

Verwenden Sie eine Tabellenfunktion, die einen Bereichsbezug als Argument fordert, so müssen Sie ein **Range**-Objekt festlegen. Sie können z. B. die Tabellenfunktion **VERGLEICH** (**Match** in VBA) verwenden, um einen Zellbereich zu durchsuchen.

In eine Zelle eines Tabellenblatts würden Sie dazu eine Formel wie =VERGLEICH(9,A1:A10,0) eingeben.

In einer Prozedur in Visual Basic jedoch legen Sie ein **Range**-Objekt fest, um das gleiche Ergebnis zu erhalten.

```
Sub FindFirst()
    myVar = Application.WorksheetFunction _
        .Match(9, Worksheets(1).Range("A1:A10"), 0)
    MsgBox myVar
End Sub
```

Anmerkung Der Kennzeichner **WorksheetFunction** wird von den Funktionen in Visual Basic nicht verwendet. Namen von Funktionen können in Visual Basic und Microsoft Excel zwar identisch sein, jedoch unterschiedliche Auswirkungen haben. Zum Beispiel geben Application.WorksheetFunction.Log und Log unterschiedliche Werte zurück.

Einfügen einer Tabellenfunktion (Formel) in eine Zelle

Um eine Tabellenfunktion in eine Zelle einzufügen, geben Sie die Funktion als Wert der **Formula**-Eigenschaft des entsprechenden **Range**-Objekts an. Im folgenden Beispiel wird die Tabellenfunktion **ZUFALLSZAHL** (**RAND** in VBA) der **Formula**-Eigenschaft des Bereichs A1:B3 in Tabelle1 der aktiven Arbeitsmappe zugewiesen.

```
Sub InsertFormula()
    Worksheets("Sheet1").Range("A1:B3").Formula = "=RAND()"
End Sub
```

WICHTIG: nicht mit den deutschen Funktionen arbeiten, weil die erst bei Neuübernahme mit F2-Enter

greifen - sondern mit englischen

daher: Cells(1,1).formula="=sum(B1:A2)" und nicht .formula = "=Summe(...)

Wie findet man die englischen Befehle heraus: einfach eine funktionierende Zelle mit Makrorecorder erneut mit F2 du dann ENTER übernehmen

FALSCH muss als FALSE übergeben werden !

="123" als =""123""

und auch "" in Formeln wie =Wenn(A1=1,1,"") muss mit """" übergeben werden.

WICHTIG 2: viele Funktionen in Deutsch verwenden den STRICHPUNKT - z.B. "=wenn(A1=1;1;0)"

im Englischen ist aber der Standard der Beistrich - daher bitte "=if(A1=1,1,0")

WICHTIG 3: der Makrorecorder in Excel 2007 erzeugt bei einer normalen Zellverknüpfung den falschen Eintrag

Konkret: Range("A1").FormulaR1C1 = "=Sprachen!E22" (Sprachen' heißt die Tabelle mit dem Wert in diesem Beispiel)

Sobald man diesen Code ausführt wird in A1 folgende Formel eingefügt, die einen Fehler erzeugt =Sprachen!'E22'.

Lösung: entferne einfach das R1C1 und mache richtigerweise draus: Range("A1").FormulaR1C1 = "=Sprachen!E22"
Damit funktioniert es

Fehler-Prüfung (ob Zelle einen Fehler enthält)

```
If IsError(Cells(QZ,1)) = True Then
```

Hyperlinks verhindern

Gibt man z.B. Emailadressen oder www.adressen ein, erzeugt Excel (Word …) automatisch anwählbare Hyperlinks. Wen das stört, der kann solche Adressen z.B. mit beginnendem ' eingeben – also z.B. 'www.test.at

In den Exceloptionen / Dokumentenprüfung / Autokorrektur-Optionen / Autoformat während Eingabe gibt es den Eintrag Internet und Netzwerkpfade durch Hyperlinks – dort das Häkchen raus geben.

Wenn nach der Eingabe von z.B. einer Email-Adresse ein Hyperlink entstanden ist, kann man diesen ganz einfach so entfernen

```
Range("F40").Select
Selection.Hyperlinks.Delete
```

Code BSP 1:

Hyperlinks are a property of a `Range` or a `sheet`, not of an individual cell.

So how about this

```
//..your original code for getting the Range Rng

//..then use this For Each loop:
Dim HL as Hyperlink
For Each HL In Rng.Hyperlinks
HL.Delete 'This will effectively replace the cells contents with the URL
Next
```

Code BSP 2:

```
Sub demo()
    Dim hl As Hyperlink
    Dim rng As Range
    Dim cl As Range
    Dim txt As String
    Dim ws As Worksheet

    Set ws = ThisWorkbook.Worksheets("Updated_UnMatched")

    For Each hl In ws.Hyperlinks
        Set cl = hl.Parent
        txt = hl.Address & hl.SubAddress
        hl.Delete
        cl.Value = txt
    Next
End Sub
```

Prozeduren generieren

Wie man einen Button generiert und mit VBA-Code hinterlegt => VBA-CODE GENERIEREN

FUNKTIONEN DATUM UND ZEIT

1904-Problem

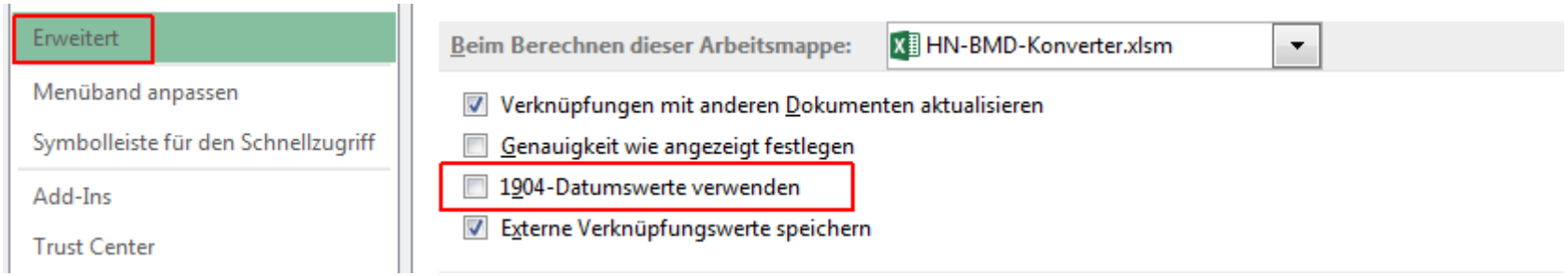
Das 1904-Datumsformat ist für Macintosh/Apple/IOS erfunden worden, weil das mit einem Datum vor 1900 nicht klar kommt und in Windows / Excel ist der erste Tag in der Zeitrechnung der 31.12.1899.

In Windows sollte man immer OHNE der 1904-Datum-Option arbeiten. Falls man doch mit 1904-Option arbeitet - GANZ WICHTIG: Zellen mit Datum nur mit VALUE übertragen werden und nicht ohne, weil sonst Excel die 4 Jahre Differenz übergibt. Also nicht: cells(1,1)=cells(2,1), sondern Cells(1,1).Value=cells(2,1).Value

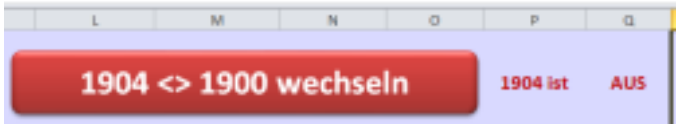
Ob eine Zelle ein Datum enhält liest man übrigens so aus:
MsgBox IsDate(ActiveCell.Value)
oder
MsgBox ActiveCell.NumberFormat

1.Lösung

Oft reicht es in der betreffenden Vorlage, die die Daten bekommt (meist bei mir über die Zwischenablage) in der ganzen Arbeitsmappe hier die 1904-Logik einmal abzudrehen, indem man hier das Häkchen rausnimmt



Bekommt jemand abwechselnd Daten mit 1904 und dann mit 1900-Logik, baut man ihm einen Wechselschalter:



```
Sub Wechsel_1904()

' Dies ist der Code für einen Wechselschalter, der die 1904-Logik auf und abdreht

    ActiveWorkbook.Date1904 = Not ActiveWorkbook.Date1904
    If ActiveWorkbook.Date1904 = False Then
       Range("Q1") = "AUS"
     Else
       Range("Q1") = "AN"
    End If


End Sub
```

2.Lösung

Es entsteht in der Regel bei einem als Text formartiertem Datum, wenn Excel es als Datum interpretieren soll – z.B. ein 02.05.69-Text wird dann schon mal ein um 4 Jahre und einen Tag verschobenen Tag. Was helfen kann ist die CDATE-Funktion in VBA, wenn man mit ihr ein Textdatum umwandelt in ein echtes Datum.

So wars verkehrt:

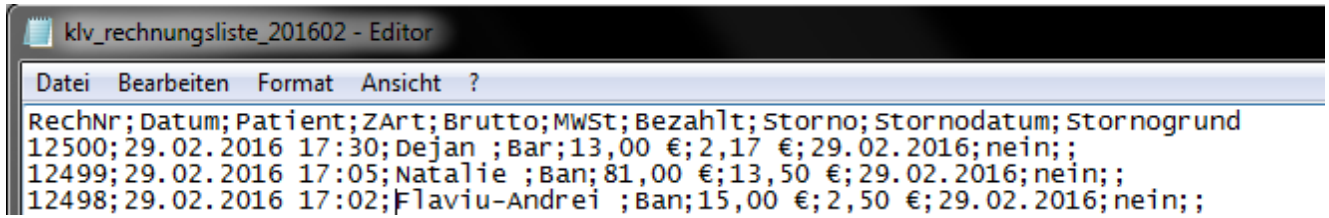
```
' Belegdatum
Tabelle1.Cells(ZZ, 3) = Cells(QZ, 2) ' in Cells (QZ,2) ist ein Datum als Text formatiert
```

So wars richtig:

```
' Belegdatum
Tabelle1.Cells(ZZ, 3) = CDate(Cells(QZ, 2)) ' ' Cells (QZ,2) wird nun korrekt in Datum umgewandelt
```

3.Lösung

wenn man eine CSV-Datei erhält, die ein Exceldatum mit Uhrzeit enthält und wenn man diese Datei aus dem Editor (und nicht aus Excel) in die Zwischenablage kopiert und in Excel einfügt, dann kommt es oft mit + 4 Jahren an:



Es reicht direkt die Datei in Excel zu öffnen, in Excel in die Zwischenablage zu kopieren und dann im anderen Excel einzufügen. (Sollte mein ZWISCHENABLAGE EINFÜGEN-Code mit der Tabelle ZWISCHENABLAGE im Einsatz sein, bitte dort die Spalten NICHT als Text formatieren, sondern auf STANDARD lassen und es funktioniert.

1904-Datumswerte reparieren

```
Sub DATUMREPAIR()

' Einmalprozedur, die alle im 1904-Datumsformat erfassten Datumswerte nach der Umstellung auf 31.12.1989 repariert
' Das 1904-Datum ist für Macintosh/IOS-Systeme gedacht, das mit Datumswerten vor 1900 nicht zurecht kam
' Für Windows verwendet man immer das Datum OHNE 1904-Option - also wo der erste Tage der 31.12.1899 ist.
' Hat man irrtümlich eine Arbeitsmappe auf 1904 eingestellt und darin die Datumswerte eingestellt, wie gewünscht,
' dann ist das Windows-intern ein falsches Datum - aber es wird zumindest immer richtig angezeigt.
' Sobald man aber mit VBA-Code auf das Datum zugreift und zB eine Zelle kopiert, kommt dort schon das falsche Datum an.
' Darum empfiehlt es sich das Datumsformat einmal korrekt auf OHNE 1904-Option einzustellen und dann alle Datumswerte zu
' korrigieren mit nachfolgendem Code.
' (In der Spalte BJ ist in unserem Fall eine Formel, die aus Monat und Jahr-Funktion einer anderen Zelle ein Monat/Jahr-Ergebnis
' anzeigt. Dieses ist zwar als Zahl formatiert, aber die ISDATE-Funktion erkennt es dennoch als Datum an. Darum wird diese Spalte
' unten aus dem Code ausgenommen)

Dim ZELLE As Range

On Error GoTo FEHLER

For Each ZELLE In ActiveSheet.UsedRange.Cells

   If IsDate(ZELLE) = True And ZELLE.Column <> 62 Then ZELLE = ZELLE + 1462

Next ZELLE

Exit Sub

FEHLER:
MsgBox ZELLE.Address
End Sub
```

Alter berechnen mit NOW und Geburtsdatum

```
TD2.Cells(Z - 3, 34) = int((CDate(Now) - CDate(Cells(Z, 8))) / 365)  ' Alter
```

Auslesen ob eine Zelle ein Datum enthält

Ob eine Zelle ein Datum enhält liest man übrigens so aus:
MsgBox IsDate(ActiveCell.Value)
oder
MsgBox ActiveCell.NumberFormat

BMD-Datum aus normalem Datum machen

```
DATUM = Range("Z2")
If Month(DATUM) < 10 Then
       If Day(DATUM) < 10 Then
          DATUM = Year(DATUM) & "0" & Month(DATUM) & "0" & Day(DATUM)
         Else
          DATUM = Year(DATUM) & "0" & Month(DATUM) & Day(DATUM)
       End If
  Else
       If Day(DATUM) < 10 Then
          DATUM = Year(DATUM) & Month(DATUM) & "0" & Day(DATUM)
         Else
          DATUM = Year(DATUM) & Month(DATUM) & Day(DATUM)
       End If
End If
```

Datumstexte in Datum umwandeln mit und ohne Punkt

Die nachfolgende Funktion kommt mit Datumswerten egal ob mit 2 stelliger Jahreszahl oder mit 4-stelliger Jahreszahl zurecht
mit Punkt: 1.1.17 wird genauso erkannt wie 01.01.2017
ohne Punkt: 10117 wird genauso erkannt wie 010117 - nur ein einstelliges Monat (1-9) MUSS zwingend mit führender 0 übergeben sein, damit ein 10117 als 1.1.17 und nicht als 10.1.17 verstanden wird
Wenn Sonderzeichen statt dem Punkt vorkommen (- _ \ /)usw. werden diese automatisch umgewandelt in einen Punkt.

Wichtig: die Prozedur kann sowohl mit einer Variable aufgerufen, die STRING ist oder auch vom Typ VARIANT ist.

```
Sub TEST1 ()
```

```
 MsgBox DATUM("3.1.17")
End Sub

Sub TEST2

Public Function DATUM(DATUMSTEXT) As Date

Dim C As Integer ' Laufvariable für Durchwandern des Datumstext
Dim Tag As String
Dim Monat As String
Dim Jahr As String


    ' Wir ersetzen ev. vorkommende Sonderzeichen

    DATUMSTEXT = Replace(DATUMSTEXT, "-", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, "_", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, "/", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, "\", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, ",", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, ":", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, ";", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, "'", ".")
    DATUMSTEXT = Replace(DATUMSTEXT, "+", ".")

    ' Im Optimalfall hat der Datumstext 10 Zeichen: 23.12.2020
    If Len(DATUMSTEXT) < 10 Then ' wenn kürzer dann

        ' ---------------------
        ' --- PUNKTE DATUM ---
        ' ---------------------

        If InStr(DATUMSTEXT, ".") > 1 Then ' wenn es Punkte gibt

            ' --- 4-stelliges Jahresdatum ---

            If Left(Right(DATUMSTEXT, 4), 2) = 20 Then ' wenn es mit einem 4-stelligem Jahr endet
                ' Ermitteln Jahr
                Jahr = Right(DATUMSTEXT, 4)
                ' Ermitteln monat
                For C = 1 To Len(DATUMSTEXT)
                    If Mid(DATUMSTEXT, C, 1) = "." Then
                        Monat = Mid(DATUMSTEXT, C + 1, 2)
                        If Right(Monat, 1) = "." Then Monat = "0" & Left(Monat, 1)
```

```
            Exit For
         End If
      Next C
      ' Ermitteln Tag
      Tag = Left(DATUMSTEXT, 2)
      If Right(Tag, 1) = "." Then Tag = "0" & Left(DATUMSTEXT, 1)
      DATUMSTEXT = Tag & "." & Monat & "." & Jahr

   Else

  ' --- 2-stelliges Jahresdatum ---

      ' Ermitteln Jahr
      Jahr = Right(DATUMSTEXT, 2)
      ' Ermitteln monat
      For C = 1 To Len(DATUMSTEXT)
         If Mid(DATUMSTEXT, C, 1) = "." Then
            Monat = Mid(DATUMSTEXT, C + 1, 2)
            If Right(Monat, 1) = "." Then Monat = "0" & Left(Monat, 1)
            Exit For
         End If
      Next C
      ' Ermitteln Tag
      Tag = Left(DATUMSTEXT, 2)
      If Right(Tag, 1) = "." Then Tag = "0" & Left(DATUMSTEXT, 1)
      DATUMSTEXT = Tag & "." & Monat & "." & Jahr

  End If

 Else ' wenn es keine Punkte gibt

' -------------------------
' --- OHNE PUNKTE DATUM ---
' -------------------------


  If Left(Right(DATUMSTEXT, 4), 2) = 20 Then ' wenn es mit einem 4-stelligem Jahr endet

      ' --- 4-stelliges Jahresdatum ---

      If Len(DATUMSTEXT) = 6 Then ' TMJJJJ
         Jahr = Right(DATUMSTEXT, 4)
         Monat = "0" & Mid(DATUMSTEXT, 2, 1)
         Tag = "0" & Left(DATUMSTEXT, 1)
```

```
            End If
            If Len(DATUMSTEXT) = 7 Then ' TMMJJJJ
               Jahr = Right(DATUMSTEXT, 4)
               Monat = Mid(DATUMSTEXT, 2, 2)
               Tag = "0" & Left(DATUMSTEXT, 1)
            End If
            If Len(DATUMSTEXT) = 8 Then ' TTMMJJJJ
               Jahr = Right(DATUMSTEXT, 4)
               Monat = Mid(DATUMSTEXT, 3, 2)
               Tag = Left(DATUMSTEXT, 2)
            End If
            DATUMSTEXT = Tag & "." & Monat & "." & Jahr

         Else

            ' --- 2-stelliges Jahresdatum ---

            If Len(DATUMSTEXT) = 4 Then ' TMJJ
               Jahr = Right(DATUMSTEXT, 2)
               Monat = "0" & Mid(DATUMSTEXT, 2, 1)
               Tag = "0" & Left(DATUMSTEXT, 1)
            End If
            If Len(DATUMSTEXT) = 5 Then ' TMMJJ
               Jahr = Right(DATUMSTEXT, 2)
               Monat = Mid(DATUMSTEXT, 2, 2)
               Tag = "0" & Left(DATUMSTEXT, 1)
            End If
            If Len(DATUMSTEXT) = 6 Then ' TTMMJJ
               Jahr = Right(DATUMSTEXT, 2)
               Monat = Mid(DATUMSTEXT, 3, 2)
               Tag = Left(DATUMSTEXT, 2)
            End If
            DATUMSTEXT = Tag & "." & Monat & "." & Jahr

         End If

      End If
   End If

   DATUM = CDate(DATUMSTEXT)

End Function
```

Datumstext 01012013 in Datum umwandeln

```
 ' Es reicht die Zielvariable vom Typ DATE zu haben und den Text mit Punkten zu übergeben

Dim DATUM as String
Dim STARTDATUM As Date

DATUM = "01012013"
STARTDATUM = Left(DATUM, 2) & "." & Mid(DATUM, 3, 2) & "." & Right(DATUM, 4)
MsgBox STARTDATUM
```

Datum ausfüllen

Angenommen ein bestimmtes Monat in einer Zelle soll um 1 erhöht werden. Die Zelle mit dem Datum war besonders formatiert - aus dem ausgefüllten 1.9.2009 wurde September 2009 angezeigt.

Schreibt man nun per VBA neu zusammengesetzt das Datum hinein - also Tag & "." & Monat & "." & Jahr, so wurde das Format erst wieder richtig, wenn man es mit F2 und Enter übernahm. Es klappte aber sofort, wenn man den oben ermittelten String noch zusätzlich in ein Datum umwandelt mit CDate:

```
Monat = Month(Range("F6"))
Jahr = Year(Range("F6"))

Monat = Monat + 1
If Monat = 13 Then
   Monat = 1
   Jahr = Jahr + 1
End If

Range("F6") = CDate("01." & Monat & "." & Jahr)
```

Datum kommt mit PLUS 4 Jahren an - siehe Eintrag 1904-Problem

Datum umwandeln Excel-Interne Datumszahl für Vergleich

Zellen im Standardformat können Datumswerte erhalten, wenn man sie vollständig tt.mm.jjjj eingibt. Dann ist intern die Excelzahl vorhanden und es wird nur im Layout ein Datum angezeigt. In diesem Format können Zellen auch verglichen werden und frühere Datumswerte sind < spätere Datumswerte.

Aber nur das Zellformat als Datum erlaubt die Eingabe von nur tt.mm ohne Jahr und Excel ergänzt automatisch das aktuelle Jahr.

Wird eine Zelle als Datumsformat formatiert, kann Excel damit nicht mehr rechnen (< / > Vergleiche erkennen nicht, welches Datum zuvor ist.

Daher: man muss für den Vergleich die Zellen mit einem Datumswert umwandeln in die Excel-Interne Zahl

MsgBox CDbl(CDate(ActiveCell))

Datum umwandeln Zahl 31.12.2014=>31122014

Da das Sortieren nach Datum bisweilen nicht klappt (und 1.2, 1.3, 1.4, 2.2, 2.3, 2.4… sortiert wird) wandle ich bisweilen das Datum in einer freien Spalte um ins die entsprechende Excel-Datumszahl (345678) und sortiere diese :

aus 1.2.2007 wird 403045

ActiveSheet.Cells(1, 2) = CLng(ActiveSheet.Cells(1, 1))

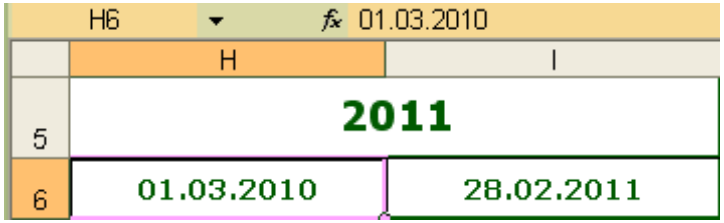
Alternativ: das Datum umwandeln ins BMD-Format: yyyymmdd

```
H = CDate(Cells(1, 1).Value)
d = Day(H)
m = Month(H)
Y = Year(H)
```

Datumseingaben standardisieren

Im nachfolgenden Programmcode der Jahresabschlusscheckliste kann entweder eine Wirtschaftsjahrbeginn-Feld (VON-Feld in H6) oder ein WJ-Ende-Feld (BIS-

Feld in I6) oder die Jahreszahl (in H5) ausgefüllt werden.



Das VON-Feld ist in Zelle H6 und das BIS-Feld ist in Zelle I6.

Zudem gibt es ein Feld, wo nur das Wirtschaftsjahr steht - also zB nur 2008 - in Zelle H5.

Man füllt nur eines der drei Felder aus und die jeweils beiden anderen Felder werden automatisch ausgefüllt.

Bei der Eingabe im VON- oder im BIS-Feld wird zuerst das jeweils ausgefüllte Feld standardisiert (dass zB aus 1.1 => 01.01.2008 wird) und dann werden die jeweiligen beiden anderen Felder automatisch auch ausgefüllt.

Der Code ist übrigens direkt im betreffenden Tabellenblatt im Worksheet_Change-Code

```
' ------------------------------------------------
 ' Wenn VON abweichendem Wirtschaftsjahr gefüllt wird
 ' ------------------------------------------------

 ' Zuerst ergänzen des "VON-Datums"

 If Target.Address = "$H$6" Then
  If AENDERUNG = 1 Then Exit Sub ' wenn Zelle nicht von User, sondern von VBA-Code hier ausgefüllt wird,
  ' dann nicht eine Endlos-Ausfüllschleife starten
  AENDERUNG = 1 ' damit wird nun aktiviert, dass der VBA-Code in den Zellen schreibend aktiv wird und
  ' dass die obige IF-Abfrage eine Endlosschleife verhindern kann
    WERT = Range("H6") ' Auslesen des aktuell eingegebenen Datums
    If Len(WERT) = 3 Then ' das könnte so eine Eingabe sein: 1.1
       WERT = "0" & Left(WERT, 1) & ".0" & Right(WERT, 1) & "." ' das ergibt 01.01.
       If Mid(WERT, 4, 2) <> "01" Then ' wenn ein abweichendes WJ
          WERT = WERT & Range("H5") - 1 ' ergänze die Jahreszahl aus der Jahreszelle H5, aber zieh 1 ab
         Else
          WERT = WERT & Range("H5") ' sonste ergänze die Jahreszahl ohne Korrektur
       End If
       Range("H6") = WERT ' Schreibe das Datum nun wieder zurück in die Zelle, in der User die Eingabe tätigte
    End If
```

```
If Len(WERT) = 4 Then '12.1 oder 1.12 oder 1.1.
     If Mid(WERT, 3, 1) = "." Then ' wenn 12.1
       WERT = Left(WERT, 3) & "0" & Right(WERT, 1) & "."
          If Mid(WERT, 4, 2) <> "01" Then ' abweichendes WJ
             WERT = WERT & Range("H5") - 1
           Else
             WERT = WERT & Range("H5")
          End If


      Else ' 1.12 oder 1.1.

          If Right(WERT, 1) = "." Then ' 1.1.
             WERT = "0" & Left(WERT, 2) & "0" & Right(WERT, 2)
             If Mid(WERT, 4, 2) <> "01" Then ' abweichendes WJ
                WERT = WERT & Range("H5") - 1
              Else
                WERT = WERT & Range("H5")
             End If

           Else ' 1.12
             WERT = "0" & WERT & "."
             If Mid(WERT, 4, 2) <> "01" Then ' abweichendes WJ
                WERT = WERT & Range("H5") - 1
              Else
                WERT = WERT & Range("H5")
             End If
          End If
     End If
  Range("H6") = WERT
End If

If Len(WERT) = 5 Then '1.12. oder 12.1. oder 01.12
    If Mid(WERT, 2, 1) = "." Then ' 1.12.
       WERT = "0" & WERT ' 01.12.
     Else ' 12.1. oder 01.12
       If Right(WERT, 1) = "." Then ' 12.1.
          WERT = Left(WERT, 3) & "0" & Right(WERT, 2) '12.01.
        Else ' 01.12
          WERT = WERT & "."
       End If
    End If

    If Mid(WERT, 4, 2) <> "01" Then ' abweichendes WJ
```

```
        WERT = WERT & Range("H5") - 1
       Else
        WERT = WERT & Range("H5")
    End If
   Range("H6") = WERT
End If

If Len(WERT) = 6 Then ' 1.1.08 - 01.01.
    If Right(WERT, 1) = "." Then ' 01.01.
       If Mid(WERT, 4, 2) <> "01" Then ' abweichendes WJ
         WERT = WERT & Range("H5") - 1
        Else
         WERT = WERT & Range("H5")
       End If
      Else ' 1.1.08
       WERT = "0" & Left(WERT, 2) & "0" & Mid(WERT, 3, 2) & "20" & Right(WERT, 2)
    End If
   Range("H6") = WERT
End If

If Len(WERT) = 7 Then ' 1.12.08 - 12.1.08
    If Mid(WERT, 2, 1) = "." Then ' 1.12.08
       WERT = "0" & Left(WERT, 5) & "20" & Right(WERT, 2)
      Else ' 12.1.08
       WERT = Left(WERT, 3) & "0" & Mid(WERT, 4, 2) & "20" & Right(WERT, 2)
    End If
   Range("H6") = WERT
End If

If Len(WERT) = 8 Then ' 01.01.08 oder 1.1.2008
  If Mid(WERT, 3, 1) = "." Then ' 01.01.08
       WERT = Left(WERT, 6) & "20" & Right(WERT, 2)
    Else ' 1.1.2008
       WERT = "0" & Left(WERT, 2) & "0" & Mid(WERT, 3, 2) & "20" & Right(WERT, 2)
  End If
   Range("H6") = WERT
End If

If Len(WERT) = 9 Then ' 1.12.2008 - 12.1.2008
    If Mid(WERT, 2, 1) = "." Then '  1.12.2008
       WERT = "0" & WERT
      Else ' 12.1.2008
       WERT = Left(WERT, 3) & "0" & Mid(WERT, 4, 6)
    End If
```

```
      Range("H6") = WERT
    End If

' Ändern der Jahreszahl in der Zelle H5

  If Mid(WERT, 4, 2) = "01" Then ' wenn Jänner, dann kein abweichendes WJ
       Range("H5") = Right(Range("H6"), 4)
    Else ' Feb - Dez ' abweichendes WJ
       Range("H5") = Right(Range("H6"), 4) + 1
  End If


' Automatisches Ändern des BIS-Zeitraumes

  If Left(WERT, 2) <> "01" Then ' wenn nicht der erste eines Monats zB 10.01.2008
       ENDDATUM = Val(Left(WERT, 2)) - 1 & Mid(WERT, 3, 4) & Right(WERT, 4) + 1  ' 09.01.2009
       If Len(ENDDATUM) < 10 Then ENDDATUM = "0" & ENDDATUM
       'MsgBox "yep"
    Else ' wenn der erste eines Monats 01.01.2008 oder 01.02.2008
       If Mid(WERT, 4, 2) = "01" Then ' 1.Jänner
            ENDDATUM = "31.12." & Right(WERT, 4)
          Else ' Feb-Dez
            If Mid(WERT, 4, 2) = "02" Then ENDDATUM = "31.01." & Right(WERT, 4) + 1 ' 1.Feb
            If Mid(WERT, 4, 2) = "03" Then
               If (Right(WERT, 4) + 1) Mod 4 = 0 Then
                 ENDDATUM = "29.02." & Right(WERT, 4) + 1 ' März
                Else
                 ENDDATUM = "28.02." & Right(WERT, 4) + 1 ' März
               End If
            End If
            If Mid(WERT, 4, 2) = "04" Then ENDDATUM = "31.03." & Right(WERT, 4) + 1 ' Apr
            If Mid(WERT, 4, 2) = "05" Then ENDDATUM = "30.04." & Right(WERT, 4) + 1 ' Mai
            If Mid(WERT, 4, 2) = "06" Then ENDDATUM = "31.05." & Right(WERT, 4) + 1 ' Jun
            If Mid(WERT, 4, 2) = "07" Then ENDDATUM = "30.06." & Right(WERT, 4) + 1 ' Jul
            If Mid(WERT, 4, 2) = "08" Then ENDDATUM = "31.07." & Right(WERT, 4) + 1 ' Aug
            If Mid(WERT, 4, 2) = "09" Then ENDDATUM = "31.08." & Right(WERT, 4) + 1 ' Sep
            If Mid(WERT, 4, 2) = "10" Then ENDDATUM = "30.09." & Right(WERT, 4) + 1 ' Okt
            If Mid(WERT, 4, 2) = "11" Then ENDDATUM = "31.10." & Right(WERT, 4) + 1 ' Nov
            If Mid(WERT, 4, 2) = "12" Then ENDDATUM = "30.11." & Right(WERT, 4) + 1 ' Dez

       End If
  End If
 Range("I6") = ENDDATUM
 Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
```

```
 Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
 AENDERUNG = 0

End If

' ' ------------------------------------------------
' 'Wenn BIS abweichendem Wirtschaftsjahr gefüllt wird
' ' ------------------------------------------------

 ' Zuerst ergänzen des "BIS-Datums"

 If Target.Address = "$I$6" Then
  If AENDERUNG = 1 Then Exit Sub
  AENDERUNG = 1
    WERT = Range("I6")
    If Len(WERT) = 3 Then ' 1.1
       WERT = "0" & Left(WERT, 1) & ".0" & Right(WERT, 1) & "." ' 01.01.
          WERT = WERT & Range("H5")
       Range("I6") = WERT
        Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
        Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)

    End If

    If Len(WERT) = 4 Then '12.1 oder 1.12 oder 1.1.
         If Mid(WERT, 3, 1) = "." Then ' wenn 12.1
           WERT = Left(WERT, 3) & "0" & Right(WERT, 1) & "."
           WERT = WERT & Range("H5")


          Else ' 1.12 oder 1.1.

              If Right(WERT, 1) = "." Then ' 1.1.
                 WERT = "0" & Left(WERT, 2) & "0" & Right(WERT, 2)
           WERT = WERT & Range("H5")

               Else ' 1.12
                  WERT = "0" & WERT & "."
           WERT = WERT & Range("H5")
              End If
         End If
       Range("I6") = WERT
        Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
        Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
```

```
End If

If Len(WERT) = 5 Then '1.12. oder 12.1. oder 01.12
   If Mid(WERT, 2, 1) = "." Then ' 1.12.
      WERT = "0" & WERT ' 01.12.
    Else ' 12.1. oder 01.12
      If Right(WERT, 1) = "." Then ' 12.1.
         WERT = Left(WERT, 3) & "0" & Right(WERT, 2) '12.01.
       Else ' 01.12
         WERT = WERT & "."
      End If
   End If

      WERT = WERT & Range("H5")
  Range("I6") = WERT
   Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
   Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
End If

If Len(WERT) = 6 Then ' 1.1.08 - 01.01.
   If Right(WERT, 1) = "." Then ' 01.01.
      WERT = WERT & Range("H5")
    Else ' 1.1.08
      WERT = "0" & Left(WERT, 2) & "0" & Mid(WERT, 3, 2) & "20" & Right(WERT, 2)
   End If
  Range("I6") = WERT
   Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
   Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
End If

If Len(WERT) = 7 Then ' 1.12.08 - 12.1.08
   If Mid(WERT, 2, 1) = "." Then ' 1.12.08
      WERT = "0" & Left(WERT, 5) & "20" & Right(WERT, 2)
    Else ' 12.1.08
      WERT = Left(WERT, 3) & "0" & Mid(WERT, 4, 2) & "20" & Right(WERT, 2)
   End If
  Range("I6") = WERT
   Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
   Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
End If

If Len(WERT) = 8 Then ' 01.01.08 oder 1.1.2008
  If Mid(WERT, 3, 1) = "." Then ' 01.01.08
```

```
            WERT = Left(WERT, 6) & "20" & Right(WERT, 2)
         Else ' 1.1.2008
            WERT = "0" & Left(WERT, 2) & "0" & Mid(WERT, 3, 2) & "20" & Right(WERT, 2)
       End If
        Range("I6") = WERT
          Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
          Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
    End If

    If Len(WERT) = 9 Then ' 1.12.2008 - 12.1.2008
       If Mid(WERT, 2, 1) = "." Then '  1.12.2008
          WERT = "0" & WERT
        Else ' 12.1.2008
          WERT = Left(WERT, 3) & "0" & Mid(WERT, 4, 6)
       End If
      Range("I6") = WERT
       Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
       Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
    End If

' Ändern des JA-Jahres

       Range("H5") = Right(Range("I6"), 4)


' Ändern des VON-Zeitraumes

  ' wenn nicht der Letzte eines Monats zB 10.01.2008
  If Left(WERT, 2) <> "28" And Left(WERT, 2) <> "29" And Left(WERT, 2) <> "30" And Left(WERT, 2) <> "31" Then
       STARTDATUM = Val(Left(WERT, 2)) + 1 & Mid(WERT, 3, 4) & Right(WERT, 4) - 1   ' 11.01.2008
       If Len(STARTDATUM) < 10 Then STARTDATUM = "0" & STARTDATUM
    Else ' wenn der letzte eines Monats 31.01.2008 oder 29.02.2008
       If Mid(WERT, 4, 2) = "01" Then ' 31.Jänner
            STARTDATUM = "01.02." & Right(WERT, 4) - 1
          Else ' Feb-Dez
            If Mid(WERT, 4, 2) = "02" Then STARTDATUM = "01.03." & Right(WERT, 4) - 1 ' Feb
            If Mid(WERT, 4, 2) = "03" Then STARTDATUM = "01.04." & Right(WERT, 4) - 1 ' März
            If Mid(WERT, 4, 2) = "04" Then STARTDATUM = "01.05." & Right(WERT, 4) - 1 ' Apr
            If Mid(WERT, 4, 2) = "05" Then STARTDATUM = "01.06." & Right(WERT, 4) - 1 ' Mai
            If Mid(WERT, 4, 2) = "06" Then STARTDATUM = "01.07." & Right(WERT, 4) - 1 ' Jun
            If Mid(WERT, 4, 2) = "07" Then STARTDATUM = "01.08." & Right(WERT, 4) - 1 ' Jul
            If Mid(WERT, 4, 2) = "08" Then STARTDATUM = "01.09." & Right(WERT, 4) - 1 ' Aug
            If Mid(WERT, 4, 2) = "09" Then STARTDATUM = "01.10." & Right(WERT, 4) - 1 ' Sep
            If Mid(WERT, 4, 2) = "10" Then STARTDATUM = "01.11." & Right(WERT, 4) - 1 ' Okt
```

```
            If Mid(WERT, 4, 2) = "11" Then STARTDATUM = "01.12." & Right(WERT, 4) - 1 ' Nov
            If Mid(WERT, 4, 2) = "12" Then STARTDATUM = "01.01." & Right(WERT, 4)  ' Dez
        End If
    End If
 Range("H6") = STARTDATUM
 AENDERUNG = 0

End If


' ------------------------------------------------
'   Wenn Jahreszahl ausgefüllt wird
' ------------------------------------------------

 If Target.Address = "$H$5" Then
    If AENDERUNG = 1 Then Exit Sub
    AENDERUNG = 1
    WERT = Range("H5")
    If Len(WERT) = 2 Then
       WERT = "20" & WERT
       Range("H5") = WERT
    End If

    STARTDATUM = Range("H6")
    ENDDATUM = Range("I6")

    If Left(STARTDATUM, 6) = "01.01." Then ' wenn WJ-Beginn 1.1
       STARTDATUM = Left(STARTDATUM, 6) & WERT
      Else
       STARTDATUM = Left(STARTDATUM, 6) & WERT - 1
    End If
    ENDDATUM = Left(ENDDATUM, 6) & WERT
    If Left(ENDDATUM, 6) = "28.02." Or Left(ENDDATUM, 6) = "29.02." Then ' wenn Februar
       If WERT Mod 4 = 0 Then
          ENDDATUM = "29.02." & WERT
        Else
          ENDDATUM = "28.02." & WERT
       End If

    End If
    Range("H6") = STARTDATUM
    Range("I6") = ENDDATUM
        Sheets("Stammdaten jP").Range("J31") = Left(Range("I6"), 6)
        Sheets("Stammdaten nP").Range("I39") = Left(Range("I6"), 6)
```

```
    AENDERUNG = 0
  End If
```

Datumsfehler 30.12 bei Leerzellen

In Excel 2010 ergibt die Month-Funktion einer leeren Zelle den falschen Wert 12 und die Day-Funktion den falschen Wert 30.

Entdeckt habe ich ihn bei einer Routine, die alle Tage im Jahr richtig rechnete, aber beim 30.12 immer versagte. Daher zusätzlich abprüfen, ob ÜBERHAUPT ein Wert in der Zelle ist, bevor man sie als 30.12 interpretiert.

Feiertage siehe Ostern

Heute und Jetzt

```
Msgbox Date ' heutiger Tag

Msgbox Now ' heutiger Tag und Uhrzeit
```

Heute und Jetzt als Dateinamensstempel YYYYMMDD_HHMMSS

```
    Dim Monat As String     ' Aktuelles Monat mit führender Null, wenn <10
    Dim Tag As String       ' Aktueller Tag mit führender Null, wenn <10

    Dim Stunden As String    ' Aktuelle Stunde mit führender Null
    Dim Minuten As String    ' Aktuelle Minute mit führender Null
    Dim Sekunden As String    ' Aktuelle Sekunde mit führender Null

    Dim Datumstempel As String ' YYYYMMDD HHMMSS

'   Auslesen des Dateinamens der aktuellen Datei

    Dateiname = Left(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") - 1)
    Dateiendung = Mid(ThisWorkbook.Name, InStrRev(ThisWorkbook.Name, ".") + 1)

' Definieren des Datums- und Zeitstempels

    If Month(Now) < 10 Then Monat = "0" & Right(Str(Month(Now)), 1) Else Monat = Right(Str(Month(Now)), 2)
    If Day(Now) < 10 Then Tag = "0" & Right(Str(Day(Now)), 1) Else Tag = Right(Str(Day(Now)), 2)
```

```
If Hour(Now) < 10 Then Stunden = "0" & Right(Str(Hour(Now)), 1) Else Stunden = Right(Str(Hour(Now)), 2)
If Minute(Now) < 10 Then Minuten = "0" & Right(Str(Minute(Now)), 1) Else Minuten = Right(Str(Minute(Now)), 2)
If Second(Now) < 10 Then Sekunden = "0" & Right(Str(Second(Now)), 1) Else Sekunden = Right(Str(Second(Now)), 2)

Datumstempel = Year(Now) & Monat & Tag & "_" & Stunden & Minuten & Sekunden
MsgBox Datumstempel
```

Industriezeit umrechnen in Stunden und Minuten

Version 1 - Aus 7,75 wird 7:45

```
Sub INDUSTRIEZEIT_UMWANDELN()

For Z = 2 To 60000
   If Cells(Z, 10) = "" Then Exit For
   ZEIT = Cells(Z, 10)
   stunden = Int(ZEIT)
   minuten = ZEIT - stunden
   Cells(Z, 10).NumberFormat = "h:mm;@"
   Cells(Z, 10) = stunden & ":" & Round(minuten * 60, 0)


Next Z

End Sub
```

Version 2

```
   Dim UNIT As String ' Menge der Zeit als Industriezeit (12,50 sind 12 Stunden und 30 Minuten)
   Dim STUNDEN As Single ' Stunden
   Dim MINUTEN As Double ' Minuten - da wir runden müssen (Industriezeit) unbedingt DOUBLE nehmen, Single macht beim Runden Probleme

      UNIT = Cells(QZ, 6) ' Auslesen der Industriezeit Stunden,Minuten - zB 7,5 oder 7 oder 7,55
      If Len(UNIT) - InStrRev(UNIT, ",") + 1 = 2 Then UNIT = UNIT & "0" ' wenn nur eine Kommastelle vorhanden ist (zB 7,5) füge eine Null dazu: 7,50
      If InStr(UNIT, ",") > 0 Then ' wenn es ein Komma gibt
         STUNDEN = CDbl(Left(UNIT, InStr(UNIT, ",") - 1))
         MINUTEN = Right(UNIT, Len(UNIT) - InStrRev(UNIT, ","))
         MINUTEN = Round(CDbl(MINUTEN) / 100 * 60)
       Else ' sonst nimm die ganze Zahl als Stunden
         STUNDEN = UNIT
         MINUTEN = 0
```

```
End If
MsgBox UNIT & " - " & STUNDEN & " - " & MINUTEN ' Zu Testzwecken
```

Internetzeit auslesen – Sommer-/Winterzeit feststellen

Timeserver:

```
.AddItem "time-a.timefreq.bldrdoc.gov"
.AddItem "time-b.timefreq.bldrdoc.gov"
.AddItem "time-c.timefreq.bldrdoc.gov"
.AddItem "utcnist.colorado.edu"
.AddItem "time-nw.nist.gov"
.AddItem "nist1.nyc.certifiedtime.com"
.AddItem "nist1.dc.certifiedtime.com"
.AddItem "nist1.sjc.certifiedtime.com"
.AddItem "nist1.datum.com"
.AddItem "ntp2.cmc.ec.gc.ca"
.AddItem "ntps1-0.uni-erlangen.de"
.AddItem "ntps1-1.uni-erlangen.de"
.AddItem "ntps1-2.uni-erlangen.de"
.AddItem "ntps1-0.cs.tu-berlin.de"
.AddItem "time.ien.it"
.AddItem "ptbtime1.ptb.de"
.AddItem "ptbtime2.ptb.de"
```

In meiner Zeiterfassung kombiniere ich die 2.Version 2015 mit der 1.Version 2015 – wenn die erste nicht klappt, wird automatisch die nachfolgende zweite hier ausgeführt

2. Version 2015 – ging im September 2015 sehr gut

```
Public Sub GetDateTime()
   Dim dtmNow As Date
   dtmNow = InternetTime
   MsgBox TimeValue(dtmNow)
   MsgBox DateValue(dtmNow)
End Sub


Private Declare Function WSAStartup Lib "ws2_32.dll" ( _
   ByVal wVersionRequired As Integer, _
```

```
    ByRef lpWSAData As WSADATA) As Long
Private Declare Function socket Lib "ws2_32.dll" ( _
    ByVal af As Long, _
    ByVal lType As Long, _
    ByVal protocol As Long) As Long
Private Declare Function connect Lib "ws2_32.dll" ( _
    ByVal s As Long, _
    ByRef Name As SOCKADDR, _
    ByVal namelen As Long) As Long
Private Declare Function htons Lib "ws2_32.dll" ( _
    ByVal hostshort As Integer) As Integer
Private Declare Function inet_addr Lib "ws2_32.dll" ( _
    ByVal cp As String) As Long
Private Declare Function recv Lib "ws2_32.dll" ( _
    ByVal s As Long, _
    ByVal buf As String, _
    ByVal lLen As Long, _
    ByVal flags As Long) As Long
Private Declare Function closesocket Lib "ws2_32.dll" ( _
    ByVal s As Long) As Long
Private Declare Function GetTimeZoneInformation Lib "kernel32.dll" ( _
    ByRef lpTZI As TIME_ZONE_INFORMATION) As Long
Private Declare Function GetTickCount Lib "kernel32.dll" () As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function WSAGetLastError Lib "ws2_32.dll" () As Long

Private Const WS_VERSION_REQD As Long = &H101&
Private Const WSADEscriptION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128

Private Const AF_INET As Long = 2

Private Const SOCK_STREAM As Long = 1
Private Const IPPROTO_TCP As Long = 6

Private Const TIME_ZONE_ID_DAYLIGHT As Long = 2

Private Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * WSADEscriptION_LEN
    szSystemStatus As String * WSASYS_STATUS_LEN
    iMaxSockets As Integer
    iMaxUdpDg As Integer
    lpVendorInfo As Long
End Type

Private Type SOCKADDR
    sin_family As Integer
```

```
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type

Private Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

Private Type TIME_ZONE_INFORMATION
    Bias As Long
    StandardName(31) As Integer
    StandardDate As SYSTEMTIME
    StandardBias As Long
    DaylightName(31) As Integer
    DaylightDate As SYSTEMTIME
    DaylightBias As Long
End Type
```

Private Function InternetTime() As Date

```
    Const SERVER_IP As String = "192.53.103.104"

    Dim udtData As WSADATA, udtAdresse As SOCKADDR
    Dim udtTimeZone As TIME_ZONE_INFORMATION
    Dim strTime As String, strRecv_Data As String * 5
    Dim dblTimeStamp As Double
    Dim lngStartup_Return As Long, lngSocket_Return As Long
    Dim lngConnect_Return As Long, lngReceive_Return As Long
    Dim lngClose_Return As Long, lngTZI_Return As Long
    Dim lngGTC_Return_1 As Long

    lngStartup_Return = WSAStartup(WS_VERSION_REQD, udtData)
    If lngStartup_Return <> 0 Then Exit Function

    lngSocket_Return = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
    If lngSocket_Return > 10000 And lngSocket_Return < 11005 Then Exit Function

    udtAdresse.sin_family = AF_INET
    udtAdresse.sin_addr = inet_addr(SERVER_IP)
    udtAdresse.sin_port = htons(37)
    lngConnect_Return = connect(lngSocket_Return, udtAdresse, Len(udtAdresse))
```

```
    If lngConnect_Return <> 0 Then Exit Function

    lngGTC_Return_1 = GetTickCount()
    lngReceive_Return = recv(lngSocket_Return, strRecv_Data, Len(strRecv_Data), 0)
    strTime = Left$(strRecv_Data, lngReceive_Return)
    If lngReceive_Return <> 4 Then Exit Function

    lngClose_Return = closesocket(lngSocket_Return)
    If lngClose_Return <> 0 Then Exit Function

    Call WSACleanup

    dblTimeStamp = Asc(Mid(strTime, 1, 1)) * 256 ^ 3 + _
       Asc(Mid(strTime, 2, 1)) * 256 ^ 2 + _
       Asc(Mid(strTime, 3, 1)) * 256 ^ 1 + _
       Asc(Mid(strTime, 4, 1)) - 3155673600#

    lngTZI_Return = GetTimeZoneInformation(udtTimeZone)

    If lngTZI_Return = TIME_ZONE_ID_DAYLIGHT Then
       dblTimeStamp = dblTimeStamp - (udtTimeZone.Bias * 60 + _
          udtTimeZone.DaylightBias * 60)
    Else
       dblTimeStamp = dblTimeStamp - udtTimeZone.Bias * 60
    End If

    lngGTC_Return_1 = Round((GetTickCount - lngGTC_Return_1) / 1000, 0)
    dblTimeStamp = dblTimeStamp + lngGTC_Return_1
    InternetTime = DateAdd("s", dblTimeStamp, "1.1.2000")

End Function
```

1. VERSION 2015 – ging am September am lokalen PC zuhause nicht mehr – bei anderen PCs schon – eventuell wäre nur Neustart notwendig gewesen

```
' ****************************************************************
'  NEUE Version der Internetzeitauslese - ist Nachfolger für erste
' Version
```

```
' ************************************************************************

Function InternetTime(Optional GMTDifference As Integer) As Date

    '-------------------------------------------------------------------------------
    'This function returns the Greenwich Mean Time retrieved from an internet server.
    'You can use the optional argument GMTDifference in order to add (or subtract)
    'an hour from the GMT time. For Example if you call the function as:
    '=InternetTIme(2) it will return the (local) hour GMT + 2. Note that the
    'GMTDifference variable is an integer number.
    '-------------------------------------------------------------------------------

    'Declaring the necessary variables.
    Dim Request     As Object
    Dim ServerURL   As String
    Dim Results     As String
    Dim NetDate     As String
    Dim NetTime     As Date
    Dim LocalDate   As Date
    Dim LocalTime   As Date

    'Check if the time difference is within the accepted range.
    If GMTDifference < -12 Or GMTDifference > 14 Then
        Exit Function
    End If

    'The server address.
    ServerURL = "http://www.timeanddate.com/worldclock/fullscreen.html?n=2"

    'Build the XMLHTTP object and check if was created successfully.
    On Error Resume Next
    Set Request = CreateObject("Microsoft.XMLHTTP")
    If Err.Number <> 0 Then
        Exit Function
    End If
```

```
On Error GoTo 0

'Create the request.
Request.Open "GET", ServerURL, False, "", ""

'Send the request to the internet server.
Request.Send

'Based on the status node result, proceed accordingly.
If Request.ReadyState = 4 Then

    'If the request succeed, the following line will return
    'something like this: Mon, 30 Sep 2013 18:33:23 GMT.
    Results = Request.getResponseHeader("date")

    'Use the Mid function to get something like: 30 Sep 2013 18:33:23.
    Results = Mid(Results, 6, Len(Results) - 9)

    'Use the Left and Right function to distinguish the date and time.
    NetDate = Left(Results, Len(Results) - 9) '30 Sep 2013
    NetTime = Right(Results, 8) '18:33:23

    'Convert the date into a valid Excel date 30 Sep 2013 -> 30/9/2013.
    'Required for countries that have some non-Latin characters at their alphabet (Greece, Russia, Serbia etc.).
    LocalDate = ConvertDate(NetDate)

    'Add the hour difference to the retrieved GMT time.
    LocalTime = NetTime + GMTDifference / 24

    'Return the local date and time.
    InternetTime = LocalDate + LocalTime

    ' ------------------------------
    ' Meine Ergänzung
    ' ------------------------------
    ' Set Your Standard Time from Greenwich Mean Time (da ich die Internettime-Funktion OHNE GMTDifference aufrufe, muss
```

```
ich es hier noch addieren
        ' ob diese neue Internettime-Funktion eine Unterscheidung brauch in Sommerzeit und Winterzeit, wie die erste, weiß ich
nicht und muss ich ab November testen.
        ' Austrian Time: Sommermonate (IST) = GMT+2 / Wintermonate = GMT+1 ?
        If IstSommerzeit(Now) = True Then
           Hr = 2      'Hours.
         Else
           Hr = 1
        End If

           Mn = 0     'Minutes.
           'Sc = 0     'Seconds.

        NewNow = DateAdd("h", Hr, InternetTime) 'Adding 1 o. 2 Hours to GMT.
        NewNow = DateAdd("n", Mn, NewNow)   'Adding 30 Minutes to GMT.
        'NewNow = DateAdd("s", Sc, NewNow)  'Adding 0 Seconds to GMT.

        InternetTime = NewNow




    End If

    'Release the XMLHTTP object.
    Set Request = Nothing

End Function

Function ConvertDate(strDate As String) As Date

    '--------------------------------------------------------------------
    'This function converts the input date into a valid Excel date.
    'For example the 30 Sep 2013 becomes 30/9/2013.
    'Required for countries that have non-Latin characters at their alphabet.
```

```
    'Written by:    Christos Samaras
    'Date:          25/09/2013
    'e-mail:        xristos.samaras@gmail.com
    'site:          http://www.myengineeringworld.net
    '--------------------------------------------------------------------

    'Declaring the necessary variables.
    Dim MyMonth     As Integer

    'Check the month and convert it to number.
    Select Case UCase(Mid(strDate, 4, 3))
        Case "JAN": MyMonth = 1
        Case "FEB": MyMonth = 2
        Case "MAR": MyMonth = 3
        Case "APR": MyMonth = 4
        Case "MAY": MyMonth = 5
        Case "JUN": MyMonth = 6
        Case "JUL": MyMonth = 7
        Case "AUG": MyMonth = 8
        Case "SEP": MyMonth = 9
        Case "OCT": MyMonth = 10
        Case "NOV": MyMonth = 11
        Case "DEC": MyMonth = 12
    End Select

    'Rebuild the date.
    ConvertDate = DateValue(Right(strDate, 4) & "/" & MyMonth & "/" & Left(strDate, 2))

End Function


Function IstSommerzeit(Datum As Date) As Boolean
Dim WiEnde As Date
Dim SoAnfang As Date
Dim SoEnde As Date
Dim WiAnfang As Date
```

```
    Application.Volatile
    WiEnde = DateSerial(Year(Datum), 4, 1) - _
             Weekday(DateSerial(Year(Datum), 4, 1), vbMonday) + TimeSerial(2, 0, 0)
    SoAnfang = DateSerial(Year(Datum), 4, 1) - _
             Weekday(DateSerial(Year(Datum), 4, 1), vbMonday) + TimeSerial(3, 0, 0)
    SoEnde = DateSerial(Year(Datum), 11, 1) - _
             Weekday(DateSerial(Year(Datum), 11, 1), vbMonday) + TimeSerial(2, 0, 0)
    WiAnfang = DateSerial(Year(Datum), 11, 1) - _
             Weekday(DateSerial(Year(Datum), 11, 1), vbMonday) + TimeSerial(3, 0, 0)
    If Datum < WiEnde Then
       IstSommerzeit = False
    ElseIf Datum >= WiEnde And Datum < SoAnfang Then
       IstSommerzeit = "#WERT!"    'nicht möglich
    ElseIf Datum >= SoAnfang And Datum < SoEnde Then
       IstSommerzeit = True
    ElseIf Datum >= SoEnde And Datum < WiAnfang Then
       IstSommerzeit = "#WERT!"    'beides möglich
    Else
       IstSommerzeit = False
    End If
End Function
```

1. Version 2014 INTERNET_TIME

Sub SCHREIBE_ABMELDEN_NM()

```
' Hilfsroutine, die die Internetzeit für ein Abmelden am Nachmittag in Spalte AM/39 einträgt

' --- Eintragen der Kontrollzeit aus dem Internet in den ausgeblendeten Bereich, der zur Kontrolle dient ---

If INTERNET_TIME <> "00:00:00" Then ' wenn eine Internetverbindung klappt
    Cells(TAG_ZEILE, 39) = Hour(INTERNET_TIME) & ":" & Minute(INTERNET_TIME)
End if
```

End Sub

Function INTERNET_TIME() As Date

```
Dim ws
Dim http
Dim GMT_Time, NewNow, NewDate, NewTime, Hr, Mn ', Sc

'Below line wont work since clock providers changed the URL.
'Const GMTTime As String = "http://wwp.greenwichmeantime.com/time/scripts/clock-8/runner.php"


'Updated URL to fetch internet time ***
'Macro updated Date & Time: 27-Oct-12 1:07 PM

Const GMTTime As String = "http://wwp.greenwichmeantime.com/time/scripts/clock-8/runner.php?tz=gmt"

On Error Resume Next
Set http = CreateObject("Microsoft.XMLHTTP")

http.Open "GET", GMTTime & Now(), False, "", ""
http.send

GMT_Time = http.getResponseHeader("Date")
' MsgBox GMT_Time
GMT_Time = Mid$(GMT_Time, 6, Len(GMT_Time) - 9)
'MsgBox GMT_Time

'Set Your Standard Time from Greenwich Mean Time.
'Austrian Time: Sommermonate (IST) = GMT+2 / Wintermonate = GMT+1
```

```
If IstSommerzeit(Now) = True Then
   Hr = 2      'Hours.
  Else
   Hr = 1
End If

   Mn = 0     'Minutes.
   'Sc = 0     'Seconds.

NewNow = DateAdd("h", Hr, GMT_Time) 'Adding 5 Hours to GMT.
NewNow = DateAdd("n", Mn, NewNow)   'Adding 30 Minutes to GMT.
'NewNow = DateAdd("s", Sc, NewNow)  'Adding 0 Seconds to GMT.

INTERNET_TIME = NewNow

Set http = Nothing
```

End Function

Function IstSommerzeit(Datum As Date) As Boolean

```
Dim WiEnde As Date
Dim SoAnfang As Date
Dim SoEnde As Date
Dim WiAnfang As Date
   Application.Volatile
   WiEnde = DateSerial(Year(Datum), 4, 1) - _
            Weekday(DateSerial(Year(Datum), 4, 1), vbMonday) + TimeSerial(2, 0, 0)
   SoAnfang = DateSerial(Year(Datum), 4, 1) - _
            Weekday(DateSerial(Year(Datum), 4, 1), vbMonday) + TimeSerial(3, 0, 0)
   SoEnde = DateSerial(Year(Datum), 11, 1) - _
            Weekday(DateSerial(Year(Datum), 11, 1), vbMonday) + TimeSerial(2, 0, 0)
   WiAnfang = DateSerial(Year(Datum), 11, 1) - _
            Weekday(DateSerial(Year(Datum), 11, 1), vbMonday) + TimeSerial(3, 0, 0)
   If Datum < WiEnde Then
      IstSommerzeit = False
   ElseIf Datum >= WiEnde And Datum < SoAnfang Then
      IstSommerzeit = "#WERT!"    'nicht möglich
   ElseIf Datum >= SoAnfang And Datum < SoEnde Then
      IstSommerzeit = True
   ElseIf Datum >= SoEnde And Datum < WiAnfang Then
      IstSommerzeit = "#WERT!"    'beides möglich
```

```
    Else
        IstSommerzeit = False
    End If
End Function
```

2.Version 2014

```
Function GetOfficialUSTime() As Boolean

On Error GoTo ErrHandler

Dim RetVal As Variant
Dim MyURL As String
Dim tim As String
Dim dat As String
Dim timStart As Integer
Dim datStart As Integer
Dim datEnd As Integer
Dim oHttpTest As Object

Set oHttpTest = CreateObject("Microsoft.XMLHTTP")
MyURL = "http://www.time.gov/timezone.cgi?Central/d/-6"
oHttpTest.Open "POST", MyURL, False
oHttpTest.Send

If CLng(oHttpTest.Status) < 300 Then 'Response is OK
datStart = InStr(oHttpTest.responseText, "<td align=" & Chr(34) & "center" & Chr(34) & "><font size=" & Chr(34) & "7" & Chr(34) & " color=" & Chr(34) &
"white" & Chr(34) & "><b>") + 116
datEnd = InStr(datStart, oHttpTest.responseText, "<br>")
timStart = InStr(oHttpTest.responseText, "<td align=" & Chr(34) & "center" & Chr(34) & "><font size=" & Chr(34) & "7" & Chr(34) & " color=" & Chr(34) &
"white" & Chr(34) & "><b>") + 51

tim = Mid(oHttpTest.responseText, timStart, 8)
dat = Mid(oHttpTest.responseText, datStart, datEnd - datStart)
MsgBox "Time = " & tim & vbCrLf & "Date = " & dat
Else
RetVal = MsgBox("Status Code: " & oHttpTest.Status & vbCrLf & "Status Text: " & oHttpTest.statusText & vbCrLf & vbCrLf & "Unexpected Response From
Server" & vbCrLf & vbCrLf & "Press 'OK' To See Full Response in Web Browser", vbExclamation + vbOKCancel, "Unexpected Server Error")
If RetVal = vbOK Then
If Dir("C:\HTTP_ERR.HTM") <> "" Then Kill "C:\HTTP_ERR.HTM"
Open "C:\HTTP_ERR.HTM" For Output As #1
```

```
Print #1, oHttpTest.responseText
Close #1
Application.FollowHyperlink "C:\HTTP_ERR.HTM"
End If
End If

Egress:
Set oHttpTest = Nothing
Exit Function

ErrHandler:
MsgBox Err.Description
Resume Egress
End Function

Sub TESTE()
    MsgBox GetOfficialUSTime
End Sub
```

Beste weitere Lösung - getestet in VBA

```
Sub GetiNetTime()

'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'
'
'   The GetiNetTime macro is written by Karthikeyan T.
'
'   Please Note: Original code adjusted here for setting Indian Standard Time,
'   India Standard Time (IST) = GMT+5:30
'
'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'

Dim ws
Dim http
Dim GMT_Time, NewNow, NewDate, NewTime, Hr, Mn ', Sc

'Below line wont work since clock providers changed the URL.
'Const GMTTime As String = "http://wwp.greenwichmeantime.com/time/scripts/clock-8/runner.php"


'Updated URL to fetch internet time ***
'Macro updated Date & Time: 27-Oct-12 1:07 PM
```

```
Const GMTTime As String = "http://wwp.greenwichmeantime.com/time/scripts/clock-8/runner.php?tz=gmt"

On Error Resume Next
Set http = CreateObject("Microsoft.XMLHTTP")

http.Open "GET", GMTTime & Now(), False, "", ""
http.send

GMT_Time = http.getResponseHeader("Date")
MsgBox GMT_Time
GMT_Time = Mid$(GMT_Time, 6, Len(GMT_Time) - 9)
MsgBox GMT_Time

'Set Indian Standard Time from Greenwich Mean Time.
'India Standard Time (IST) = GMT+5:30
    Hr = 0      'Hours.
    Mn = 0      'Minutes.
    'Sc = 0     'Seconds.

NewNow = DateAdd("h", Hr, GMT_Time) 'Adding 5 Hours to GMT.
NewNow = DateAdd("n", Mn, NewNow)   'Adding 30 Minutes to GMT.
'NewNow = DateAdd("s", Sc, NewNow)  'Adding 0 Seconds to GMT.

MsgBox "Current Date & Time is:  IST " & NewNow, vbOKOnly, "GetiNetTime"

'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'
'
'   If you want to insert the new date & time in excel worksheet just unquote
'   the following lines,
'
'   Sheets("Sheet1").Select
'   Range("A1").Select
'   ActiveCell.Value = NewNow
'
'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'

'Insert current date & time in cell on selected worksheet.
'Sheets("Sheet1").Select        'Select worksheet as you like
'Range("A1").Select             'Change the destination as you like
'ActiveCell.Value = NewNow

'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'
'
```

```
'   If you want to change the system time just unquote the following lines,
'
'   Set ws = CreateObject("WScript.Shell")
'   NewDate = DateValue(NewNow)
'   NewTime = Format(TimeValue(NewNow), "hh:mm:ss")
'   ws.Run "%comspec% /c time " & NewTime, 0
'   ws.Run "%comspec% /c date " & NewDate, 0
'   Set ws = Nothing
'
'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'

'Set ws = CreateObject("WScript.Shell")
'Split out date.
'NewDate = DateValue(NewNow)

'Split out time.
'NewTime = Format(TimeValue(NewNow), "hh:mm:ss")

'Run DOS Time command in hidden window.
'ws.Run "%comspec% /c time " & NewTime, 0

'Run DOS Date command in hidden window.
'ws.Run "%comspec% /c date " & NewDate, 0

Cleanup:
'Set ws = Nothing
Set http = Nothing
```

VARIANTE 2 die auch in VBA geht

```
Function GetOfficialUSTime() As Boolean

On Error GoTo ErrHandler

Dim RetVal As Variant
Dim MyURL As String
Dim tim As String
Dim dat As String
Dim timStart As Integer
Dim datStart As Integer
Dim datEnd As Integer
Dim oHttpTest As Object
```

```
Set oHttpTest = CreateObject("Microsoft.XMLHTTP")
MyURL = "http://www.time.gov/timezone.cgi?Central/d/-6"
oHttpTest.Open "POST", MyURL, False
oHttpTest.Send

If CLng(oHttpTest.Status) < 300 Then 'Response is OK
datStart = InStr(oHttpTest.responseText, "<td align=" & Chr(34) & "center" & Chr(34) & "><font size=" & Chr(34) & "7" & Chr(34) & " color=" & Chr(34) &
"white" & Chr(34) & "><b>") + 116
datEnd = InStr(datStart, oHttpTest.responseText, "<br>")
timStart = InStr(oHttpTest.responseText, "<td align=" & Chr(34) & "center" & Chr(34) & "><font size=" & Chr(34) & "7" & Chr(34) & " color=" & Chr(34) &
"white" & Chr(34) & "><b>") + 51

tim = Mid(oHttpTest.responseText, timStart, 8)
dat = Mid(oHttpTest.responseText, datStart, datEnd - datStart)
MsgBox "Time = " & tim & vbCrLf & "Date = " & dat
Else
RetVal = MsgBox("Status Code: " & oHttpTest.Status & vbCrLf & "Status Text: " & oHttpTest.statusText & vbCrLf & vbCrLf & "Unexpected Response From
Server" & vbCrLf & vbCrLf & "Press 'OK' To See Full Response in Web Browser", vbExclamation + vbOKCancel, "Unexpected Server Error")
If RetVal = vbOK Then
If Dir("C:\HTTP_ERR.HTM") <> "" Then Kill "C:\HTTP_ERR.HTM"
Open "C:\HTTP_ERR.HTM" For Output As #1
Print #1, oHttpTest.responseText
Close #1
Application.FollowHyperlink "C:\HTTP_ERR.HTM"
End If
End If

Egress:
Set oHttpTest = Nothing
Exit Function

ErrHandler:
MsgBox Err.Description
Resume Egress
End Function
```

Variante 3 für VB

http://www.mrexcel.com/forum/excel-questions/542483-excel2003-get-time-internet.html

Variante 4 für VB

http://vbnet.mvps.org/index.html?code/network/netremotetodsync.htm

Jahresdatum kommt als PLUS 4 Jahre an - siehe Eintrag 1904-Problem

Kalenderwoche berechnen

Variante 1 - Deutsche DIN

Definition gemäss DIN 1355:
Die deutschen Kalenderwochen berechnen sich wie folgt:
Der 1. Januar eines Jahres gehört erst dann zur ersten Kalenderwoche, wenn dieser Tag auf einen Montag, Dienstag, Mittwoch oder Donnerstag fällt.
Falls der 1. Januar ein Freitag, Samstag oder Sonntag ist, zählt er, und ggf. auch der 2. und 3. Januar, noch zur letzten Kalenderwoche des vorherigen Jahres.
Weiterhin können der 29., 30. und 31.12. eines Jahres schon zur Kalenderwoche 1 des neuen Jahres gehören.
Das ist genau dann der Fall, wenn der 31.12. auf einen Montag, Dienstag oder Mittwoch fällt.

```
Option Explicit

Public Sub Test_KW()
 Dim chkDate As Date
 On Error GoTo Fehler
 chkDate = InputBox("Bitte geben Sie ein Datum in dieser Form ein:", _
  "DIN KW Berechnung", "1.1.2007")
 If IsEmpty(chkDate) Then Exit Sub
 If Not IsDate(chkDate) Then
  MsgBox "Ungültiges Datum"
  Exit Sub
 End If
 MsgBox "Die DIN KW für den " & Chr$(13) & _
  chkDate & Chr$(13) & "lautet: " & DIN_KW(chkDate)
 Exit Sub
 Fehler:
 MsgBox Err.Description
End Sub

Public Function DIN_KW(DasDatum As Date) As Byte
 Dim KW As Date
 KW = 4 + DasDatum - Weekday(DasDatum, 2)
```

```
 DIN_KW = (KW - DateSerial(Year(KW), 1, -6)) \ 7
End Function
```

Variante 2

```
Sub test()
MsgBox KWoche("1.2.2011")
End Sub

Function KWoche(d As Date) As Integer
'von Christoph Kremer, Aachen
Dim t&
   t = DateSerial(Year(d + (8 - WeekDay(d)) Mod 7 - 3), 1, 1)
   KWoche = (d - t - 3 + (WeekDay(t) + 1) Mod 7) \ 7 + 1
End Function
```

Version 3

```
Function DKW(dat As Date) As Integer
Dim a As Integer
a = Int((dat - DateSerial(Year(dat), 1, 1) + _
((WeekDay(DateSerial(Year(dat), 1, 1)) + 1) _
Mod 7) - 3) / 7) + 1
If a = 0 Then
a = DKW(DateSerial(Year(dat) - 1, 12, 31))
ElseIf a = 53 And (WeekDay(DateSerial(Year(dat), 12, 31)) - 1) _
Mod 7 <= 3 Then
a = 1
End If
DKW = a
End Function
```

Version 4 direkt als Excelformel

Ermittlung der Kalenderwoche nach DIN-Norm * (7)

Aufgabe

Die Funktion KALENDERWOCHE() rechnet bis einschließlich Excel 2007 falsch bzw. nach amerikanischen Standard. (Zudem muß bis einschließlich Excel 2003 das Add-In Analysefunktionen geladen sein.)

Beispiel:

	A	B	C
1	Datum	falsch	richtig
2	31.12.99	53	52
3	01.01.00	1	52
4	02.01.00	1	52
5	16.09.00	38	37

1976 wurde der Wochenbeginn auf Montag festgelegt. (Vorher gab's die Kalenderwoche in dem Sinne also nicht.)
Die erste Woche des Jahres ist definiert als die Woche, in die mindestens 4 Tage fallen = DIN 1355. Entspricht der internationalen Norm ISO 8601 (1988); übernommen von der EU als EN 28601 (1992) und in Deutschland als DIN EN 28601 (1993) umgesetzt (vereinfacht: die Woche, die den 04. Januar enthält).

(Amerikanisch: immer die Woche, die den 01. Januar enthält.)

Lösung

das Datum (größer 31.12.1900) steht in A1

```
=KÜRZEN((A1-WOCHENTAG(A1;2)-DATUM(JAHR(A1+4-WOCHENTAG(A1;2));1;-10))/7)
```

Alternative von Franz Pölt:

```
=KÜRZEN((A1-DATUM(JAHR(A1+3-REST(A1-2;7));1;REST(A1-2;7)-9))/7)
```

Zelle benutzerdefiniert formatieren mit 0". KW"

Neu ab Excel 2010:
Wenn man den 2. Parameter *Zahl_Typ* der Funktion KALENDERWOCHE mit 21 belegt, erhält man nun auch für Deutschland die richtigen Ergebnisse:
=KALENDERWOCHE(A1;21)
funktioniert tadellos.

Soll zusätzlich zur KW noch die entsprechende Jahreszahl ausgegeben werden - ist nur kompliziert um Silvester herum, denn z.B. der 03.01.2010 ist 53/2009, dann (eine der obigen Formeln steht in B1):
=B1&"/"&MIN(JAHR(A1-1-REST(A1-2;7)+4);JAHR(A1-REST(A1-1;7)+4))
verkürzt von Erich Gier:
=B1&"/"&JAHR(A1+3-REST(A1-2;7))

Um herauszufinden, wieviel Kalenderwochen (52 oder 53) das Jahr des Datums in A1 hat (Franz Pölt):
=52+(TAG(346-REST("2.1."&JAHR(A1);7)+("1-"&JAHR(A1)))<8)
bzw. allgemein, wenn in der Systemsteuerung ein anderes Datumsformat als TT.MM.JJJJ eingestellt wurde (Horst Schmid):
=52+(TAG(346-REST(DATUM(JAHR(A1);1;2);7) +(DATUM(JAHR(A1);1;1)))<8)

Datum aus Kalenderwoche berechnen

In A1 steht die Kalenderwoche.
In A2 die vierstellige Jahreszahl
=DATUM(A2;1;1)+$A$1*7-WOCHENTAG(DATUM(A2;1;1);2)+WENN(WOCHENTAG(DATUM(A2;1;1);2)>4;1;-6)

Theorie Kalenderwoche

Die Kalenderwoche (abgekürzt: KW) ist für Deutschland durch die Norm DIN 1355 definiert.
Diese gilt seit dem 1. Januar 1976.

Danach besteht die Kalenderwoche aus 7 Tagen und beginnt am Montag und endet am Sonntag (Tage Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag). Abgekürzt werden die Wochentagsnamen durch die ersten beiden Buchstaben des Wochentagnamens - also Mo für Montag.
Ein Jahr kann 52 bis 53 Kalenderwochen haben. Die Zahl hängt davon ab, wie die erste Kalenderwoche begann.

Folgende Regeln gelten zur Ermittlung der ersten Kalenderwoche eines Jahres:

1. Die erste Kalenderwoche eines Kalenderjahres ist die Woche, in die mind. 4 Tage der ersten 7 Januartage fallen.
2. An der Grenze der Kalenderjahre gehören also entweder die ersten 3 Tage des kommenden Jahres zur letzten Kalenderwoche des alten Jahres oder die letzten 3 Tage des alten Jahres zur ersten Kalenderwoche des neuen Jahres.
3. Kalenderjahre, die mit dem Wochentag Donnerstag beginnen, haben 53 Kalenderwochen. Im Fall von Schaltjahren gilt dies auch für Jahre, die an einem Mittwoch beginnen.

Zählweise nach ISO 8601

Die deutschsprachige Kalender-Industrie hält sich ausnahmslos an die internationale Norm ISO 8601, die als letzten Tag der Woche den Sonntag bestimmt, statt des Samstags/Sonnabends/Sabbats, wie es in der jüdisch-christlich-islamischen Tradition üblich ist.

Im Geltungsbereich der Normen des DIN Deutschen Instituts für Normung e. V. werden seit 1976 durch Normung folgende Regeln empfohlen:

- Jeden Montag und nur montags beginnt eine neue Kalenderwoche.
- Die erste Kalenderwoche ist diejenige, die mindestens vier Tage des neuen Jahres enthält.

Aus diesen Punkten ergeben sich folgende Eigenschaften:

- Es gibt keine unvollständigen Kalenderwochen, ausnahmslos jede KW enthält genau sieben Tage.
- Jedes Jahr hat entweder 52 oder 53 Kalenderwochen.
- Ein Jahr hat genau dann 53 Kalenderwochen, wenn es mit einem Donnerstag beginnt oder endet:
    - Ein Normaljahr mit 53 Wochen beginnt *und* endet an einem Donnerstag.
    - Ein Schaltjahr mit 53 Wochen beginnt entweder an einem Mittwoch und endet somit mit Donnerstag oder beginnt an einem Donnerstag und endet an einem Freitag.
- Der 4. Januar liegt immer in Kalenderwoche 1.
- Der 29., 30. und 31. Dezember können schon zur ersten Kalenderwoche des Folgejahres gehören.
- Der 1., 2. und 3. Januar können noch in der letzten Kalenderwoche des Vorjahres liegen.
- Der Donnerstag ist ausschlaggebend, zu welchem Jahr die Woche gezählt wird. Liegt er im neuen Jahr, ist es die Kalenderwoche 1.

Kalenderwochen-Anzahl eines Jahres

```
Sub Test_AnzWo()
        MsgBox AnzWo(2012)
        Msgbox Year(Date)
End Sub

Public Function AnzWo(XJahr As Integer)
    Dim DieWochen As Integer, i As Integer
    For i = 31 To 28 Step -1
        DieWochen = DIN_KW(DateSerial(XJahr, 12, i))
        If DieWochen > 1 Then Exit For
    Next
    AnzWo = DieWochen
End Function

Public Function DIN_KW(DasDatum As Date) As Byte
 Dim KW As Date
 KW = 4 + DasDatum - Weekday(DasDatum, 2)
 DIN_KW = (KW - DateSerial(Year(KW), 1, -6)) \ 7
End Function
```

Mittels Excelformeln

Hi;

wenn du nur aus der Jahrezahl die Anzahl der Kalenderwochen berechnen
willst hier ein Beispiel;

	A	B
920	2008	52
921	2009	53

B920=KÜRZEN((DATUM(A920;12;28)-WOCHENTAG(DATUM(A920;12;28);2)-DATUM(JAHR(DATUM(A920;12;28)+4-WOCHENTAG(DATUM(A920;12;28);2));1;-10))/7)
B921 =KÜRZEN((DATUM(A921;12;28)-WOCHENTAG(DATUM(A921;12;28);2)-DATUM(JAHR(DATUM(A921;12;28)+4-WOCHENTAG(DATUM(A921;12;28);2));1;-10))/7)

Monatsname aus Monatszahl

```
msgbox monthname(month(now()))
```

Monatsende

```
Sub Test()
    Dim DaDatum As Date
    DaDatum = "12.12.04"
    MsgBox "Monatsende " & DateSerial(Year(DaDatum), Month(DaDatum) + 1, 1) - 1
End Sub
```

Ostern berechnen und die übrigen beweglichen Feiertage

1) Direkt in Excel

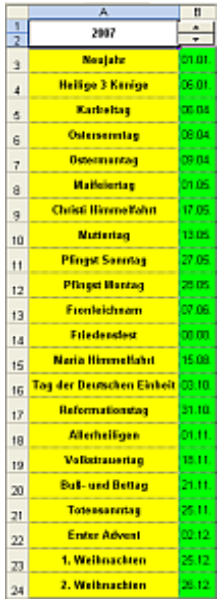
Es gibt Feiertage, welche jedes Jahr an einem festen Datum stattfinden. Des Weiteren gibt es welche, wo das Datum wechselt. Diese Feiertage zu ermitteln soll hier unser Problem sein. Es muss aber gesagt werden, dass es gar keines ist. Im Großen und Ganzen benötigen wir zwei Formeln. Von diesen errechneten Tagen, die uns die Formeln liefern, hängen alle anderen Feiertage ab. Ist Euch das schon einmal aufgefallen? Mir ist das erst bewusst, seit ich meinen ersten Kalender mit Feiertagen erstellt hatte. Wir benötigen also zwei Formeln für die Feiertage - und welche? Ein Feiertag nach dem sich viele bewegliche Feiertage richten ist der Ostersonntag. Die Formel zur Ermittlung lautet, sofern in der Zelle A1 die aktuelle Jahreszahl steht: *=DM((TAG(MINUTE($A$1/38)/2+55)&".4."&$A$1)/7;)*7-6* .

Ich habe diese Formel nicht erstellt. Hand aufs Herz - wie diese Formel aufgebaut ist, dass kann ich Euch nicht erklären. Es gibt sicherlich eine Erklärung dafür, wie man den Ostersonntag und sonstige Tage berechnet, aber das gehört eigentlich nicht hier hin. Deshalb setze ich eine ganz kurze Erläuterung, zumindest für den Ostersonntag, ans Ende dieses Textes.

Die zweite Formel, welche wir benötigen, errechnet den ersten Advent. Auch von dieser Formel leitet man einige Feiertage ab. Sie lautet unter Bezugnahme auf die aktuelle Jahreszahl in Zelle A1:

=DATUM($A$1;12;25)-WOCHENTAG(DATUM($A$1;12;25);2)-21.

Wenn Ihr Euere Jahreszahl in einer anderen Zelle stehen habt, so müsst Ihr die Formel anpassen. Überall wo $A$1 steht muss der Buchstabe und die Zahl angepasst werden. Für unser Beispiel möchte ich aber, dass Ihr Euch eine Tabelle wie im Bild gezeigt anlegt. Die Feiertage braucht Ihr nicht eintragen, denn dafür setzen wir Formeln ein.



	A	B
1	2007	
2		
3	Neujahr	01.01.
4	Heilige 3 Könige	06.01.
5	Karfreitag	06.04.
6	Ostersonntag	08.04.
7	Ostermontag	09.04.
8	Maifeiertag	01.05.
9	Christi Himmelfahrt	17.05.
10	Muttertag	13.05.
11	Pfingst Sonntag	27.05.
12	Pfingst Montag	28.05.
13	Fronleichnam	07.06.
14	Friedensfest	08.08.
15	Maria Himmelfahrt	15.08.
16	Tag der Deutschen Einheit	03.10.
17	Reformationstag	31.10.
18	Allerheiligen	01.11.
19	Volkstrauertag	15.11.
20	Buß- und Bettag	21.11.
21	Totensonntag	25.11.
22	Erster Advent	02.12.
23	1. Weihnachten	25.12.
24	2. Weihnachten	26.12.

Was Ihr in die Zelle B6 eintragt, sollte Euch jetzt eigentlich schon klar sein. In Zelle A6 steht der Ostersonntag, also gehört in die Zelle B7 die Formel dazu und das wäre unsere Formel *=DM((TAG(MINUTE($A$1/38)/2+55)&".4."&$A$1)/7;)*7-6*

In Zelle B22 könnt Ihr somit auch schon einmal unsere zweite Formel

=DATUM($A$1;12;25)-WOCHENTAG(DATUM($A$1;12;25);2)-21

eintragen.

Nun, da wir unsere wichtigsten Formeln eingetragen haben, wird es Zeit zwischen festen und beweglichen Feiertagen zu unterscheiden.

Die beweglichen Feiertage, welche sich nach Ostern richten, sind:

Name des Feiertages	Verhältnis zum Ostersonntag	Formel
Karfreitag	2 Tage zuvor	=B6-2
Ostermontag	1 Tag danach	=B6+1
Christi Himmelfahrt	39 Tage danach	=B6+39
Pfingstsonntag	49 Tage danach	=B6+49
Pfingstmontag	50 Tage danach	=B6+50
Fronleichnam	60 Tage danach	=B6+60

Nach dem ersten Advent richten sich diese Feiertage:

Name des Feiertages	Verhältnis zum 1. Advent	Formel
Totensonntag	7 Tage davor	=B22-7
Buß – und Bettag	11 Tage davor	=B22-11
Volkstrauertag	14 Tage davor	=B22-14

Die restlichen Feiertage auf unserer Liste sind somit feste Feiertage. Um diese in die Zellen einzutragen nutzen wir die Formel *=DATUM()* . Diese wird in unserem Fall wie folgt aufgebaut: *=DATUM(A1;MONAT,TAG).* Warum der Bezug auf Zelle A1? Weil wir doch hier unsere Jahreszahl eingetragen haben auf welche wir uns in der Formel beziehen. Um das noch einmal zu verdeutlichen wollen wir uns noch einmal die Formel für den ersten Weihnachtstag anschauen. Wir holen die Jahreszahl aus A1; darauf folgt der Monat (alles mit Semikolon getrennt) und erst dann der Tag. Die Formel würde also =Datum(A1;12;25) lauten.

Hier kommen jetzt die Daten für die festen Feiertage:

Name des Feiertages	Datum	Formel
Neujahr	01.01.	=Datum(A1;1;1)
Heilige 3 Könige	06.01.	=Datum(A1;1;6)
Maifeiertag	01.05.	=Datum(A1;5;1)
Friedensfest	08.08.	=Datum(A1;8;8)
Mariä Himmelfahrt	15.08.	=Datum(A1;8;15)
Tag der Deutschen Einheit	03.10.	=Datum(A1;10;3)
Reformationstag	31.10.	=Datum(A1;10;31)
Allerheiligen	01.11.	=Datum(A1;11;1)
1. Weihnachten	25.12.	=Datum(A1;12;25)
2. Weihnachten	26.12.	=Datum(A1;12;26)
Silvester	31.12.	=Datum(A1;12;31)

Wenn Ihr diese Daten jetzt alle in Euere Tabelle eingetragen habt, dann habt Ihr Euch einen flexiblen Feiertagskalender gebastelt.

Erläuterung: Wie fällt Ostern?

Ostern fällt immer zwischen den 22. März und dem 25. April. Ausschlaggebend ist hier der erste Sonntag nach dem ersten Vollmond nach Frühlingsanfang. An diesem Sonntag feiern wir das Osterfest. Der große deutsche Mathematiker Gauß hat dazu eine Berechnungsformel im 19. Jahrhundert entwickelt, die in der von uns hier eingesetzten ihren Niederschlag gefunden hat.

2) Mittels VBA

MEINE KURZE VARIANTE

```
Sub Bewegliche_Feiertage()

Dim OSTERN As Date
Dim D As Integer
Dim JAHR As Integer

JAHR = InputBox("Welches Jahr : " & Year(Now), "Jahr")

D = (((255 - 11 * (JAHR Mod 19)) - 21) Mod 30) + 21
OSTERN = DateSerial(JAHR, 3, 1) + D + (D > 48) + 6 - ((JAHR + JAHR \ 4 + D + (D > 48) + 1) Mod 7)

MsgBox "Karfreitag ist am " & OSTERN - 2
MsgBox "Ostermontag ist am " & OSTERN + 1
MsgBox "Pfingstmontag ist am " & OSTERN + 50
MsgBox "Fronleichmann ist am " & OSTERN + 60
MsgBox "Christi Himmelfahrt ist am " & OSTERN + 39


End Sub
```

LANGE VARIANTE 1

```
'Funktion zur Berechnung des Ostersonntags

Function Ostern(JahresZahl As Variant)

   Dim A, B, C, D, E, TagesZahl As Integer

   A = JahresZahl Mod 19                    ' Formel nach Gauß
   B = JahresZahl Mod 4                     ' Werte für den Zeitraum
   C = JahresZahl Mod 7                     ' 1900 - 2099
   D = (19 * A + 24) Mod 30                  ' M=24
   E = (2 * B + 4 * C + 6 * D + 5) Mod 7          ' N=5
   TagesZahl = 22 + D + E                    ' Ermittlung des Tages

   If TagesZahl > 31 Then                   ' März oder April ?

      TagesZahl = TagesZahl - 31               ' wenn April dann 31 Tage abziehen

      If TagesZahl = 26 Then TagesZahl = 19        ' wenn Ostern am 26.April
                                           ' dann immer 19.April
                                           ' 1. Ausnahme

      If (TagesZahl = 25 And D = 28 And A > 10) Then TagesZahl = 18
                                           ' 2. Ausnahme

     Ostern = DateSerial(JahresZahl, 4, TagesZahl)  ' Ostern im April
   Else
     Ostern = DateSerial(JahresZahl, 3, TagesZahl)  ' Ostern im März
   End If

End Function

'   Rosenmontag: 48 Tage vor Ostern
'
Function Rosenmontag(JahresZahl)
   Rosenmontag = Ostern(JahresZahl) - 48
End Function
'
'   Faschingsdienstag: 47 Tage vor Ostern
'
Function Fastnacht(JahresZahl)
   Fastnacht = Ostern(JahresZahl) - 47
End Function
'
'   Aschermittwoch: 46 Tage vor Ostern
'
Function Aschermittwoch(JahresZahl)
   Aschermittwoch = Ostern(JahresZahl) - 46
End Function
```

```
'
'   Karfreitag: 2 Tage vor Ostern
'
Function Karfreitag(JahresZahl)
    Karfreitag = Ostern(JahresZahl) - 2
End Function
'
'   Ostermontag: 1 Tag nach Ostern
'
Function Ostermontag(JahresZahl)
    Ostermontag = Ostern(JahresZahl) + 1
End Function
'
'   Christi Himmelfahrt: 39 Tage nach Ostern
'
Function ChrHimmelfahrt(JahresZahl)
    ChrHimmelfahrt = Ostern(JahresZahl) + 39
End Function
'
'Pfingstsonntag: 49 Tage nach Ostern
Function Pfingsten(JahresZahl)
    Pfingsten = Ostern(JahresZahl) + 49
End Function
'
'Pfingstmontag: 50 Tage nach Ostern
Function Pfingstmontag(JahresZahl)
    Pfingstmontag = Ostern(JahresZahl) + 50
End Function
'
'Fronleichnam: 60 Tage nach Ostern
Function Fronleichnam(JahresZahl)
    Fronleichnam = Ostern(JahresZahl) + 60
End Function
'
'Neujahr (1.Jan)
Function Neujahr(JahresZahl)
    Neujahr = DateSerial(JahresZahl, 1, 1)
End Function
'
'Heilige 3 Könige (6.Jan)
Function DreiKönig(JahresZahl)
    DreiKönig = DateSerial(JahresZahl, 1, 6)
End Function
'
'Tag der Arbeit (1.Mai)
Function TagDArbeit(JahresZahl)
    TagDArbeit = DateSerial(JahresZahl, 5, 1)
End Function
'
```

```
'Maria Himmelfahrt (15.Aug)
Function MHimmelfahrt(JahresZahl)
    MHimmelfahrt = DateSerial(JahresZahl, 8, 15)
End Function
'
'Tag der deutschen Einheit (3.Okt)
Function DtschEinheit(JahresZahl)
    DtschEinheit = DateSerial(JahresZahl, 10, 3)
End Function
'
' Reformationstag (31.Okt)
Function Reformation(JahresZahl)
    Reformation = DateSerial(JahresZahl, 10, 31)
End Function
'
'Allerheiligen (1.Nov)
Function Allerheiligen(JahresZahl)
    Allerheiligen = DateSerial(JahresZahl, 11, 1)
End Function
'
'Heilig Abend (24.Dez)
Function Weihnacht(JahresZahl)
    Weihnacht = DateSerial(JahresZahl, 12, 24)
End Function
'
' 1.Weihnachtsfeiertag (25.Dez)
Function Weihnacht1(JahresZahl)
    Weihnacht1 = DateSerial(JahresZahl, 12, 25)
End Function
'
'2.Weihnachtsfeiertag (26.Dez)
Function Weihnacht2(JahresZahl)
    Weihnacht2 = DateSerial(JahresZahl, 12, 26)
End Function
'
'Silvester (31.Dez)
Function Silvester(JahresZahl)
    Silvester = DateSerial(JahresZahl, 12, 31)
End Function
'
'Advent
Function Advent(JahresZahl)
    Const Sonntag = 1
    If WeekDay(DateSerial(JahresZahl, 12, 24)) = Sonntag Then
        Advent = DateSerial(JahresZahl, 12, 3)
    Else
        Advent = DateSerial(JahresZahl, 12, 24) - WeekDay(DateSerial(JahresZahl, 12, 24))
        Advent = Advent - 20
    End If
```

```
End Function
'
'   Buß- und Bettag (11 Tage vor dem 1. Advent)
Function BetTag(JahresZahl)
   BetTag = Advent(JahresZahl) - 11
End Function
```

LANGE VARIANTE 2

```
Public Function Feiertage(intYear As Integer) As Variant
Dim varDates(13, 1) As Variant
Dim dEaster As Date

dEaster = Easter(intYear)

varDates(0, 0) = DateSerial(intYear, 1, 1)
varDates(0, 1) = "Neujahr"

varDates(1, 0) = DateSerial(intYear, 1, 6)
varDates(1, 1) = "Dreikönig"

varDates(2, 0) = dEaster - 2
varDates(2, 1) = "Karfreitag"

varDates(3, 0) = dEaster + 1
varDates(3, 1) = "Ostermontag"

varDates(4, 0) = DateSerial(intYear, 5, 1)
varDates(4, 1) = "Tag der Arbeit"

varDates(5, 0) = dEaster + 39
varDates(5, 1) = "Christi Himmelfahrt"

varDates(6, 0) = dEaster + 50
varDates(6, 1) = "Pfingstmontag"

varDates(7, 0) = dEaster + 60
varDates(7, 1) = "Fronleichnam"

varDates(8, 0) = DateSerial(intYear, 10, 3)
varDates(8, 1) = "Tag der Einheit"

varDates(9, 0) = DateSerial(intYear, 11, 1)
varDates(9, 1) = "Allerheiligen"

varDates(10, 0) = DateSerial(intYear, 12, 24)
varDates(10, 1) = "Heiligabend"

varDates(11, 0) = DateSerial(intYear, 12, 25)
varDates(11, 1) = "1. Weihnachtstag"
```

```
varDates(12, 0) = DateSerial(intYear, 12, 26)
varDates(12, 1) = "2. Weihnachtstag"

varDates(13, 0) = DateSerial(intYear, 12, 31)
varDates(13, 1) = "Silvester"

Feiertage = varDates

End Function



Private Function Easter(Year As Integer) As Date
Dim D As Integer
D = (((255 - 11 * (Year Mod 19)) - 21) Mod 30) + 21
Easter = DateSerial(Year, 3, 1) + D + (D > 48) + 6 - _
  ((Year + Year \ 4 + D + (D > 48) + 1) Mod 7)
End Function



Sub test()
Dim dummy As Variant

dummy = Feiertage(2006)

Range("A1:B" & UBound(dummy, 1) + 1) = dummy
End Sub
```

Text-Datum umwandeln in echtes Datum

CDate(Format(""1.Oktober", "dd.mm.yyyy"))

Uhrzeit von Datum entfernen

Enthält eine Excelzelle neben dem Datum auch eine Uhrzeit, so kommt diese in eine als DATE-Typ definierte Variable und beim Vergleich mit einer anderen Zelle, in der eine andere Uhrzeit steht, sind die beiden Datums-Werte nicht gleich, selbst wenn der Tag selbst gleich ist: 1.1.2020 10:00:00 ist eben nicht gleicht mit 1.1.2020 11:00:00

Zum reinen Vergleichen von den Tages-Datumswerten müssen wir die Uhrzeit wegschneiden - da Tage in Excel Ganzzahlen sind und die Stunden und Minuten und Sekunden die Nachkommastellen sind, reicht es mit der INT-Funktion zu arbeiten, die die Kommastellen wegschneidet

Dim DATUM As Date ' Datum

Uhrzeit in Zelle standardisiert ausgeben

Dies ist der Code von 2013 Dr. Vogels Stundenliste - und zwar ist der Code nicht in einem einzelnen Tabellenblatt sondern gleich im Hauptblatt DIESEARBEITSMAPPE

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)

Dim A As String ' Target.Adress

On Error Resume Next ' (wenn man mehrere Zellen markiert und löscht, kommt es zu Fehler)

' wenn mehr als eine Zelle markiert ist
If Selection.Columns.Count > 1 Or Selection.Rows.Count > 1 Then
   VBA_SCHREIBT = 0
   Exit Sub
End If

' wenn das automatische Zeitformat deaktiviert ist in den Paramtern
If Sheets("Parameter").Range("E21") = False Then Exit Sub

' wenn VBA-Code (z.B: der An- und Abmeldetasten) Zeiten schreibt, müssen diese nicht formatiert werden
' oder auch hier: wenn die SheetChange-Prozedur Zellinhalt ändert, dann ruft sie sich ja sofort selbst wieder auf - und diese Zirkelläufe sollen sofort beendet werden
If VBA_SCHREIBT = 1 Then Exit Sub

' hier merken wir uns nun genau in dieser Variable, dass wir den Code-Bereich der SheetChange-Prozedur betreten, der Zellen verändert
' und durch das Ändern der Variable VBA_SCHREIBT auf 1, wird festgelegt, dass es zu keinen Zirkelläufen kommen soll
VBA_SCHREIBT = 1


' Kontrolle, ob wir überhaupt in einem Monats-Tabellenblatt sind - denn diese beginnen alle mit einer Zahl
If Val(Left(ActiveSheet.Name, 1)) > 0 Then ' wenn aktuelles Tabellenblatt mit einer Zahl beginnt - daher eines der 12 Monatstabellen ist

   ' wenn in der Spalte B irgendwo ein OK gesetzt wurde, sind alle Kommentarzellen (Spalte W-AA) oberhalb dieser Zeile zu sperren
   If Target.Column = "2" And UCase(Target.Value) = "OK" Then
      Range("W14:AA" & Target.Row).Locked = True
   End If

   ' Einschränkung der Zeitformatierung auf die Spalten E, F, G, H, K und L
   A = Mid(Target.Address, 2, 1) ' die Spalte
   If A = "E" Or A = "F" Or A = "G" Or A = "H" Or A = "K" Or A = "L" Then  ' wenn ein Wert in den Spalten E-H oder K-L geändert wurde

      ' Einschränkung nur auf die Zellen der Zeile 14 bis 44
      A = Target.row ' Zeilennummern
```

```
    If A >= 14 And A <= 44 Then ' wenn ein Wert zwischen den Zeilen 13 und 35 geändert wurde

        ' Die Eingaben können 3 Formate haben:
        ' 1.) 1, 2, 3, 12, 24 ... wenn man nur die Stunden eingibt - sie kommen intern als Werte >=1 an und werden einfach um ":00" ergänzt
        ' 2.) 100, 130, 200, 300, 1200, 2400 ... wenn man die Stunden und Minuten ohne Doppelpunkt eingibt - sie kommen auch als Wert >1 ein und müssen nur einen Doppelpunkt vor den letzten beiden Stellen eingefügt erhalten
        ' 3.) 0:30, 1:00, 1:30, 12:00, 12:30 ... dies sind Zeiteingaben mit Doppelpunkt - und sie erhalten alle einen Wert unter 1 (24:00 wäre genau 1)

        ' zu Testzwecken kann man hier den Wert und die Länger der Eingabe anzeigen lassen
        ' MsgBox "Wert " & Target.Value & " - Länge " & Len(Target.Value)

        ' zwischen den drei Eingabearten gibt es nur eine einzige Kollision: die Eingabe von 1 und von 24:00 hat immer den Wert 1 - aber die Eingaben haben eine verschiedene Länge:

        ' wenn man 24 oder 2400 eingegeben hat
        If Target.Value = 24 Or Target.Value = 2400 Then
           Target.Value = "24:00"
           VBA_SCHREIBT = 0
           Exit Sub
        End If
        ' wenn man 24:00 eingegeben hat (interner Wert 1)
        If Target.Value = 1 And Len(Target.Value) > 1 Then
           Target.Value = "24:00"
           VBA_SCHREIBT = 0
           Exit Sub
        End If
        ' wenn man 12 oder 1200 eingegeben hat
        If Target.Value = 12 Or Target.Value = 1200 Then
           Target.Value = "12:00"
           VBA_SCHREIBT = 0
           Exit Sub
        End If
        ' wenn man 12:00 eingegeben hat
        If Target.Value = 0.5 Then
           Target.Value = "12:00"
           VBA_SCHREIBT = 0
           Exit Sub
        End If
        ' wenn man 1 eingegeben hat
        If Target.Value = 1 And Len(Target.Value) = 1 Then
           Target.Value = "1:00"
           VBA_SCHREIBT = 0
           Exit Sub
        End If
```

```
        ' Eingaben, deren Wert > als 1 sind Eingaben ohne Doppelpunkt (da diese immer einen Wert von 0-1 haben) - zB 101 (für 1:01) oder 1200 etc
        If Val(Target.Value) > 1 Then

            ' 4-ziffrige Werte gehen von 1000 bis 2400
            If Len(Target.Value) = 4 Then
                If Left(Target.Value, 1) = "1" Or Left(Target.Value, 1) = "2" Then
                    Target.Value = Left(Target.Value, 2) & ":" & Right(Target.Value, 2)
                    GoTo FERTIG
                End If
            End If

            ' 3-ziffrige Werte gehen von 100 bis 959
            If Len(Target.Value) = 3 Then
                Target.Value = Left(Target.Value, 1) & ":" & Right(Target.Value, 2)
            End If

            ' 2-ziffrige Werte gehen von 10-24 und stellen Stunden dar
            If Len(Target.Value) = 2 Then
                Target.Value = Left(Target.Value, 2) & ":" & "00"
            End If

            ' 1-ziffrige Werte gehen von 1-9 und sind auch Stunden
            If Len(Target.Value) = 1 Then
                Target.Value = Left(Target.Value, 1) & ":" & "00"
            End If
        End If

    End If
  End If
End If

FERTIG:
  VBA_SCHREIBT = 0

End Sub
```

Uhrzeit in Zelle in Userform-Textfeld eintragen

Excel hat ja keine echte Uhrzeiten, sondern rechnet in Tagen und Uhrzeiten sind Kommastellen, die nur als Formatwunsch in Stunden angegeben werden können. Darum ist z.B. auch schon mal das Aufrunden von 10:00-9:00 nicht 1 Stunde, sondern 2, weil Excel 1,000000001 als Ergebnis hat und dies aufrundet. (Daher einfach beim Berechnen von Stundenperioden vor dem Aufrunden 0,000005 abziehen)

Möchte man eine Zeit, die in einer Zelle steht, in ein Textfeld (z.B. einer Userform) übernehmen, so sind 12:00 immer 0,5

Abhilfe schafft hier die Format-Funktion (ist quasi die VBA-Variante der Text-Formelfunktion)

```
TextBox1.Value = Format(Sheets(1).Range("A1").Value, "hh:mm")
```

Wenn es um mehr als 24 h gehen soll:

Problem ist ich brauche ein Format was mehr als 24 Stunden anzeigt.
nun funktioniert alles bis hier. An dieser stelle sollen die werte in der Form angezeigt werden und bei Fehlern manuell verändert werden funkt auch super nur zeigt er mir egal was ich mache immer Komma werte an.

bei textbox7 sind einige versuche zu sehen aber es funkt irgend wie nicht.

Code:

```
Private Sub UserForm Activate()
    Dim aa As Time
    aa = Sheets(1).Cells(24, 3).Value '.NumberFormat = "[hh]:mm")
    TextBox1.Text = Sheets(1).Cells(8, 2).Value
    TextBox2.Text = Sheets(1).Cells(9, 2).Value
    TextBox3.Text = Sheets(1).Cells(10, 2).Value
    TextBox4.Text = Sheets(1).Cells(11, 2).Value
    TextBox5.Text = Sheets(1).Cells(12, 2).Value
    TextBox6.Text = Sheets(1).Cells(13, 2).Value

    'TextBox7 = aa
    'TextBox7.Value = (Sheets(1).Cells(24, 3).NumberFormat = "[hh]:mm")
    'TextBox7.Value = Format(Sheets(1).Cells(24, 3).Value, NumberFormat:="[hh]:mm:ss")
    TextBox7 = Format(aa, "[hh]:mm")

    TextBox8.Text = Sheets(1).Cells(24, 4).Value
    TextBox9.Text = Sheets(1).Cells(14, 2).Value
    'TextBox10.Text = Sheets(1).Cells(14, 2).Value
    TextBox11.Text = Sheets(1).Cells(22, 3).Value
End Sub
```

Ok habe das Problem dann doch gefunden.

Code:

```
    TextBox7.Text = Sheets(1).Cells(24, 3).Text 'hier muss .text sein sonst falsches format
```

Vergleich von Datumswerten

zum Vergleichen ob ein Datum früher oder später ist als ein anderes kann man direkt mit dem Datum - wenn die Variable vom Typ DATE ist, rechnen. Man muss nicht in die von den Tabellenblättern gewohnte Datumszahl "312123" umwandeln und die Datums-Tageszahl vergleichen.

Werktag eines Datums ermitteln

msgbox Weekday(Date, vbMonday)

Weekday ermittelt den Werktag - Date ist das heutige System-Datum und vbmonday gibt der weekdayfunktion den Hinweis, dass die Woche mit Montag beginnt - Montag ist also 1, Dienstag 2 ….

Werktage eines Monats + Schaltjahr Februar-Tage

von mir erstellt für Porschetool

```
Sub CALCULATE_WORKING_DAYS()

' This procedure calculates the working days (Monday until Friday) of the month

Dim Integer_month As Integer
Dim Integer_lastday As Integer
Dim Integer_year As Integer
Dim I As Integer ' counter
Dim Integer_workingdays As Integer

Integer_month = Val(Left(Sheets("employees").Range("I6"), 2))
Integer_year = Val(Right(Sheets("employees").Range("I6"), 4))

' what is the lastday in this month

If Integer_month = 1 Then Integer_lastday = 31
If Integer_month = 2 Then
   If Integer_year Mod 4 = 0 Then
      Integer_lastday = 29
     Else
      Integer_lastday = 28
   End If
End If
If Integer_month = 3 Then Integer_lastday = 31
If Integer_month = 4 Then Integer_lastday = 30
If Integer_month = 5 Then Integer_lastday = 31
```

```
If Integer_month = 6 Then Integer_lastday = 30
If Integer_month = 7 Then Integer_lastday = 31
If Integer_month = 8 Then Integer_lastday = 31
If Integer_month = 9 Then Integer_lastday = 30
If Integer_month = 10 Then Integer_lastday = 31
If Integer_month = 11 Then Integer_lastday = 30
If Integer_month = 12 Then Integer_lastday = 31

' how many working days are between the first and the last day of this periode

For I = 1 To Integer_lastday

   If Weekday(I & "." & Integer_month & "." & Integer_year) > 1 And Weekday(I & "." & Integer_month & "." & Integer_year) < 7 Then Integer_workingdays
= Integer_workingdays + 1

Next I

Sheets("employees").Range("J6") = Integer_workingdays

End Sub
```

Werktage zwischen 2 Datumswerten berechnen

```
Function fncArbeitstage(datStart As Date, datEnde As Date) As Integer

   'Deklaration
   Dim intWochen As Integer
   Dim intTage As Integer
   Dim varDatum As Variant

   'Anzahl ganze Wochen zwischen den Daten
   intWochen = DateDiff("w", datStart, datEnde)

   'Anzahl Arbeitstage zwischen den Daten (5 Tage pro Woche)
   intTage = intWochen * 5

   ' Abzug für den ersten Tag, der ja nicht mitgezählt werden soll, da wir die Werktage ZWISCHEN zwei Terminen wollen (der erste Tag zählt also nicht, der
letzte Tag schon)
   ' BSP Mo. 5.8.13 bis Mi 7.8.2013: Montag wird nicht gezählt, Di+Mi=2 Tage
   abzug = 1

   'Datum, das nach der letzten vollen Woche im Datumsbereich liegt
```

```
    varDatum = DateAdd("ww", intWochen, datStart)
    ' Wenn es Sa oder So ist, dann erhöhe zum nächsten Werktag Montag
    If Weekday(varDatum) = vbSaturday Then
        varDatum = DateAdd("d", 2, varDatum)
        abzug = 0 ' da wir den ersten Tag nun schon verlassen haben, braucht es keinen Abzug mehr
    End If
    If Weekday(varDatum) = vbSunday Then
        varDatum = DateAdd("d", 1, varDatum)
        abzug = 0 ' da wir den ersten Tag nun schon verlassen haben, braucht es keinen Abzug mehr
    End If

    'Für restliche Tage:
    'Daten, die nicht auf ein Wochenende fallen dazuzählen
    Do While varDatum <= datEnde

        If Weekday(varDatum) <> vbSunday _
        And Weekday(varDatum) <> vbSaturday Then
            intTage = intTage + 1
        End If
        'Nächstes zuprüfendes Datum
        varDatum = DateAdd("d", 1, varDatum)
    Loop

    'Rückgabe
    fncArbeitstage = intTage - abzug

 End Function

Sub test()

MsgBox fncArbeitstage("05.08.2013", "07.08.2013")
MsgBox fncArbeitstage("04.08.2013", "06.08.2013")


End Sub
```

Werktage zu Datum dazurechnen

```
Sub test()
    MsgBox AddArbeitstage("3.6.2013", 10)
End Sub
Public Function AddArbeitstage(datStart As Date, intDauer As Integer) As Date
```

```
    'Deklaration
    Dim intWochen As Integer
    Dim intTage As Integer
    Dim varDatum As Variant


    AddArbeitstage = datStart ' Das Enddatum beginnt mit dem Startdatum

For d = 1 To intDauer
    AddArbeitstage = AddArbeitstage + 1
    If Weekday(AddArbeitstage, vbMonday) = 6 Then AddArbeitstage = AddArbeitstage + 2
    If Weekday(AddArbeitstage, vbMonday) = 7 Then AddArbeitstage = AddArbeitstage + 1
Next d

 End Function
```

Wochentag aus Datum erzeugen

```
Public Function Datum(Datumstag As Date)

Tag = Weekday(Datumstag, vbMonday)

Select Case Tag
        Case 1
            B.Cells(BZ, 13) = "Mo " & B.Cells(BZ, 13)
        Case 2
            B.Cells(BZ, 13) = "Di " & B.Cells(BZ, 13)
        Case 3
            B.Cells(BZ, 13) = "Mi " & B.Cells(BZ, 13)
        Case 4
            B.Cells(BZ, 13) = "Do " & B.Cells(BZ, 13)
        Case 5
            B.Cells(BZ, 13) = "Fr " & B.Cells(BZ, 13)
        Case 6
            B.Cells(BZ, 13) = "Sa " & B.Cells(BZ, 13)
        Case 7
            B.Cells(BZ, 13) = "So " & B.Cells(BZ, 13)

End Select
```

Zeiteingaben ohne Doppelpunkt ermöglichen (7 wird 7:00 , 123 wird 1:23)

Wir arbeiten mit einer Variable VBA_SCHREIBT - diese bitte vorher Public setzen !
Und: Es gibt ja die Workbook und die Worksheet_Change-Variante

```
Public VBA_SCHREIBT

Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
' oder: Private Sub Worksheet_Change(ByVal Target As Range)

Dim A As String ' Target.Adress

On Error Resume Next ' (wenn man mehrere Zellen markiert und löscht, kommt es zu Fehler)

' wenn mehr als eine Zelle markiert ist
If Selection.Columns.Count > 1 Or Selection.Rows.Count > 1 Then
   VBA_SCHREIBT = 0
   Exit Sub
End If

' wenn das automatische Zeitformat deaktiviert ist in den Paramtern
If Sheets("Parameter").Range("E21") = False Then Exit Sub

' wenn VBA-Code (z.B: der An- und Abmeldetasten) Zeiten schreibt, müssen diese nicht formatiert werden
' oder auch hier: wenn die SheetChange-Prozedur Zellinhalt ändert, dann ruft sie sich ja sofort selbst wieder auf - und diese Zirkelläufe sollen sofort beendet werden
If VBA_SCHREIBT = 1 Then Exit Sub

' hier merken wir uns nun genau in dieser Variable, dass wir den Code-Bereich der SheetChange-Prozedur betreten, der Zellen verändert
' und durch das Ändern der Variable VBA_SCHREIBT auf 1, wird festgelegt, dass es zu keinen Zirkelläufen kommen soll
VBA_SCHREIBT = 1


' Kontrolle, ob wir überhaupt in einem Monats-Tabellenblatt sind - denn diese beginnen alle mit einer Zahl
If Val(Left(ActiveSheet.Name, 1)) > 0 Then ' wenn aktuelles Tabellenblatt mit einer Zahl beginnt - daher eines der 12 Monatstabellen ist

   ' wenn in der Spalte B irgendwo ein OK gesetzt wurde, sind alle Kommentarzellen (Spalte W-AA) oberhalb dieser Zeile zu sperren
   If Target.Column = "2" And UCase(Target.Value) = "OK" Then
      Range("W14:AA" & Target.Row).Locked = True
   End If
```

```
' Einschränkung der Zeitformatierung auf die Spalten E, F, G, H, K und L
A = Mid(Target.Address, 2, 1) ' die Spalte
If A = "E" Or A = "F" Or A = "G" Or A = "H" Or A = "K" Or A = "L" Then  ' wenn ein Wert in den Spalten E-H oder K-L geändert wurde

    ' Einschränkung nur auf die Zellen der Zeile 14 bis 44
    A = Right(Target.Address, 2) ' Zeilennummern

    If Val(A) >= 14 And Val(A) <= 44 Then ' wenn ein Wert zwischen den Zeilen 13 und 35 geändert wurde

        ' Die Eingaben können 3 Formate haben:
        ' 1.) 1, 2, 3, 12, 24 ... wenn man nur die Stunden eingibt - sie kommen intern als Werte >=1 an und werden einfach um ":00" ergänzt
        ' 2.) 100, 130, 200, 300, 1200, 2400 ... wenn man die Stunden und Minuten ohne Doppelpunkt eingibt - sie kommen auch als Wert >1 ein und müssen nur einen Doppelpunkt vor den letzten beiden Stellen eingefügt erhalten
        ' 3.) 0:30, 1:00, 1:30, 12:00, 12:30 ... dies sind Zeiteingaben mit Doppelpunkt - und sie erhalten alle einen Wert unter 1 (24:00 wäre genau 1)

        ' zu Testzwecken kann man hier den Wert und die Länger der Eingabe anzeigen lassen
        ' MsgBox "Wert " & Target.Value & " - Länge " & Len(Target.Value)

        ' zwischen den drei Eingabearten gibt es nur eine einzige Kollision: die Eingabe von 1 und von 24:00 hat immer den Wert 1 - aber die Eingaben haben eine verschiedene Länge:

        ' wenn man 24 oder 2400 eingegeben hat
        If Target.Value = 24 Or Target.Value = 2400 Then
            Target.Value = "24:00"
            VBA_SCHREIBT = 0
            Exit Sub
        End If
        ' wenn man 24:00 eingegeben hat (interner Wert 1)
        If Target.Value = 1 And Len(Target.Value) > 1 Then
            Target.Value = "24:00"
            VBA_SCHREIBT = 0
            Exit Sub
        End If
        ' wenn man 12 oder 1200 eingegeben hat
        If Target.Value = 12 Or Target.Value = 1200 Then
            Target.Value = "12:00"
            VBA_SCHREIBT = 0
            Exit Sub
        End If
        ' wenn man 12:00 eingegeben hat
        If Target.Value = 0.5 Then
            Target.Value = "12:00"
            VBA_SCHREIBT = 0
            Exit Sub
```

```
            End If
            ' wenn man 1 eingegeben hat
            If Target.Value = 1 And Len(Target.Value) = 1 Then
               Target.Value = "1:00"
               VBA_SCHREIBT = 0
               Exit Sub
            End If

            ' Eingaben, deren Wert > als 1 sind Eingaben ohne Doppelpunkt (da diese immer einen Wert von 0-1 haben) - zB 101 (für 1:01) oder 1200 etc
            If Val(Target.Value) > 1 Then

               ' 4-ziffrige Werte gehen von 1000 bis 2400
               If Len(Target.Value) = 4 Then
                  If Left(Target.Value, 1) = "1" Or Left(Target.Value, 1) = "2" Then
                     Target.Value = Left(Target.Value, 2) & ":" & Right(Target.Value, 2)
                     GoTo FERTIG
                  End If
               End If

               ' 3-ziffrige Werte gehen von 100 bis 959
               If Len(Target.Value) = 3 Then
                  Target.Value = Left(Target.Value, 1) & ":" & Right(Target.Value, 2)
               End If

               ' 2-ziffrige Werte gehen von 10-24 und stellen Stunden dar
               If Len(Target.Value) = 2 Then
                  Target.Value = Left(Target.Value, 2) & ":" & "00"
               End If

               ' 1-ziffrige Werte gehen von 1-9 und sind auch Stunden
               If Len(Target.Value) = 1 Then
                  Target.Value = Left(Target.Value, 1) & ":" & "00"
               End If
            End If

         End If
      End If
   End If

FERTIG:
   VBA_SCHREIBT = 0

End Sub
```

Zeiten aus einer Zelle auslesen

Ist eine Zeit als Text in einer Zelle, so kann man sie mit

mit der Zellfunktion: =Zeitwert(A1)

mit der VBA-Funktion: CDate(Range("A1")

auslesen

Zeit umwandeln zum Rechnen (6:00 => 6,0)

Angenommen in der aktuellen Zelle, die als Zeit formatiert ist, steht der z.B. 10:00

Ich möchte die Differenz zu jetzt (z.B. genau 16:00:00) wissen:

MsgBox CDbl(Time - ActiveCell) * 24

Ergebnis: 6,0

Ohne CDBL und 24 wäre das Ergebnis: "6:00"
Mit CDBL ohne 24 erhält man den Bruchteil eines Tages: 0,25

FUNKTIONEN EXCELTABELLENFUNKTIONEN IN VBA

Möchte man Tabellenfunktionen, die man normalerweise direkt in die Exceltabellen in den Zellen einträgt, in VBA verwenden, so braucht man dazu die Funktion

```
Application.WorksheetFunction

zB.

Cells(Zeile,Spalte) = Application.Worksheetfunction.SumIf(KriterienBereich,Kriterium, _
Wertebereich)
```

viele Funktionen sind im VBA schon integriert.
Schreib mal in der VBA-Umgebung **Application.WorksheetFunction.**

… nach dem letzten Punkt bekommst du die Funktionen schon vorgeschlagen.

also statt in A1 die Formel = SUMME(B1:E100)
schreibst du Code:
Worksheets("Tabelle1").Cells(1,1) = Application.WorksheetFunction.Sum(Worksheets("Tabelle1").Range("B1:E100")

BEISPIEL

Hallo,

ich versuche vergeblich diese Formel in VBA umzusetzen, sodass ich die ermittelte Anzahl in eine Variable speichern kann:

=SUMMENPRODUKT((A4:A10000<>"")/ZÄHLENWENN(A4:A10000;A4:A10000&""))

Vielen Dank!

```
- gewünschte Formel in Zelle schreiben
- Formelzelle mit Zellcursor markieren
- Tastenkombination Alt+F11 (Vba-Editor) aufgerufen
- wenn im Vba-Editor das Fenster 'Direktbereich' nicht sichtbar ist
```

```
- dann mit Tastenkombination Strg+g sichtbar schalten
- im Direktbereich folgendes Statemant schreiben:
-  ? activecell.formulalocal
-  und Entertaste drücken
- Ergebnis:
  =SUMMENPRODUKT((A4:A10000<>"")/ZÄHLENWENN(A4:A10000;A4:A10000&""))
- Vba-Code für Formel in Zelle 'K1':
  Range("K1").FormulaLocal = "=SUMMENPRODUKT((A4:A10000<>"""")/ZÄHLENWENN(A4:A10000;A4:A10000&""""))
```

so klapt es jetzt bei mir:
Code:

```
Sub test1()
    Dim d As Double, formel As String

    formel = "=SUMPRODUCT((A4:A10000<>"""")/COUNTIF(A4:A10000,A4:A10000&""""))"
    d = Application.Evaluate(formel)
    MsgBox d
End Sub
```

Hallo Lupo,

genau das hatte ich gesucht. Das heißt im Klartext für mich, dass Formeln aus Zellen direkt mit einer eckigen Klammer 1:1 verwendet werden kann.

Klasse genau das habe ich gesucht.

FUNKTIONEN MATHEMATIK

Ausrechnen von Usereingaben mit + - x und /

Beispiel ist aus Simplefibu,wenn User in Betragsfelder (Userform) nicht nur einen Wert eingibt, sondern + und - und / und * verwendet, um Beträge zu verknüpfen. Nachfolgender Code errechnet das Ergebnis

Public Sub ZELLKONVERTER()

```
    ' Dient für die Userforms um Beträge mit Rechenoperationen auszuwerten (zB 2,14+3,14) und wird von dort aus aufgerufen
    Dim i As Integer ' Laufvariable
    Dim RECHNEN As Integer ' merkt sich, ob die Eingabe eine Rechenoperation enhält
    Dim ZEICHEN As String ' zum Zerschneiden des Strings in einzelne Zeichen
    Dim ERGEBNIS As String ' zum Umwandeln von Variable ZELLEKONVERTIERT
    ZELLEKONVERTIERT = ""
    ERGEBNIS = ""
    RECHNEN = 0
    ' Abschneiden eventueller Rechenzeichen (+,-,=) am Ende der Eingabe
    If Right(ZELLEINGABE, 1) = "=" Or Right(ZELLEKONVERTIERT, 1) = "+" Or Right(ZELLEKONVERTIERT, 1) = "-" Then
        ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)
    End If

    ' Leider hat "evaluate"-Befehl einen Fehler - ein "evaulate ("24")" ergibt 90 und nicht 24 =>
    ' evaluate-Befehl nur ausführen, wenn eine Rechenoperation im String enthalten ist
    ' weiters verdaut "evaluate" keine "," - er nimmt nur "."
    ' und zudem löscht Excel beim Übergeben einer Zahl, die ein "." enthält, an eine Variable
    ' diesen Punkt (aus "2.12" wird 212)
    ' netterweise gibt der Evaluate-Befehl, der zwar "." braucht, das Ergebnis korrekt wieder mit "," aus
    ' => Im zu berechnenden String nur dann die "," in "." umwandeln, wenn anschließend ein Evaluate erfolgt


' Wenn Zelleingabe mit einem + o. - endet : hinzufügen von 0, damit gerechnet werden kann - aber das Ergebnis sich nicht ändert

    If Right(ZELLEINGABE, 1) = "+" Or Right(ZELLEINGABE, 1) = "-" Then ZELLEINGABE = ZELLEINGABE + "0"

' Wenn Zelleingabe mit * oder / endet : hinzufügen von 1, damit gerechnet werden kann, aber Ergebnis gleich bleibt

    If Right(ZELLEINGABE, 1) = "*" Or Right(ZELLEINGABE, 1) = "/" Then ZELLEINGABE = ZELLEINGABE + "1"

' Wenn Zelleingabe mit "/0" endet : abschneiden, damit gerechnet werden kann

    If Right(ZELLEINGABE, 2) = "/0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 2)

' Wenn Zelleingabe mit "/0," endet : abschneiden, damit gerechnet werden kann

    If Right(ZELLEINGABE, 3) = "/0," Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 3)
```

```
' Wenn Zelleingabe mit "/0,0" endet : abschneiden, damit gerechnet werden kann

   If Right(ZELLEINGABE, 4) = "/0,0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 4)

' Wenn Zelleingabe mit " " endet - abschneiden

   If Right(ZELLEINGABE, 1) = " " Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)

' KONTROLLE OB EINE RECHENOPERATION ENTHALTEN IST (nur dann wird Evaluate ausgeführt)
   For i = 1 To Len(ZELLEINGABE)
      ZEICHEN = Mid(ZELLEINGABE, i, 1)
      If ZEICHEN = "+" Or ZEICHEN = "-" Or ZEICHEN = "*" Or ZEICHEN = "/" Then RECHNEN = 1
   Next i

' WENN RECHENOPERATION ENTHALTEN DANN VOR EVALUATE DAS "," in "." umwandeln
   If RECHNEN = 1 Then

      ' Zeichenweise "umschaufeln" in Zielvariable und aus "," einen "." machen
      For i = 1 To Len(ZELLEINGABE)
         If Mid(ZELLEINGABE, i, 1) = "," Then
            ERGEBNIS = ERGEBNIS + "."
         Else
            ERGEBNIS = ERGEBNIS + Mid(ZELLEINGABE, i, 1)
         End If
      Next
      ERGEBNIS = Evaluate(ERGEBNIS)
     Else
      ERGEBNIS = ZELLEINGABE
   End If
  If ERGEBNIS = "" Then ERGEBNIS = "0"
  ERGEBNIS = Str(Round(ERGEBNIS, 2))


 ' Anschließend das Ergebnis, falls mit "." wieder umwandeln auf ","

       For i = 1 To Len(ERGEBNIS)
         If Mid(ERGEBNIS, i, 1) = "." Then
            ZELLEKONVERTIERT = ZELLEKONVERTIERT + ","
         Else
            ZELLEKONVERTIERT = ZELLEKONVERTIERT + Mid(ERGEBNIS, i, 1)
         End If
      Next
```

End Sub

Anzahl der Stellen auslesen

```
Function CountDigits(s As String) As Integer
Dim i
For i = 1 To Len(s)
If Mid(s, i, 1) Like "#" Then
CountDigits = CountDigits + 1
End If
Next i
End Function
```

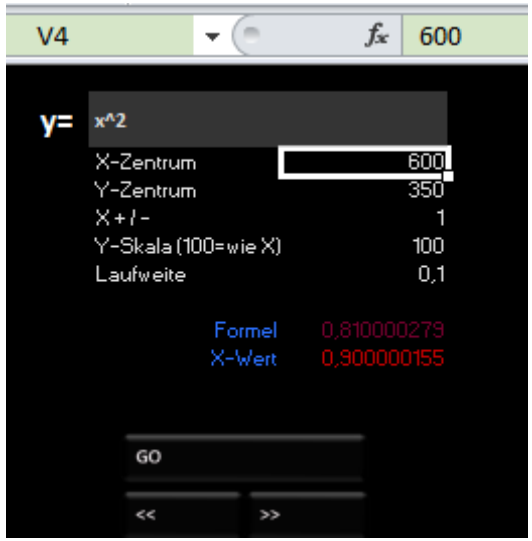
Formeln in Zellen schreiben

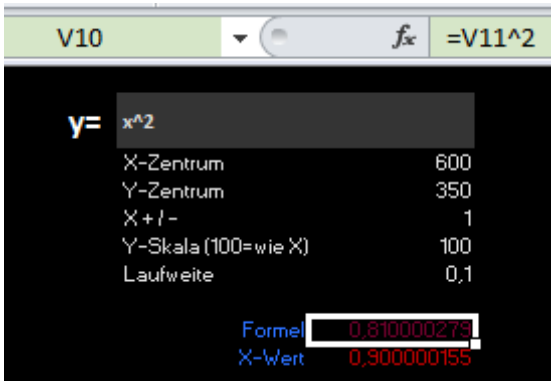
Nachfolgend mein 2D-Funktionen-Zeichner - in der Zelle U2 schreibe ich meine Formel rein und diese möchte ich auswerten.

Dazu nehme ich die Formel und ersetze deren x mit dem tatsächlichen x-Wert.

Diese Formel schreibe ich dann in die Zelle V10.

Ursprünglich hab ich diesen X-Wert direkt in die Formel in V10 geschrieben - aber bei ganz kleinen Werten hat Excel dann das berühmte 9,99999 E-2 drausgemacht und damit kam die Excelformel natürlich nicht zurecht. Lösung: ich schreibe auch den X-Wert interim in eine Zelle (V11) und verweise in der Formel dann auf diese Zelle:





Wichtig war es auch mit .FormulaLocal zu arbeiten. Denn übergab ich z.B. die Wurzel mit SQR, so kam mein deutsches Excel damit nicht zurecht.

```
Sub T_01_Funktionen_46()

' Freie Formeleingabe

Dim X_ZENTRUM ' Mittelpunkt der X-Achse im Tabellenblatt
Dim Y_ZENTRUM ' Mittelpunkt der Y-Achse
Dim X_BEREICH ' der betrachtete X-Bereich im negativen und positiven
Dim LAUFWEITE
Dim Y_SKALA ' wenn 100, dann ist Y-Achse 1:1 gleich mit X, wenn > 100, dann wird Y-Achse komprimiert, wenn <100 wird gezoomt
```

```
Dim X_ALT ' wir zeichnen immer eine Linie zum Punkt der vorigen Berechnung
Dim Y_ALT ' diese beiden Variablen hier enthalten die echten Funktionswerte

Dim X1, X2, Y1, Y2 ' da wir beim Zeichnen nur mit ganzen Zahlen arbeiten können, wandeln wir die Werte um

Dim X As Single

X_ZENTRUM = Range("V4")
Y_ZENTRUM = Range("V5")
X_BEREICH = Range("V6")
LAUFWEITE = Range("V8")
Y_SKALA = Range("V7")

LEEREN

PI = 4 * Atn(1) ' entweder 4xArcustangens oder 3.1415926535897932 fix nehmen
E = 2.71828182845904    ' Eulersche Zahl

'On Error Resume Next

For X = -X_BEREICH To X_BEREICH Step LAUFWEITE

      Range("V11") = X
      'If X < 0 And Abs(X) < 0.000001 Then X = -0.000001
      'If X > 0 And Abs(X) < 0.000001 Then X = 0.000001
      Formel = "=" & Replace(UCase(Range("U2")), "X", "V11")
      Formel = Replace(UCase(Formel), "PI", PI)
      Formel = Replace(UCase(Formel), "E", E)
      'Formel = Replace(UCase(Formel), ",", ".")
      Formel = Replace(UCase(Formel), "SQR", "WURZEL")
      Range("V10").FormulaLocal = Formel
      Y = Range("V10")

      If X > -Range("V6") Then ' sobald wir einen Durchgang schon mal hatten
         X1 = Round(X_ZENTRUM - X_ZENTRUM * X_ALT / X_BEREICH, 0)
         Y1 = Round(Y_ZENTRUM - Y_ZENTRUM * Y_ALT / X_BEREICH * 100 / Y_SKALA, 0)
         X2 = Round(X_ZENTRUM - X_ZENTRUM * X / X_BEREICH, 0)
         Y2 = Round(Y_ZENTRUM - Y_ZENTRUM * Y / X_BEREICH * 100 / Y_SKALA, 0)
         If X1 <= 0 Or X2 <= 0 Or Y1 <= 0 Or Y2 <= 0 Then GoTo KEINZEICHNEN
         Set L = ActiveSheet.Shapes.AddLine(X1, Y1, X2, Y2)
         L.Line.ForeColor.RGB = RGB(120, 77, 250)
KEINZEICHNEN:
      End If
```

```
        X_ALT = X
        Y_ALT = Y


Next X

' Zeichnen X und Y-Achse

        X1 = Round(X_ZENTRUM - X_ZENTRUM * X_ALT / X_BEREICH, 0)
        Y1 = Round(Y_ZENTRUM - Y_ZENTRUM * Y_ALT / X_BEREICH * 100 / Y_SKALA, 0)
        X2 = Round(X_ZENTRUM - X_ZENTRUM * X / X_BEREICH, 0)
        Y2 = Round(Y_ZENTRUM - Y_ZENTRUM * Y / X_BEREICH * 100 / Y_SKALA, 0)

        Set L = ActiveSheet.Shapes.AddLine(X_ZENTRUM - X_ZENTRUM + 10, Y_ZENTRUM, X_ZENTRUM + X_ZENTRUM - 10, Y_ZENTRUM)
        L.Line.ForeColor.RGB = RGB(255, 0, 0)
        Set L = ActiveSheet.Shapes.AddLine(X_ZENTRUM, Y_ZENTRUM - Y_ZENTRUM + 10, X_ZENTRUM, Y_ZENTRUM + Y_ZENTRUM - 10)
        L.Line.ForeColor.RGB = RGB(255, 0, 0)


End Sub
```

Kommazahl (amerikanisch) in Formel hinzufügen

beim Dionex-Reporting wurden Kommabeträge in Zellen mit Formeln so hinzugefügt, dass die bestehende Formel erweitert wurde - zB aus =1+2 wurde =1+2+3,14

Es war eine amerikanische Tabelle, und sie führte zu Fehlermeldungen, wenn man die Beträge mit ","-Komma übergab - sie wollte nur "."-Kommabeträge

Mit nachfolgender Funktion konnte ich die dvo-Beträge (sie enthielten noch echten Tausendertrennpunkt, der zuerst entfernt wird) die mit ","-Komma waren, umwandeln in Kommabeträge mit "."-Komma

```
Public Function KOMMAZAHL_WANDELN(KOMMAZAHL As String)

        KOMMAZAHL = Replace(KOMMAZAHL, ".", "")

        If Asc(Right(KOMMAZAHL, 3)) = 44 Then
           KOMMAZAHL_WANDELN = Left(KOMMAZAHL, Len(KOMMAZAHL) - 3) & "." & Right(KOMMAZAHL, 2)
        End If
        If Asc(Right(KOMMAZAHL, 2)) = 44 Then
           KOMMAZAHL_WANDELN = Left(KOMMAZAHL, Len(KOMMAZAHL) - 2) & "." & Right(KOMMAZAHL, 1)
        End If
```

End Function

Runden von Zahlen - Problem der Round-Funktion

Ich hatte das Problem, dass eine Variable, die als Single definiert war und errechnet wurde erst mal eine Zahl mit vielen Kommastellen war.
Als ich diese mit Round (VARIABLE,2) auf zwei Stellen runden wollte, klappte das um die Burg nicht.

Im Internet fand ich dazu zwei Lösungen, die beide funktioniert haben:

1) Entweder die Variable nicht als SINGLE sondern als DOUBLE definieren

2) Oder statt dem normalen Code

Range("A12")=Round(-Abs(STEUER),2)

die erweiterte WorksheetFunction davor zu setzen

Range("A12")=WorksheetFunction.Round(-Abs(STEUER), 2)

ALLGEMEINE THEORIE ZUR RUNDENFUNKTION

Die Round-Funktion gibt eine Zahl zurück, die auf eine festgelegte Anzahl an Dezimalpunkten gerundet wurde.
Syntax

Round(*Ausdruck* [,*AnzahlAnDezimalpunktn*])

Die Syntax der Round-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
Ausdruck	Erforderlich. Numerischer Ausdruck, der gerundet wird.
AnzahlAnDezimalpunkten	Optional. Zahl, die angibt, wie viele Stellen rechts vom Dezimalpunkt beim Runden berücksichtigt werden. Wird dieser Wert ausgelassen, gibt die Round-Funktion Ganzzahlen zurück.

Sinus, Bogenmaß, Gradmaß

Excel rechnet alles im Bogenmaß. Im Bogenmaß ist ein ganzer Kreis (=360 Grad im Gradmaß) genau 2 Pi.

Man könnte auch sagen:

Bogenmaß = Grad * 2 Pi / 360

Man vereinfacht dies, indem man sagt

Bogenmaß = Grad * Pi / 180

Die Zahl Pi kann man in Excel entweder direkt eingeben - 3.1415926535897932 – oder man berechnet sie mit 4 * ATN (1) , da der Arcustangens von 1 genau ein Viertel Pi ist.

```
Sub test()

   Dim PI As Double ' 16 Kommastellen
   PI = 4 * Atn(1)

   For W = 0 To 90 Step 10
      MsgBox "Sinus von " & W & " Grad =" & Sin(W * PI / 180)
   Next W

End Sub
```

UST-Funktion mit mehrfacher Parameterübergabe und mehrfachen Rückgabewerten

Wir wollen eine Funktion, die aus an sie übergebenen Werten – Brutto, Netto, Steuerbetrag und Steuerprozent – jene Werte errechnet, die nicht an sie übergeben wurden (also mit dem Wert 0). Konkret wollen wir – egal welche 2 Werte wir übergeben – dass die restlichen zwei Werte ermittelt werden.

```
Sub USTTESTER()

   BRUTTO = Range("F11")
   NETTO = Range("G11")
```

```
    STEUER = Range("I11")
    PROZENT = Range("H11")

    A = UST(BRUTTO, NETTO, STEUER, PROZENT)

    MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT & "%"
End Sub

Public Function UST(BRUTTO, NETTO, STEUER, PROZENT) As Boolean

' - diese Function rechnet von den an sie übergebenen Teilen die restlichen Teile aus
' WICHTIG: NICHT übergebene Werte müssen vor dem Aufruf auf Null gesetzt werden - wenn zB KEIN Netto-Betrag übergeben wird, MUSS er zuvor auf 0 gesetzt werden,
'          weil er beim letzten Durchlauf der Funktion einen Wert erhalten hat
' - alle Variablen werden als Variant geführt, damit sie nicht definiert sein müssen
' - vor der Rückgabe werden sie alle auf 2 Stellen gerundet, die Prozent auf 0 Stellen (wobei auch die erste Kommastelle vor dem Runden geprüft wird,
' - sprich 20,04 % werden auf 20,0 gerunden und werden toleriert, 20,05 werden auf 20,1 aufgerundet und als Fehler ausgewiesen
' - Wenn kein eindeutiger Steuerprozentsatz ermittelt werden kann, wird der Prozentsatz 99 zurückgegeben
' - Beim Ermitteln von Beträgen (Brutto/Netto/Steuer) werden diese prinzipiell nur berechnet, wenn sie leer übergeben wurden
' - Die Function UST selbst ist als Boolean definiert, weil sie mit einer Hilfvariabel aufgerufen werden muss;
'   diese Hilfsvariable ist hier a, aber man kann jede beliebige Variable nehmen, da sie keine Relevanz hat.
' - der Aufruf dieser Funktion hier lautet:
'   a = UST(BRUTTO, NETTO, STEUER, PROZENT)
'   MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT


' ----------------------------
' --  Berechnen der Beträge --
' ----------------------------

' --- Wenn Bruttobetrag vorhanden ---

If BRUTTO <> 0 Then
    ' wenn Netto vorhanden => Steuer
    If NETTO <> 0 And STEUER = 0 Then
        STEUER = BRUTTO - NETTO
        GoTo WEITER1
    End If
    ' wenn Steuer vorhanden => Netto
    If STEUER <> 0 And NETTO = 0 Then
        NETTO = BRUTTO - STEUER
        GoTo WEITER1
    End If
    ' wenn Prozent vorhanden => Netto + Steuer
```

```
   If PROZENT <> "" And NETTO = 0 Then
      NETTO = BRUTTO / (1 + PROZENT / 100)
   End If
   If PROZENT <> "" And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If

WEITER1:

' --- Wenn Nettobetrag vorhanden ---

If NETTO <> 0 Then
   ' wenn Brutto vorhanden => Steuer
   If BRUTTO <> 0 And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
      GoTo WEITER2
   End If
   ' wenn Steuer vorhanden => Netto
   If STEUER <> 0 And BRUTTO = 0 Then
      BRUTTO = NETTO + STEUER
      GoTo WEITER2
   End If
   ' wenn nur Prozent vorhanden sind => Brutto + Steuer
   If PROZENT <> "" And BRUTTO = 0 Then
      BRUTTO = NETTO * (1 + PROZENT / 100)
   End If
   If PROZENT <> "" And STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If

WEITER2:

' --- Wenn Steuerbetrag vorhanden ---

If STEUER <> 0 Then
   ' wenn Brutto vorhanden => Netto
   If BRUTTO <> 0 And NETTO = 0 Then
      NETTO = BRUTTO - STEUER
      GoTo WEITER3
   End If
   ' wenn Netto vorhanden => Brutto
   If NETTO <> 0 And BRUTTO = 0 Then
```

```
      BRUTTO = NETTO + STEUER
      GoTo WEITER3
   End If
   ' wenn nur Prozent vorhanden sind => Brutto + Steuer
   If PROZENT <> "" And BRUTTO = 0 Then
      BRUTTO = STEUER * 100 / PROZENT + STEUER
   End If
   If PROZENT <> "" And NETTO = 0 Then
      NETTO = STEUER * 100 / PROZENT
   End If
End If

WEITER3:

' --- Von den Prozent ev. Fehlendes errechnen ---

If PROZENT <> "" Then
   ' Brutto ermitteln
   If BRUTTO = 0 Then
      If NETTO <> 0 Then
         BRUTTO = NETTO + NETTO * PROZENT / 100
      End If
      If STEUER <> 0 And BRUTTO = 0 Then ' nur wenn Brutto immer noch 0 ist
         BRUTTO = STEUER * 100 / PROZENT + STEUER
      End If
   End If
   ' Netto ermitteln
   If NETTO = 0 Then
      If BRUTTO <> 0 Then
         NETTO = BRUTTO / (1 + PROZENT / 100)
      End If
      If STEUER <> 0 And NETTO = 0 Then ' nur wenn Netto immer noch 0 ist
         NETTO = STEUER * 100 / PROZENT
      End If
   End If
   ' Steuer ermitteln
   If STEUER = 0 Then
      If BRUTTO <> 0 Then
         STEUER = BRUTTO * PROZENT / (100 + PROZENT)
      End If
      If NETTO <> 0 And STEUER = 0 Then ' nur wenn Steuer immer noch 0 ist
         STEUER = NETTO * PROZENT / 100
      End If
      If STEUER <> 0 And STEUER = 0 Then
```

```
         NETTO = STEUER * 100 / PROZENT
      End If
   End If

End If

If PROZENT = 0 Then
   If BRUTTO = 0 Then
      BRUTTO = NETTO + STEUER
   End If
   If NETTO = 0 Then
      NETTO = BRUTTO - STEUER
   End If
   If STEUER = 0 Then
      STEUER = BRUTTO - NETTO
   End If
End If


' --- die Prozent berechnen ---

If PROZENT = 0 Then

   ' default Prozent auf ERROR stellen 99
   PROZENT = 99

   If BRUTTO <> 0 Then
      If BRUTTO = NETTO Then
         PROZENT = 0
       Else
         If NETTO <> 0 Then ' Abfangen, falls Netto nicht bekannt
            PROZENT = Round((BRUTTO - NETTO) / NETTO * 100, 1) ' wir runden auf eine Kommastelle
         End If
      End If
    Else ' wenn brutto nicht bekannt ist
      PROZENT = Round(STEUER / NETTO * 100, 1)
   End If

End If

' --- zuletzt noch alles runden ---

BRUTTO = WorksheetFunction.Round(BRUTTO, 2) ' mit vorangestelltem WorksheetFunction rundet Round kaufmännisch korrekt - 1,25 wird 1,3 statt 1,2
NETTO = WorksheetFunction.Round(NETTO, 2)
```

```
STEUER = WorksheetFunction.Round(STEUER, 2)
PROZENT = WorksheetFunction.Round(PROZENT, 0)

' Wenn Prozentwert vorhanden, aber kein existierender echter => setze ihn auf 99
If PROZENT <> 20 And PROZENT <> 10 And PROZENT <> 13 And PROZENT <> 0 And PROZENT <> "" Then
   PROZENT = 99
End If



End Function

Sub TEST()

Dim BRUTTO
Dim NETTO
Dim STEUER
Dim PROZENT

NETTO = 24
STEUER = 4

a = UST(BRUTTO, NETTO, STEUER, PROZENT) ' A ist nur eine Hilfsvariable zum Aufrufen vom Typ Boolean

MsgBox BRUTTO & " - " & NETTO & " - " & STEUER & " - " & PROZENT


End Sub
```

Hier das Ergebnis der Funktion – oben die Eingaben für den UST-Rechner – und unten das Ergebnis – 99 bei Prozent steht für nicht korrekte Prozent.
Die Prozent werden relativ streng gerundet und geprüft ob sie 20, 13, 10 oder 0% sind – bei 100 € darf bei der Steuer eine Unschärfe von 5 Cent sein – bei 6 Cent gilt es nicht mehr als richtig

Angabe	4 Werte	3 Werte					2 Werte					1 Wert				Fehlangabe => 99 %				Unschärfen	
Netto	10,00	10,00	10,00	10,00		10,00	10,00		10,00				10,00			8,00	10,00	10,00		100,00	100,00
Prozent	20	20	20		20	20		20		20				20							
Steuer	2,00	2,00		2,00	2,00		2,00	2,00			2,00				2,00		3,00		3,00	20,06	20,05
Brutto	12,00		12,00	12,00	12,00				12,00	12,00	12,00	12,00				12,00		13,00	13,00		
Ergebnis																					
Netto	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	12,00	10,00	0,00	0,00	8,00	10,00	10,00	10,00	100,00	100,00
Prozent	20	20	20	20	20	20	20	20	20	20	20	0	0	20	99	99	99	99	99	99	20
Steuer	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	0,00	0,00	0,00	2,00	4,00	3,00	3,00	3,00	20,06	20,05
Brutto	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	12,00	10,00	0,00	2,00	12,00	13,00	13,00	13,00	120,06	120,05

VAL und CDBL

Bisweilen addiere ich gerne Zahlen in Zellen hinein und möchte dies über eine Formel machen, damit man die Beträge einzeln sieht.

Leider mag der VBA-Code beim Addieren von Zahlen in Formeln hinein nur das amerikanische Zahlenformat – also zB 123.45 und nicht 123,45
Hier mit VAL zu arbeiten bringt nichts. Was aber klappt ist, dass man mit Replace-Funktion das deutsche Zahlenformat umwandelt ins amerikanische

```
If Cells(ZZ, 7) = "" Then
   Cells(ZZ, 7).Formula = "=" & Replace(Sheets("BV " & MONAT).Cells(Z, 2), ",", ".")
 Else
   Cells(ZZ, 7).Formula = Cells(ZZ, 7).Formula & "+" & Replace(Sheets("BV " & MONAT).Cells(Z, 2), ",", ".")
End If
```

Sollte mal ein Datenmüll aus NTCS kommen mit echten Tausenderpunkten, dann habe ich eine Hilfsprozedur, die beim befüllen der Formeln verwendet wird:

So sieht die Anwendung der Hilfsfunktion aus:

```
If Cells(ZZ, 17) = "" Then
   Cells(ZZ, 17).Formula = "=" & BETRAG(Sheets("BV " & MONAT).Cells(Z, 15))
 Else
   Cells(ZZ, 17).Formula = Cells(ZZ, 17).Formula & "+" & BETRAG(Sheets("BV " & MONAT).Cells(Z, 15))
End If
```

Und so die Hilfsfunktion selbst:

```
Public Function BETRAG(ZAHL As Variant)

    BETRAG = ZAHL
    BETRAG = Replace(BETRAG, ".", "") ' Tausenderpunkte entfernen
    BETRAG = Replace(BETRAG, ",", ".") ' Dezimalpunkt umwandeln in Kommapunkt (wichtig für Formelerweiterung, die keine deutschen Beträge mit Komma nimmt)

End Function
```

Ich hatte eine Zelle als Zahl formatiert und mittels WENN-Formeln wird eine Summe darin ermittelt. Die Formel und deren Quellzellen waren so beschaffen, dass das Ergebnis auch nur eine leer Zelle ergeben kann. Dieses konnte dann nicht mittels VBA ausgelesen und mit anderen Zahlen/Zell-Zahlen addiert werden, und ich bekam immer eine Fehlermeldung.

Lösung: NICHT mit CDBL arbeiten, weil dies nur dann Zellwerte in Zahlen umwandelt, wenn dieses irgendwo als Zahlen zu interpretierende Werte hat - aber nicht, wenn es NICHTS enthält. Mit NICHTS kommt nur VAL zurecht - daher habe ich die Zelle mit VAL(Range("A1") ausgelesen

Val erwartet übrigens den internationalen Punkt als Komma. daher:

Val("1234,5") ergeibt nur 1234

Arbeite stattdessen mit CDBL und schon erkennt es auch den Beistrich als Komma an

WICHTIG: CDBL("ABC") führt zu Fehlermeldung, da CDBL nur mit Strings zurecht kommt, die echte Zahlen darstellen - VAL("ABCD") hingegen ergibt den Wert 0 ohne Fehlermeldung

Zahlen umwandeln in ausgeschriebenen Text

```
Function DollarText(vNumber) As Variant
'see also Function SpellNumber(ByVal MyNumber), PSS ID Number: Q140704
Dim sDollars As String
Dim sCents As String
Dim iLen As Integer
Dim sTemp As String
Dim iPos As Integer
Dim iHundreds As Integer
Dim iTens As Integer
```

```
Dim iOnes As Integer
Dim sUnits(2 To 5) As String
Dim bHit As Boolean
Dim vOnes As Variant
Dim vTeens As Variant
Dim vTens As Variant
If Not IsNumeric(vNumber) Then
Exit Function
End If
sDollars = Format(vNumber, "###0.00")
iLen = Len(sDollars) - 3
If iLen > 15 Then
DollarText = CVErr(xlErrNum)
Exit Function
End If
sCents = Right$(sDollars, 2) & "/100 Dollars"
If vNumber < 1 Then
DollarText = sCents
Exit Function
End If
sDollars = Left$(sDollars, iLen)
vOnes = Array("", "One", "Two", "Three",
"Four", "Five", _
"Six", "Seven", "Eight", "Nine")
vTeens = Array("Ten", "Eleven", "Twelve",
"Thirteen", "Fourteen", _
"Fifteen", "Sixteen", "Seventeen",
"Eighteen", "Nineteen")
vTens = Array("", "", "Twenty", "Thirty",
"Forty", "Fifty", _
"Sixty", "Seventy", "Eighty", "Ninety")
sUnits(2) = "Thousand"
sUnits(3) = "Million"
sUnits(4) = "Billion"
sUnits(5) = "Trillion"
sTemp = ""
For iPos = 15 To 3 Step -3
If iLen >= iPos - 2 Then
bHit = False
If iLen >= iPos Then
iHundreds = Asc(Mid$(sDollars, iLen - iPos + 1, 1)) - 48
If iHundreds > 0 Then
sTemp = sTemp & " " & vOnes(iHundreds) & "
Hundred"
```

```
bHit = True
End If
End If
iTens = 0
iOnes = 0
If iLen >= iPos - 1 Then
iTens = Asc(Mid$(sDollars, iLen - iPos + 2, 1)) - 48
End If
If iLen >= iPos - 2 Then
iOnes = Asc(Mid$(sDollars, iLen - iPos + 3, 1)) - 48
End If
If iTens = 1 Then
sTemp = sTemp & " " & vTeens(iOnes)
bHit = True
Else
If iTens >= 2 Then
sTemp = sTemp & " " & vTens(iTens)
bHit = True
End If
If iOnes > 0 Then
If iTens >= 2 Then
sTemp = sTemp & "-"
Else
sTemp = sTemp & " "
End If
sTemp = sTemp & vOnes(iOnes)
bHit = True
End If
End If
If bHit And iPos > 3 Then
sTemp = sTemp & " " & sUnits(iPos \ 3)
End If
End If
Next iPos
DollarText = Trim(sTemp) & " and " & sCents
End Function 'DollarText
```

Zufallszahlen erzeugen

Version 1 von mir

```
' RND erzeugt eine Zahl <1 und >0
```

```
Randomize ' Reset des internen Zufallgenerators

Msgbox Int((Obergrenze - Untergrenze + 1) * Rnd + Untergrenze)
```

Version 2

```
Sub RandomNumbers()
Dim Number()
Dim MyRange As Range
Dim c As Range
Set MyRange = Selection
LastNumber = 100000
ReDim Number(LastNumber)
For i = 1 To LastNumber
Number(i) = i
Next i
For Each c In MyRange
Placement = Int(Rnd() * LastNumber + 1)
c.Value = Number(Placement)
dummy = Number(LastNumber)
Number(LastNumber) = Number(Placement)
Number(Placement) = dummy
LastNumber = LastNumber - 1
Next c
End Sub
```

FUNKTIONEN TEXT

Aus Zahl Text machen CSTR() vs STR()

Ganz wichtig: die beste Funktion um Zahlen in Text umzuwandeln ist die CSTR-Funktion. Die STR-Funktion macht dies zwar auch, aber mit einem meist unerwünschten Nebeneffekt: Sie übergibt IMMER ein Zeichen für das Vorzeichen der Zahl: konkret ein Minus bei negativen Zahlen und ein Leerzeichen bei positiven Zahlen. Die CSTR-Funktion wandelt Zahlen besser um - ein negatives Vorzeichen kommt an, positive Zahlen erhalten nur die Zahlen ohne Vorzeichen oder einen Leerzeichen-Platzhalter.
Alternativ mit der TRIM-Funktion (LTRIM würde reichen) die STR-Funktion zähmen: Msgbox Trim(Str("2018")) ergibt 2018

Funktion	Rückgabetyp	Bereich des Arguments *Ausdruck*
CBool	Boolean	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck.
CByte	Byte	0 bis 255.
CCur	Currency	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807.
CDate	Date	Ein beliebiger gültiger Datumsausdruck.
CDbl	Double	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
CDec	Decimal	+/-79.228.162.514.264.337.593.543.950.335 für skalierte Ganzzahlen, d.h. Zahlen ohne Dezimalstellen. Für Zahlen mit 28 Dezimalstellen gilt der Bereich +/-7,9228162514264337593543950335. Die kleinste mögliche Zahl ungleich Null ist 0,0000000000000000000000000001.
CInt	Integer	-32.768 bis 32.767; Nachkommastellen werden gerundet.
CLng	Long	-2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet.

CLngLng	LongLong	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807; Dezimalen werden gerundet (nur auf 64-Bit-Plattformen zulässig)
CLngPtr	LongPtr	-2.147.483.648 bis 2.147.483.647 auf 32-Bit-Systemen, -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807 auf 64-Bit-Systemen; Dezimalen werden auf 32-Bit- und 64-Bit-Systemen gerundet.
CSng	Single	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Cvar	Variant	Numerische Werte im Bereich des Typs **Double**. Nichtnumerische Werte im Bereich des Typs **String**.
CStr	String	Rückgabe für CStr hängt vom Argument *Ausdruck* ab.

Anzahl Vorkommen eines Zeichen in String

```
s = "1.1.2.3sxy"
AnzahlPunkteInString = UBound(Split(s, "."))
MsgBox AnzahlPunkteInString
```

Alternative

```
Dim Text As String
Text = "Hallo Du da."
MsgBox Len(Text) - Len(Replace(Text, ".", vbNullString))
```

Alphanummerische Umwandlung (Entfernen alle Sonderzeichen)

Sollen alle Sonderzeichen entfernt werden und nur Ziffern und Buchstaben übrig bleiben:

```
Sub Test()
        MsgBox ALPHANUMMERISCH("öäü?\")
End Sub
```

```
Public Function ALPHANUMMERISCH(ALPHATEXT) As String

   ALPHATEXT = Replace(ALPHATEXT, " ", "")
   ALPHATEXT = Replace(ALPHATEXT, "<", "")
   ALPHATEXT = Replace(ALPHATEXT, ">", "")
   ALPHATEXT = Replace(ALPHATEXT, ",", "")
   ALPHATEXT = Replace(ALPHATEXT, ";", "")
   ALPHATEXT = Replace(ALPHATEXT, ".", "")
   ALPHATEXT = Replace(ALPHATEXT, ":", "")
   ALPHATEXT = Replace(ALPHATEXT, "-", "")
   ALPHATEXT = Replace(ALPHATEXT, "_", "")
   ALPHATEXT = Replace(ALPHATEXT, "#", "")
   ALPHATEXT = Replace(ALPHATEXT, "'", "")
   ALPHATEXT = Replace(ALPHATEXT, "|", "")
   ALPHATEXT = Replace(ALPHATEXT, "+", "")
   ALPHATEXT = Replace(ALPHATEXT, "~", "")
   ALPHATEXT = Replace(ALPHATEXT, "*", "")
   ALPHATEXT = Replace(ALPHATEXT, "^", "")
   ALPHATEXT = Replace(ALPHATEXT, "°", "")
   ALPHATEXT = Replace(ALPHATEXT, "!", "")
   ALPHATEXT = Replace(ALPHATEXT, """", "")
   ALPHATEXT = Replace(ALPHATEXT, "§", "")
   ALPHATEXT = Replace(ALPHATEXT, "%", "")
   ALPHATEXT = Replace(ALPHATEXT, "&", "")
   ALPHATEXT = Replace(ALPHATEXT, "/", "")
   ALPHATEXT = Replace(ALPHATEXT, "(", "")
   ALPHATEXT = Replace(ALPHATEXT, ")", "")
   ALPHATEXT = Replace(ALPHATEXT, "{", "")
   ALPHATEXT = Replace(ALPHATEXT, "}", "")
   ALPHATEXT = Replace(ALPHATEXT, "[", "")
   ALPHATEXT = Replace(ALPHATEXT, "]", "")
   ALPHATEXT = Replace(ALPHATEXT, "=", "")
   ALPHATEXT = Replace(ALPHATEXT, "ß", "ss")
   ALPHATEXT = Replace(ALPHATEXT, "ä", "ae")
   ALPHATEXT = Replace(ALPHATEXT, "Ä", "AE")
   ALPHATEXT = Replace(ALPHATEXT, "Ö", "OE")
   ALPHATEXT = Replace(ALPHATEXT, "ö", "oe")
   ALPHATEXT = Replace(ALPHATEXT, "ü", "ue")
   ALPHATEXT = Replace(ALPHATEXT, "Ü", "UE")
   ALPHATEXT = Replace(ALPHATEXT, "?", "")
   ALPHATEXT = Replace(ALPHATEXT, "´", "")
   ALPHATEXT = Replace(ALPHATEXT, "`", "")
   ALPHATEXT = Replace(ALPHATEXT, "\", "")
   ALPHATEXT = Replace(ALPHATEXT, "@", "")
```

```
    ALPHATEXT = Replace(ALPHATEXT, "€", "")

    ALPHANUMMERISCH = ALPHATEXT

End Function
```

Aus String alle Nichtzahlen entfernen

Es bleiben nur die Zeichen 0-9, alle andere wird entfernt - auch Komma und Tausenderpunkte !

```
Public Function ENTFERNE_NICHTZAHLEN(ZAHL As Variant) As Long

Dim NEUZAHL

    ' For T = 1 To Len(Str(ZAHL))-1 ' meist geht es besser ohne STR-Funktion
     For T = 1 To Len(ZAHL)-1

        If Val(Mid(ZAHL, T, 1)) > 0 Or Mid(ZAHL, T, 1) = 0 Then
            NEUZAHL = NEUZAHL & Mid(ZAHL, T, 1)
        End If
    Next T
ZAHL = NEUZAHL

End Function
```

Aus Text nur die Zahlen überlassen

Siehe auch Funktion davor "Aus String alle Nichtzahlen entfernen"

Wenn in Zelle A1 z.B. der Text 1.2.3.4 steht, erhält man die Zahl 1234 durch folgenden Code

```
Dim Zahl as Integer

Zahl=Range("A1")
```

Einzelne Zeichen in einer Zelle formatieren

n = Range("E3").Characters.Count ' Anzahl der Zeichen dieser Zelle auslesen
Range("E3").Characters(n, 1).Font.Superscript = True ' das letzte Zeichen zu einer Hochziffer (Superscript-Eigenschaft) machen.
Range("E3").Characters(n-1, 2).Font.Superscript = True ' das letzten 2 Zeichen zu einer Hochziffer machen.

Das Character-Objekt benötigt einen Hinweis,
- welches das erste Zeichen seini soll
- und dann, wieviele Zeichen ab dieser Position betroffen sein sollen

Ersetzen von Zeichen in String

MsgBox WorksheetFunction.Substitute("HANSI", "A", "a")
MsgBox Replace("HANSI", "A", "a")

Ersetzen von Umlaute-Fehlerzeichen in Text

Bisweilen exportieren Schnittstellen wie DPW die Sonderzeichen falsch nach Excel - mittels ASC-Code kann man sie leicht ersetzen

```
MYTEXT = Cells(Z, 5) ' 1.Text
MYTEXTNEU = ""
For L = 1 To Len(MYTEXT)
   LETTER = Mid(MYTEXT, L, 1)
   If Asc(LETTER) = 132 Then LETTER = "ä"
   If Asc(LETTER) = 148 Then LETTER = "ö"
   If Asc(LETTER) = 129 Then LETTER = "ü"
   If Asc(LETTER) = 142 Then LETTER = "Ä"
   If Asc(LETTER) = 153 Then LETTER = "Ö"
   If Asc(LETTER) = 154 Then LETTER = "Ü"
   If Asc(LETTER) = 225 Then LETTER = "ß"
   MYTEXTNEU = MYTEXTNEU & LETTER
Next L
Cells(Z, 5) = MYTEXTNEU
```

Groß-Kleinschreibung wechseln

```
Sub ToggleCase()
Dim Upr, Lwr, Ppr
'Originaladresse speichern
Set OriginalCell = ActiveCell
Set OriginalSelection = Selection
If IsEmpty(ActiveCell) Then GoTo NoneFound
'Errorhandling
```

```
On Error GoTo Limiting
If OriginalCell = OriginalSelection Then
Selection.Select
GoTo Converting
Else
Resume Next
End If
Limiting:
'Auswahl auf gültige Zellen begrenzen
On Error GoTo NoneFound
Selection.SpecialCells(xlCellTypeConstants, 3).Select
Converting:
'Statusbar ändern
Application.StatusBar = "Ändere Gross- und Kleinschreibung..."
For Each DCell In Selection.Cells
Upr = UCase(DCell)
Lwr = LCase(DCell)
If Upr = DCell.Value Then
DCell.Value = Lwr
Else
DCell.Value = Upr
End If
Next DCell
'Statusbar zurücksetzen
Application.StatusBar = False
Exit Sub
NoneFound:
MsgBox "Alle Zellen der aktuelllen Auswahl enthalten Formeln oder sind
leer!", vbExclamation, " Fehler aufgetreten"
OriginalSelection.Select
OriginalCell.Activate
End Sub
```

Leerzeichen entfernen

LTrim(" 123") entfernt beginnende Leerzeichen
RTrim("12345 ") entfernt abschließende Leerzeichen
Trim(" 123 ") entfernt beginnende und abschließende Leerzeichen

Replace(" 123 456 "," ","") entfernt alle Leerzeichen, egal wo sie sind - auch wenn mehrere nebeneinander sind

Will man mehrere Leerzeichen auf nur EIN Leerzeichen reduzieren gibt es ja in Word die GLÄTTEN-Funktion. Diese entfernt führende und abschließende Leerzeichen und doppelte oder mehrfache Leerzeichen innerhalb der Zelle werden auf ein Leerzeichen reduziert.

Die Glättenfunktion heißt in VBA Trim – aber Achtung, die reine Trim-Funktion schneidet nur führende und abschließende Leerzeichen weg, entfernt aber keine doppelten/mehrfachen Leerzeichen INNERHALB der Zelle. Ruft man die Glättenfunktion über die Application-Methode auf, dann hat man die echte Excel-Glättenfunktion.

Msgbox Application.Trim(" AB CD EF ") & Trim (" AB CD EF ") zeigt dir sofort den Unterschied - nur der erste Befehl reduziert auch doppelte/mehrfache Leerzeichen in der Zelle.

Klassisches Beispiel daher:

Range("A1") = Application.Trim("A1")

Will man eine ganze Tabelle säubern:

```
Sub LeerRaus()
With ActiveSheet
.UsedRange.Cells.Value = Application.Trim(.UsedRange.Cells.Value)
End With
End Sub
```

Leerzeichen finden - ASCI-Code anzeigen

Um nicht benötigte Leerzeichen zu entfernen wird man in der Regel mit " " - Strings arbeiten, die man sucht und entfern.

Dies ist aber nur das europäische / deutsche Leerzeichen (ASC-Code 32).

Bei einem internationalen Reporting gab es jedoch ein Leerzeichen, das genauso aussah wie ein normales Leerzeichen (nona :o), aber das den ASC-Code 160 hatte.

Daher im Bedarfsfall eine Zelle zerlegen und die ASC-Codes abfragen:

```
For T = 1 To Len(ActiveCell)
        MsgBox Asc(Mid(ActiveCell, T, 1))
Next T
```

SONDERZEICHEN-REPARATUR

Regelmäßig kommen Umlaute in Excel falsch an und wollen repariert werden.

Variante 1 korrigiert ganzes Blatt

```
Sub Sonderzeichen_UTF8()


' fehlerhafte UTF8-Codes korrigieren  in gesamten Tabellenblatt Export
    ' Aktuelles Tabellenblatt komplett korrigieren
     Cells.Replace What:="Ã¤", Replacement:="ä", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ã""", Replacement:="Ä", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ã¼", Replacement:="ü", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ãœ", Replacement:="Ü", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ã", Replacement:="ö", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ã-", Replacement:="Ö", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="Ã", Replacement:="ö", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="ÃŸ", Replacement:="ß", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False
     Cells.Replace What:="¶", Replacement:="", LookAt:=xlPart, SearchOrder:=xlByRows,
MatchCase:=False, SearchFormat:=False, ReplaceFormat:=False

End Sub
```

Variante 2 korrigiert nur eine bestimmte Textvariable

```
Public Function REPARATUR_ASCII(REPARATUR_TEXT)

    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ã„", "Ä")

    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ã¤", "ä")

    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ã-", "Ö")
```

```
    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ã¶", "ö")
    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ãœ", "Ü")
    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "Ã¼", "ü")
    REPARATUR_TEXT = Replace(REPARATUR_TEXT, "ÃŸ", "ß")
    REPARATUR_ASCII = REPARATUR_TEXT
End Function

Sub test2()
    MsgBox REPARATUR_ASCII(ActiveCell.Value)
End Sub
```

Hier die Theorie

Zeichensalat	Sonderzeichen	Hexadez. Zeichen-Nr.	Dezimale Zeichen-Nr.	Benennung
=E4	ä	#x00E4	228	Kleines a mit Diaeresis (a-Umlaut)
=F6	ö	#00F6	246	Kleines o mit Diaeresis (o-Umlaut)
=FC	ü	#x00FC	252	Kleines u mit Diaeresis (u-

				Umlaut)
=DF	ß	#x00DF	223	Scharfes s, Esszett, Ligatur aus lang-s und z
=C4	Ä	#x00C4	196	Großes A mit Diaeresis (A-Umlaut)
=D6	Ö	#x00D6	214	Großes O mit Diaeresis (O-Umlaut)
=DC	Ü	#x00DC	220	Großes U mit Diaeresis (U-Umlaut)

Nun können Sie den Zeichenkauderwelsch in Klartexte zurückübersetzen; zum Beispiel in einem Editor wie WordPad, TextPad, Scite oder auch in dem Textverarbeitungsprogramm Microsoft Word: Mit dem Wechselbefehl ("Suchen/ Ersetzen"; Strg. + H); "suchen nach" dem Zeichenkauderwelsch in der ersten Tabellenspalte, "ersetzen durch" das richtige Zeichen in der zweiten Tabellenspalte; dabei auf Groß- und Kleinschreibung achten!

falsche Zeichen	**Benennung der falschen Zeichen**	**Richtiges Zeichen**	**Benennung des richtigen Zeichens**	**hexadez. Unicode des richtigen Zeichens**
Ã„	großes A mit Tilde und gekrümmte Anführungszeichen unten ("99")	Ä	Großes A mit Umlaut (Diaeresis)	00C4
Ã¤	großes A mit Tilde und allgem. Währungszeichen (= Ricardo-Sonne, "Filzlaus")	ä	Kleines a mit Umlaut (Diaeresis)	00E4
Ã–	großes A mit Tilde und langer Gedankenstrich (n-dash)	Ö	großes O mit Umlaut	00D6

			(Diaeresis)	
Ã¶	großes A mit Tilde und Absatzzeichen (Pilcrow)	ö	kleines o mit Umlaut (Diaeresis)	00F6
Ãœ	großes A mit Tilde und Ligatur aus kleinem o und kleinem e	Ü	großes U mit Umlaut (Diaeresis)	00DC
Ã¼	großes A mit Tilde und Bruchzahl "Ein Viertel"	ü	Kleines u mit Umlaut (Diaeresis)	00FC
ÃŸ	großes A mit Tilde und großes Y mit Diaeresis	ß	Esszett, scharfes S	00DF
Ã¡	großes A mit Tilde und auf dem Kopf stehendes (spanisches) Ausrufungszeichen	á	Kleines a mit Akut (accent aigu)	00E1
Ã©	großes A mit Tilde und "Copyright"-Zeichen (umkreistes kleines c)	é	Kleines e mit Akut (accent aigu)	00E9
Ã¨	großes A mit Tilde und Diaeresis (Umlaut-Punkte)	è	Kleines e mit Gravis (accent grave)	00E8

Zeichensalat	**Beschreibung des Zeichensalats**	**Richtiges Zeichen**	**Benennung des richtigen Zeichens**
â€ž	kleines a mit Zirkumflex und Euro-Währungszeichen und kleines z mit Caron	„	gekrümmte Anführungsstriche unten ("Gänsefüßchen unten") ("99")
â€“	kleines a mit Zirkumflex und Euro-Währungszeichen und gekrümmte	–	langer Gedankenstrich (Geviertstrich)

	Anfürungszeichen oben ("66")		
â€œ	kleines a mit Zirkumflex und Euro-Währungszeichen und oe-Ligatur (Ligatur aus kleinem o und kleinem e)	“	gekrümmte Anführungsstriche oben ("Gänsefüßchen oben") ("66")

Zeichensalat	Beschreibung des Zeichensalats	Richtiges Zeichen	Benennung des richtigen Zeichens
+ANw-	Plus-Zeichen, großes A, großes N, kleines W, Bindestrich	Ü	großes U mit Umlaut (Diaeresis)
+APw-	Plus-Zeichen, großes A, großes P, kleines W, Bindestrich	ü	kleines u mit Umlaut (Diaeresis)
+AOQ-	Plus-Zeichen, großes A, großes O, großes Q, Bindestrich	ä	kleines a mit Umlaut (Diaeresis)
+APY-	Plus-Zeichen, großes A, großes P, großes Y, Bindestrich	ö	Kleines o mit Umlaut (Diaeresis)
+AEA-	Plus-Zeichen, großes A, großes E, großes A, Bindestrich	@	'at'-Zeichen (Klammeraffe, Affenschaukel)

STR-Funktion Problem Leerzeichen

die STR-Funktion übergibt immer das Vorzeichen - bei positiven Zahlen in Form eines Leerzeichens.

1.) Entweder die STR-Funktion mit der TRIM (bzw LTRIM würde reichen)-Funktion zähmen
Msgbox Trim(Str("2018")) ergibt 2018, da TRIM ein vorlaufendes und nachlaufendes Leerzeichen entfernt (da STR-Funktion nur vorlaufendes Leerzeichen erzeugt, würde LTRIM reichen)

2.) Gleich mit der CSTR-Funktion arbeiten, die keine Leerzeichen erzeugt

Textteil in String herausschneiden

```
msgbox Mid("Bengerl",2,6)
```

Texte normieren (Leerzeichen + Nicht-Buchstaben löschen)

```
Public Function NORMTEXT(TEXT As String)
    ' dient zum Wegschneiden Leerzeichen und Sonderzeichen
    Dim T
    Dim TEXTNEU

    TEXTNEU = ""
    For T = 1 To Len(TEXT)
        If Mid(TEXT, T, 1) <> " " And Mid(TEXT, T, 1) <> "/" And Mid(TEXT, T, 1) <> "." And Mid(TEXT, T, 1) <> "-" Then
            TEXTNEU = TEXTNEU & Mid(TEXT, T, 1)

        End If
    Next T

    NORMTEXT = TEXTNEU
    'MsgBox "NORMTEXT: " & NORMTEXT


End Function
```

Umlaute ersetzen

SIEHE AUCH SONDERZEICHEN REPARATUR UM ÄÖÜ und ß zu reparieren

```
Sub UmlauteWandeln()
Dim MyRange As Range
Dim Cell As Range
Application.ScreenUpdating = False
Set MyRange = Selection
For Each Cell In MyRange
Selection.Replace What:="ss", Replacement:="ss",
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="ü", Replacement:="ue",
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="Ü", Replacement:="Ue",
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="ö", Replacement:="oe",
```

```
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="Ö", Replacement:="Oe",
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="ä", Replacement:="ae",
LookAt:=xlPart, MatchCase:=True
Selection.Replace What:="Ä", Replacement:="Ae",
LookAt:=xlPart, MatchCase:=True
Next Cell
End Sub
```

Verschlüsselung => Registrierung

Zeichenkonstanten (Tabulator, Zeilenvorschub...)

um z.B. ein "-Zeichen in einen String zu bekommen, setzt man es einfach 2x

z.B. TEXT = "Ich kam mir etwas ""blöd"" vor "
oder .HTMLBody = "<FONT COLOR=""#D00000"">Dies ist mein Text"

Konstante	Entsprechung	Erklärung
vbBack	Chr(8)	Rückschrittzeichen.
vbCr	Chr(13)	CarrigeReturn oder auch Wagenrücklaufzeichen
vbCrLf	Chr(13)+Chr(10)	CarrigeReturn-LineFeed. Dies dient unter Windows dem Zeilenumbruch.
vbFormFeed	Chr(12)	Seitenvorschub. Findet (denk ich mal) nur im Druckbereich verwendung
vbLf	Chr(10)	LineFeed oder Zeilenvorschub(eine Zeile Weiter). Unter Windows sollte/darf man CarrigeReturn nicht vergessen.
vbNewLine		NewLine ist Plattformabhängig und daher ideal. Es wählt selbstständig aus, wie auf dem System ein Zeilenumbruch dargestellt wird(unter Windows **vbCrLf**)
vbNullChar	Chr(0)	Chr(0) ist immer das Abschliessende Zeichen einer Zeichenkette oder eines Arrays. Unter Basic findet es eher selten Verwendung(nur in Verbindung mit API-Funktionen)
vbNullString	""	eine wohl selten benutze Konstante, da "" schneller geht und leichter Lesbar ist.
vbTab	Chr(9)	Ein Tabulatorzeichen. Da ein Tabulator eigentlich gar kein Zeichen ist, wird es oft als eine bestimmte(4) Anzahl von Leerzeichen dargestellt. In professionelleren Bereichen(zB Word) kann man die Tabulatorweite genau definieren.
vbVerticalTab	Chr(11)	Das gleiche nur Vertikal.. bis jetzt hab ich noch keinen Einsatz dafür gefunden?!

vbCrLf	**Chr(**13**) + Chr(**10**)**	Kombination aus Wagenrücklauf und Zeilenvorschub
vbCr	**Chr(**13**)**	Wagenrücklaufzeichen

vbLf	**Chr(**10**)**	Zeilenvorschubzeichen
vbNewLine	**Chr(**13**) + Chr(**10**)** oder, auf dem Macintosh, **Chr(**13**)**	Plattformspezifisches Zeilenumbruchzeichen; je nachdem, welches für die aktuelle Plattform geeignet ist
vbNullChar	**Chr(**0**)**	Zeichen mit dem Wert 0
vbNullString	Zeichenfolge mit dem Wert 0	Nicht identisch mit der Null-Zeichenfolge (""); wird verwendet, um externe Prozeduren aufzurufen.
vbObjectError	-2147221504	Benutzerdefinierte Fehlernummern sollten größer als dieser Wert sein. Zum Beispiel: Err.Raise Number = vbObjectError + 1000
vbTab	**Chr(**9**)**	Tabulatorzeichen
vbBack	**Chr(**8**)**	Rückschrittzeichen

Zeichen in String finden - vorwärts und rückwärts

INSTR (Suchstartposition beginnend mit 1, zu durchsuchender String, Suchstring)

```
MsgBox InStr(1, "1.1.1", ".") ' ergibt 2
MsgBox InStr(2, "1.1.1", ".") ' ergibt 2
MsgBox InStr(3, "1.1.1", ".") ' ergibt 4
MsgBox InStr(4, "1.1.1", ".") ' ergibt 4

Möchte man Teile finden bis zum nächsten Auftreten eines Zeichens:

a = "1.14.1"
MsgBox Mid(a, 3, InStr(3, a, ".") - 3)

Wir wollen die mittlere 14 herausschneiden, egal ob 1, 2 oder 3 Stellig - daher, wir brauchen
ab der 3 Position alles bis zum Auftreten des zweiten Punktes
```

Die InStr-Funktion sucht nach einem SUCHTEXT in einem ZUDURCHSUCHENDENTEXT - sie beginnt dabei beim ersten Zeichen, es sei denn die STARTPOSITION wird angegeben:

*InStr([*STARTPOSITION*,]* ZUDURCHSUCHENDENTEXT*,* SUCHTEXT *)*

Die InStrRev-Funktion sucht auf dieselbe Weise, jedoch beginnt sie beim letzten Zeiten (wenn Startposition nicht angegeben ist) und sucht von rechts nach links:

*InstrRev(*ZUDURCHSUCHENDENTEXT*,* SUCHTEXT *[,*STARTPOSITION*])*

WICHTIG: Das Ergebnis ist aber NICHT das wievielte Zeichen es von hinten gezählt ist, sondern wieder gleich wie bei der InStr-Funktion an der wievielten Stelle es von vorne gezählt vorkommt.

Bsp: Instr("1234,678," , ",") gibt den Wert 5 aus - InstrRev("1234,678," , ",") gibt den Wert 9 aus !

Möchte man wissen, an der wievielten Stelle der Suchbegriff von rechts aus gesehen vorkommt, muss man so arbeiten:

Len("1234,678," , ",") - InstrRev("1234,678," , ",")+1

Beispiel: Vom Ordnerpfad der Excelarbeitsmappe, die den VBA-Code enthält (daher ThisWorkbook und nicht ActiveWorkbook) soll der übergeordnete Ordnerpfad herausgefunden werden:

```
PFAD = ThisWorkbook.Path ' ergibt zB: C:\OP_Lexor\Schnittstelle\Unterordner
MsgBox Left(PFAD, InStrRev(PFAD, "\")-1) ' ergibt C:\OP_Lexor\Schnittstelle
```

Zeilenumbruch in Zelle entfernen

Wenn in einer Zelle Zeilenumbrüche sind in Excel, dann werden diese beim Export leider mitexportiert und verursachen einen Datenkauderwelsch

Der Zeilenumbruch in Excel innerhalb einer Zelle ist VBLF - ich entferne ihn mit Replace

Replace(Textvariable,vblf,““) ' ev. auch VBCr und vbcrlf probieren

Funktion zum wiederholten entfernen von Umbrüchen und Leerzeichenplatzhaltern

```
Public Function ENTFERNEZEILENUMBRUCH(ZELLINHALT) As String
' Hilfsprozedur, die nicht erwünschte Zell/Zeilen-Umbrüche innerhalb einer Zelle entfernt
```

```
    ZELLINHALT = Replace(ZELLINHALT, "  ", " ")
    ZELLINHALT = Replace(ZELLINHALT, "  ", " ") ' wiederholen, damit aus drei Leerzeichen eines wird
    ZELLINHALT = Replace(ZELLINHALT, vbCr, " ")
    ZELLINHALT = Replace(ZELLINHALT, vbCrLf, " ")
    ZELLINHALT = Replace(ZELLINHALT, vbLf, " ")

    ENTFERNEZEILENUMBRUCH = ZELLINHALT

End Function
```

BSP Bank Austria

CSV-Dateien der Bank Austria haben viele Zeilen, die durch verschiedene Zeichen entstehen, sowohl interne Zellumbrüche als auch viele Leerzeichen. Folgender Inhalt kommt einer einzelnen Zelle vor

```
SEPA-AUFTRAGSBESTÄTIGUNG
                        Belegnr.: 162872.867.054.346
                        13.10.2016          EUR   10017 474 437
 Zahlungsempf.: Agentur ABC GmbH
       IBAN: AT622011182766xxxxxx          BIC: GIBAATWW
         ID:
 Zahlungsgrund:
  Zahlungsref.: Re 201610210
  Auftraggeber: Dr. Harald XXXX                    Valuta:   13.10
       IBAN: AT941200010xxxxxxx          BIC: BKAUATWW
         ID:                             Betrag:    EUR
                                           4.140,00
 Auftr.geb.Ref:
```

Das Ergebnis sieht so aus …

SEPA-AUFTRAGSBESTÄTIGUNG Belegnr.: 162872.867.054.346 13.10.2016 EUR 10017 474 437 Zahlungsempf.: Agentur MINT GmbH IBAN: AT622011182766763800 BIC: GIBAATWW ID: Zahlungsgrund: Zahlungsref.: Re 201610210 Auftraggeber: Dr. Harald Beck Valuta: 13.10 IBAN: AT941200010017474437 BIC: BKAUATWW ID: Betrag: EUR 4.140,00 Auftr.geb.Ref:

… Wenn man diesen Code drübersausen lässt:

```
ActiveCell.Value = Replace(ActiveCell.Value, vbCr, " ")
ActiveCell.Value = Replace(ActiveCell.Value, vbCrLf, " ")
ActiveCell.Value = Replace(ActiveCell.Value, vbLf, " ")
ActiveCell.Value = Replace(ActiveCell.Value, "     ", " ")
ActiveCell.Value = Replace(ActiveCell.Value, "    ", " ")
ActiveCell.Value = Replace(ActiveCell.Value, "    ", " ")
ActiveCell.Value = Replace(ActiveCell.Value, "   ", " ")
ActiveCell.Value = Replace(ActiveCell.Value, "  ", " ")
```

Zellinhalt auftrennen nach Zellumbruch

Sind in einer Zelle mehrere Zellumbrüche und dadurch einzelne Zeilen in der zelle, so kann man sie mit dieser Funktion (für Unitreu entworfen) auftrennen:

```
Sub Auftrennen()

' Dieser Code wandert die ab der aktuellen Zelle nach unten
' und trennt die Zeilen in jeder Zelle (ALT-ENTER-ZellUmbruch)
' in die rechts stehenden Spalten (die leer sein müssen !) auf

Dim Erstezeile As Integer
Dim Erstespalte As Integer
Dim Spaltenzaehler As Integer
Dim Inhalt As String ' gesamter Zellinhalt
Dim Naechsteszeichen As String ' jeweils 1 Zeichen für die Schleife durch Inhalt
Dim Z As Integer ' Schleife durch die Anzahl von Zeichen des Inhalts
Dim Ergebnis As String

Erstezeile = ActiveCell.Row
Erstespalte = ActiveCell.Column

Application.ScreenUpdating = False

' Schleife durch alle Zellen in dieser Spalte
For Zeile = Erstezeile To 9999

 Inhalt = Cells(Zeile, Erstespalte)
 Spaltenzaehler = 1

 ' wegschneiden von max. 10 Zeilenumbrüchen VOR der ersten Zeile mit Text
 For i = 1 To 10
    If Left(Inhalt, 1) = vbCr Or Left(Inhalt, 1) = vbLf Then
        Inhalt = Right(Inhalt, Len(Inhalt) - 1)
```

```
   End If
 Next i

 Ergebnis = "" ' Reset des Zieltextes

 ' Schleife durch alle Zeichen des Zellinhalts
 For Z = 1 To Len(Inhalt)
   Naechsteszeichen = Mid(Inhalt, Z, 1)
   ' MsgBox Asc(Naechsteszeichen = Mid(Inhalt, Z, 1)) ' ASCII-Code anzeigen

   If Naechsteszeichen <> vbCr And Naechsteszeichen <> vbLf Then ' solange das vbcr/vblf-Zeichen nicht kommt
      Ergebnis = Ergebnis & Naechsteszeichen
     Else
      ' schreiben des bisherigen Textes
      Cells(Zeile, Erstespalte + Spaltenzaehler) = Ergebnis
      Spaltenzaehler = Spaltenzaehler + 1
      Ergebnis = ""
      If Mid(Inhalt, Z + 1, 1) = vbCr Or Mid(Inhalt, Z + 1, 1) = vbLf Then Z = Z + 1 ' wenn das nächste Zeichen auch noch ein Spezialzeichen ist, dann auch
wegschneiden
   End If

 Next Z

 ' Schreiben des Rests

 Cells(Zeile, Erstespalte + Spaltenzaehler) = Ergebnis

Next Zeile


Application.ScreenUpdating = True

End Sub
```

Und so siehts aus

Hier ist mein Datensatz				
mit einer zweiten Zeile	Hier ist	mit einer	und einer	
und einer dritten Zeile	mein	zweiten	dritten	
und noch einer vierten Zeile	Datensatz	Zeile	Zeile	und noch einer vierten Zeile

FUNKTIONEN UMWANDELN VON VARIABLENTYPEN

Aus Text nur die Zahlen überlassen

Wenn in Zelle A1 z.B. der Text 1.2.3.4 steht, erhält man die Zahl 1234 durch folgenden Code

Dim Zahl as Integer

Zahl=Range("A1")

Wenn Zahl nicht als Zahl formatierbar ist

Ich (Stefan) hatte einen „Parser", der eine Zahl, die als Text eingelesen wurde, schön umwandelte in eine echte Zahl (Kommas / Tausenderpunkte etc korrigieren) – zuletzt hatte ich eine wunderschöne Zahl – aber die war noch immer ein String

(aus ZB 1.234.567.89 wurde 1234567,89)

Wenn ich nun diesen String einfügte in die Zelle, konnte ich die Zelle als Zahl formatieren, aber Excel machte das nicht – erst wenn ich manuell in die Zeile hineinging mit F2 und das ganze mit Return beendet, wurde ein echtes Zahlenformat draus.

Grund: der Zelleninhalt stammte aus einem String und das vergisst Excel nicht. Daher VOR dem einfügen in die Zelle, den String umwandeln in eine echte Zahl:

```
Cells(ZEILE, SPALTE).NumberFormat = "#,##0.00"
Cells(ZEILE, SPALTE) = CDbl(TEXTZAHLUMGEWANDELT)
```

Alternativ könnte man auch mit der SENDKEYS-Anweisung zuerst mit F2 die Zelle aktivieren können und mit Enter wieder übernehmen:

```
Range(STARTSPALTENTEXT & STARTZEILE & ":" & ENDSPALTENTEXT & ENDZEILE).Select
For Each BEREICH In Selection
    SendKeys "{F2}", True
    SendKeys "{ENTER}", True
Next BEREICH
```

WICHTIG: man darf bei der SENDKEYS-Anweisung die Bildschirmaktualisierung (Application.ScreenUpdating) nicht deaktivieren, sonst kommt es zu Fehlern oder Crashes

Zahl in Userform-Textfeld ausrechnen (Currency + EVALUATE)

List man ein Textfeld einer Userform direkt aus in eine Currency-Variable, dann kommt es zu einem Fehler, wenn man mit der Löschtaste den Zellinhalt leert und den Inhalt eines leeren Textfeldes an die Variable übergibt.

1.LÖSUNG - OHNE RECHENOPERATION

man fragt den Inhalt des Textfeldes vor der Übergabe aus und übergibt ein ev. leeres Textfeld direkt mit dem Wert 0:

```
If TextBox3.Value <> "" Then
        BRUTTO20 = TextBox3.Value
   Else
        BRUTTO20 = 0
End If
```

2.LÖSUNG - MIT RECHENOPERATION

Alternativ nimmt man eine Stringvariable, die die Zelle ausliest und konvertiert sie mittels CDBL-Funktion (die im Gegensatz zu Val ein "," als Komma deuten kann) in die Currency-Variable um.

Ich habe eine eigene Subroutine, die den Textfeldwert über die Stringvariable ZELLEINGABE erhält und das Ergebnis (ausgrechnet, falls Rechenoperationen enthalten waren in der Eingabe wie 1+1) zurückgibt über die Variable ZELLEKONVERTIERT.

Wichtig: man kann mit Sub TextBox5_**Change** und nicht nur mit Textbox5**_Exit** arbeiten - Vorteil: während Eingabe wird laufend das Ergebnis gerechnet und kann angezeigt werden

allerdings ist zusätzlicher Code notwendig um Eingaben wie "1+...." "1/..." "1/0..." etc abzufangen, da die Evaluate-Methode bei "1+" zu einem Fehler führt.

```
Sub ZELLKONVERTER()
   ' Dient für die Userforms um Beträge mit Rechenoperationen auszuwerten (zB 2,14+3,14) und wird von dort aus aufgerufen
   Dim I As Integer ' Laufvariable
   Dim RECHNEN As Integer ' merkt sich, ob die Eingabe eine Rechenoperation enhält
   Dim ZEICHEN As String ' zum Zerschneiden des Strings in einzelne Zeichen
   Dim ERGEBNIS As String ' zum Umwandeln von Variable ZELLEKONVERTIERT
   ZELLEKONVERTIERT = ""
   ERGEBNIS = ""
   RECHNEN = 0

   ' Abschneiden eventueller Rechenzeichen (+,-,=) am Ende der Eingabe
   If Right(ZELLEINGABE, 1) = "=" Or Right(ZELLEKONVERTIERT, 1) = "+" Or Right(ZELLEKONVERTIERT, 1) = "-" Then
```

```
        ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)
    End If
    ' Leider hat "evaluate"-Befehl einen Fehler - ein "evaulate ("24")" ergibt 90 und nicht 24 =>
    ' evaluate-Befehl nur ausführen, wenn eine Rechenoperation im String enthalten ist
    ' weiters verdaut "evaluate" keine "," - er nimmt nur "."
    ' und zudem löscht Excel beim Übergeben einer Zahl, die ein "." enthält, an eine Variable
    ' diesen Punkt (aus "2.12" wird 212)
    ' netterweise gibt der Evaluate-Befehl, der zwar "." braucht, das Ergebnis korrekt wieder mit "," aus
    ' => Im zu berechnenden String nur dann die "," in "." umwandeln, wenn anschließend ein Evaluate erfolgt
' Wenn Zelleingabe mit einem + o. - endet : hinzufügen von 0, damit gerechnet werden kann - aber das Ergebnis sich nicht ändert
    If Right(ZELLEINGABE, 1) = "+" Or Right(ZELLEINGABE, 1) = "-" Then ZELLEINGABE = ZELLEINGABE + "0"
' Wenn Zelleingabe mit * oder / endet : hinzufügen von 1, damit gerechnet werden kann, aber Ergebnis gleich bleibt
    If Right(ZELLEINGABE, 1) = "*" Or Right(ZELLEINGABE, 1) = "/" Then ZELLEINGABE = ZELLEINGABE + "1"
' Wenn Zelleingabe mit "/0" endet : abschneiden, damit gerechnet werden kann
    If Right(ZELLEINGABE, 2) = "/0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 2)
' Wenn Zelleingabe mit "/0," endet : abschneiden, damit gerechnet werden kann
    If Right(ZELLEINGABE, 3) = "/0," Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 3)
' Wenn Zelleingabe mit "/0,0" endet : abschneiden, damit gerechnet werden kann
    If Right(ZELLEINGABE, 4) = "/0,0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 4)
' Wenn Zelleingabe mit " " endet - abschneiden
    If Right(ZELLEINGABE, 1) = " " Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)
' KONTROLLE OB EINE RECHENOPERATION ENTHALTEN IST (nur dann wird Evaluate ausgeführt)
    For I = 1 To Len(ZELLEINGABE)
        ZEICHEN = Mid(ZELLEINGABE, I, 1)
        If ZEICHEN = "+" Or ZEICHEN = "-" Or ZEICHEN = "*" Or ZEICHEN = "/" Then RECHNEN = 1
    Next I
' WENN RECHENOPERATION ENTHALTEN DANN VOR EVALUATE DAS "," in "." umwandeln
    If RECHNEN = 1 Then
        ' Zeichenweise "umschaufeln" in Zielvariable und aus "," einen "." machen
        For I = 1 To Len(ZELLEINGABE)
            If Mid(ZELLEINGABE, I, 1) = "," Then
                ERGEBNIS = ERGEBNIS + "."
            Else
                ERGEBNIS = ERGEBNIS + Mid(ZELLEINGABE, I, 1)
            End If
        Next
        ERGEBNIS = Evaluate(ERGEBNIS)
      Else
        ERGEBNIS = ZELLEINGABE
    End If
   If ERGEBNIS = "" Then ERGEBNIS = "0"
   ERGEBNIS = Round(ERGEBNIS, 2)
' Anschließend das Ergebnis, falls mit "." wieder umwandeln auf ","
        For I = 1 To Len(ERGEBNIS)
```

```
        If Mid(ERGEBNIS, I, 1) = "." Then
            ZELLEKONVERTIERT = ZELLEKONVERTIERT + ","
        Else
            ZELLEKONVERTIERT = ZELLEKONVERTIERT + Mid(ERGEBNIS, I, 1)
        End If
    Next
End Sub
```

Anmerkung: Die Verwendung eckiger Klammern (etwa "[A1:C5]") entspricht dem Aufruf der **Evaluate**-Methode mit einem String-Argument. Die folgenden Ausdruckspaare beispielsweise haben dieselbe Bedeutung:

```
[a1].Value = 25
Evaluate("A1").Value = 25

trigVariable = [SIN(45)]
trigVariable = Evaluate("SIN(45)")

Set firstCellInSheet = Workbooks("BOOK1.XLS").Sheets(4).[A1]
Set firstCellInSheet = Workbooks("BOOK1.XLS").Sheets(4).Evaluate("A1")
```

VAL und CDBL

Ich hatte eine Zelle als Zahl formatiert und mittels WENN-Formeln wird eine Summe darin ermittelt. Die Formel und deren Quellzellen waren so beschaffen, dass das Ergebnis auch nur eine leer Zelle ergeben kann. Dieses konnte dann nicht mittels VBA ausgelesen und mit anderen Zahlen/Zell-Zahlen addiert werden, und ich bekam immer eine Fehlermeldung.

Lösung: NICHT mit CDBL arbeiten, weil dies nur dann Zellwerte in Zahlen umwandelt, wenn dieses irgendwo als Zahlen zu interpretierende Werte hat - aber nicht, wenn es NICHTS enthält. Mit NICHTS kommt nur VAL zurecht - daher habe ich die Zelle mit VAL(Range("A1") ausgelesen

Val erwartet übrigens den internationalen Punkt als Komma. daher:

Val("1234,5") ergeibt nur 1234

Arbeite stattdessen mit CDBL und schon erkennt es auch den Beistrich als Komma an

WICHTIG: CDBL("ABC") führt zu Fehlermeldung, da CDBL nur mit Strings zurecht kommt, die echte Zahlen darstellen - VAL("ABCD") hingegen ergibt den Wert 0 ohne Fehlermeldung

Verschlüsselung => Registrierung

Zelle mit Formel/Verweis umwandeln in Fixwert

Dazu gibt es viele Möglichkeiten. Am schnellsten ist

Range("D1:D10")= Range("D1:D10").value

INSTALLATION VON VORLAGEN

Allgemeines PERSONL.XLS und ADD-INS

Ich arbeite - wie bei den Powertools - sehr gerne mit einer Excel-Datei deren Fenster ausgeblendet ist und die nur eine Symbolleiste enthält und VBA-Code. In den XL-Start-Ordner kopiert, wird steht sie bei jedem Excelstart automatisch zur Verfügung.

In der Regel befinden sich alle Infos im VBA-Code. Möchte man jedoch auch mal irgendwelche Parameter in der Vorlage selbst speichern (um nicht die Windows-Registrierung bemühen zu müssen), schreibt man in eines ihrer Tabellenblätter die Werte hinein.

Dort sind sie sicher, da die Tabellenblätter durch das ausgeblendete Fenster der Exceldatei ohnedies nicht sichtbar sind.

Da jedoch alle Tabellenblatt-Befehle (Sheets, Activesheet) sich auf die aktuelle Arbeitsmappe beziehen - und durch das ausgeblendete Fenster diese Vorlagendatei NIE die aktuelle Arbeitsmappe sein kann (außer man blendet das Fenster ein - was auch per VBA gehen würde und mittels Deaktivieren der Screenupdating-Funktion auch unsichtbar bleiben könnte), darum muss man vor jedem Befehl, der auf eine Tabelle in der unsichtbaren Excelvorlage zugreifen soll immer ein 'ThisWorkbook." voransetzen - also zB ThisWorkbook.Sheets("Tabelle1").Range("B8")= …

Mit ThisWorkbook. zwingt man den VBA-Code in die Vorlage selbst hinein.

Hier zunächst eine Gegenüberstellung von personl.xls und Add-In, damit Sie für sich entscheiden, welches der passende Weg für Ihren Code ist:

	Vergleich	
	personl.xls	Add-In

Zweck	"Behälter" zum Abruf persönlicher Makros und persönlicher benutzerdefinierter Funktionen	"Behälter" zum Abruf allgemeiner Makros und allgemeiner benutzerdefinierter Funktionen
Übertragbarkeit	starke Bindung an den PC	kann leicht weitergegeben werden - bessere Übertragbarkeit an andere Anwender
Funktionsaufruf	bei Aufruf von Funktionen umständlichere Formel, z.B. =PERSONL.XLS!Gesperrt(B2)	bei Aufruf von Funktionen einfachere Formel, der Mappenname braucht nicht vorangestellt zu werden, z.B. =Gesperrt(B2)
excelweite Verfügbarkeit	da die Datei automatisch bei jedem Excelstart mitgeöffnet wird, ist der enthaltene Code immer verfügbar	ein Add-In ist nur verfügbar, wenn es installiert und aktiviert ist
Deaktivierung	dazu muss bei geschlossenem Excel die Datei personl.xls vorübergehend aus dem Verzeichnis XLStart in ein anderes Verzeichnis verschoben werden	einfach durch abhaken des entsprechenden Add-Ins unter Menü > Extras > Add-Ins…
Dateiendung	*.xls	*.xla
Speicherort	im aktuellen Verzeichnis XLStart	frei definierbar - empfehlenswert ist jedoch ein Sammelverzeichnis AddIn, um Dateien bei Installation nicht suchen zu müssen
Ausführung	Makros werden angezeigt unter Menü > Extras > Makro > Makros.... und können von dort aus auch aufgerufen werden	Makros werden nicht angezeigt unter Menü > Extras > Makro > Makros.... von daher können sie nicht auf diesem Weg aufgerufen werden
Tempo	kein messbarer Unterschied	
Offene Dateien	wird von Excel immer mitgezählt beim Zählen offener Dateien, auch wenn ausgeblendet	wird von Excel nicht mitgezählt beim Zählen offener Dateien; standardmäßig ausgeblendet

Wie erstellt man die Datei personl.xls?

Wie zu Beginn erwähnt, bringt Excel die Datei personl.xls nicht von Hause aus mit, Sie müssen sie selbst anlegen.

Damit Sie die personl.xls sicher im richtigen Verzeichnis XLStart anlegen, empfiehlt es sich, diese durch Aufzeichnung eines kleinen Makros zu erstellen. Dieses Makro kann ja später aus der Datei gelöscht werden.

Das gehe ich jetzt hier Schritt für Schritt mit Ihnen durch, am Beispiel eines simplen und ungefährlichen Makros:

- öffnen Sie eine neue Datei
- gehen Sie über *Menü* > *Extras* > *Makro* > *Aufzeichnen....*
- die Dialogbox *Makro aufzeichnen* springt auf
- geben Sie im Dialogfeld *Makroname* ihrem Makro einen aussagekräftigen Namen, z.B. blaue_Zelle, die anderen Voreinstellungen können Sie unverändert so beibehalten
- wählen Sie dort im Dropdown aus, dass Ihr Makro in der **Persönlichen Makroarbeitsmappe** gespeichert werden soll
- die Makroaufzeichnung starten Sie mit Klick auf die Schaltfläche *OK*
- klicken Sie auf das Füllfarbeicon und wählen blau aus
- danach stoppen Sie die Aufzeichnung durch Klick auf das *Aufzeichnen-Beenden-Kästchen* links auf der gleichnamigen Symbolleiste
- schließen Sie Excel - dabei springt die Rückfrage auf, ob die Änderungen der persönlichen Makroarbeitsmappe gespeichert werden sollen?
- bestätigen Sie diese mit *Ja* und ab nun haben Sie eine personl.xls, auch wenn Sie diese nicht sehen, da sie ausgeblendet ist. Excel vergibt den Dateinamen personl.xls automatisch und speichert ebenso automatisch in das Verzeichnis XLStart

Wo ist die Datei personl.xls gespeichert?

Die personl.xls muss zwingend im Verzeichnis **XLStart** gespeichert sein, sonst wird sie nicht beim Excel-Start mitgeöffnet und die enthaltenen Makros sowie benutzerdefinierten Funktionen stehen nicht zur Verfügung.

Je nach Windows-Version kann das XLStart-Verzeichnis an verschiedenen Stellen liegen. Haben Sie früher auf Ihrem System andere Excelversionen installiert gehabt, kann es auch sein, dass das XLStart-Verzeichnis mehrfach vorkommt. Es gibt aber jedenfalls nur ein aktuell Verwendetes und das müssen sie finden.

Suchen Sie direkt im Explorer nach der Datei personl.xls und lassen Sie sicherheitshalber den ganzen Arbeitsplatz mit Unterverzeichnissen durchsuchen:

- wenn diese nicht gefunden wird, existiert sie noch nicht und Sie müssen sie anlegen
- finden sich mehrere Dateien dieses Namens, dann wird die mit dem jüngsten Datum die aktuelle personl.xls im aktuell von Excel herangezogenen XLStart-Verzeichnis sein

Dies hier als Beispiel, warum Suche nach der aktuellen Version wichtig ist:
Bei mir selbst (Excel 2002) ist die aktuelle Datei in diesem Ordner gespeichert:
C:\Dokumente und Einstellungen\Schmitz\Application Data\Microsoft\Excel\XLStart
In anderen Versionen kann sie aber auch im Office-Verzeichnis liegen, so findet sich auch noch diese Altlast auf meinem System:
C:\Programme\Microsoft Office\Office\XLStart
Nun verstehen Sie wohl, wie wichtig es ist, die Aktuelle herauszufinden, denn nur die springt beim Excelstart an.

Pfad des aktuellen XLStart-Verzeichnisses per Makro ermitteln

Beide Codes gehören in ein Modul:

Möglichkeit 1: Ausgabe in einer MsgBox:

```
Sub ZeigePfadMeinesStartordners()
  MsgBox Application.StartupPath
End Sub
```

Möglichkeit 2: Ausgabe im Direktfenster; praktisch, wenn Sie ihn rauskopieren möchten:

```
Sub PfadMeinesStartordners_ins_Direktfenster()
  Debug.Print Application.StartupPath
End Sub
```

Abhilfe: Trotz korrektem Speicherort startet die Datei personl.xls nicht

Problem: Ihre personl.xls startet nicht beim Excelstart und Sie sind sich sicher, dass die Datei im aktuellen XLStart-Verzeichnis gespeichert ist.

Grund: Dann ist die Datei personl.xls von Excel deaktiviert worden, weil sie ein Problem verursacht hat.

Abhilfe: So können Sie die personl.xls (oder auch andere deaktivierte Elemente) reaktivieren: Guckst Du: Deaktivierte Elemente aktivieren

Wie blendet man die Datei personl.xls ein oder aus?

Wenn die personl.xls nicht gerade bearbeitet wird (dazu muss sie auch nicht zwingend eingeblendet ein, siehe hier), sollte sie ausgeblendet sein. Dann wird sie Ihnen auch nicht in der Liste der geöffneten Dateien aufgeführt. Zur Verfügung stehen die Makros und benutzerdefinierten Funktion auch im ausgeblendeten Zustand. Sollten Sie aus irgendeinem Grund die Datei eingeblendet haben, blenden sie diese nach Speichern Ihrer Änderungen auch bitte wieder aus. Beim Schließen von Excel werden Sie gefragt, ob Sie die Änderungen an der personl.xls speichern möchten, bestätigen Sie dies, weil diese Rückfrage sich auf den ausgeblendeten Zustand bezieht, den Inhalt haben Sie ja bereits gespeichert.

Um die **personl.xls einzublenden** gehen Sie über *Menü* > *Fenster* > *Einblenden....* . Dann wählen Sie mit der Maus im aufspringenden Dialogfenster die Datei personl.xls aus und verlassen den Dialog über die Schaltfläche OK.

Um die **personl.xls auszublenden** aktivieren Sie die Datei personl.xls und gehen über *Menü* > *Fenster* > *Ausblenden....*

Wie kann man die personl.xls nachträglich bearbeiten, ergänzen oder Code löschen?

Möglichkeit 1:

- dafür müssen Sie bei geöffnetem Excel die personl.xls wie hier beschrieben einblenden
- nun können Sie in die Module Code durch kopieren einfügen oder Code nachbearbeiten oder auch löschen. Dies funktioniert wie in jeder anderen Exceldatei; bei Bedarf siehe: Wie und wo fügt man ein Makro bzw. Code ein
- nach den gewünschten Änderungen speichern Sie die personl.xls und blenden sie wie hier beschrieben aus

Möglichkeit 2:

Wenn Sie sich in der Entwicklungsumgebung für VBA befinden (dahin gelangen Sie über die Tastenkombination Alt + F11), können Sie auch von dort aus direkt auf die Module der personl.xls zugreifen und deren Inhalt bearbeiten. Dann ist Ein- und Ausblenden nicht nötig. Denken Sie aber ans Speichern Ihrer Änderungen.

Möglichkeit 3:

Sie können natürlich auch per Aufnahme Makros der personl.xls zufügen, wenn Sie zu Beginn der Aufnahme im Dropdown ausgewählt haben, dass Ihr Makro in der **Persönlichen Makroarbeitsmappe** gespeichert werden soll. Dabei entfallen die Arbeitsschritte ein- und ausblenden.

Wie löscht man die Datei personl.xls?

Löschen können Sie die aktuelle personl.xls nur bei geschlossenem Excel über den Explorer. Dies nur für den Fall, falls Ihre derzeitige nur unrelevanten Code enthält, der nicht im Einsatz ist. Dann können Sie sich schneller eine neue personl.xls anlegen, als die Bestehende umfangreich aufzuräumen.

Finden können Sie im Explorer die aktuelle personl.xls auf diesem Weg.

Also: Vorsicht mit diesem Schritt, wenn Sie unsicher sind, ob Code aus der aktuellen personl.xls irgendwo im Einsatz ist!

Wie überträgt man die Datei personl.xls auf einen anderen PC?

Lesen Sie diesen Absatz bitte vor Durchführung komplett, weil Gefahren lauern!

Sie können diese Datei wie jede andere Datei kopieren und per Mail oder Datenträger auf einem anderen PC übertragen. Auf dem Quell- sowie dem Zielcomputer finden Sie im Explorer die aktuelle personl.xls bzw. das aktuelle XLStart-Verzeichnis auf diesem Weg.

Wenn auf dem Zielcomputer keine personl.xls existiert, legen Sie dort eine neue personl.xls an, um alle Unsicherheiten über den korrekten Pfad auszuräumen. Die so entstandene neue personl.xls können Sie dann an der richtigen Stelle über den Explorer mit Ihrer Version der Datei überschreiben.

Existiert auf dem Zielcomputer bereits eine aktuelle personl.xls ist Vorsicht geboten!!! Eventuell greifen Dateien auf Code dieser Datei zu, die nach Überschreiben der personl.xls durch Ihre Version ins Leere liefen. Dann benennen Sie lieber Ihre Datei bis Sicherheit besteht in personl2.xls um und fügen Sie danach erst ins aktuelle XLStart-Verzeichnis ein. Dann stehen Ihnen ab dem nächsten Excelstart ausgeblendet beide Dateien mit ihrem Codeangebot zur Verfügung.

Wie ruft man ein Makro der personl.xls aus einer anderen Datei heraus auf?

Berücksichtigen Sie grundsätzlich:
Die Routine muss eine Sub sein, darf nicht als Private definiert sein und darf keine Parameter besitzen, sonst geht es nicht.

Möglichkeit 1: Per Tastenkombination

Sie weisen dem Makro der personl.xls eine Tastenkombination zu. Dann springt das Makro aus jeder Excel-Datei heraus an, wenn Sie diese Tastenkombination drücken (also in Kombination mit *Strg* oder *Strg+Umschalt*). Da man sich nur beschränkt Tastenkombinationen merken kann, sollte man diesen Weg nur bei sehr häufig genutzten Makros wählen. Nachträglich können Sie eine Tastenkombination wie folgt zuweisen:

- gehen Sie über *Menü > Extras > Makro > Makros....*
- wählen Sie im Dropdown unter *Makros in* die Personl.xls aus

- markieren Sie mit der Maus das gewünschte Makro, so dass es im Feld *Makroname* erscheint
- klicken Sie auf die Schaltfläche *Optionen...*
- ein weiteres Dialogfenster springt auf und Sie können dort die gewünschte Tastenkombination eingeben
- berücksichtigen Sie, welche Tastenkombinationen Sie bereits vergeben haben bzw. welche Tastenkombinationen von Excel selbst vorbelegt sind
- schließen Sie das Dialogfenster über die Schaltfläche *OK*
- verlassen Sie das nun noch offene Dialogfenster *Makro* über das Schließenkreuz

Möglichkeit 2: Aus einem Makro heraus

Über ein Makro in der aufrufenden Datei, so wird z.B. unser oben erstelltes Makro blaue_Zelle aufgerufen:

```
Sub Aufruf_blaue_Zelle()
Application.Run ("Personl.xls!Blaue_Zelle")
End Sub
```

Auch in längeren Code der Ausgangsdatei kann diese Codezeile an der passenden Stelle eingebunden werden.

Möglichkeit 3: Über eine Schaltfläche auf einer Symbolleiste

Für häufig genutzte Makros der personl.xls lohnt sich eventuell der Einbau einer entsprechenden Schaltfläche, der Sie das Makro zuweisen. Das geht so:

- gehen Sie über *Menü > Ansicht > Symbolleisten > Anpassen.... > Registerblatt: Befehle*
- wählen Sie unter *Kategorien:* Makros aus
- unter Befehle sehen sie ein Smiley-Symbol (das ist austauschbar, aber wir nehmen das für unser Beispiel), dieses markieren Sie durch anklicken
- ziehen Sie das Symbol mit der Maus an die gewünschte Stelle in der gewünschten Symbolleiste
- klicken Sie nun das neu erstellte Symbol mit der rechten Maustaste an und wählen dann aus: *Makro zuweisen...*
- das Dialogfenster *Makro zuweisen* springt auf
- wählen Sie im Dropdown unter *Makros in* die Personl.xls aus
- markieren Sie mit der Maus das gewünschte Makro, so dass es im Feld *Makroname* erscheint

- verlassen Sie das Dialogfenster über die Schaltfläche *OK*
- schliessen Sie nun das Dialogfenster *Anpassen* über die Schaltfläche *Schließen*

Ab nun startet Ihr Makro bei Klick auf die Schaltfläche. Diese Änderung wirkt sich auf Ihre *.xlb-Datei aus, in der Ihre Menüeinstellungen und Symbolleisten hinterlegt sind.

Das Icon können Sie auch wieder löschen, indem Sie

- über *Menü > Ansicht > Symbolleisten > Anpassen.... > Registerblatt: Befehle* gehen
- wenn das Dialogfenster aktiv ist, können Sie das Icon einfach aus der Symbolleiste mit der Maus auf das Dialogfenster ziehen
- dieses dann über die Schaltfläche *Schließen* beenden

Möglichkeit 4: Zu Fuß

- gehen Sie über *Menü > Extras > Makro > Makros....*
- die Dialogbox *Makro* springt auf
- wählen Sie im Dropdown unter *Makros in* die Personl.xls aus
- markieren Sie mit der Maus das gewünschte Makro, so dass es im Feld *Makroname* erscheint
- klicken Sie auf die Schaltfläche *Ausführen*

Wie benutzt man eine benutzerdefinierte Funktion der personl.xls in einer anderen Datei?

Wenn Sie eine benutzerdefinierte Funktion in der personl.xls haben, können Sie diese Funktion aus jeder Exceldatei heraus einsetzen. Sie müssen allerdings in der Formel bzw. im Code angeben, dass sie sich in der personl.xls befindet.

Möglichkeit 1: Als Funktion in einer Zelle

Beispiel: Angenommen, Sie haben diese benutzerdefinierte Funktion *Gesperrt* in ihrer personl.xls:

```
Function Gesperrt(ByVal Text As String) As String
Dim n As Integer
   If Text <> "" Then
      For n = 1 To Len(Text)
      Gesperrt = Gesperrt & Mid(Text, n, 1) & " "
```

```
      Next
   End If
End Function
```

Dann muss die Formel in einer Zelle (hier C2) so lauten, damit sie ausgeführt wird:

	A	B	C	D
1				
2		Anton Gimpel	A n t o n G i m p e l	
3				

Formeln der Tabelle

C2 : =PERSONL.XLS!Gesperrt(B2)

Anmerkung: Wäre der gleiche Code in einem Add-In würde diese Formel ausreichen: =Gesperrt(B2)

Möglichkeit 2: Funktion der personl.xls in ein Makro einer anderen Datei einbinden - ohne Parameter

Wie schon oben erwähnt, muss die Routine eine Sub sein, darf nicht als Private definiert sein und darf keine Parameter besitzen. Deshalb müssen wir hier einen Umweg gehen.

Zusätzlich zu obiger Funktion *Gesperrt* müssen wir in die personl.xls folgendes Makro einfügen:

```
Sub Aufruf_Gesperrt()
   MsgBox Gesperrt(ActiveCell)
End Sub
```

Und in die aufrufende Datei dann folgendes Makro (dieses Makro hier gibt dann den Inhalt der aktiven Zelle in einer MsgBox in gesperrter Schrift wieder):

```
Sub Aufruf_Gesperrt2()
   Application.Run ("Personl.xls!Aufruf_Gesperrt")
End Sub
```

Möglichkeit 3: Funktion der personl.xls in ein Makro einer anderen Datei einbinden - mit Parametern

Wenn Sie der Run-Methode Parameter mitgeben wollen, müssen Sie folgendes beachten:

- rufen Sie eine Sub oder eine Funktion ohne Rückgabewert auf, dann darf der Ausdruck nicht in Klammern stehen
- rufen Sie eine Funktion mit Rückgabewert auf, dann muss der Ausdruck in Klammern stehen.

Dafür verändern wir unsere bisherige benutzerdefinierte Funktion *Gesperrt* wie folgt in der personl.xls:

```
Public Function Gesperrt3(ByVal sText As String, _
    ByRef iBlanks As Integer) As String
    Dim iCount As Integer
    For iCount = 1 To Len(sText) - 1
        Gesperrt3 = Gesperrt3 & Mid$(sText, iCount, 1) & _
            String(iBlanks, vbKeySpace)
    Next
    Gesperrt3 = Gesperrt3 & Right$(sText, 1)
End Function
```

Aufruf aus einer anderen Datei ohne Rückgabewert ohne Klammern:

```
Sub Aufruf_Gesperrt3()
    Application.Run "Personl.xls!Gesperrt3", ActiveCell.Text, 2
End Sub
```

Aufruf aus einer anderen Datei mit Rückgabewert mit Klammern:

```
Sub Aufruf_Gesperrt4()
    MsgBox Application.Run("Personl.xls!Gesperrt3", ActiveCell.Text, 4)
End Sub
```

Wie schützt man die Datei personl.xls vor Virenattacken oder ungewollter Veränderung?

Grund für folgenden Tipp ist, dass Viren häufig so programmiert werden, dass sie sich in genau diese Datei einschleichen.

Nachdem Sie Ihre wichtigen Codes in die Datei eingefügt haben, sollten Sie überlegen, ob Sie diese Datei nicht schreibschützen wollen. Finden können Sie im Explorer die aktuelle personl.xls auf diesem Weg. Das Setzen des Schreibschutzes funktioniert so:

- Rechtsklick im Explorer auf die Datei personl.xls

- *Eigenschaften* anklicken
- im *Registerblatt Allgemein* haken Sie unter *Dateiattribute: "Schreibgeschützt"* an
- und verlassen dann das Dialogfenster über die Schaltfläche *OK*

Dieser Weg schützt Ihre Datei weitmöglichst auch vor ungewollter und unbemerkter Veränderung durch Sie selbst oder Mitbenutzer, neben dem "kleinen" Schutz, dass sie normalerweise ausgeblendet ist.

Natürlich können Sie die Datei trotzdem später noch verändern, indem Sie vorübergehend dann den Schreibschutz wieder aufheben.

Arbeiten mit Verknüpfungen

Man muss eine Vorlagendatei, die unsichtbar sein soll und nur eine Symbolleiste dem User anbieten soll, nicht zwingend in den XLStart-Ordner von Excel kopieren, damit sie bei jedem Excelstart automatisch ihre Funktionen und ihre Symbolleiste zur Verfügung stellt.

Man kann auch nur eine Verknüpfung dieser Datei in den XLStart-Ordner kopieren.

Vorteil: Man kann eine Vorlage zentral im Netzwerk anlegen und der Mitarbeiter, der damit arbeitet, bekommt nur eine Verknüpfung davon in seinen XLStart-Ordner. Die Wartung kann aber nach wie vor in der Vorlage im Netzwerk erfolgen.

Nachteil: man kann diese Vorlage dann nicht wirklich fein an mehrere Mitarbeiter ausrollen, die gleichzeitig damit arbeiten wollen. Denn der zweite erhält dann beim Öffnen von seinem Excel schon den Hinweis, dass die Datei von jemand anderem genutzt wird und dass er sie nur schreibgeschützt öffnen kann.

Dateitypen

man kann eine xls - Datei manuell umbenennen auf xlt oder xla und sie funktionieren gleich. Wenn in den Ordner XLStart kopiert wird eine XLT-Datei NICHT wie normal geöffnet (nämlich als XLS-Datei mit einem angehängten 1 - zB Test.xlt wird zu Test1.xls - sondern immer als originale Test.XLT.

Sonst wäre es ja nett gewesen von einer XLT-Vorlage beliebigig viele Links in die XLStart-Ordner mehrerer Mitarbeiter im Netzwerk zu kopieren und sie arbeiten dann alle mit einer zentralen XLT-Vorlage, ohne sich gegenseitig zu stören. Änderungen im Code könnten aber zentral gemacht werden und müssten nicht immer ausgerollt werden

Installation mittels BATCH-Datei

Eine eigene Setuproutine ist natürlich was feines, weil dann auch die Deinstallation klappt - (wobei man die Symbolleiste dann dennoch manuell entfernen muss :o)

Mittels BATCH-Datei kann aber eine Excel-Vorlage auch recht fein installiert werden - für Excel lautet ihr Inhalt zB

xcopy Symbol.xls "%userprofile%\Anwendungsdaten\Microsoft\Excel\XLStart" /R /Y

Verwenden des XLStart-Ordners

Office / Excel hat zwei Möglichkeiten, um ein Add-In bei jedem Excel-Start automatisch zu starten: den XLStart-Ordner - es gibt ihn zweimal :

C:\Dokumente und Einstellungen\[username]\Anwendungsdaten\Microsoft\Excel\XLSTART
C:\Programme\Microsoft Office\OFFICE11\XLSTART

Egal wo ein Addin ist - es wird bei jedem Excelstart automatisch ausgeführt.

Ganz großer Vorteil: eine normale Excelvorlage mit VBA-Code führt bei jedem Aufruf - wenn die Sicherheitseinstellungen nicht grad auf niedrig gestellt sind - eine Abfrage an den User oder wird gleich ganz gesperrt. Dies gilt jedoch nicht für Vorlagen, die in diesem XLStart-Ordner sind

Theoretisch kann man bei Einzelplatzinstallationen auch eine Verknüpfung in diesen Ordner kopieren.

Vorlage / Addin soll sich automatisch installieren

Meine Powertools maile ich an jemand - und er öffnet sie.

Die Powertools sollen automatisch erkennen, dass die aktuelle Datei NICHT im XLSTART-Ordner ist und vorschlagen, dass es sich gleich dorthin von selbst speichert um in Zukunft automatisch vorhanden zu sein.

Die Lösung ist allerdings nicht optimal, weil es nur bei der Erstinstallation geht. Wenn die Powertools schon installiert sind, dann werden Sie beim nächsten Start automatisch geöffnet und die neue Vorlagenversion startet gar nicht mehr.

Aber die Theorie dennoch:

In DieseArbeitsmappe kommt folgender Code:

```
Private Sub Workbook_Open()
 INSTALLATIONS_PRUEFUNG
End Sub
```

In ein Modul kommt der Rest:

```
' ------------------------
' KONSTANTEN
' ------------------------
```

```
Private Const CSIDL_ADMINTOOLS          As Long = &H30   '{user}\Start Menu _                                    '\Programs\Administrative Tools
Private Const CSIDL_COMMON_ADMINTOOLS    As Long = &H2F   '(all users)\Start Menu\Programs\Administrative Tools
Private Const CSIDL_APPDATA             As Long = &H1A   '{user}\Application Data
Private Const CSIDL_COMMON_APPDATA       As Long = &H23   '(all users)\Application Data
Private Const CSIDL_COMMON_DOCUMENTS     As Long = &H2E   '(all users)\Documents
Private Const CSIDL_COOKIES             As Long = &H21
Private Const CSIDL_HISTORY             As Long = &H22
Private Const CSIDL_INTERNET_CACHE       As Long = &H20   'Internet Cache folder
Private Const CSIDL_LOCAL_APPDATA        As Long = &H1C   '{user}\Local Settings\Application Data (non roaming)
Private Const CSIDL_MYPICTURES          As Long = &H27   'C:\Program Files\My Pictures
Private Const CSIDL_PERSONAL            As Long = &H5    'My Documents
Private Const CSIDL_PROGRAM_FILES        As Long = &H26   'Program Files folder
Private Const CSIDL_PROGRAM_FILES_COMMON As Long = &H2B   'Program Files\Common
Private Const CSIDL_SYSTEM              As Long = &H25   'system folder
Private Const CSIDL_WINDOWS             As Long = &H24   'Windows directory or SYSROOT()
Private Const CSIDL_FLAG_CREATE = &H8000&               'combine with CSIDL_ value to force
Private Const MAX_PATH = 260
Private Const SHGFP_TYPE_CURRENT = &H0
Private Const S_OK = 0

Private Declare Function SHGetFolderPath Lib "shfolder" _
       Alias "SHGetFolderPathA" (ByVal hwndOwner As Long, _
       ByVal nFolder As Long, ByVal hToken As Long, _
       ByVal dwFlags As Long, ByVal pszPath As String) As Long




' ------------------------
' FUNKTION
' ------------------------

Function ActualApplicationDataFolder() As String

    Dim lngCSIDL   As Long
    Dim strPath    As String
    Dim lngReturn  As Long

    lngCSIDL = CSIDL_APPDATA
    strPath = String(MAX_PATH, 0)

    'Get the folder's path.
    'If the "Create" flag is used, the folder will be created if it does not exist.
```

```
    lngReturn = SHGetFolderPath(0, lngCSIDL, 0, SHGFP_TYPE_CURRENT, strPath)
'    lngReturn = SHGetFolderPath(0, lngCSIDL Or CSIDL_FLAG_CREATE, 0, SHGFP_TYPE_CURRENT, strPath)

    Select Case lngReturn
    Case S_OK
        ActualApplicationDataFolder = Left$(strPath, InStr(1, strPath, Chr(0)) - 1)

    Case Else
        ActualApplicationDataFolder = ""

    End Select

End Function



' ------------------------
' Prozedur
' ------------------------


Sub INSTALLATIONS_PRUEFUNG()

Dim Wahl

If ThisWorkbook.Path <> ActualApplicationDataFolder & "\Microsoft\Excel\XLSTART" Then

    Wahl = MsgBox("Die Vorlage wurde nicht im Excelstartordner geöffnet." & vbCrLf & vbCrLf & _
    "Falls Sie die Vorlage zum ersten Mal erhalten, kann sie direkt in Ihren Startordner kopiert werden," & vbCrLf & _
    "damit sie in Zukunft automatisch startet" & vbCrLf & vbCrLf & _
    "Möchten Sie diese Vorlage in Zukunft automatisch starten lassen ?", vbYesNoCancel, "Frage")

    If Wahl = vbYes Then
        ThisWorkbook.SaveAs ActualApplicationDataFolder & "\Microsoft\Excel\XLSTART\Testinstallation.xls"
      Else
        MsgBox "Die Vorlage wurde nicht installiert."
    End If
Else ' Vorlage startet schon im richtigen Ordner => nichts tun
End If
End Sub
```

MAC IOS – UNTERSCHIEDE

Es soll eigentlich kaum Unterschiede geben beim VBA-Code. Ausnahme alles, was mit Dateinamen zu tun hat und die Zeilenumbrüche.

Was auch nicht geht sind die ACTIVE-X-Steuerelemente – die normalen Formularsteuerelemente gehen schon.

VBA war bereits unter Office 2004 mit gewissen Einschränkungen recht brauchbar und mit Office 2011 ist es weitgehend identisch mit VBA unter Office 2010. Nur bei Office 2008 gabs auf dem Mac keine VBA-Umgebung.

Die paar wenigen eingebauten VBA-Funktionen, die unter Office:mac fehlen, lassen sich mit ein paar Zeilen Code leicht nachbauen.

Mit folgender Funktion kann man abrufen, ob man auf einem MAC ist:

```
Function isMac() As Boolean
isMac = False
If InStr(Application.OperatingSystem, "Macintosh") Then
        isMac = True
    End If
End Function

Ganz ähnliches macht dieser Code:

Sub PCorMAC()
   If Application.OperatingSystem Like "*Mac*" Then
      importPicturesMac
   Else
      importPicturesWin
   End If
End Sub
```

Unterschied 1: Zeilenumbrüche entfernen

```
If (isMac()) Then
    zeilenumbruch = 13
Else
    zeilenumbruch = 10
End If

ActiveSheet.UsedRange.Replace Chr(zeilenumbruch), ""
```

Unterschied 2: Datei-Pfade

```
sheetName = ActiveSheet.Name
    If (isMac()) Then
        sheetPath = ActiveWorkbook.Path & ":"
    Else
        sheetPath = ActiveWorkbook.Path & "\"
    End If

    ActiveWorkbook.SaveAs Filename:= _
         sheetPath & sheetName & ".txt" _
        , FileFormat:=xlText, CreateBackup:=False
```

Unter OS X ist der Trenner für Pfade nicht "\" (Backslash) sondern "/" Slash). Auch gibt es z.B. keine Laufwerksbuchstaben wie z.B. "C:\". Das muss bei der Bildung von Pfaden in Programmen beachtet werden.

Was als Trenner verwendet wird kommt auf die Programmiersprache bzw die Umgebung an. Im Terminal ist es der Slash, bei Applescript meistens der Doppelpunkt. VB scheint auch den Doppelpunkt zu akzeptieren. Wenn man nicht weiß was gerade aktuell ist läßt man sich halt einen Pfad ausgeben und schaut dann die Ausgabe an. Da oben im Quellcode ein Applescript für Teilfunktionen verwendet wird wird dort eben der ":" verwendet.

MENÜLEISTEN UND SYMBOLLEISTEN

Allgemeines

Menü- und Symbolleisten sind sowohl manuell wie auch über VBA zu erstellen, zu verändern und zu löschen.

Seit der Excel-Version 8.0 (Office 97) handelt es sich bei den Menü- und Symbolleisten um das Objektmodell der **Commandbars** mit den zugehörigen **Control**-Elementen *CommandBarButton*, *CommandBarPopUp* und *CommandBarComboBox* unter dem Oberbegriff *CommandBarControl*.

Grundsätzlich empfiehlt es sich, zu einer Arbeitsmappe gehörende CommandBars oder CommandBarControls beim Öffnen der Arbeitsmappe über das **Workbook_Open**-Ereignis zu erstellen und über das **Workbook_BeforeClose**-Ereignis zu löschen. Nur so ist gewährleistet, dass der Anwender nicht durch Auswirkungen von CommandBar-Programmierungen oder -Anbindungen belästigt wird.

Der Commandbars-Auflistung fügt man mit der **Add**-Methode eine neue Leiste hinzu. Erfolgt die Erstellung der neuen CommandBar in einem Klassenmodul, ist die Syntax **Application.CommandBars.Add...** zwingend erforderlich, erfolgt die Erstellung in einem Standardmodul, reicht ein **CommandBars.Add...**. Um später mögliche Kollisionen mit anderen Office-Anwendungen zu vermeiden, wird allerdings auch hier die **Application**-Nennung empfohlen.

Die Add-Methode kann mit bis zu 4 Parameter aufgerufen werden:

- **Name**
  Der Name der Symbolleiste, zwingend erforderlich

- **Position**
  optional, folgende Konstanten sind möglich:

    - msoBarLeft (am linken Bildschirmrand)

- msoBarRight (am rechten Bildschirmrand)
- msoBarTop (wird an die bestehenden Symbolleisten angegliedert)
- msoBarBottom (am unteren Bildschirmrand, über der Statusleiste)
- msoBarFloating (nicht verankerte Symbolleiste, die Position kann festgelegt werden)
- msoBarPopUp (Kontext-Symbolleiste, mit der rechten Maustaste im Tabellenblatt aufrufbar)

- **MenuBar**
  optional, legt fest, ob es sich um eine Menü- oder eine Symbolleiste handelt (TRUE = Menüleiste, FALSE = Symbolleiste, Voreinstellung ist FALSE).

- **Temporary**
  optional, legt fest, ob die Menü- oder Symbolleiste mit Microsoft Excel geschlossen werden soll (TRUE = temporär, FALSE = bestehenbleibend, Voreinstellung ist FALSE). Wird also TRUE festgelegt, wird die CommandBar gelöscht, wenn Excel geschlossen wird und taucht auch in der CommandBar-Auflistung nicht mehr auf.

EIN PAAR BEISPIELE:

1. Untermenüs durch Makro erstellen

```
Sub MenuErstellen()
   Dim MB As CommandBar
   Dim Ctrl1 As CommandBarControl
   Dim Ctrl2 As CommandBarControl
   Dim Ctrl1a As CommandBarControl
   Dim Ctrl1b As CommandBarControl
   Set MB = CommandBars.Add(Name:="Neues Menü", MenuBar:=True)
   Set Ctrl1 = MB.Controls.Add(Type:=msoControlPopup)
   Ctrl1.Caption = "Untermenü1"
   Set Ctrl2 = MB.Controls.Add(Type:=msoControlPopup)
   Ctrl2.Caption = "Untermenü2"
   Set Ctrl1a = Ctrl1.Controls.Add(Type:=msoControlPopup)
   Ctrl1a.Caption = "Daten"
   Set Ctrl1b = Ctrl1.Controls.Add(Type:=msoControlPopup)
   Ctrl1b.Caption = "Übertragen"
   CommandBars("Neues Menü").Visible = True
End Sub
```

2. Menü "Symbolleisten" deaktivieren/aktivieren

```
Sub DisableToolbarMenu()
      CommandBars("Toolbar List").Enabled = False
End Sub
```

```
Sub DisableToolbarMenu()
    CommandBars("Toolbar List").Enabled = True
End Sub
```

3. Menüs dynamisch ein- und ausblenden

```
Private Sub Workbook_Activate()
MenuBars(xlWorksheet).Menus.Add "&Test Menü"
Set ml = MenuBars(xlWorksheet).Menus("Test Menü")
With ml
.MenuItems.Add Caption:="&Daten erfassen", _
OnAction:="DatenSpeichern"
.MenuItems.AddMenu Caption:="&Auswertungen"
With .MenuItems("Auswertungen")
.MenuItems.Add Caption:="&Auswertung1", _
OnAction:=""
.MenuItems.Add Caption:="A&uswertung2", _
OnAction:=""
End With
End With
End Sub
Private Sub Workbook_Deactivate()
MenuBars(xlWorksheet).Reset
End Sub
Private Sub Workbook_Open()
MenuBars(xlWorksheet).Menus.Add "&Test Menü"
Set ml = MenuBars(xlWorksheet).Menus("Test Menü")
With ml
.MenuItems.Add Caption:="&Daten erfassen", _
OnAction:="DatenSpeichern"
.MenuItems.AddMenu Caption:="&Auswertungen"
With .MenuItems("Auswertungen")
.MenuItems.Add Caption:="&Auswertung1", _
OnAction:=""
.MenuItems.Add Caption:="A&uswertung2", _
OnAction:=""
End With
End With
End Sub
```

4. Symbolleisten ausblenden

```
Sub Verstecken()
For Each tb in Toolbars
```

```
tb.Visible = False
Next tb
End Sub
```

5. Shortcut-Menü ein- und ausschalten

```
Sub ShortCutOnOff()
Application.ShortcutMenus(xlWorksheetCell).Enabled = False
End Sub
```

6. Icons in Symbolleiste deaktivieren

```
Sub SymbolGrauen()
CommandBars("Standard").Controls(1).Enabled = False
End Sub
```

7. Symbolleiste positionieren

```
Sub NeueSymbolleiste()
Dim cmdB As CommandBar
   Set cmdB = CommandBars.Add("MyToolbar", temporary:=True)
   With cmdB
      .Left = 50
      .Top = 100
      .Visible = True
   End With
End Sub
```

8. ID von Symbolleisten und Symbolen auslesen

Drei verschiedene Makros könne verwendet werden:

1. CommandBarControlID_List liefert die IDs der Symbolleisten mit Menüpunkt, ID-Nr und Beschreibung

2. CommandBarFaceID_List liefert alle FaceIDs mit Bild und ID

3. CommandBar_List liest die Excel-internen Bezeichnungen der Menüs, Menüpunkte, deren Typ und ID aus

```
Dim cbb As CommandBarButton, ComBar As CommandBar, cbc As CommandBarControl

Sub CommandBarControlID_List()
Dim a, b, c
Application.ScreenUpdating = False
```

```
For Each ComBar In Application.CommandBars
If ComBar.Name = "test" Then ComBar.Delete
Next
Set ComBar = Application.CommandBars.Add(Name:="test",
Position:=msoBarTop)
b = 0
c = 1
For a = 1 To 50000
On Error Resume Next
Set cbb = ComBar.Controls.Add(Id:=a)

If Err.Number <> 0 Then GoTo weiter
cbb.CopyFace
With Workbooks("FaceIDs").Sheets(1)
.Cells((c Mod 100) + 1, (c \ 100) + b + 1).Formula = a
.Cells((c Mod 100) + 1, (c \ 100) + b + 2).Activate
ActiveSheet.Paste
.Cells((c Mod 100) + 1, (c \ 100) + b + 3).Formula = cbb.Caption
End With
If (c + 1) Mod 100 = 0 Then b = b + 3
c = c + 1
weiter:
Application.CommandBars("test").FindControl(Id:=a).Delete
Err.Clear
Next
End Sub

Sub CommandBarFaceID_List()
Dim a, b
Application.ScreenUpdating = False
For Each ComBar In Application.CommandBars
If ComBar.Name = "test" Then ComBar.Delete
Next
On Error Resume Next
Set ComBar = Application.CommandBars.Add(Name:="test",
Position:=msoBarTop)
Set cbb = ComBar.Controls.Add(Id:=1)
b = 0

For a = 1 To 3518
With cbb
.FaceId = a
.CopyFace
End With
```

```
With ThisWorkbook.Sheets(1)
.Cells((a Mod 100) + 1, (a \ 100) + b + 1).Formula = a
.Cells((a Mod 100) + 1, (a \ 100) + b + 2).Activate
ActiveSheet.Paste
End With
If (a + 1) Mod 100 = 0 Then b = b + 2
Next
End Sub

Sub CommandBar_List()
Application.ScreenUpdating = False
Dim a, b, c, cbc, d
b = 1
d = 0
For Each a In Application.CommandBars
Cells(b + d, 1) = a.Name
Cells(b + d, 2) = "Item-no: " & b
For Each cbc In a.Controls
d = d + 1
Cells(b + d, 3) = cbc.Caption
Cells(b + d, 4) = Cells(cbc.Type, 10)
Cells(b + d, 5) = "Type: " & cbc.Type
Cells(b + d, 6) = "ID: " & cbc.Id
Next
b = b + 1
Next
End Sub
```

9. Menüeintrag neuen Befehl zuordnen

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target
As Excel.Range, Cancel As Boolean)

Set chgEinfügen =
Application.ShortcutMenus(xlWorksheetCell).MenuItems("Einfügen")
With chgEinfügen
.OnAction = "mkrEinfügen"
End With
End Sub
Sub mkrEinfügen()
Selection.PasteSpecial Paste:=xlValues
End Sub
```

10. Quickinfo zuordnen

```
Sub QuickInfo()
    Application.Toolbars("SybolleistenName").
       _ToolbarButtons(Indexzahl).Name = "Infotext"
End Sub
```

Ausblenden Menüleiste, Registerkarten, Beschriftungen, Leisten ab Excel 2007

Siehe auch MINIMIEREN und MAXIMIEREN VON MENÜBAND

BEISPIEL VON REISEKOSTENERFASSUNG

```
Sub EDITOR_MODUS_EIN_AUS()

' Office-Wechselschalter, der den Blattschutz, die Spaltenüberschriften-Anzeige und die Menüleisten hin und her schaltet

    If ActiveWindow.DisplayHeadings = False Then
        ActiveWindow.DisplayHeadings = True
        MenuebandMaximieren
        BLATTSCHUTZ_AUS
      Else
        ActiveWindow.DisplayHeadings = False
        MenuebandMinimieren
        BLATTSCHUTZ_EIN
    End If
End Sub


Public Function MenuebandIstMinimiert() As Boolean
   'Abfragen, ob das Menüband (Ribbon) minimiert ist
   MenuebandIstMinimiert = (CommandBars("Ribbon").Controls(1).Height < 100)
End Function

Public Sub MenuebandMinimieren()
  'Menüband minimieren
  'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.
   If MenuebandIstMinimiert = False Then
```

```
    CommandBars.ExecuteMso "MinimizeRibbon"
  End If
End Sub

Public Sub MenuebandMaximieren()
  'Menüband maximieren
  'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.
  If MenuebandIstMinimiert Then
    CommandBars.ExecuteMso "MinimizeRibbon"
  End If
End Sub

Sub BLATTSCHUTZ_EIN()
   ' Wenn Tabellenblattschutz nicht gesetzt ist
   If ActiveSheet.ProtectContents = False Then
      ActiveSheet.Protect DrawingObjects:=True, Contents:=True, Scenarios:=True
   End If
End Sub

Sub BLATTSCHUTZ_AUS()
   ' Wenn Tabellenblattschutz gesetzt ist
   If ActiveSheet.ProtectContents = True Then
      ActiveSheet.Unprotect
   End If
End Sub
```

BEISPIEL 2 VON MEINEM SPIEL

Hier kommt der Code von meinem Spiel, das ich in Excel programmiert habe und für das ich möglichst ALLES von Excel ausblenden wollte

```
   Application.Caption = " " ' Titelleiste: das Wort Excel entfernen – Achtung, das Leerzeichen ist zwingend nötig
   ActiveWindow.Caption = "" ' Titelleiste: DAS ERSETZT DEN DATEINAMEN

   Application.WindowState = xlMaximized ' Excelfenster maximieren

   Application.DisplayFullScreen = True ' Menüleisten ausblenden und Bildschirm auf Vollbildschirm schalten
   Application.DisplayStatusBar = False ' Statuszeile ausblenden
   Application.DisplayFormulaBar = False ' Zelladresse und Formelzeile ausblenden

   With ActiveWindow
```

```
    .DisplayHorizontalScrollBar = False ' horizontale Bildlaufleiste ausblenden für alle Tabellen
    .DisplayVerticalScrollBar = False ' vertikale Leiste für alle Tabellen
    .DisplayWorkbookTabs = False ' Blattregisterkarten für alle Tabellen
    .DisplayHeadings = False ' Zeilen und Spaltenüberschriften ausblenden
End With
```

Und hier noch ein weiterer Code, der sich auch um STRG-C/V/X kümmert:

```
Public Sub AUSBLENDEN ()
   Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",False)" 'blendet die Exceloberfläche aus und lässt nur die Tabelle übrig
   Application.DisplayFormulaBar = False ' blendet Standard u. Formatzeile aus
   Application.OnKey "^x", ""
   Application.OnKey "^c", ""
   Application.OnKey "^v", ""
   Application.OnKey "+{DEL}", ""
   Application.OnKey "+{INSERT}", ""

   'Drag & Drop ausschalten
   Application.CellDragAndDrop = False

   'Schaltflaechen in Menüleiste => Bearbeiten deaktivieren
   procControlEnableDisable 21, False ' Ausschneiden
   procControlEnableDisable 19, False 'Kopieren
   procControlEnableDisable 22, False 'Einfuegen
   procControlEnableDisable 755, False 'Inhalte einfuegen
   procControlEnableDisable 809, False 'Office-&Zwischenablage

End Sub

Public Sub EINBLENDEN()
   Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",True)" 'blendet die Exceloberfläche wieder ein
   Application.DisplayFormulaBar = True
   Application.OnKey "^x"
   Application.OnKey "^c"
   Application.OnKey "^v"
   Application.OnKey "+{DEL}"
   Application.OnKey "+{INSERT}"

   'Drag & Drop wieder erlauben
   Application.CellDragAndDrop = True

   'Schaltflaechen in Menüleiste => Bearbeiten aktivieren
   procControlEnableDisable 21, True ' Ausschneiden
   procControlEnableDisable 19, True 'Kopieren
   procControlEnableDisable 22, True 'Einfuegen
   procControlEnableDisable 755, True 'Inhalte einfuegen
   procControlEnableDisable 809, True 'Office-&Zwischenablage
```

```
End Sub

Sub procControlEnableDisable(intId As Integer, bolStatus As Boolean)
   On Error Resume Next 'sonst Fehler bei clipboard
   Dim cmbSuche As CommandBar
   Dim cmbcSteuerelement As CommandBarControl

   For Each cmbSuche In Application.CommandBars
      Set cmbcSteuerelement = _
      cmbSuche.FindControl(ID:=intId, recursive:=True)

      If Not cmbcSteuerelement Is Nothing Then
         cmbcSteuerelement.Enabled = bolStatus
      End If
   Next

End Sub
```

VERSION 2:

```
Private Sub TabellePur()  'blendet die Exceloberfläche aus und lässt nur die Tabelle übrig
   Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",False)"
   'Application.DisplayFormulaBar = False ' blendet Standard u. Formatzeile aus
   Application.OnKey "^x", ""
   Application.OnKey "^c", ""
   Application.OnKey "^v", ""
   Application.OnKey "+{DEL}", ""
   Application.OnKey "+{INSERT}", ""

   'Drag & Drop ausschalten
   Application.CellDragAndDrop = False

   'Schaltflaechen in Menüleiste => Bearbeiten deaktivieren
   procControlEnableDisable 21, False ' Ausschneiden
   procControlEnableDisable 19, False 'Kopieren
   procControlEnableDisable 22, False 'Einfuegen
   procControlEnableDisable 755, False 'Inhalte einfuegen
   procControlEnableDisable 809, False 'Office-&Zwischenablage

End Sub
```

```
Private Sub TabellePur_zurueck()  'blendet die Exceloberfläche wieder ein
    Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",True)"
    'Application.DisplayFormulaBar = True
    Application.OnKey "^x"
    Application.OnKey "^c"
    Application.OnKey "^v"
    Application.OnKey "+{DEL}"
    Application.OnKey "+{INSERT}"

    'Drag & Drop wieder erlauben
    Application.CellDragAndDrop = True

    'Schaltflaechen in Menüleiste => Bearbeiten aktivieren
    procControlEnableDisable 21, True ' Ausschneiden
    procControlEnableDisable 19, True 'Kopieren
    procControlEnableDisable 22, True 'Einfuegen
    procControlEnableDisable 755, True 'Inhalte einfuegen
    procControlEnableDisable 809, True 'Office-&Zwischenablage

End Sub
```

Ausblenden Menüpunkte ab Excel 2007

In Excel 2010 lässt sich das Ribbon wie folgt minimieren bzw. wieder herstellen:
CommandBars.ExecuteMso "MinimizeRibbon"

Ab 2007:

Wie lautet die VBA-Anweisung um das Menüband komplett ein- und auszublenden?

Das folgende Beispiel benutzt eine alte Excel4-Makroanweisung. Es blendet allerdings zusätzlich die Registerkartennamen und die Symbolleiste für den Schnellzugriff aus:

```
Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",False)"
```

Menüband wieder einblenden:

```
Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",True)"
```

BIS 2003:

Das Makro **Menu_ID_ermitteln** gibt im Direktfernster die ID´s der einzelnen Menüpunkte aus,
mit der das entsprechende Menü angesprochen werden kann.

```
Sub Menu_ID_ermitteln()
Dim Menüleiste As CommandBar
Dim i As Integer
Dim n As Integer

Set Menüleiste = CommandBars(1)
n = Menüleiste.Controls.Count
For i = 1 To n
Debug.Print Menüleiste.Controls(i).ID & _
" ---> " & Menüleiste.Controls(i).Caption
Next
End Sub
```

Die Ausgabe im Direktbereich sieht dann in etwa wie folgt aus:

```
30002 ---> &Datei
30003 ---> &Bearbeiten
30004 ---> &Ansicht
30005 ---> &Einfügen
30006 ---> Forma&t
30007 ---> E&xtras
30011 ---> Date&n
30083 ---> A&ktion
30009 ---> &Fenster
30010 ---> &?
```

Um einen Menüpunkt zu deaktivieren ändern Sie die ID-Nr. in die gewünschte
in dem im Direktbereich ausgegebene ID-Nr. ab. Dieser Menüpunkt wird nach Ausführung
des Makros **'ausblenden'** deaktiviert.

```
Sub ausblenden()
Dim ctrl As CommandBarPopup
Set ctrl = Application.CommandBars.FindControl(ID:=30009) 'hier z. B. Fenster
If ctrl Is Nothing Then Else ctrl.Enabled = False
End Sub
```

Das Makro **'einblenden'** aktiviert den gewünschten Menüpunkt wieder! Es ist entsprechend wieder die gewünsche Menü-Id-Nr. anzugeben.

```
Sub einblenden()
Dim ctrl As CommandBarPopup
Set ctrl = Application.CommandBars.FindControl(ID:=30009) 'hier z. B. Fenster
If ctrl Is Nothing Then Else ctrl.Enabled = True
End Sub
```

http://www.excel-inside.de/vba-menuprogrammierung/einzelne-menupunkte-deaktivieren-ausblenden

http://www.herber.de/forum/archiv/1072to1076/1073726_Menuepunkte_Speichern_sperren.html

http://www.office-loesung.de/ftopic96057_0_0_asc.php

Bestimmte Zellen nicht anwählen können

1.) Oft wird man mit Blattschutz arbeiten und nur die veränderbaren Zellen bleiben ungeschützt

2.) Man kann mit Selection.Change arbeiten und analysieren wo der User hinklickt - Nachteil: Bilder kann er anklicken, ohne dass Selection.Change ausgelöst wird. Man kann die natürlich mit einem Makro belegen, das sie schützt. Oder man setzt den Tabellenblattschutz, dann werden Grafiken auch geschützt

3.) ActiveSheet.Scrollarea ="$D$1:$E$10": man kann nur den erlaubten Zellbereich anwählen. Empfiehlt sich nur für überschaubare Tabellen. Der User kann in Tabellen, die nicht am Bildschirm vollständig sichtbar sind, nur durch den erlaubten Bereich scrollen und nicht erlaubte Bereiche sind durch Scrollen nicht sichtbar machenbar

4.) Worksheet.EnableSelection (greift nur bei Tabellenblattschutz)

Worksheet.EnableSelection-Eigenschaft

Gibt die Elemente zurück, die auf dem Blatt ausgewählt werden können, oder legt diese fest. **XlEnableSelection**-Wert mit Lese-/Schreibzugriff.

Syntax

Ausdruck.**EnableSelection**

Ausdruck Eine Variable, die ein **Worksheet**-Objekt darstellt.

Anmerkungen

Diese Eigenschaft ist nur wirksam, wenn das Arbeitsblatt geschützt ist: **xlNoSelection**xlNoSelection verhindert jegliche Markierung auf dem Blatt, **xlUnlockedCells**xlUnlockedCells gestattet nur die Markierung der Zellen, deren **Locked**-Eigenschaft **False**False ist, und **xlNoRestrictions**xlNoRestrictions erlaubt die Markierung einer beliebigen Zelle.

Beispiel

In diesem Beispiel wird festgelegt, dass im ersten Arbeitsblatt keine Elemente markiert werden können.

Visual Basic für Applikationen

```
With Worksheets(1)
    .EnableSelection = xlNoSelection
    .Protect Contents:=True, UserInterfaceOnly:=True
End With
```

GRUNDLAGEN

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Druckversion&action=edit&section=16Menü- und Symbolleisten

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-1Grundsätzliches

Menü- und Symbolleisten sind sowohl manuell wie auch über VBA zu erstellen, zu verändern und zu löschen.

Seit der Excel-Version 8.0 (Office 97) handelt es sich bei den Menü- und Symbolleisten um das Objektmodell der **Commandbars** mit den zugehörigen **Control**-Elementen *CommandBarButton*, *CommandBarPopUp* und *CommandBarComboBox* unter dem Oberbegriff *CommandBarControl*.

Grundsätzlich empfiehlt es sich, zu einer Arbeitsmappe gehörende CommandBars oder CommandBarControls beim Öffnen der Arbeitsmappe über das **Workbook_Open**-Ereignis zu erstellen und über das **Workbook_BeforeClose**-Ereignis zu löschen. Nur so ist gewährleistet, dass der Anwender nicht durch Auswirkungen von CommandBar-Programmierungen oder -Anbindungen belästigt wird.

Der Commandbars-Auflistung fügt man mit der **Add**-Methode eine neue Leiste hinzu. Erfolgt die Erstellung der neuen CommandBar in einem Klassenmodul, ist die Syntax **Application.CommandBars.Add...** zwingend erforderlich, erfolgt die Erstellung in einem Standardmodul, reicht ein **CommandBars.Add...**. Um später mögliche Kollisionen mit anderen Office-Anwendungen zu vermeiden, wird allerdings auch hier die **Application**-Nennung empfohlen.

Die Add-Methode kann mit bis zu 9 Parameter aufgerufen werden:

- **Name**
  Der Name der Symbolleiste, zwingend erforderlich
- **Position**
  optional, folgende Konstanten sind möglich:
  - msoBarLeft (am linken Bildschirmrand)
  - msoBarRight (am rechten Bildschirmrand)
  - msoBarTop (wird an die bestehenden Symbolleisten angegliedert)
  - msoBarBottom (am unteren Bildschirmrand, über der Statusleiste)
  - msoBarFloating (nicht verankerte Symbolleiste, die Position kann festgelegt werden)
  - msoBarPopUp (Kontext-Symbolleiste, mit der rechten Maustaste im Tabellenblatt aufrufbar)

- **MenuBar**
  optional, legt fest, ob es sich um eine Menü- oder eine Symbolleiste handelt (TRUE = Menüleiste, FALSE = Symbolleiste, Voreinstellung ist FALSE).

- **Temporary**
  optional, legt fest, ob die Menü- oder Symbolleiste mit Microsoft Excel geschlossen werden soll (TRUE = temporär, FALSE = bestehenbleibend, Voreinstellung ist FALSE). Wird also TRUE festgelegt, wird die CommandBar gelöscht, wenn Excel geschlossen wird und taucht auch in der CommandBar-Auflistung nicht mehr auf.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-2

Beispiele für das VBA-Handling von CommandBars

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-3

Menüleiste ein-/ausblenden

- Prozedur: CmdBarEinAus
- Art: Sub
- Modul: Standardmodul
- Zweck: Arbeitsblattmenüleiste aus- und einblenden.
- Ablaufbeschreibung:
  - Rahmen mit dem CommandBar-Objekt bilden
  - Wenn eingeschaltet ausschalten, sonst einschalten
- Code:

```
Sub CmdBarEinAus()
   With Application.CommandBars("Worksheet Menu Bar")
      .Enabled = Not .Enabled
   End With
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-4

Neue Menüleiste erstellen und einblenden

- Prozedur: NewMenueBar
- Art: Sub
- Modul: Standardmodul
- Zweck: Es wird eine neue Menüleiste erstellt und eingeblendet, wobei die Arbeitsblattmenüleiste ausgeblendet wird.
- Ablaufbeschreibung:
    - Variablendeklaration
    - Prozedur zum Löschen der evtl. bereits bestehenden Menüleiste aufrufen
    - Menüleiste erstellen
    - 1. Menü erstellen
    - Schleife über 12 Monate bilden
    - Monatsschaltfläche erstellen
    - Rahmen um das Schaltflächenobjekt erstellen
    - Aufschriftung festlegen
    - Der Schaltfläche keine Prozedur zuweisen
    - Den Aufschrifttyp festlegen
    - 2. Menü erstellen
    - Schleife über 12 Monate bilden
    - Monatsschaltfläche erstellen
    - Rahmen um das Schaltflächenobjekt erstellen
    - Aufschriftung festlegen

- Der Schaltfläche keine Prozedur zuweisen
- Den Aufschrifttyp festlegen
- Arbeitsblattmenüleiste ausblenden
- Neue Menüleiste einblenden

- Code:

```
Sub NewMenueBar()
   Dim oCmdBar As CommandBar
   Dim oPopUp As CommandBarPopup
   Dim oCmdBtn As CommandBarButton
   Dim datDay As Date
   Dim iMonths As Integer
   Call DeleteNewMenueBar
   Set oCmdBar = Application.CommandBars.Add( _
      Name:="MyNewCommandBar", _
      Position:=msoBarTop, _
      MenuBar:=True, _
      temporary:=True)
   Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
   oPopUp.Caption = "Prüfung"
   For iMonths = 1 To 12
      Set oCmdBtn = oPopUp.Controls.Add
      With oCmdBtn
         .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & " Druck"
         .OnAction = ""
         .Style = msoButtonCaption
      End With
   Next iMonths
   Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
   oPopUp.Caption = "Monatsbericht"
   For iMonths = 1 To 12
      Set oCmdBtn = oPopUp.Controls.Add
      With oCmdBtn
         .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & " Druck"
         .OnAction = ""
         .Style = msoButtonCaption
      End With
   Next iMonths
   Application.CommandBars("Worksheet Menu Bar").Enabled = False
   oCmdBar.Visible = True
End Sub
```

- Prozedur: DeleteNewMenueBar
- Art: Sub
- Modul: Standardmodul
- Zweck: Evtl. bestehende Menüleiste löschen
- Ablaufbeschreibung:
    - Fehlerroutine für den Fall starten, dass die Menüleiste nicht existiert
    - Benutzerdefinierte Menüleiste löschen
    - Arbeitsblattmenüleiste einblenden
- Code:

```
Private Sub DeleteNewMenueBar()
   On Error GoTo ERRORHANDLER
   Application.CommandBars("MyNewCommandBar").Delete
   Application.CommandBars("Worksheet Menu Bar").Enabled = True
   Exit Sub
ERRORHANDLER:
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-5**Alle Menüleisten ein-/ausblenden**

- Prozedur: AllesAusEinBlenden
- Art: Sub
- Modul: Standardmodul
- Zweck: Alle Menü- und Symbolleisten aus- und einblenden.
- Ablaufbeschreibung:
    - Objektvariable für CommandBar erstellen

- Rahmen um das CommandBar-Objekt erstellen
- Wenn die Arbeitsblattmenüleiste eingeblendet ist...
- Arbeitsblattmenüleiste ausblenden
- Auf Vollbildschirm schalten
- Eine Schleife über die CommandBars bilden
- Wenn es sich bei der aktuellen CommandBar nicht um die Arbeitsblattmenüleiste handelt...
- Wenn die aktuelle CommandBar sichtbar ist...
- Die aktuelle Commandbar ausblenden
- Aktive Arbeitsmappe schützen, wobei der Windows-Parameter auf **True** gesetzt wird (hierdurch werden die Anwendungs- und Arbeitsmappen-Schließkreuze ausgeblendet)
- Wenn die Arbeitsblattmenüleiste nicht sichtbar ist...
- Arbeitsmappenschutz aufheben
- Arbeitsblattmenüleiste anzeigen
- Vollbildmodus ausschalten

- Code:

```
Sub AllesAusEinBlenden()
   Dim oBar As CommandBar
   With CommandBars("Worksheet Menu Bar")
      If .Enabled Then
         .Enabled = False
         Application.DisplayFullScreen = True
         For Each oBar In Application.CommandBars
            If oBar.Name <> "Worksheet Menu Bar" Then
               If oBar.Visible Then
                  oBar.Visible = False
               End If
            End If
         Next oBar
```

```
         ActiveWorkbook.Protect Structure:=True , Windows:=False
      Else
         ActiveWorkbook.Unprotect
         .Enabled = True
         Application.DisplayFullScreen = False
      End If
   End With
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-6**Jahreskalender als Symbolleiste erstellen bzw. löschen**

- Prozedur: NewCalendar
- Art: Sub
- Modul: Standardmodul
- Zweck: Jahreskalender als Symbolleiste anlegen
- Ablaufbeschreibung:
  - Variablendeklaration
  - Fehlerroutine einschalten
  - Jahreskalender-Symbolleiste löschen
  - Prozedur beenden
  - Wenn keine Jahreskalender-Symbolleiste vorhanden war...
  - Neue Symbolleiste erstellen
  - Schleife über 12 Monate bilden
  - Menü für jeden Monat anlegen
  - Menüaufschrift festlegen
  - Wenn der Monatszähler durch 4 teilbar ist, eine neue Gruppe beginnen

- Die Tagesanzahl des jeweiligen Monats ermitteln
- Eine Schleife über die Tage des jeweiligen Monats bilden
- Das jeweilig aktuelle Datum ermitteln
- Tagesschaltfläche erstellen
- Aufschrift der Tagesschaltfläche festlegen
- Aufschriftart der Tagesschaltfläche festlegen
- Aufzurufende Prozedur festlegen
- Wenn es sich um einen Montag handelt, eine neue Gruppe beginnen
- Neue Symbolleiste anzeigen

- Code:

```
Sub NewCalendar()
   Dim oCmdBar As CommandBar
   Dim oPopUp As CommandBarPopup
   Dim oCmdBtn As CommandBarButton
   Dim datDay As Date
   Dim iMonths As Integer, iDays As Integer, iCount As Integer
   On Error GoTo ERRORHANDLER
   Application.CommandBars(CStr(Year(Date))).Delete
   Exit Sub
ERRORHANDLER:
   Set oCmdBar = Application.CommandBars.Add( _
      CStr(Year(Date)), msoBarTop, False, True)
   For iMonths = 1 To 12
      Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
      With oPopUp
         .Caption = Format(DateSerial(1, iMonths, 1), "mmmm")
         If iMonths Mod 3 = 1 And iMonths <> 1 Then .BeginGroup = True
         iCount = Day(DateSerial(Year(Date), iMonths + 1, 0))
         For iDays = 1 To iCount
            datDay = DateSerial(Year(Date), iMonths, iDays)
            Set oCmdBtn = oPopUp.Controls.Add
            With oCmdBtn
               .Caption = Day(datDay) & " - " & Format(datDay, "dddd")
               .Style = msoButtonCaption
```

```
               .OnAction = "GetDate"
               If Weekday(datDay, vbUseSystemDayOfWeek) = 1 And iDays <>; 1 Then .BeginGroup = True
            End With
         Next iDays
      End With
   Next iMonths
   oCmdBar.Visible = True
End Sub
```

- Prozedur: GetDate
- Art: Sub
- Modul: Standardmodul
- Zweck: Das aufgerufene Tagesdatum melden
- Ablaufbeschreibung:
  - Variablendeklaration
  - Aktuelles Jahr ermitteln
  - Monat ermitteln, aus dem der Aufruf erfolgte
  - Tag ermitteln, der ausgewählt wurde
  - Ausgewähltes Datum melden
- Code:

```
Sub GetDate()
   Dim iYear As Integer, iMonth As Integer, iDay As Integer
   Dim iGroupM As Integer, iGroupD As Integer
   iYear = Year(Date)
   iMonth = WorksheetFunction.RoundUp(Application.Caller(2) - _
      (Application.Caller(2) / 4), 0)
   iDay = Application.Caller(1) - GetGroups(iMonth, Application.Caller(1))
   MsgBox Format(DateSerial(iYear, iMonth, iDay), "dddd - dd. mmmm yyyy")
End Sub
```

- Prozedur: GetGroups

- Art: Function
- Modul: Standardmodul
- Zweck: Gruppe auslesen
- Ablaufbeschreibung:
  - Variablendeklaration
  - Zählvariable initialisieren
  - Eine Schleife über alle Monate der Jahreskalender-Symbolleiste bilden
  - Solange die Zählvariable kleiner/gleich die Anzahl der Controls...
  - Wenn eine neue Gruppe beginnt...
  - Gruppenzähler um 1 hochzählen
  - Wenn die Zählvariable gleich dem übergebenen Tag minus dem Gruppenzähler, dann Schleife beenden
  - Zählvariable um 1 hochzählen
  - Gruppenzähler als Funktionswert übergeben
- Code:

```
Private Function GetGroups(iActMonth As Integer, iActDay As Integer)
   Dim iGroups As Integer, iCounter As Integer
   iCounter = 1
   With Application.CommandBars(CStr(Year(Date))).Controls(iActMonth)
      Do While iCounter <= .Controls.Count
         If .Controls(iCounter).BeginGroup = True Then
            iGroups = iGroups + 1
         End If
         If iCounter = iActDay - iGroups Then Exit Do
         iCounter = iCounter + 1
      Loop
      GetGroups = iGroups
   End With
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Men%C3%BC-_und_Symbolleisten&action=edit&section=T-7

Alle Menü- und Symbolleisten auflisten

- Prozedur: ListAllCommandbars
- Art: Sub
- Modul: Standardmodul
- Zweck: Alle Symbolleisten mit dem englischen und dem Landesnamen mit der Angabe, ob sichtbar oder nicht, auflisten
- Ablaufbeschreibung:
  - Variablendeklaration
  - Bildschirmaktualisierung ausschalten
  - Neue Arbeitsmappe anlegen
  - Kopfzeile schreiben
  - Kopfzeile formatieren
  - Zeilenzähler initialisieren
  - Eine Schleife über alle - eingebauten und benutzerdefinierten - CommandBars bilden
  - Den englischen Namen eintragen
  - Den Landesnamen eintragen
  - Den Sichtbarkeitsstatus eintragen
  - Spaltenbreiten automatisch anpassen
  - Nicht genutzte Spalten ausblenden
  - Nicht genutzte Zeilen ausblenden

- Bildschirmaktualisierung einschalten
- Speichernstatus der Arbeitsmappe auf WAHR setzen (um beim Schließen eine Speichern-Rückfrage zu übergehen)

- Code:

```
Sub ListAllCommandbars()
   Dim oBar As CommandBar
   Dim iRow As Integer
   Application.ScreenUpdating = False
   Workbooks.Add 1
   Cells(1, 1) = "Name"
   Cells(1, 2) = "Lokaler Name"
   Cells(1, 3) = "Sichtbar"
   With Range("A1:C1")
      .Font.Bold = True
      .Font.ColorIndex = 2
      .Interior.ColorIndex = 1
   End With
   iRow = 1
   For Each oBar In Application.CommandBars
      iRow = iRow + 1
      Cells(iRow, 1) = oBar.Name
      Cells(iRow, 2) = oBar.NameLocal
      Cells(iRow, 3) = oBar.Visible
    Next oBar
    Columns("A:C").AutoFit
    Columns("D:IV").Hidden = True
    Rows(iRow + 1 & ":" & Rows.Count).Hidden = True
    Application.ScreenUpdating = True
    ActiveWorkbook.Saved = True
End Sub
```

Alle Menüleiste ein-/ausblenden

Prozedur: AllesAusEinBlenden

Art: Sub

Modul: Standardmodul

Zweck: Alle Menü- und Symbolleisten aus- und einblenden.

Ablaufbeschreibung:

- Objektvariable für CommandBar erstellen

- Rahmen um das CommandBar-Objekt erstellen
- Wenn die Arbeitsblattmenüleiste eingeblendet ist...
- Arbeitsblattmenüleiste ausblenden
- Auf Vollbildschirm schalten
- Eine Schleife über die CommandBars bilden
- Wenn es sich bei der aktuellen CommandBar nicht um die Arbeitsblattmenüleiste handelt...
- Wenn die aktuelle CommandBar sichtbar ist...
- Die aktuelle Commandbar ausblenden
- Aktive Arbeitsmappe schützen, wobei der Windows-Parameter auf **True** gesetzt wird (hierdurch werden die Anwendungs- und Arbeitsmappen-Schließkreuze ausgeblendet)
- Wenn die Arbeitsblattmenüleiste nicht sichtbar ist...
- Arbeitsmappenschutz aufheben
- Arbeitsblattmenüleiste anzeigen
- Vollbildmodus ausschalten

Code:

```
Sub AllesAusEinBlenden()
  Dim oBar As CommandBar
  With CommandBars("Worksheet Menu Bar")
    If .Enabled Then
      .Enabled = False
      Application.DisplayFullScreen = True
      For Each oBar In Application.CommandBars
        If oBar.Name &lt;&gt; "Worksheet Menu Bar" Then
          If oBar.Visible Then
            oBar.Visible = False
          End If
        End If
      Next oBar
      ActiveWorkbook.Protect Structure:=True , Windows:=False
    Else
      ActiveWorkbook.Unprotect
      .Enabled = True
```

```
        Application.DisplayFullScreen = False
    End If
  End With
End Sub
```

Alle Menü- und Symbolleisten auflisten

Prozedur: ListAllCommandbars

Art: Sub

Modul: Standardmodul

Zweck: Alle Symbolleisten mit dem englischen und dem Landesnamen mit der Angabe, ob sichtbar oder nicht, auflisten

Ablaufbeschreibung:

- Variablendeklaration
- Bildschirmaktualisierung ausschalten
- Neue Arbeitsmappe anlegen
- Kopfzeile schreiben
- Kopfzeile formatieren
- Zeilenzähler initialisieren
- Eine Schleife über alle - eingebauten und benutzerdefinierten - CommandBars bilden
- Den englischen Namen eintragen
- Den Landesnamen eintragen
- Den Sichtbarkeitsstatus eintragen
- Spaltenbreiten automatisch anpassen
- Nicht genutzte Spalten ausblenden
- Nicht genutzte Zeilen ausblenden
- Bildschirmaktualisierung einschalten

- Speichernstatus der Arbeitsmappe auf WAHR setzen (um beim Schließen eine Speichern-Rückfrage zu übergehen)

Code:

```
Sub ListAllCommandbars()
  Dim oBar As CommandBar
  Dim iRow As Integer
  Application.ScreenUpdating = False
  Workbooks.Add 1
  Cells(1, 1) = "Name"
  Cells(1, 2) = "Lokaler Name"
  Cells(1, 3) = "Sichtbar"
  With Range("A1:C1")
    .Font.Bold = True
    .Font.ColorIndex = 2
    .Interior.ColorIndex = 1
  End With
  iRow = 1
  For Each oBar In Application.CommandBars
    iRow = iRow + 1
    Cells(iRow, 1) = oBar.Name
    Cells(iRow, 2) = oBar.NameLocal
    Cells(iRow, 3) = oBar.Visible
   Next oBar
   Columns("A:C").AutoFit
   Columns("D:IV").Hidden = True
   Rows(iRow + 1 & ":" & Rows.Count).Hidden = True
   Application.ScreenUpdating = True
   ActiveWorkbook.Saved = True
End Sub
```

Einfügen eines Buttons mit Hyperlink in einer Symbolleiste

Erstellen Sie eine temporäre Symbolleiste mit einem Button hinter dem ein Hyperlink hinterlegt ist.

```
Sub Set_Hyperlink_Commandbar()
Dim myBar As CommandBar
Dim mybutton As CommandBarButton
Set myBar = CommandBars.Add(Name:="Hyper-Test", Position:=msoBarTop, Temporary:=True)
Set mybutton = myBar.Controls.Add(Type:=msoControlButton)
With mybutton
    .FaceId = 277
    .HyperlinkType = msoCommandBarButtonHyperlinkOpen
```

```
    'Nur wenn der Hyperlink im Tooltip steht funkioniert es :-))
    .TooltipText = "http://www.microsoft.com"
End With
Application.CommandBars("Hyper-Test").Visible = True
End Sub
```

Löschen Sie die Symbolleiste wieder.

```
Sub Del_Commandbar()
Application.CommandBars("Hyper-test").Delete
End Sub
```

Sie können diese beiden Codeteile auch in ein Workbook_Open und ein Workbook_Close Ereignis zuweisen. Dann wird die Symbolleiste beim öffnen der Datei erstellt und beim schliessen der Datei wieder entfernt.

Icon ID und Bezeichnung in Tabelle schreiben

Wird manchmal benötigt um die entsprechenden ID's herauszufinden

```
Sub print_control_name()
'(C) Ramses
'Schreibt die ID und die Bezeichnung der einzelnen Symbole
'und Buttons in die aktive Tabelle                                                    !!
'Achtung: Existierende Daten in dieser Tabelle werden überschrieben
Dim cc As Integer, n As Long, i As Long
cc = 1
For n = 1 To Application.Commandbars.count
   Cells(1, cc) = Application.Commandbars(n).Name
   For i = 1 To Application.Commandbars(n).Controls.count
      Cells(i + 1, cc) = Application.Commandbars(n).Controls(i).ID
      Cells(i + 1, cc + 1) = Application.Commandbars(n).Controls(i).Caption
   Next i
   cc = cc + 2
Next n
End Sub
```

Jahreskalender als Symbolleiste erstellen bzw. löschen

Prozedur: NewCalendar

Art: Sub

Modul: Standardmodul

Zweck: Jahreskalender als Symbolleiste anlegen

Ablaufbeschreibung:

- Variablendeklaration
- Fehlerroutine einschalten
- Jahreskalender-Symbolleiste löschen
- Prozedur beenden
- Wenn keine Jahreskalender-Symbolleiste vorhanden war...
- Neue Symbolleiste erstellen
- Schleife über 12 Monate bilden
- Menü für jeden Monat anlegen
- Menüaufschrift festlegen
- Wenn der Monatszähler durch 4 teilbar ist, eine neue Gruppe beginnen
- Die Tagesanzahl des jeweiligen Monats ermitteln
- Eine Schleife über die Tage des jeweiligen Monats bilden
- Das jeweilig aktuelle Datum ermitteln
- Tagesschaltfläche erstellen
- Aufschrift der Tagesschaltfläche festlegen
- Aufschriftart der Tagesschaltfläche festlegen
- Aufzurufende Prozedur festlegen
- Wenn es sich um einen Montag handelt, eine neue Gruppe beginnen
- Neue Symbolleiste anzeigen

Code:

```
Sub NewCalendar()
  Dim oCmdBar As CommandBar
  Dim oPopUp As CommandBarPopup
```

```
  Dim oCmdBtn As CommandBarButton
  Dim datDay As Date
  Dim iMonths As Integer, iDays As Integer, iCount As Integer
  On Error GoTo ERRORHANDLER
  Application.CommandBars(CStr(Year(Date))).Delete
  Exit Sub
ERRORHANDLER:
  Set oCmdBar = Application.CommandBars.Add( _
    CStr(Year(Date)), msoBarTop, False, True)
  For iMonths = 1 To 12
    Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
    With oPopUp
      .Caption = Format(DateSerial(1, iMonths, 1), "mmmm")
      If iMonths Mod 3 = 1 And iMonths &lt;&gt; 1 Then .BeginGroup = True
      iCount = Day(DateSerial(Year(Date), iMonths + 1, 0))
      For iDays = 1 To iCount
        datDay = DateSerial(Year(Date), iMonths, iDays)
        Set oCmdBtn = oPopUp.Controls.Add
        With oCmdBtn
          .Caption = Day(datDay) & " - " & Format(datDay, "dddd")
          .Style = msoButtonCaption
          .OnAction = "GetDate"
          If Weekday(datDay) = 1 And iDays &lt;&gt; 1 Then .BeginGroup = True
        End With
      Next iDays
    End With
  Next iMonths
  oCmdBar.Visible = True
End Sub
```

Prozedur: GetDate

Art: Sub

Modul: Standardmodul

Zweck: Das aufgerufene Tagesdatum melden

Ablaufbeschreibung:

- Variablendeklaration
- Aktuelles Jahr ermitteln
- Monat ermitteln, aus dem der Aufruf erfolgte

- Tag ermitteln, der ausgewählt wurde
- Ausgewähltes Datum melden

Code:

```
Sub GetDate()
  Dim iYear As Integer, iMonth As Integer, iDay As Integer
  Dim iGroupM As Integer, iGroupD As Integer
  iYear = Year(Date)
  iMonth = WorksheetFunction.RoundUp(Application.Caller(2) - _
    (Application.Caller(2) / 4), 0)
  iDay = Application.Caller(1) - GetGroups(iMonth, Application.Caller(1))
  MsgBox Format(DateSerial(iYear, iMonth, iDay), "dddd - dd. mmmm yyyy")
End Sub
```

Prozedur: GetGroups

Art: Function

Modul: Standardmodul

Zweck: Gruppe auslesen

Ablaufbeschreibung:

- Variablendeklaration
- Zählvariable initialisieren
- Eine Schleife über alle Monate der Jahreskalender-Symbolleiste bilden
- Solange die Zählvariable kleiner/gleich die Anzahl der Controls...
- Wenn eine neue Gruppe beginnt...
- Gruppenzähler um 1 hochzählen
- Wenn die Zählvariable gleich dem übergebenen Tag minus dem Gruppenzähler, dann Schleife beenden
- Zählvariable um 1 hochzählen
- Gruppenzähler als Funktionswert übergeben

Code:

```
Private Function GetGroups(iActMonth As Integer, iActDay As Integer)
  Dim iGroups As Integer, iCounter As Integer
  iCounter = 1
  With Application.CommandBars(CStr(Year(Date))).Controls(iActMonth)
    Do While iCounter &gt;= .Controls.Count
      If .Controls(iCounter).BeginGroup = True Then
        iGroups = iGroups + 1
      End If
      If iCounter = iActDay - iGroups Then Exit Do
      iCounter = iCounter + 1
    Loop
    GetGroups = iGroups
  End With
End Function
```

Kontextmenü des Tabellenregisters mit eigenem Befehl erweitern

Häufig gebraucht aber gut versteckt im Kontextmenü, geht es damit etwas einfacher.
Das Code "Create_New_Context_Command" kann auch in das Workbook_Open einer Mappe geschrieben werden

```
'Globale Konstante für den Namen des neuen Menüpunktes
'damit kann auch das "Delete_New_Context_Command" Makro
'auf den gleichen Namen zugreifen, und muss nur an einem Ort geändert werden
'Die Konstante gehört in ein MODUL
'(C)                                                                                    Ramses
Const myContext As String = "Neue Mappe aus Tabelle erstellen"

Sub Create_New_Context_Command()
Dim myNewContext As Object
Dim myContext As String
'Löschen eines bereits existierenden Menüs
'mit gleichem Namen
On Error Resume Next
'Commandbars("Ply") ist das Kontextmenü der Tabellenregister
Application.CommandBars("Ply").Controls(myContext).Delete
On Error GoTo 0
Set myNewContext = Application.CommandBars("Ply").Controls.Add
With myNewContext
   .Caption = myContext
   .OnAction = "Create_New_Workbook"
End With
End Sub

Sub Delete_New_Context_Command()
'Falls keine mehr existiert oder noch nicht ausgeführt
On Error Resume Next
Application.CommandBars("Ply").Controls(myContext).Delete
On Error GoTo 0
End Sub

'Das benötigte Makro für die "OnAction"                    -                    Methode des neuen
'Menüpunktes
Sub Create_New_Workbook()
'Erstellt aus der gerade aktiven Tabelle eine Arbeitsmappe
'in der nur die gerade aktive Tabelle enthalten ist
Worsheets.Copy
End Sub
```

Maximieren und Minimieren der Ribbon-Menüleiste

MEINE VARIANTE 1 INKL. SPALTENÜBERSCHRIFTEN

```
Sub ANZEIGE_UMSCHALTEN()
    If ActiveWindow.DisplayHeadings = False Then
       ActiveWindow.DisplayHeadings = True
       MenuebandMaximieren
     Else
       ActiveWindow.DisplayHeadings = False
       MenuebandMinimieren
    End If
End Sub


Public Function MenuebandIstMinimiert() As Boolean
   'Abfragen, ob das Menüband (Ribbon) minimiert ist
   MenuebandIstMinimiert = (CommandBars("Ribbon").Controls(1).Height < 100)
End Function

Public Sub MenuebandMinimieren()
  'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.

   If MenuebandIstMinimiert = False Then
      CommandBars.ExecuteMso "MinimizeRibbon"
   End If

End Sub

Public Sub MenuebandMaximieren()
   'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.

   If MenuebandIstMinimiert Then
      CommandBars.ExecuteMso "MinimizeRibbon"
   End If

End Sub
```

VARIANTE 2
Wenn ein einfacher Wechselschalter reicht:

```
Public Sub Ein_und_ausklappen()
    CommandBars.ExecuteMso "MinimizeRibbon"
End Sub
```

VARIANTE 3
Möchte man aber verlässlich Maximieren oder Minimieren, braucht man folgende 3:

```
Public Function MenuebandIstMinimiert() As Boolean

   'Abfragen, ob das Menüband (Ribbon) minimiert ist
```

```
  MenuebandIstMinimiert = (CommandBars("Ribbon").Controls(1).Height < 100)

End Function

Public Sub MenuebandMinimieren()

 'Menüband minimieren
 'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.

  If MenuebandIstMinimiert = False Then
    CommandBars.ExecuteMso "MinimizeRibbon"
  End If

End Sub

Public Sub MenuebandMaximieren()

  'Menüband maximieren
  'Die Funktion "MenuebandIstMinimiert()" muß vorhanden sein.

  If MenuebandIstMinimiert Then
    CommandBars.ExecuteMso "MinimizeRibbon"
  End If

End Sub
```

Menüleiste beim Schließen Killen

Menüleisten, die man fix bei einer bestimmten Exceldatei anhängt, bleiben in Excel auch nach Schließen dieser Datei ewig hängen - es sei denn man löscht sie vorher per VBA. Dieses Löschen hängt man am besten in den Workbook_BeforeClose-Code hinein.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   On Error Resume Next
   Application.CommandBars("BUCHHALTUNG").Delete
End Sub
```

Menüleiste ein-/ausblenden

Prozedur: CmdBarEinAus

Art: Sub

Modul: Standardmodul

Zweck: Arbeitsblattmenüleiste aus- und einblenden.

Ablaufbeschreibung:

Rahmen mit dem CommandBar-Objekt bilden

Wenn eingeschaltet ausschalten, sonst einschalten

Code:

```
Sub CmdBarEinAus()
  With Application.CommandBars("Worksheet Menu Bar")
    .Enabled = Not .Enabled
  End With
End Sub
```

Neue Menüleiste erstellen und einblenden

Prozedur: NewMenueBar

Art: Sub

Modul: Standardmodul

Zweck: Es wird eine neue Menüleiste erstellt und eingeblendet, wobei die Arbeitsblattmenüleiste ausgeblendet wird.

Ablaufbeschreibung:

- Variablendeklaration
- Prozedur zum Löschen der evtl. bereits bestehenden Menüleiste aufrufen
- Menüleiste erstellen
- 1. Menü erstellen
- Schleife über 12 Monate bilden
- Monatsschaltfläche erstellen
- Rahmen um das Schaltflächenobjekt erstellen

- Aufschriftung festlegen
- Der Schaltfläche keine Prozedur zuweisen
- Den Aufschrifttyp festlegen
- 2. Menü erstellen
- Schleife über 12 Monate bilden
- Monatsschaltfläche erstellen
- Rahmen um das Schaltflächenobjekt erstellen
- Aufschriftung festlegen
- Der Schaltfläche keine Prozedur zuweisen
- Den Aufschrifttyp festlegen
- Arbeitsblattmenüleiste ausblenden
- Neue Menüleiste einblenden

Code:

```
Sub NewMenueBar()
  Dim oCmdBar As CommandBar
  Dim oPopUp As CommandBarPopup
  Dim oCmdBtn As CommandBarButton
  Dim datDay As Date
  Dim iMonths As Integer
  Call DeleteNewMenueBar
  Set oCmdBar = Application.CommandBars.Add( _
    Name:="MyNewCommandBar", _
    Position:=msoBarTop, _
    MenuBar:=True, _
    temporary:=True)
  Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
  oPopUp.Caption = "Prüfung"
  For iMonths = 1 To 12
    Set oCmdBtn = oPopUp.Controls.Add
    With oCmdBtn
      .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & " Druck"
      .OnAction = ""
      .Style = msoButtonCaption
```

```
    End With
  Next iMonths
  Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
  oPopUp.Caption = "Monatsbericht"
  For iMonths = 1 To 12
    Set oCmdBtn = oPopUp.Controls.Add
    With oCmdBtn
      .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & " Druck"
      .OnAction = ""
      .Style = msoButtonCaption
    End With
  Next iMonths
  Application.CommandBars("Worksheet Menu Bar").Enabled = False
  oCmdBar.Visible = True
End Sub
```

Prozedur: DeleteNewMenueBar

Art: Sub

Modul: Standardmodul

Zweck: Evtl. bestehende Menüleiste löschen

Ablaufbeschreibung:

- Fehlerroutine für den Fall starten, dass die Menüleiste nicht existiert
- Benutzerdefinierte Menüleiste löschen
- Arbeitsblattmenüleiste einblenden

Code:

```
Private Sub DeleteNewMenueBar()
  On Error GoTo ERRORHANDLER
  Application.CommandBars("MyNewCommandBar").Delete
  Application.CommandBars("Worksheet Menu Bar").Enabled = True
  Exit Sub
ERRORHANDLER:
End Sub
```

Neue Menü-Punkte hinzufügen

```
Sub Auto_Open()
Dim mb As Object
Dim mi As Object
   On Error Resume Next
   If Year(Now) > maxYear Then
      MsgBox ("This version of """ & progName & """ has expired." & Chr(13) & Chr(13) & "To get a newer version check out:" & Chr(13) &
"http://www.straxx.com/excel/password.html")
      Application.DisplayAlerts = False
      ThisWorkbook.Close
      Exit Sub
   End If
   Call Auto_Close
   For Each mb In MenuBars
      With mb.Menus("Tools")
         Call .MenuItems.Add("Unprotect sheet", "UnprotectSheet")
         Call .MenuItems.Add("Unprotect workbook", "UnprotectWorkBook")
      End With
   Next
   Call MsgBox(progName & " now loaded." & Chr(13) & Chr(13) & "Choose 'Unprotect workbook' on the 'Tools'-menu to unprotect workbook." & Chr(13) &
"Choose 'Unprotect sheet' on the 'Tools'-menu to unprotect sheet." & Chr(13), vbOKOnly + vbInformation, progName & "     http://www.straxx.com")
End Sub

Sub Auto_Close()
Dim mi As Object
Dim mb As Object
   On Error Resume Next
   For Each mb In MenuBars
      For Each mi In mb.Menus("Tools").MenuItems
         If mi.Caption = "Unprotect workbook" Or mi.Caption = "Unprotect sheet" Then
            mi.Delete
         End If
      Next
   Next
End Sub
```

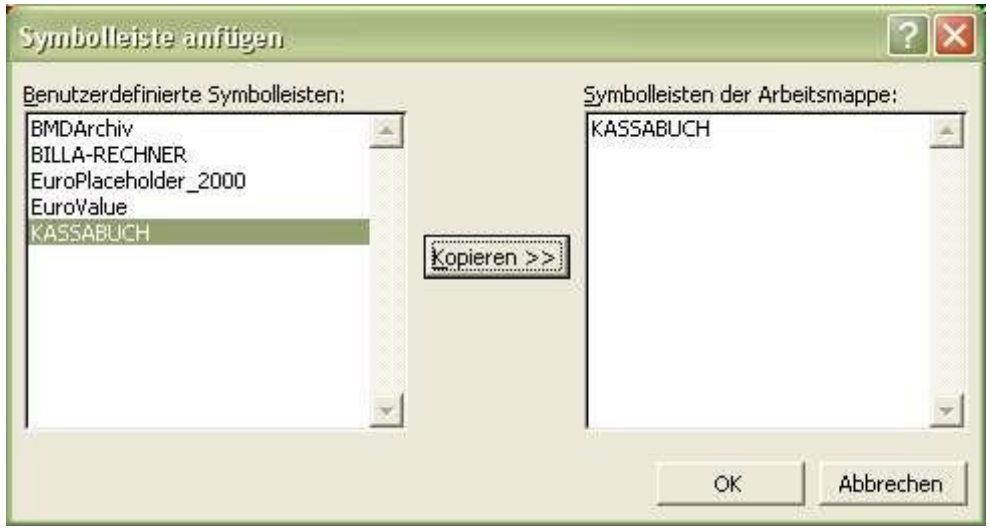
Symbolleisten Arbeitsmappenspezifisch

Problem: Excel kann sie nicht nur einer Arbeitsmappe zuweisen - wenn man sie einer Arbeitsmappe zuweist und diese dann öffnet, wird die Symbolleiste gespeichert und immer, auch nach dem Schließen, bei anderen Dateien angezeigt.

Lösung 1: Die Symbolleiste in VBA erst erstellen - dann ist sie nach dem Schließen der Datei auch wieder weg - (aber man kann die Icons dann nicht so schön einstellen)

Lösung 2: Man erzeugt eine Symbolleiste auf normalen Weg, kann 16x16-8Bit-Bilder als Icons einfügen für die Symbolleiste und fügt anschließend (und auch nach jeder Änderung der Leiste bitte) diese an die Datei an.

ANPASSEN / SYMBOLLEISTEN / ANFÜGEN



Zusätzlich braucht diese Arbeitsmappe dann einen VBA-Code, der bei ihrem Schließen die Symbolleiste, die bei jedem Öffnen in die allgemeinen Symbolleisten-Datei hineinkopiert wird, dort wieder löscht:

Dies erfolgt im VBA-Editor direkt im VBA-Code der Arbeitsmappe (hier Kassa-BMD.xls)


```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    On Error Resume Next
    Application.CommandBars("KASSABUCH").Delete
End Sub
```

Private Sub Workbook_BeforeClose(Cancel As Boolean)
On Error Resume Next ' falls Leiste schon geschlossen ist
Application.CommandBars("KASSABUCH").Delete ' KASSABUCH ist der Name der Leiste
End Sub

Wers ganz supergenau machen will, lässt die Symbolleiste beim Deaktiveren (sprich Datei ist im Hintergrund offen und eine andere Datei ist in Excel im

Vordergrund) ausblenden und beim Aktivieren der Datei, diese wieder einblenden mit folgenden zwei Subs direkt im obigen Codeteil "DIESE ARBEITSMAPPE" - (die Symbolleiste heißt hier "BANKBUCH"):

```
Private Sub Workbook_Activate()
    Application.CommandBars("BANKBUCH").Visible = True
End Sub

Private Sub Workbook_Deactivate()
    Application.CommandBars("BANKBUCH").Visible = False
End Sub
```

Suche-Textbox in einer Menüleiste einblenden

Alle Codeteile gehören in die jeweiligen Module. Nach dem öffnen der so präparierten Arbeitsmappe steht eine neue zusätzliche Suchfunktion in der Symbolleiste zur Verfügung.

```
'Private Sub Workbook_Open()
CreateSearchBar
'End Sub

'allgemeines Modul
Sub CreateSearchBar()
'Menüleiste mit Textfeld und Button zum Suchen
Dim myCMB As CommandBar
Dim myCombo As CommandBarControl
Dim myBtn As CommandBarButton
'Bereits eine eventuell noch existierende Symbolleiste löschen
DeleteSearchBar
'Erstellen der Symbolleiste
With Application.Commandbars.Add(Name:="MySearch")
    'sichtbar
   .Visible = True
   'oberhalb der Tabelle
   .Position = msoBarTop
   Set myCombo = .Controls.Add(Type:=msoControlEdit, temporary:=True)
   Set myBtn = .Controls.Add(Type:=msoControlButton, temporary:=True)
End With
'Action der Editbox definieren wenn sie
'verlassen wird !!
With myCombo
   .OnAction = "Search"
   .Width = 150
```

```
End With
'bietet die gleiche Funktion nochmal
With myBtn
   .Caption = "Suchen"
   .OnAction = "Search"
   .Style = msoButtonIconAndCaption
   .FaceId = 141
End With
End Sub


Sub Search()
'Das Makro das ausgelöst wird zur Suche
Dim strSrch As String
strSrch = Application.Commandbars("MySearch").Controls(1).Text
If strSrch = "" Then Exit Sub
    Cells.Find(What:=strSrch, After:=ActiveCell, LookIn:=xlValues, LookAt:= _
       xlPart, SearchOrder:=xlByColumns, SearchDirection:=xlNext, MatchCase:= _
       False).Activate
End Sub


Sub DeleteSearchBar()
'Sollte in das Workbook_BeforeClose Ereignis um
'die erstellte Symbolleiste zu netfernen
On Error Resume Next
Application.Commandbars("MySearch").Delete
End Sub
```

Wechselschalter in eigener Symbolleiste

Umändern eines bestehenden Buttons in einer selbsterstellten Menüleiste:

Konkret gibt es den Button AUSFÜLLMODUS, der, wenn man ihn anklickt, wechseln soll in die Beschriftung "AUSDURCKMODUS" (und zudem ein Hintergrundgrau in manchen Zellen zur besseren Dokumentstrukturierung und Ausfüllhilfe auf Weiß schalten für den Ausdruck)

Sub FARBWECHSELN()

On Error GoTo FEHLER ' wenn die Taste bereits Ausfüllmodus heißt, führt die nächste Zeile zu einem Fehler

Application.CommandBars("Checkliste").Controls("Ausdruckmodus").Caption = "Ausfüllmodus"

```
    ActiveWorkbook.Colors(15) = RGB(230, 230, 230)
    Exit Sub

FEHLER:

    Application.CommandBars("Checkliste").Controls("Ausfüllmodus").Caption = "Ausdruckmodus"
    ActiveWorkbook.Colors(15) = RGB(255, 255, 255)

End Sub
```

--- W e i t e r e M e n ü - R e z e p t e ---

Umbenennen vorhandener Menüeinträge

Diese Beispiele demonstrieren das Umbenennen vorhandener Menüeinträge:

```
Const Neue_Bezeichnung = "&Ich wurde umbenannt !"
'Benennt "Extras" um
Sub Menüpunkt_umbenennen()
   On Error Resume Next
   CommandBars("Worksheet Menu Bar"). _
     Controls("E&xtras").Caption = Neue_Bezeichnung
End Sub
'Setzt "Extras" wieder zurück
Sub Menüpunkt_zurücksetzen()
   On Error Resume Next
   CommandBars("Worksheet Menu Bar"). _
     Controls(Neue_Bezeichnung).Caption = "E&xtras"
End Sub

'Benennt "Speichern" im Menü "Datei um
Sub Menüeintrag_umbenennen()
   CommandBars("Worksheet Menu Bar"). _
     Controls("Datei").Controls("Speichern").Caption = Neue_Bezeichnung
End Sub
'Setzt "Speichern" im Menü "Datei" wieder zurück
Sub Menüeintrag_zurücksetzen()
   CommandBars("Worksheet Menu Bar"). _
     Controls("Datei").Controls(Neue_Bezeichnung).Caption = "&Speichern"
```

```
End Sub
```

Neue Menüleiste erstellen

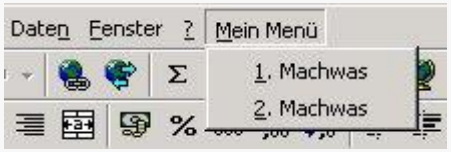
Dieses Beispiel erstellt eine neue Menüleiste.
Die bereits vorhandene Menüleiste wird dabei entfernt.


```
Const MenüName = "Mein_Menü"
Sub Neues_Menü()
   Call Menü_zurücksetzen

   Application.CommandBars("Worksheet Menu Bar").Enabled = False

   Set MB = CommandBars.Add(Name:=MenüName, MenuBar:=True)
   Set M1 = MB.Controls.Add(Type:=msoControlPopup)
   Set M2 = MB.Controls.Add(Type:=msoControlPopup)
   Set M3 = MB.Controls.Add(Type:=msoControlPopup)
   Set M4 = MB.Controls.Add(Type:=msoControlPopup)

   M1.Caption = "Menü 1"
   M2.Caption = "Menü 2"
   M3.Caption = "Menü 3"
   M4.Caption = "Menü 4"

   Set cmdButton = M1.Controls.Add
   With cmdButton
     .Caption = "Eintrag 1"
     .OnAction = "machwas1"
     .Style = msoButtonCaption
   End With

   Set cmdButton = M2.Controls.Add
```

```
    With cmdButton
      .Caption = "Eintrag 2"
      .OnAction = "machwas2"
      .Style = msoButtonCaption
    End With

    Set cmdButton = M3.Controls.Add
    With cmdButton
      .Caption = "Eintrag 3"
      .OnAction = "machwas3"
      .Style = msoButtonCaption
    End With

    Set cmdButton = M4.Controls.Add
    With cmdButton
      .Caption = "Eintrag 4"
      .OnAction = "machwas4"
      .Style = msoButtonCaption
    End With

    CommandBars(MenüName).Visible = True
End Sub
Sub Menü_zurücksetzen()
    On Error Resume Next
    Application.CommandBars(MenüName).Delete
    Application.CommandBars("Worksheet Menu Bar").Enabled = True
End Sub
Sub MachWas1()
    MsgBox "Hallo 1"
End Sub
Sub MachWas2()
    MsgBox "Hallo 2"
End Sub
Sub MachWas3()
    MsgBox "Hallo 3"
End Sub
Sub MachWas4()
    MsgBox "Hallo 4"
End Sub
```

Neues Menü mit Menüeinträgen erstellen

Dieses Beispiel erstellt einen neuen Menüeintrag
mit 2 Untereinträgen, aus denen entsprechende Routinen gestartet
werden können:

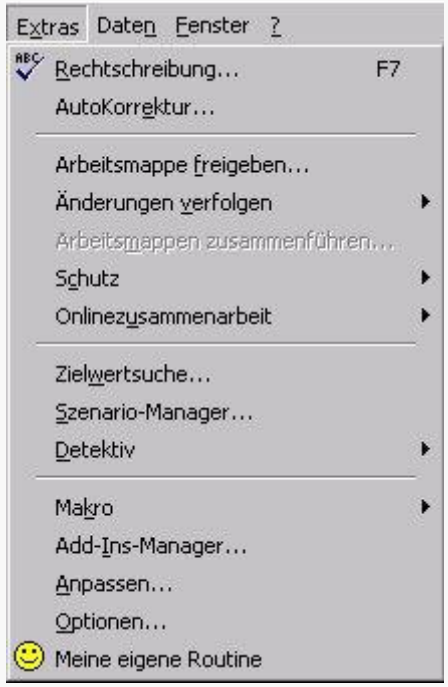



```
Const MenueName = "&Mein Menü"
Const Befehl1 = "&1. Machwas"
Const Befehl2 = "&2. Machwas"
Sub Menü_Erstellen()
   Dim MB As Object, MeinMenü As Object, Befehl As Object
   Call Menü_Löschen
   Set MB = CommandBars.ActiveMenuBar
   Set MeinMenü = MB.Controls.Add(Type:=msoControlPopup, Temporary:=True)
   MeinMenü.Caption = MenueName
   Set Befehl = MeinMenü.Controls.Add(Type:=msoControlButton, Id:=1)
   With Befehl
      .Caption = Befehl1
      .OnAction = "Machwas1"
   End With
   Set Befehl = MeinMenü.Controls.Add(Type:=msoControlButton, Id:=1)
   With Befehl
      .Caption = Befehl2
      .OnAction = "Machwas2"
   End With
End Sub
Sub Menü_Löschen()
   On Error Resume Next
   CommandBars.ActiveMenuBar.Controls(MenueName).Delete
End Sub
Sub Machwas1()
```

```
   MsgBox "1. Routine", vbExclamation
End Sub
Sub Machwas2()
   MsgBox "2. Routine", vbExclamation
End Sub
```

Vorhandenes Menü erweitern

Dieses Beispiel erweitert das Menü "Extras" um einen weiteren Menüpunkt:




```
Const MenüEintrag = "Meine eigene Routine"
Sub Menü_erweitern()
```

```
    Call Menüeintrag_löschen
    Set CB = Application.CommandBars.ActiveMenuBar
    Set CBC = CB.Controls("Extras")
    Set CBCC = CBC.Controls.Add
    With CBCC
        .Caption = MenüEintrag
        .OnAction = "MachWas"
        .FaceId = 59
    End With
End Sub
Sub Menüeintrag_löschen()
    On Error Resume Next
    Application.CommandBars.ActiveMenuBar. _
      Controls("Extras").Controls(MenüEintrag).Delete
End Sub
Sub MachWas()
    MsgBox "Hallo, da bin ich !"
End Sub
```

Menüeinträge löschen oder deaktivieren

Diese Routinen sollen zeigen, wie man vorhandene Menüeinträge
löschen oder disablen (deaktivieren) kann.

Löschen von Menüeinträgen

```
'Löschen des Menüpunktes "Extras"
Sub Menüpunkt_löschen()
    On Error Resume Next
    CommandBars("Worksheet Menu Bar"). _
      Controls("Extras").Delete
End Sub

'Löscht den Eintrag "Speichern" im Menü "Datei"
Sub Menüeintrag_löschen()
    CommandBars("Worksheet Menu Bar"). _
```

```
    Controls("Datei").Controls("Speichern").Delete
End Sub
'Wiederherstellen der gesamten Menüleiste
Sub Menüleiste_wiederherstellen()
   CommandBars("Worksheet Menu Bar").Reset
End Sub
```

Deaktivieren von Menüeinträgen

```
'Deaktiviert den Menüpunkt "Extras"
Sub Menüpunkt_deaktivieren()
   CommandBars("Worksheet Menu Bar"). _
      Controls("Extras").Enabled = False
End Sub
'Reaktiviert den Menüpunkt "Extras"
Sub Menüpunkt_reaktivieren()
   CommandBars("Worksheet Menu Bar"). _
      Controls("Extras").Enabled = True
End Sub
'Deaktiviert den Menüeintrag "Speichern" im Menü "Datei"
Sub Menüeintrag_deaktivieren()
   CommandBars("Worksheet Menu Bar"). _
      Controls("Datei").Controls("Speichern").Enabled = False
End Sub
'Reaktiviert den Menüeintrag "Speichern" im Menü "Datei"
Sub Menüeintrag_reaktivieren()
   CommandBars("Worksheet Menu Bar"). _
      Controls("Datei").Controls("Speichern").Enabled = True
End Sub
```

Prüfen ob ein Menüeintrag vorhanden ist

Vor dem Zugriff auf einen Menüpunkt sollte geprüft werden
ob dieser überhaupt vorhanden ist:

Ist ein Menüpunkt vorhanden ("Extras")

```
Function Menüpunkt_vorhanden(Bezeichnung) As Boolean
   Menüpunkt_vorhanden = False
   For Each MNU In Application.CommandBars _
    ("Worksheet Menu Bar").Controls
      If UCase(MNU.Caption) = UCase(Bezeichnung) Then
         Menüpunkt_vorhanden = True
         Exit Function
      End If
   Next MNU
End Function

'Ist der Menüpunkt "Extras" vorhanden ?
Sub Testen()
   MsgBox Menüpunkt_vorhanden("E&xtras")
End Sub
```

Ist ein Menüeintrag vorhanden ("Extras" - "Optionen")

```
Function Menüeintrag_vorhanden(SubMenü, Bezeichnung) As Boolean
   Menüeintrag_vorhanden = False
   For Each MNU In Application.CommandBars _
    ("Worksheet Menu Bar").Controls(SubMenü).Controls
      If UCase(MNU.Caption) = UCase(Bezeichnung) Then
         Menüeintrag_vorhanden = True
         Exit Function
      End If
   Next MNU
End Function

'Ist der Menüeintrag "Optionen..." unter "Extras" vorhanden ?
Sub Testen()
   MsgBox Menüeintrag_vorhanden("E&xtras", "&Optionen...")
End Sub
```

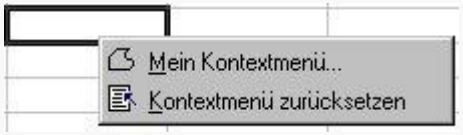
Menüeinträge umbenennen

Diese Beispiele demonstrieren das Umbenennen vorhandener Menüeinträge:

```
Const Neue_Bezeichnung = "&Ich wurde umbenannt !"
'Benennt "Extras" um
Sub Menüpunkt_umbenennen()
   On Error Resume Next
   CommandBars("Worksheet Menu Bar"). _
     Controls("E&xtras").Caption = Neue_Bezeichnung
End Sub
'Setzt "Extras" wieder zurück
Sub Menüpunkt_zurücksetzen()
   On Error Resume Next
   CommandBars("Worksheet Menu Bar"). _
     Controls(Neue_Bezeichnung).Caption = "E&xtras"
End Sub

'Benennt "Speichern" im Menü "Datei um
Sub Menüeintrag_umbenennen()
   CommandBars("Worksheet Menu Bar"). _
     Controls("Datei").Controls("Speichern").Caption = Neue_Bezeichnung
End Sub
'Setzt "Speichern" im Menü "Datei" wieder zurück
Sub Menüeintrag_zurücksetzen()
   CommandBars("Worksheet Menu Bar"). _
     Controls("Datei").Controls(Neue_Bezeichnung).Caption = "&Speichern"
End Sub
```

Eigenes Zell-Kontextmenü

Ein neues Zell-Kontextmenü erstellen.
Das vorhandene (Standard) Kontextmenü wird dabei gelöscht.




```
Sub NeuesKontextMenü()
   Dim M As CommandBarButton
   With CommandBars("Cell")
      Do While .Controls.Count > 0
         On Error Resume Next
         .Controls(1).Delete
      Loop
      Set M = .Controls.Add(msoControlButton)
      With M
         .Caption = "&Mein Kontextmenü..."
         .OnAction = "MachWas"
         .FaceId = 200
      End With
      Set M = .Controls.Add(msoControlButton)
      With M
         .Caption = "&Kontextmenü zurücksetzen"
         .OnAction = "Reset"
         .FaceId = 44
      End With
   End With
End Sub
Sub MachWas()
   MsgBox "Hallo !", vbExclamation
End Sub
Sub Reset()
   CommandBars("Cell").Reset
End Sub
```

Vorhandene Kontextmenü's erweitern

Unterscheiden Sie zwischen den verschiedenen Arten
der Kontextmenü's:

Zell-Kontextmenü, Spalten-Kontextmenü und Zeilen-Kontextmenü

Diese werden unterschiedlich angesprochen.

Kontextmenü einer Spalte:

```
Const t = "Meine eigene Routine"
Sub Kontextmenü_Erweitern()
    Call Kontext_Löschen
    Dim Kontext As Object
    Set Kontext = CommandBars("Column").Controls.Add
    Kontext.BeginGroup = True
    With Kontext
        .Caption = t
        .OnAction = "MachWas"
        .FaceId = 122
    End With
End Sub
Sub Kontext_Löschen()
   On Error Resume Next
   CommandBars("Column").Controls(t).Delete
End Sub

Sub MachWas()
    MsgBox "Hallo, da bin ich !", vbExclamation
End Sub
```

Kontextmenü einer Zeile:

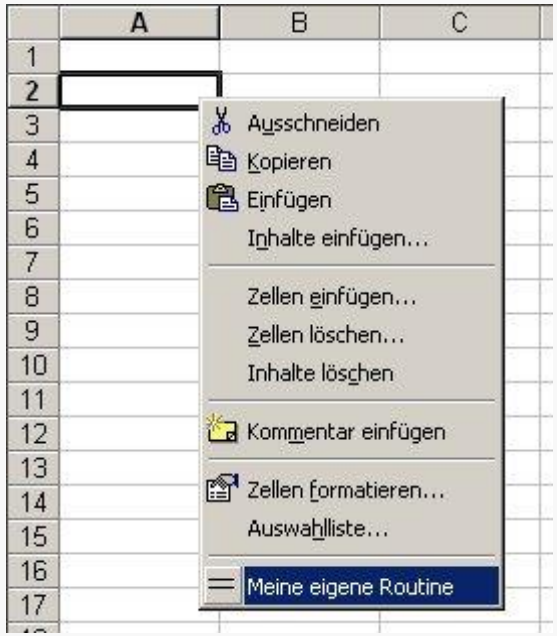
```
Const t = "Meine eigene Routine"
Sub Kontextmenü_Erweitern()
    Call Kontext_Löschen
    Dim Kontext As Object
    Set Kontext = CommandBars("Row").Controls.Add
    Kontext.BeginGroup = True
    With Kontext
        .Caption = t
        .OnAction = "MachWas"
        .FaceId = 122
    End With
End Sub
Sub Kontext_Löschen()
   On Error Resume Next
   CommandBars("Row").Controls(t).Delete
End Sub
```

```
Sub MachWas()
   MsgBox "Hallo, da bin ich !", vbExclamation
End Sub
```

Kontextmenü einer Zelle:

```
Const t = "Meine eigene Routine"
Sub Kontextmenü_Erweitern()
   Call Kontext_Löschen
   Dim Kontext As Object
   Set Kontext = CommandBars("Cell").Controls.Add
   Kontext.BeginGroup = True
   With Kontext
      .Caption = t
      .OnAction = "MachWas"
      .FaceId = 122
   End With
End Sub
```

```
Sub Kontext_Löschen()
   On Error Resume Next
   CommandBars("Cell").Controls(t).Delete
End Sub
Sub MachWas()
    MsgBox "Hallo, da bin ich !", vbExclamation
End Sub
```



Neue Symbolleiste mit Symbolen erstellen

Dieses Beispiel erstellt eine neue Symbolleiste mit 3 Symbolen und
weist den Symbolen 3 unterschiedliche Makros zu:


```
Const Symbolleistenname = "Meine Symbolleiste"
Sub Symbolleiste_erstellen()
   Set CB = Application.CommandBars.Add(Name:=Symbolleistenname, _
     temporary:=True, Position:=msoBarTop)
   CB.Visible = True
   Set CBC = CB.Controls.Add(Type:=msoControlButton)
   With CBC
      .FaceId = 59
      .Caption = "Symbol 1"
      .OnAction = "Makro1"
   End With
   Set CBC = CB.Controls.Add(Type:=msoControlButton)
   With CBC
      .FaceId = 66
      .Caption = "Symbol 2"
      .OnAction = "Makro2"
   End With
   Set CBC = CB.Controls.Add(Type:=msoControlButton)
   With CBC
      .FaceId = 67
      .Caption = "Symbol 3"
      .OnAction = "Makro3"
   End With
End Sub
Sub Symbolleiste_löschen()
   On Error Resume Next
   Application.CommandBars(Symbolleistenname).Delete
End Sub
Sub Makro1()
   MsgBox "Makro 1"
End Sub
Sub Makro2()
```

```
    MsgBox "Makro 2"
End Sub
Sub Makro3()
    MsgBox "Makro 3"
End Sub
```

Damit die neue Symbolleiste nur dann erscheint,
wenn diese Datei geöffnet wird genügen diese beiden Routinen
unter "DieseArbeitsmappe":

```
Private Sub Workbook_Open()
    Call Symbolleiste_erstellen
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call Symbolleiste_löschen
End Sub
```

Symbolleiste mit Textbutton erstellen

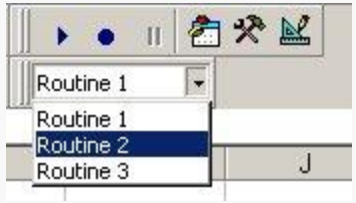
Das folgende Beispiel erstellt eine neue Symbolleiste mit einem Textbutton,
dem ein Makro zugewiesen wird:


```
Const Symbolleistenname = "Eigene Symbolleiste"

Sub Symbolleiste_erstellen()
    On Error Resume Next
```

```
    Dim CB As CommandBar
    Dim CBC As CommandBarButton
    Set CB = Application.CommandBars(Symbolleistenname)
    Symbolleiste_löschen
    Set CB = Application.CommandBars.Add(Name:=Symbolleistenname, _
      temporary:=False, Position:=msoBarTop)
    Set CBC = CB.Controls.Add(Type:=msoControlButton)
    With CBC
        .Caption = "Meine Routine"
        .OnAction = "MachWas"
        .Style = msoButtonCaption
    End With
    CB.Visible = True
End Sub

Sub Symbolleiste_löschen()
    On Error Resume Next
    Application.CommandBars(Symbolleistenname).Delete
End Sub
Sub MachWas()
    MsgBox "Da bin ich", vbExclamation
End Sub
```

Dropdownliste in Symbolleiste

Anstelle vieler Symbole in einer Symbolleiste kann man sich helfen,
wenn man beispielsweise zusammenhängende Routinen
in eine Dropdownliste setzt:




```
Const CBName = "Meine Makros"
Sub Symbolleiste_erstellen()
   Call Symbolleiste_löschen
   CommandBars.Add Name:=CBName
   With CommandBars(CBName)
      .Position = msoBarTop
      .Visible = True
      .Controls.Add Type:=msoControlDropdown
   End With
   With CommandBars(CBName).Controls(1)
      .Caption = CBName
      .AddItem "Routine 1"
      .AddItem "Routine 2"
      .AddItem "Routine 3"
      .ListIndex = 1
      .OnAction = "MachWas"
      .TooltipText = "Meine Routinen..."
   End With
End Sub
Sub Symbolleiste_löschen()
   On Error Resume Next
   CommandBars(CBName).Delete
End Sub
Sub MachWas()
   Dim objList As CommandBarControl
   Dim Auswahl As Byte
   Set objList = CommandBars.ActionControl
   Auswahl = objList.ListIndex
   Select Case Auswahl
      Case 1:
         MsgBox "Routine 1", vbExclamation
      Case 2:
```

```
        MsgBox "Routine 2", vbExclamation
      Case 3:
        MsgBox "Routine 3", vbExclamation
  End Select
End Sub
```

Vorhandene Symbolleiste erweitern

Dieses Beispiel erweitert die Symbolleiste "Formtierung"
um ein weiteres Symbol und weist diesem Symbol ein Makro zu:

```
Sub Symbolleiste_erweitern()
  Call Symbol_löschen
  Set CB = Application.CommandBars("Formatting"). _
    Controls.Add(msoControlButton)
  With CB
    .Caption = "Befehl"
    .FaceId = 66
    .OnAction = "MachWas"
    .Visible = True
  End With
End Sub
Sub Symbol_löschen()
    On Error Resume Next
    Application.CommandBars("Formatting"). _
      Controls("Befehl").Delete
End Sub
Sub MachWas()
    MsgBox "Hallo, da bin ich !"
End Sub
```

Ein Symbol für einen Makronamen in verschiedenen Dateien

Bitte beachten Sie, dass dieser Beitrag
fortgeschrittene Kenntnisse in VBA erfordert !

Was zunächst recht verwirrend klingen mag ist recht schnell erklärt:
Stellen Sie sich vor, Sie arbeiten mit mehreren Dateien,
die alle ein gleichnamiges Makro beinhalten,
jedoch verschiedene Aufgaben erledigen soll.

Eine neue Symbolleiste mit einem Symbol zu erstellen
ist keine große Sache, jedoch steckt hinter diesem Symbol
das unter "OnAction" angegebene Makro in einer bestimmten Datei,
und zwar nur in "dieser einen" Datei !

Die folgende Lösung soll verdeutlichen,
wie einfach es ist, "OnAction" je nach geöffneter Datei anzupassen.

Diese Datei hat die Aufgabe eine neue Symbolleiste mit einem
Symbol zu erstellen. Zunächst wird "OnAction" das Makro "MachWas"
zugewiesen, welches sich in der gleichen Datei befindet.
Des weiteren überwacht diese Datei mit Hilfe eines Klassenmodules
das Öffnen von Dateien, um somit die Eigenschaft "OnAction" zu ändern.
Wird also eine andere Datei geöffnet, soll sich die "OnAction" Eigenschaft
auf die gerade geöffnete Datei beziehen, ebenfalls auf den
Makronamen "MachWas".

```
'** DieseArbeitsmappe **
Dim AppObject As New CAppLog
Private Sub Workbook_Open()
    Call Symbolleiste_erstellen
    Set AppObject.app = Application
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Set AppObject.app = Nothing
End Sub
```

```
'** Modul1 **
Const Symbolleistenname = "Eigene Symbolleiste"
Const Symbolname = "Test"
Const Makroname = "MachWas"

'
'Erstellt eine neue Symbolleiste
'
Sub Symbolleiste_erstellen()
```

```
    On Error Resume Next
    Application.CommandBars(Symbolleistenname).Delete
    Set CB = Application.CommandBars.Add(Name:=Symbolleistenname, _
      temporary:=True, Position:=msoBarTop)
    CB.Visible = True
    Set CBC = CB.Controls.Add(Type:=msoControlButton)
    With CBC
       .Name = Symbolname
       .FaceId = 59
       .Caption = "Test"
       .OnAction = Makroname
    End With
End Sub

'
'Ändert die Eigenschaften der Symbolleiste (OnAction)
'
Sub SymbolMakroÄndern(Dateiname)
    On Error Resume Next
    Application.CommandBars(Symbolleistenname). _
      Controls(Symbolname).OnAction = Dateiname & "!" & Makroname
End Sub

'
'Testmakro, steckt hinter dem Symbol
'
Sub MachWas()
    MsgBox "Hier meldet sich ein Makro aus der 'Hauptdatei'", vbExclamation
End Sub
```

```
'** Klassenmodul "CAppLog" **
Public WithEvents app As Application
Private Sub app_WorkbookOpen(ByVal WBook As Excel.Workbook)
    Dateiname = WBook.FullName
    Call SymbolMakroÄndern(Dateiname)
End Sub
```

Erstellen Sie sich nun einige Testdateien,
die alle ein Makro namens "MachWas" beinhalten.
Zum Testen reicht es aus, eine Msgbox in das Makro zu setzen.

Im untenstehenden Beispiel finden Sie das Beispiel und 3 Testdateien.

Natürlich könnte man den Namen des zu startenden Makros
abhängig vom Dateinamen festlegen.

Dazu würde es ausreichen eine entsprechende Abfrage in
der Routine "SymbolMakroÄndern" einzubauen.

Ein Anwendungsbeispiel könnte sein,
wenn Sie regelmäßig mit verschiedenen Dateien arbeiten,
die unterschiedliche Routinen beinhalten.
Sie müssen sich dazu nicht unbedingt mehrere Symbole in einer
Symbolleiste anlegen, sondern nur dieses eine.
Je nach geöffneter Datei wird die entsprechend richtige Routine gestartet.
Um die Sache perfekt zu gestalten, könnte man mit Hilfe der VBE-Programmierung
vor dem Ändern der "OnAction" Eigenschaft prüfen, ob sich das entsprechende
Makro überhaupt in der geöffneten Datei befindet.
Falls dies nicht der Fall ist, weist man einfach ein Makro zu,
welches dies in Form einer entsprechenden Meldung dem Benutzer mitteilt.

Symbolleiste mit eigenen Symbolen erstellen

Folgende Routine, als Add-In einsetzbar, erstellt eine neue Symbolleiste,
falls diese noch nicht vorhanden ist.
Diese Symbolleiste wird mit 2 neuen Symbolen versehen,
die beide eine entsprechende Sub zugewiesen bekommen.

Es ist ein sofort einsetzbares Add-In,
welches Sonderzeichen einfügt:

Schritt 1
Öffnen Sie eine neue leere Arbeitsmappe

Schritt 2
In "Tabelle1" fügen Sie über die Zwischenablage
2 Symbole ein, die Sie zuvor in einer Grafikbearbeitungssoftware
erstellt haben.
Ein Tipp dazu: Kopieren Sie sich zunächst ein vorhandenes Symbol
aus einer Symbolleiste in Excel, um die genaue Größe zu kennen.

Benennen Sie diese Symbole mit "Symbol1" und "Symbol2".

Schritt 3

Wechseln Sie nun in den Visual Basic Editor (Alt+F11)
Unter "DieseArbeitsmappe" fügen Sie im Codefenster
folgendes ein:

```
Const Symbolleistenname = "Sonderzeichen"
Const Caption_1 = "&Durchmesserzeichen"
Const Caption_2 = "&Toleranz"
Private Sub Workbook_Open()
  On Error Resume Next
  Dim CB As CommandBar
  Dim CMB As CommandBarButton
  Set CB = Application.CommandBars(Symbolleistenname)
  'Symbolleiste ist nicht vorhanden
  If Err.Number <> 0 Then
    Application.CommandBars(Symbolleistenname).Delete
    Set CB = Application.CommandBars.Add(Name:= _
      Symbolleistenname, temporary:=False, Position:=msoBarTop)
    Set CMB = CB.Controls.Add(Type:=msoControlButton)
    '1. Symbol
    With CMB
      .Caption = Caption_1
      .OnAction = "Durchmesserzeichen"
       ThisWorkbook.Sheets("Tabelle1").Shapes _
        ("Symbol1").Copy
       Application.CommandBars(Symbolleistenname)._
```

```
        Controls(1).PasteFace
    End With
    '2. Symbol
    Set CMB = CB.Controls.Add(Type:=msoControlButton)
    With CMB
      .Caption = Caption_2
      .OnAction = "PlusMinus"
      ThisWorkbook.Sheets("Tabelle1").Shapes("Symbol2").Copy
      Application.CommandBars(Symbolleistenname)._
        Controls(2).PasteFace
    End With
   CB.Visible = True
  End If
End Sub
```

Schritt 4

Fügen Sie ein neues Modul ein,
welches die Routinen beinhalten soll, die beim Klick auf die
jeweiligen Schaltflächen ausgeführt werden:

```
Sub Durchmesserzeichen()
  'falls keine Datei geöffnet ist
  If Workbooks.Count = 0 Then
    MsgBox "Es ist keine Datei geöffnet !", vbCritical, "Hinweis"
    Exit Sub
  End If
  'aktuellen Wert ermitteln
  Merk = ActiveCell.Value
  With ActiveCell
    'steht etwas darin ?
    If Len(Merk) > 0 Then
      'ist das 1. Zeichen kein Durchmesserzeichen,
      'dann Durchmesserzeichen voran stellen
      If Asc(Mid(Merk, 1, 1)) <> 198 Then  .Value = "Æ " & Merk
    Else
      'ansonsten nur Durchmesserzeichen
      .Value = "Æ "
    End If
    '1. Zeichen auf Schriftart "Symbol" setzen
```

```
      .Characters(Start:=1, Length:=1).Font.Name = "Symbol"
      'restliche Zeichen mit Schriftart "Arial"
      .Characters(Start:=3, Length:=100).Font.Name = "Arial"
   End With
End Sub
Sub PlusMinus()
   'falls keine Datei geöffnet ist
   If Workbooks.Count = 0 Then
      MsgBox "Es ist keine Datei geöffnet !", vbCritical, "Hinweis"
      Exit Sub
   End If
   'aktuellen Wert ermitteln
   Merk = ActiveCell.Value
   With ActiveCell
      'steht etwas darin ?
      If Len(Merk) > 0 Then
         'ist das 1. Zeichen kein "±",
         'dann "±" voran stellen
         If Asc(Mid(Merk, 1, 1)) <> 241 Then .Value = "± " & Merk
      Else
         'ansonsten nur Durchmesserzeichen
         .Value = "± "
      End If
   End With
End Sub
```

Schritt 5

Zum Schluss speichern Sie die Datei ab und können Sie als Add-In einbinden.

Zusätzlicher Menüpunkt für eine Arbeitsmappe erstellen

```
Private Sub Workbook_Open()
Dim cbMenu As CommandBar
Dim cbSpecialMenu As CommandBarPopup
Dim cbCommand As CommandBarControl
```

```
'Zuweisen der Objectvariablen
Set cbMenu = Application.CommandBars("Worksheet Menu Bar")
Set cbSpecialMenu = cbMenu.Controls.Add(Type:=msoControlPopup)
'Titelbeschriftung der Menübar
cbSpecialMenu.Caption = "Mein Spezialmenu"
'Einen Button hinzufügen und diesen gleich beschriften
Set cbCommand = cbSpecialMenu.Controls.Add(Type:=msoControlButton)
cbCommand.Caption = "Mein Befehl"
cbCommand.OnAction = "Ihr Makro das ausgeführt werden soll"
End Sub
```

Mit diesen Anweisungen fügen Sie einen zusätzlichen Menüpunkt namens "Mein Spezialmenü" in die Standard-Menüleiste von Excel ein. Außerdem erweitern Sie das Menü um einen Befehl namens "Mein Befehl". Wenn dieser Befehl eine Aktion ausführen soll, müssen Sie nicht nur die "Caption"- sondern auch die "OnAction"-Eigenschaft setzen und ihr den Namen eines Makros zuweisen.

Erstellen Sie eine Prozedur, die das "Spezialmenü" beim Schließen der Datei wieder entfernt.
Im rechten Dropdown-Feld im VB-Editor aktivieren Sie dieses Mal aber das "BeforeClose"-Ereignis. Als Prozedurcode geben Sie folgendes ein:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Dim cbSpecialMenu As CommandBarControl
'Fehlerbehandlung ausschalten
'sonst entsteht ein Fehler wenn die Menübar nicht mehr sichtbar ist
On Error Resume Next
'Objektvariablen füllen
Set cbSpecialMenu = Application.CommandBars("Worksheet Menu Bar").Controls("Spezialmenü")
'Menüleiste löschen
cbSpecialMenu.Delete
End Sub
```

Den Menübefehl müssen Sie nicht gesondert entfernen, da er zusammen mit dem übergeordneten Menü gelöscht wird.

Um den Menüzugriff ausschließlich im Rahmen der passenden Arbeitsmappe zu erlauben, fügen Sie auch Prozeduren für das "Deactivate"- sowie das "Activate"-Ereignis des "Workbook"-Objekts hinzu. Damit stellen sie nun endgültig sicher, dass die Menüleiste nur zur Verfügung steht, wenn ihre Arbeitsmappe auch tatsächlich aktiv ist.

```
Private Sub Workbook_Deactivate()
On Error Resume Next
Application.CommandBars("Worksheet Menu Bar").Controls("Spezialmenü").Visible = False
```

```
End Sub

Private Sub Workbook_Activate()
On Error Resume Next
Application.CommandBars("Worksheet Menu Bar").Controls("Spezialmenü").Visible = True
End Sub
```

Nun können Sie die aktuelle Arbeitsmappe speichern und schließen. Beim nächsten Öffnen wird am rechten Ende der Excel-Menüleiste das neue Menü "Spezialmenü" erscheinen. Es verschwindet, sobald Sie die Mappe schließen oder eine andere Excel-Datei aktivieren. Wenn Sie zur ursprünglichen Mappe zurückkehren, steht Ihnen das Menü wieder zur Verfügung

NETZWERK

Stündliches prüfen der eigenen IP-Adresse.

Sollten Sie eine DSL-Flatrate haben und einen eigenen FTP-Server ohne eigene fixe IP-Adresse, wäre es doch ganz praktisch, wenn Sie bei einer Änderung über die neue Adresse informiert würden.,... oder ? :

```
 'Gehört zusammen für automatische Übermittlung der IP-Adresse
Sub Start_Call_IP()
 'Startet stündlich das nächste Makro
Application.OnTime Now() + TimeValue("01:00:00"), "Get_my_IP_and_send_Message"
End Sub


Sub Get_my_IP_and_send_Message()
'Hilfsvariable für Anzahl Datensätze
Dim Text1 As String
'Variablen für den Array nötig
Dim txtlines As Long, i As Long
'Für Office97 muss das Array "TextArr" als String definiert werden
'Entdeckt duch Gerd Z aus dem Herber Forum
Dim textArr As Variant, myCmd As Variant, myIp As String
Dim ReadFile As String
'Export der IP-Adress in ein txt File
'Den Namen und Pfad bitte anpassen
ReadFile = "C:\myIP.txt"
'Schliessen einer geöffneten Datei
Close #1
'Abfragen der eigenen IP-Adresse und Ausgabe in die Datei
myCmd = Shell("cmd.exe /C ipconfig > C:\myIP.txt")
'1. Öffnen der Datei
Open ReadFile For Input As #1
'Die anzahl ist nötig um die Grösse des Arrays zu deklarieren
'Zähler auf 0 setzen
txtlines = 0
Do While Not EOF(1)    ' Schleife bis Dateiende.
   Input #1, Text1    ' Hilfsvariable zum einlesen verwenden
   'Zähler hochzählen
   txtlines = txtlines + 1
Loop
'Schliessen der Datei weil Dateiende erreicht wurde
Close #1
'Erneutes Öffnen um zum Dateianfang zu kommen
Open ReadFile For Input As #1    ' Datei zum Einlesen öffnen.
'Array neu auf die Anzahl der Linien initialisieren
ReDim textArr(txtlines)
'Einlesen der Dateien in das Array
For i = 1 To txtlines
```

```
    Line Input #1, textArr(i)
Next i
Close #1
'IP String auslesen
For i = 1 To txtlines
    'Mit "IP-Adresse" beginnt der String mit der eigenen IP
    'Hast du noch eine zweite IP musst du den String anpassen
    If InStr(1, textArr(i), "IP-Adresse") Then
        myIp = Trim(Right(textArr(i), Len(textArr(i)) - InStrRev(textArr(i), ":", -1)))
    End If
Next i
'Vergleichen der IP's
If myPrivIp = "" Then
    'beim ersten Start
    myPrivIp = myIp
Else
    'Ip bereits vorhanden aber unterschiedlich
    If myPrivIp <> myIp Then
        myPrivIp = myIp
        ActiveWorkbook.SendMail "dein.name@dein.provider", "Neue IP:" & myIp
    End If
End If
'Rekursiver Aufruf um in einer Stunde erneut zu prüfen
Start_Call_IP
End Sub
```

Ist doch praktisch,... oder ;-)
Zurück

Prüfen externer Rechner mittels einer Liste mit IP-Nummern

Kann eine ganze Liste mit IP-Adressen testen und das Ergebnis in einer Tabelle ausgeben. Praktisch für WEB-Server oder sonstige Rechnerkontrolle im Netzwerk

```
Sub Check_IP_Number()
'by Ramses
'----
'Das ganze kann mit einer zusätzlichen Schleife aussenrum verwendet werden
'um zum Beispiel eine ganze Liste mit IP Adressen zui testen
'----
'Hilfsvariable für Anzahl Datensätze
Dim Text1 As String, ipAd As String
'Variablen für den Array nötig
Dim txtlines As Long, i As Long, n As Long, x As Long, wrC As Integer
'Für Office97 muss das Array "TextArr" als String definiert werden
'Entdeckt duch Gerd Z aus dem Herber Forum
Dim textArr As Variant, myCmd As Variant, myIp As String
Dim ReadFile As String
'Export der IP-Adress in ein txt File
'Den Namen und Pfad bitte anpassen
ReadFile = "C:\myIP.txt"
'Die zu pingenden Adressen beginnen in Zeile 2 der Spalte
For x = 2 To Cells(65536, 1).End(xlUp).Row
    'für einzelne Ping
    'myCmd = Shell("cmd.exe /C ping 192.168.0.2 > " & ReadFile)
    'Für Serienpings
    myCmd = Shell("cmd.exe /C ping " & Cells(x, 1).Text & " > " & ReadFile)
    'Warten um Datei schreiben zu können
    Application.Wait (Now + TimeValue("00:00:05"))
    'Erzeugt diese Einträge in der Datei
    '
    'Ping wird ausgef• hrt f• r 192.168.0.2 mit 32 Bytes Daten:
    'Antwort von 192.168.0.2: Bytes=32 Zeit=8ms TTL=128
    'Antwort von 192.168.0.2: Bytes=32 Zeit=2ms TTL=128
    'Antwort von 192.168.0.2: Bytes=32 Zeit=3ms TTL=128
    'Antwort von 192.168.0.2: Bytes=32 Zeit=3ms TTL=128
    '
    'Ping-Statistik f• r 192.168.0.2:
    '
    'Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0 (0% Verlust),
    'Ca. Zeitangaben in Millisek.:
    'Minimum = 2ms, Maximum = 8ms, Mittelwert = 4ms
    '------
    'Schliessen einer geöffneten Datei
    Close #1
```

```
    Open ReadFile For Input As #1
    'Die anzahl ist nötig um die Grösse des Arrays zu deklarieren
    'Zähler auf 0 setzen
    txtlines = 0
    Do While Not EOF(1)    ' Schleife bis Dateiende.
       Line Input #1, Text1    ' Hilfsvariable zum einlesen verwenden
       'Zähler hochzählen
       txtlines = txtlines + 1
    Loop
    'Schliessen der Datei weil Dateiende erreicht wurde
    Close #1
    'Erneutes Öffnen um zum Dateianfang zu kommen
    Open ReadFile For Input As #1    ' Datei zum Einlesen öffnen.
    'Array neu auf die Anzahl der Linien initialisieren
    ReDim textArr(txtlines)
    'Einlesen der Dateien in das Array
    For i = 1 To txtlines - 1
       Line Input #1, textArr(i)
    Next i
    Close #1
    'IP String auslesen
    For i = 1 To txtlines
       'Die IP-Adresse muss vorkommen
       If Left(textArr(i), 7) = "Antwort" And InStr(1, textArr(i), Cells(x, 1).Text) Then
          'zu füllende Spalten definieren
          wrC = 2
          ' in der übernächsten Zeile kommen die Antwortzeiten
          'Step 2 weil mit LineInput wohl noch ein Steuerzeichen importiert                                                    wird
          'das Zeilenschaltungen macht die nicht da sind
          For n = 0 To 8 Step 2
             'in den nächsten 4 Zeilen kommen die Antwortzeiten
             'Ausgabe im Direktfenster
             'Debug.Print Trim(Right(textArr(i + n), Len(textArr(i + n)) - InStrRev(textArr(i + n), ":", -1)))
             'Ausgabe in die Zellen daneben
             Cells(x, wrC) = Trim(Right(textArr(i + n), Len(textArr(i + n)) - InStrRev(textArr(i + n), ":", -1)))
             wrC = wrC + 1
          Next n
       End If
    Next i
Next x
End Sub
```

Temporäres Netzlaufwerk zuweisen wenn Laufwerksname nicht bekannt sondern nur der Server bzw. Freigabename

```
Sub Open_File_on_Not_Yet_Shared_Network_Drive()
'(C) by Ramses
'Öffnet den GetOpenFile-Dialog direkt auf eine Datei
'auf einem Netzlaufwerk dessen Server und Freigabe-Name zwar bekannt
'dessen LW-Bezeichnung jedoch nicht bekannt ist
'Kommt häufig vor in Firmen oder wenn manuelle
'Laufwerke gemappt werden, oder USB-Laufwerke
'verwendet werden, welche die normalen Laufwerksbezeichnungen
'verändern
Dim i As Byte, x As Variant
Dim OrigDrive As String, FreeDrive As String
Dim defShare As String, defPath As String, defName As String
Dim openFile As String, defDrive As String
Dim shUsername As String, shPassword As String, shDomain As String
OrigDrive = Left(ThisWorkbook.FullName, 1)
'Backslash beachten !!!
'Zur Freigabe auf dem jeweiligen Rechner
defShare = "\\ServerName\Freigabename"
'Unterstruktur bis zur Datei
defPath = "\Unterordner\"
'Dateiname
defName = "Datei.xls"
'Username der auf die Netzwerkfreigabe zugreifen darf
shUsername = "Ramses"
'Passwort des berechtigten Users
shPassword = "Ramses"
'Eventuel Domäne in der der User bekannt ist
'wenn nicht aus der lokalen Domäne
shDomain = "Domäne"
If myFreeDrive = "Null" Then
   'normales öffnen zum Browsen
   'wenn kein Laufwerksname mehr zur Verfügung steht
   MsgBox "Datei kann nicht direkt angezeigt werden." & Chr$(13) & _
      "Die Datei: """ & defName & """ liegt auf: """ & defShare & defPath & """"
   'Alle Laufwerksbezeichnungen sind verwendet
   'Der User muss nun manuell auf die Datei browsen
   openFile = Application.GetOpenFilename(defName & " (*.xls), *.xls")
Else
   defDrive = myFreeDrive
   'Erstellen eines temporären Netzlaufwerkes ohne Username und Passwort
   x = Shell("cmd.exe /C net use " & defDrive & ": " & defShare)
```

```
    'Erstellen eines temporären Netzlaufwerkes mit Username und Passwort
    'Ohne Domäne
    'x = Shell("cmd.exe /C net use " & defDrive & ": " & defShare & _
        " " & shPassword & " /User:" & shUsername)
    'Mit fremder Domäne
    'x = Shell("cmd.exe /C net use " & defDrive & ": " & defShare & _
        " " & shPassword & " /User:" & shDomain & "\" & shUsername)
    'in das Verzeichnis wechseln
    ChDrive defDrive
    'in den Unterordner wechseln
    ChDir defDrive & ":" & defPath
    'dialog anzeigen
    openFile = Application.GetOpenFilename(defName & " (*.xls), *.xls")
    'Temporäres Laufwerk wieder löschen
    x = Shell("cmd.exe /C net use " & defDrive & " /Delete")
End If
'Die gewählte Datei öffnen
Workbooks.Open (openFile)
End Sub
```

Nächsten Freien Laufwerksbuchstaben ermitteln

```
Function myFreeDrive() As String
'(c) ramses
'Sucht den nächsten freien Laufwerksnamen
Dim myFSO As Object, myDrv As Object, drvCount, drvStr As String, vName As String, drvTyp As String
Dim drvmax As Byte, drvNum As Byte
Set myFSO = CreateObject("Scripting.FileSystemObject")
Set drvCount = myFSO.drives
For Each myDrv In drvCount
   Select Case myDrv.drivetype
      'DriveType 3 sind netzlaufwerke
      'diese werden normalerweise von Z absteigend
      'gemappt. Relative Fehlerquelle !!!
      'Wenn sich der Admin nicht an die Convention hält :-)
      Case 3: drvmax = 90 - 1
      Case Else:
      'Val(Asc(myDrv.DriveLetter))
      'erstellt einen nummerischen Wert aus dem Buchstaben
         If drvNum = 0 Then
            drvNum = Val(Asc(myDrv.DriveLetter))
         End If
         If drvNum < Val(Asc(myDrv.DriveLetter)) Then
            drvNum = Val(Asc(myDrv.DriveLetter))
         End If
   End Select
Next
If drvmax <= drvNum                                    +                                    1 Then
   myFreeDrive = "Null"
Else
   'Wandelt die entsprechende Nummer + 1
   'in einen zulässigen Laufwerksbuchstaben um
   myFreeDrive = Chr(drvNum + 1)
End If
End Function
```

Wer bin ich ?

Sollte man eigentlich wissen,,.. in einem Netzwerk aber nicht immer selbstverständlich ;-)

```
                                                                                                                    'Eigenes Kürzel definieren
Private Declare Function GCN Lib "kernel32" Alias "GetComputerNameA" (ByVal myPara As String, myLen As Long) As Long
Private Declare Function GUN Lib "advapi32.dll" Alias "GetUserNameA" (ByVal myPara As String, myLen As Long) As Long
'Standardeklarationen lauten sonst im allgemeinen
'Private Declare Function GetComputerNameA Lib "kernel32" (ByVal lpBuffer As String, nSize As Long) As Long
'Private Declare Function GetUserNameA Lib "advapi32.dll" (ByVal lpBuffer As String, nSize As Long) As Long

'Prozeduren:
Public Function ActiveUserName() As String
'Benutzernamen auslesen
Dim AUN As String * 100
Dim AunLen As Byte
'100                Zeichen                reichen                in                den                meisten                Fällen                aus
AunLen = 100
If GUN(AUN, Len(AUN)) Then
   'Siehe                                                                Hinweis                                                                *
                                                                                                                    ActiveUserName = Left(AUN, AunLen)
Else
   ActiveUserName = "User can not be Identified"
End If
End Function


Public Function ActiveComputerName() As String
'Benutzernamen auslesen
Dim ACN As String * 100
Dim AcnLen As Byte
AcnLen = 100
If GCN(ACN, Len(ACN)) Then
                                                                        'Siehe                                                                Hinweis*
   ActiveComputerName = Left(ACN, AcnLen)
Else
   ActiveComputerName = "User can not be Identified"
End If
End Function


Sub wer_und_was_bin_ich()
Dim Qe As Byte
MsgBox ("Mein Rechner heisst" & ActiveComputerName)
MsgBox ("Aktuell                                                                                                    angemeldeter User ist: " & ActiveUserName)
```

```
End Sub
```

Hinweis

Das Problem das nun auftritt, ist, dass eine Überprüfung des Benutzernamens mit grosser Wahrscheinlichkeit/Sicherheit fehlschlägt. Ihr Benutzer heisst "Uwe" und eine Überprüfung in der Art

If ActiveUserName = "Uwe" Then

schlägt fehl, obwohl im Debug.Fenster der richtige Name angezeigt wird. Das Problem liegt in der Dimensionierung der Variablen "AUN As String * 100" Nach der Rückgabe aus der API-Funktion *"GetUserNameA"* wird die Variable mit sogenannten 0-Zeichen (ASCII 0) aufgefüllt bis der String eben die definierten 100 Zeichen enthält.

Testen können Sie dies, indem Sie am Punkt "If GUN...:" einen Haltepunkt setzen und mit dem Mauszeiger über die Variable AUN fahren. EXCEL zeigt Ihnen nun den Inhalt der Variablen "AUN" in einem Kommentarfeld an. Dieses wird in der Regel dann in etwa so aussehen:

"USERNAME┐┐┐┐┐┐┐┐┐┐.....

Um diese *0*-Zeichen zu eliminieren können Sie diese Funktion verwenden

Left(AUN, InStr(AUN, vbNullChar) - 1)

oder

Left(ACN, InStr(ACN, vbNullChar) - 1)

Mit "InStr" suchen Sie innerhalb des Strings AUN nach der Position des ersten Zeichen das diesem ASCII 0 Zeichen entspricht. Dieses Zeichen können sie mit "vbNullChar" identifizieren.

Von dieser Positionsnummer subtrahieren Sie 1 und haben den "reinen" Benutzernamen, bzw. den "reinen" Computernamen.

Alternativ geht's auch mit einer, etwas umständlichen ;-), eigenen Funktion der Sie die Variable "AUN" übergeben

```
Function PureName(tmpName As String)
Dim i
For i = 1 To 100
   Select Case UCase(Mid(tmpName, i, 1))
      Case vbNullChar
         'Hier wird die Position des
         'Zeichens ebenfalls gefunden
         Debug.Print "Pos: " & i
         PureName = Left(tmpName, i - 1)
         Exit For
   End Select
Next i
End Function
Sub wer_und_was_bin_ich()
MsgBox ("Mein Rechner ist" & PureName(ActiveComputerName) & PureName(ActiveUserName))
End Sub
```

Dann klappt's auch mit dem Vergleich ;-)

REGISTRIERUNG + VERSCHLÜSSELUNG

° MEINE REGISTRIERUNG UND ALLGEMEINE MÖGLICHKEITEN DER REGISTRIERUNG

Ist aus Gründen der "Aushebel-Sicherheit" nicht in diesem öffentlichen Dokument, sondern bei mir im Unterordner EXCEL/REGISTRIERUNG gespeichert inkl. der Userformen vom RGC-Tool.

Ich füge hier aber eine allgemeine Beschreibung von Registrierungsszenarien ein:

Möglichkeiten der Registrierung bzw Nutzungsbeschränkung

Da man eine Exceldatei kopieren kann, macht es nur wenig Sinn irgendwelche "Tool darf 10x gestartet werden"-Limits in die Exceldatei selbst einzubauen. Denn der Kunde zieht sich einmal am Beginn eine Kopie, deren Zähler noch auf Null ist, und zieht sich jedes Mal, wenn 10 Starts gemacht wurden, eine neue Kopie davon.

Folgende Möglichkeiten dafür gibt es:

Windows-Registrierung

Solche Start-Limits können besser in der Windows-Registrierung eingetragen werden, da diese in der Regel vom Kunden nicht manipuliert werden kann. Da müsste er schon wissen, wie man die Windows-Registrierung speichert und wiederherstellt, was normale User nicht können.

Nachteil dieser Methode: hat der Kunde zwei PCs und drei Laptops, kann er so ein Limit zumindest 4x aushebeln, indem er das Tool auf einen neuen Laptop / PC migriert.

Ein bisschen Abhilfe ist möglich, indem man die Registrierungsdaten (Datum / Uhrzeit) zusätzlich auch in das Excel-Tool selbst speichert und mit der Windows-Registrierung vergleicht. Gab es beim letzten Start des Exceltools z.B. bereits 15 registrierte Starts in der Windows-Registrierung und beim nächsten Start gar keinen, weiß man, dass das Tool auf einem neuen PC zum Einsatz kommt. Die Frage ist, ob man es dann gleich automatisch sperren darf – denn eigentlich sollte ein gekauftes Exceltool ja auch auf einem zweiten PC lauffähig sein. Diese Beschränkung auf die Nutzung nur von einem PC muss daher Teil des Kaufvertrages sein.

Sinn macht diese Überprüfung dann, wenn der User seine Daten, die er ins Tool einpflegt, weiterhin verwenden will. Beispiel: jemand erfasst seine Buchhaltung. Nach 15 Starts könnte er zwar eine Kopie der originalen Exceldatei wieder neu und leer starten, aber davon hat er dann keinen wirklichen Vorteil, da seine bisher erfassten Daten verloren gegangen sind.

Ungeeignet ist diese Methode allerdings bei einem Tool, wo der Kunde ohnedies mit einer noch nie ausgefüllten Exceldatei arbeiten möchte.

Zeit-Limits

Man kann Exceltools mit einem variablen Zeitlimit (z.B. 60 Tage) oder mit einem fixen Zeitlimit (z.B. bis 31.12.2015) versehen.

Bei der ersten Version schreibt man am ersten Tag der Nutzung dessen Datum sowohl in die Registrierung als auch in das Tool selbst und überprüft bei jedem Start ob die z.B. 60 Tage schon vorbei sind.

Nachteil: bei Tools, wo der User ohnedies mit einer noch nicht bearbeiteten Version starten möchte und er keine zuvor erfassten Daten benötigt, kann er durch Kopiervorgänge und durch Wechseln auf andere PCs/Laptops so ein Limit mehrfach aushebeln.

Für Exceltools wie eine Buchhaltungssoftware, wo der User seine bereits erfassten Daten weiter verwenden will, ist dies aber eine gute Lösung. Da man das Kopieren und Einfügen von erfassten Daten per VBA unterbinden kann, kann der User daher auch nicht die Daten aus einer abgelaufenen Exceltool-Version in eine noch nicht benutzte Kopie des Tools übertragen.

Ein sehr sicheres Limit ist ein absolutes Zeitlimit. Wenn man festlegt, dass ein Tool ab dem 1.1.2016 nicht mehr läuft, kann der User auch durch ein Verschieben des Tools auf einen neuen PC dieses Limit nicht aufheben.

Wichtig ist, dass man nicht mit dem Datum des PCs arbeitet, da dieses vom User manipuliert werden kann – sondern man über das Internet einen Zeitserver abfrägt.

Zusätzlich sollte man das aktuelle Datum beim Start des Tools eintragen, damit wenn die Meldung am 1.1.2016 an den User kommt, dass die Befristung abgelaufen ist, dieser nicht das Internet am PC abdreht (wodurch keine Zeitprüfung mehr möglich ist) und er das PC-Systemdatum zurückdreht.

Solche Manipulationsvorgänge sind durch die betreffende Protokollierung im Tool selbst gut unterbindbar. Allerdings benötigt man zur Datumsüberprüfung einen Internetzugang – daher muss es Teil des Kaufvertrages sein, dass der User bei der Verwendung des Tools seinen PC online mit dem Internet verbunden hat.

Nachteil dieser Methode: man muss in seinem Tool regelmäßig dieses fixes Zeitlimit in die Zukunft schieben. Wenn man etwa ein zwei-monatiges Zeitlimit setzen möchte z.B. anfangs fix auf den 31.12.2015, muss man es im nächsten Monat auf den 31.1.2016 setzen, im Monat darauf auf den 29.2.2016… und hat daher einen monatlichen Anpassungsbedarf.

Zweistufige Freischaltung über das Internet

Die strengste und beinahe nicht auszuhebende Registrierung geht über einen zweistufigen Freischaltungsprozess.

Die aufwendige Variante ist die von z.B. Microsoft: die Software wird so programmiert, dass sie sich mit z.B. einem SQL-Server im Internet verbindet und den Lizenzschlüssel, den der User einmalig eingeben musste, dort überprüft, ob er noch gültig ist. Bei diesem Vorgang wird in der SQL-Datenbank auch die aktuelle Nutzung erfasst – sodass dort mitgezählt wird, wie oft das Tool registriert wurde. (Microsoft erlaubt ja oft innerhalb eines gewissen Zeitraumes eine mehrfache Nutzung eines Lizenzschlüssels).

Programmtechnisch ist so eine Registrierung recht aufwändig, sowohl innerhalb des Exceltools selbst, als auch in der SQL-Datenbank. Ich persönlich habe damit noch keine Erfahrungen und kann diese Lösung daher leider nicht anbieten.

Eine schlankere Variante würde so aussehen:

Startet der User sein Excel-Tool, wird ein einmaliger Code generiert: in der Regel baut man diesen ausgehend vom Tagesdatum und der genauen Uhrzeit (inklusive Sekunde) zusammen und verschlüsselt ihn leicht, sodass er für den User nicht nachvollziehbar wird.

Aus dem Startzeitpunkt "2015-12-24 11:45:30" wird z.B. ein Hexadezimalcode im Format E2AF47. Diesen muss der Kunde mit seinem Kauflizenzschlüssel online in einem Webformular eingeben. Dieses errechnet aus diesem Startzeitpunkt-Schlüssel einen Freischaltschlüssel, den der User anschließend bei sich im Exceltool eingibt, und damit die Lauffähigkeit des Tools freischaltet.

Auf diese Weise können auch "das Tool darf nur 15x verwendet werden"-Szenarien abgebildet werden, wobei der User sich bei jeder Anwendung einmal von der Webseite online einen neuen Freischaltcode generieren lassen muss. Die Webseite muss natürlich protokollieren, wie viele Male der User schon seine Nutzung freigeschalten hat. Oder der User bekommt beim Kauf von z.B. 15 Nutzungen 15 einzelne Links zum Generieren des Freischaltcodes, die alle nur einmal aufgerufen werden können.

Die Logik auf der Webseite für die Generierung des Freischaltscodes muss natürlich gleich sein mit der Logik der Freischaltcode-Überprüfung im Exceltool. Im Prinzip reicht es eine dreifache Rechenoperation mit dem Startzeitpunkt-Schlüssels zu machen, um diesen in den Freischaltcode umzuwandeln.

Beispiel wenn der Startzeitpunktschlüssel ein 6-stelliger Hexadezimalcode ist:

1. Modulation: tausche 1. Zeichen mit 4. Zeichen, 2. Zeichen mit 6.Zeichen und 3. mit 5. Zeichen.

2. Modulation: zähle die Zahl 51842 dazu

3. Modulation: multipliziere das Ganze mal 2

Hier ein Beispielcode für eine Generierung des ersten Codes, der dann freigechalten werden muss durch einen Freischaltcode

```
Sub Lizenzcode()

Dim Datumwert As Long
Dim Zeitwert As Long
Dim Gesamtzahl As Long

Dim Meldecode
Dim Freischaltcode

Zeitwert = Second(Now) * 10000 + Minute(Now) * 100 + Hour(Now)
Datumwert = Day(Now) * 10000 + Month(Now) * 100 + Year(Now)

Gesamtzahl = 100000 + Zeitwert + Datumwert
Meldecode = Hex(Gesamtzahl)

MsgBox "Gesamtzahl: " & Gesamtzahl & " - Umgewandelt Hexadezimal " & Meldecode
```

```
End Sub
```

Demovorlage mit Demofrist versehen

Der einfachste Weg eine Vorlage mit einer Demofrist zu versehen geht über nachfolgenden Code direkt in den WORKBOOK_OPEN-Bereich

Man muss vor dem Ausrollen der Vorlage die Demofrist manuell anpassen. Ein Hochladen auf eine Homepage klappt nur, wenn die Demofrist 1 Jahr ist und man einmal jährlich eine neue Vorlage hochlädt.

Nachteil: der Klient kann sie nicht durch Eingabe eines Kennwortes in eine Vollversion umschalten.

```
Private Sub Workbook_Open()

' ----------------------------------------
' DEMOFRIST

  Dim DEMOFRIST As Date

  DEMOFRIST = "1.6.2011"
  If DEMOFRIST - Date < 0 Then ' Demofrist abgelaufen
      MsgBox "Die Demofrist dieser Vorlage ist leider abgelaufen." & vbCrLf & vbCrLf & _
      "Für eine Verlängerung der Demofrist wenden Sie sich bitte an info@simplesoft.at"
      ActiveWorkbook.Close savechanges:=False
  End If
' ----------------------------------------

End Sub
```

Excel in Registrierung neu eintragen

Excel startet nicht mehr oder verhält sich "komisch" beim öffnen von Dateien durch Doppelklick im Explorer

Start .- Ausführen - CMD

excel.exe /unregserver

anschliessend

excel.exe /regserver

eingeben.

Damit wird EXCEL neu in der Registry registriert und die meisten Probleme sollten behoben sein

Einträge in der Registry machen

Es ist von EXCEL aus möglich die Registry zu bearbeiten,... aber nicht so wie wir uns das vorstellen.
Es gibt einen speziellen Bereich in der Registry
"HKEY_USERS\UserID\Software\VB and VBA Program Settings\"
in dem Sie mit der Anweisung "SaveSetting" bzw. "GetSetting" Werte hinterlegen können, bzw. wieder auslesen können.
Auf die gesamte Registry kann man nur mit API-Funktionen oder dem **W**indows **S**cripting **H**ost (WSH) zugreifen. Aber für die meisten Anwendungen reicht das aus.

Eigene VBA-Registry Keys anlegen

```
'Option Explicit
Const RegAppName As String = "TestAnw"
Const RegKeySection As String = "StartUp"

Sub Reg_Key_setzen()
Dim i As Integer
For i = 1 To 10
    'Legt 10 verschiedene Schlüssel "i" an und setzt Werte
                                'Appname ist der Schlüsselname
        'Section der jeweilige Unterschlüssel der unter dem Schlüsselname angelegt wird
    SaveSetting appname:="TestAnw", section:="StartUp", key:=i, setting:=i + 75
    'Eintrag mit Konstanten
                'siehe oben "Const RegAppName As String = "TestAnw"
    'SaveSetting appname:=RegAppKey, section:=RegKeySection, key:=i, setting:=i + 75
Next i
End Sub
```

Registry Keys auslesen

```
Sub Reg_Key_Alle_Settings_auslesen()
Dim Einstellungen As Variant, intsettings As Integer
' Einträge Probehalber                                                    in der Registrierung vornehmen.
'       Hier werden z.B. Platzierungen für eine Userform in der Registry hinterlegt
SaveSetting appname:="TestAnw", section:="Startup", key:="Top", setting:=75
'Ohne detaillierte Anweisung
SaveSetting "TestAnw", "Startup", "Left", 50
SaveSetting "TestAnw", "Startup", "Right", 100
' Einstellungen abrufen.
Einstellungen = GetAllSettings(appname:="TestAnw", section:="Startup")
   'Anzahl der Einträge bestimmen
   For intsettings = LBound(Einstellungen, 1) To UBound(Einstellungen, 1)
      'Im Debugfenster ausdrucken
      Debug.Print Einstellungen(intsettings, 0), Einstellungen(intsettings, 1)
   Next intsettings
'Alles auf einmal im Debug.Fenster ausdrucken
Debug.Print intsettings
End Sub


Sub Reg_Key_Einzwlwert_auslesen()
Dim x As Variant
x = GetSetting(appname:="TestAnw", section:="Startup", key:="Top")
End Sub
```

Registry Keys löschen

```
Sub Reg_Key_löschen()
'Hauptschlüssel löschen
DeleteSetting "TestAnw" ', "Startup"
'Unterschlüssel löschen
'DeleteSetting "TestAnw", "Startup"
End Sub
```

So,... ich hoffe Ihnen damit ein klein wenig Informationen gegeben zu haben um Ihre Daten (z.B. Registrierungsnummern für ein Programm) Benutzer bezogen zu speichern

Gesamte Registry bearbeiten

Hier wird der Einsatz des WSH erklärt. Diese Art des Zugriffs ist dem VBA sehr ähnlich und kommt ohne API Funktionen aus. Der WSH ist normalerweise ab Windows > W2000 automatisch installiert

ACHTUNG:

Mit diesen Code-Beispielen haben Sie das Grundgerüst um die gesamte Registry zu bearbeiten, d.h. Schlüssel anlegen, ändern und löschen

Jeglicher Einsatz dieser Code-Beispiel geschieht auf eigene Gefahr. Der Autor übernimmt keinerlei Verantwortung !!!

```
Sub Create_Specific_RegKey()
'(C) by Ramses
Dim myWSH As Object, myNewRegKey As String
Dim myRegResKey As String, myRegToWriteKey As String
Set myWSH = CreateObject("WScript.Shell")
'Es wird ein spezifischer Schlüssel in DEM Registryzweig angelegt,
'der normalerweise auch von EXCEL aus mit Get- und SaveSetting erreicht werden kann
'Mit WSH kann jedoch die gesamte Registry beschrieben,
'verändert und gelesen werden
'Hier wird zu Beispielzwecken NUR ein Unterschlüssel DemoWSH Script mit Unterschlüssel Setting
'und dem Wert "Wert1" erstellt
myNewRegKey = "HKEY_CURRENT_USER\Software\VB and VBA Program Settings\DemoWSH Script\Setting\Wert1"
'Dem zu erstellen Schlüssel wird der Wert 100 zugewiesen
myWSH.regWrite myNewRegKey, "100"
End Sub


Sub Read_Specific_RegKey()
'(C) by Ramses
Dim myWSH As Object, myReadRegKey As String
Dim myRegResKey As String
'WSH Object erstellen
Set myWSH = CreateObject("WScript.Shell")
'Es muss auch der Unter-Eintrag im Key angegeben werden
'hier der                                                    Wert                                                    "Wert1"
myReadRegKey = "HKEY_CURRENT_USER\Software\VB and VBA Program Settings\DemoWSH Script\Setting\Wert1"
myRegResKey = myWSH.regread(myReadRegKey)
MsgBox "Aktueller Wert:" & myRegResKey
End Sub


Sub Change_Specific_RegKey()
'(C) by Ramses
Dim myWSH As Object, myReadRegKey As String
Dim myRegResKey As String, myRegToWriteKey As String
Set myWSH = CreateObject("WScript.Shell")
'Es muss auch der Unter-Eintrag im Key angegeben werden
'hier "Wert1"
myReadRegKey = "HKEY_CURRENT_USER\Software\VB and VBA Program Settings\DemoWSH Script\Setting\Wert1"
myRegResKey = myWSH.regread(myReadRegKey)
MsgBox "Aktueller Wert:" & myRegResKey
```

```
myRegToWriteKey = InputBox("Neuen Wert bitte eintragen:", "Registry Wert ändern", myRegResKey + 10)
If Not IsNumeric(CDbl(myRegToWriteKey)) Then
   MsgBox "Der Wert muss eine Zahl sein"
   Exit Sub
End If
'Hier wird der Schlüssel auf den neuen Wert geändert
myWSH.regWrite myReadRegKey, myRegToWriteKey
End Sub
```

Und hier zur Demonstration über die Mächtigkeit dieses Tools, das aus Ihrem Rechner die Dateien ausliest, welche als ausführbar gelten

```
Sub Read_Executable_Programs()
'(C) by Ramses
Dim myWSH As Object, myReadRegKey As String
Dim myRegResKey As String
'WSH Object erstellen
Set myWSH = CreateObject("WScript.Shell")
myReadRegKey = "HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\Programs"
myRegResKey = myWSH.regread(myReadRegKey)
MsgBox "Auf Ihrem Rechner werden: """ & myRegResKey & """ als ausführbare Dateien erkannt!"
End Sub
```

Code vopn http://www.excel-center.de/excel/handbuch.php

Registry-Eintrag auslesen

```
Const MAX_STRING As Long = 128
Public Const REG_BINARY = 3&
Public Const REG_DWORD = 4&
Declare Function RegOpenKeyA Lib "ADVAPI32.DLL" _
(ByVal hkey As Long, _
ByVal sKey As String, _
ByRef plKeyReturn As Long) As Long
Declare Function RegQueryValueExA Lib "ADVAPI32.DLL" _
(ByVal hkey As Long, _
ByVal sValueName As String, _
ByVal dwReserved As Long, _
ByRef lValueType As Long, _
ByVal sValue As String, _
ByRef lResultLen As Long) As Long
```

```
Declare Function RegCloseKey Lib "ADVAPI32.DLL" _
(ByVal hkey As Long) As Long
Public Const HKEY_CURRENT_USER = &H80000001
' Show the value of an Excel 7 entry
Sub TestShowExcelText()
MsgBox GetRegistryValue(HKEY_CURRENT_USER, _
"software\microsoft\excel\7.0\microsoft excel",
"DefaultPath")
End Sub
'Pass:
' (1) the KEY (e.g., HKEY_CLASSES_ROOT),
' (2) the SUBKEY (e.g., "Excel.Sheet.5"),
' (3) the value's name (e.g., "" [for default] or "whatever")
Function GetRegistryValue(KEY As Long, SubKey As String, _
ValueName As String) As String
Dim Buffer As String * MAX_STRING, ReturnCode As Long
Dim KeyHdlAddr As Long, ValueType As Long, ValueLen As Long
Dim TempBuffer As String, Counter As Integer
ValueLen = MAX_STRING
ReturnCode = RegOpenKeyA(KEY, SubKey, KeyHdlAddr)
If ReturnCode = 0 Then
ReturnCode = RegQueryValueExA(KeyHdlAddr, ValueName, _
0&, ValueType, Buffer, ValueLen)
RegCloseKey KeyHdlAddr
'If successful ValueType contains data type
' of value and ValueLen its length
If ReturnCode = 0 Then
Select Case ValueType
Case REG_BINARY
For Counter = 1 To ValueLen
TempBuffer = TempBuffer & _
Stretch(Hex(Asc(Mid(Buffer, Counter, 1)))) & " "
Next
GetRegistryValue = TempBuffer
Case REG_DWORD
TempBuffer = "0x"
For Counter = 4 To 1 Step -1
TempBuffer = TempBuffer & _
Stretch(Hex(Asc(Mid(Buffer, Counter, 1))))
Next
GetRegistryValue = TempBuffer
Case Else
GetRegistryValue = Buffer
End Select
```

```
Exit Function
End If
End If
GetRegistryValue = "Error"
End Function
Function Stretch(ByteStr As String) As String
If Len(ByteStr) = 1 Then ByteStr = "0" & ByteStr
Stretch = ByteStr
End Function
```

Verschlüsselung

HEXADEZIMAL-VERSCHLÜSSELUNG

Die einfachste Form eine Zahl zu verschlüsseln geht durch die Umwandlung in eine Hexidezimalzahl

Msgbox hex(255)

HEXADEZIMAL-ENTSCHLÜSSELUNG

Sub test()

HEXCODE = "FF"
MsgBox Val("&H" & HEXCODE)

End Sub

Viel kürzer werden die Zahlen dadurch aber nicht – aus 5-stelligem 99999 wird immer noch 5-stellig 1869F – aus 39000 wird zumindest schon vierstellig 9858

ALPHANUMMERISCHE-VER-/ENTSCHLÜSSELUNG von Zahlen

Ich wandle gerne Zahlen in Zeichenfolgen um, die mit den Zahlen 1-9 und den Buchstaben a-z arbeiten (ohne o). Da keine 0 und kein o vorkommt, fallen auch die Verwechslungen dabei weg. Wir haben also 34 Zeichen zur Verfügung. Und damit kann man bis zur Zahl 39303 einen dreistelligen Code machen: 39303 ist zzz. Die Zahl 0 ist 111, die Zahl 1 ist 112, …

Hier der Code

```
Public Function ENCODE(ORIGINALZAHL As Single)

' Diese Funktion wandelt eine Zahl zwischen 0 und 39303 um in eine dreistellige Ziffernfolge: 0=>111,39303=>zzz
' im Ergebnis kommen nur die Ziffern 1-9 und die Buchstaben a-z ohne o vor

Dim ZAHL As Single ' Laufvariable zum herunterzählen, um die Anzahl der 34-Zeichen-Zyklen zu ermitteln - sie enthält
danach die letzte Stelle, die man umwandeln muss
Dim ZYKLEN As Single ' Laufvariable zum herunterzählen der Gesamt-Zyklen-Anzahl - sie enthält danach die vorletzte Stelle
Dim ZYKLENGESAMT As Single ' Gesamtanzahl der 34-er Zyklen
Dim ZYKLEN34 As Single ' die Anzahl der 34x34-Zyklen
```

```
Dim STELLE1 ' 1.Stelle des Codes
Dim STELLE2 ' 2.Stelle
Dim STELLE3 ' letzte Stelle

ZAHL = ORIGINALZAHL
ZYKLENGESAMT = 0
ZYKLEN34 = 0

' Ermitteln der Gesamtanzahl 34-Zyklen

For Z = 1 To 1156 ' maximal 34x34 Zyklen
    If ZAHL >= 34 Then
        ZYKLENGESAMT = ZYKLENGESAMT + 1
        ZAHL = ZAHL - 34
    End If
Next Z

' Zählen, wieviele 34x34-Zyklen in der Gesamtzahl der 34-Zyklen enthalten sind

ZYKLEN = ZYKLENGESAMT

For Z = 1 To 34
    If ZYKLEN >= 34 Then
        ZYKLEN34 = ZYKLEN34 + 1
        ZYKLEN = ZYKLEN - 34
    End If
Next Z

STELLE3 = HELPENCODE(ZAHL)
STELLE2 = HELPENCODE(ZYKLEN)
STELLE1 = HELPENCODE(ZYKLEN34)

' MsgBox "Originalzahl " & ORIGINALZAHL & " - Verschlüsselt " & STELLE1 & STELLE2 & STELLE3
ENCODE = STELLE1 & STELLE2 & STELLE3

End Function
Public Function HELPENCODE(ZAHL As Single)

' Hilfsfunktion für die Funktion ENCODE, die eine Zahl zwischen 0 und 33 umwandelt in die Ziffern 1-9 und die Buchstaben
A-Z ohne O

    Select Case ZAHL
```

```
Case 0 To 8
    HELPENCODE = ZAHL + 1
Case 9
    HELPENCODE = "a"
Case 10
    HELPENCODE = "b"
Case 11
    HELPENCODE = "c"
Case 12
    HELPENCODE = "d"
Case 13
    HELPENCODE = "e"
Case 14
    HELPENCODE = "f"
Case 15
    HELPENCODE = "g"
Case 16
    HELPENCODE = "h"
Case 17
    HELPENCODE = "i"
Case 18
    HELPENCODE = "j"
Case 19
    HELPENCODE = "k"
Case 20
    HELPENCODE = "l"
Case 21
    HELPENCODE = "m"
Case 22
    HELPENCODE = "n"
Case 23
    HELPENCODE = "p"
Case 24
    HELPENCODE = "q"
Case 25
    HELPENCODE = "r"
Case 26
    HELPENCODE = "s"
Case 27
    HELPENCODE = "t"
Case 28
    HELPENCODE = "u"
```

```
        Case 29
            HELPENCODE = "v"
        Case 30
            HELPENCODE = "w"
        Case 31
            HELPENCODE = "x"
        Case 32
            HELPENCODE = "y"
        Case 33
            HELPENCODE = "z"
    End Select

End Function
Public Function DECODE(ZEICHEN As String)
    DECODE = HELPDECODE(Left(ZEICHEN, 1)) * 34 * 34 + HELPDECODE(Mid(ZEICHEN, 2, 1)) * 34 + HELPDECODE(Right(ZEICHEN, 1))
End Function

Public Function HELPDECODE(ZEICHEN As String)

' Hilfs-Funktion für die Funktion DECODE, die ein Zeichen der Ziffern 1-9 und der Buchstaben a-z (ohne o) umwandelt in
eine Ziffer

    Select Case ZEICHEN
        Case "1"
            HELPDECODE = 0
        Case "2"
            HELPDECODE = 1
        Case "3"
            HELPDECODE = 2
        Case "4"
            HELPDECODE = 3
        Case "5"
            HELPDECODE = 4
        Case "6"
            HELPDECODE = 5
        Case "7"
            HELPDECODE = 6
        Case "8"
            HELPDECODE = 7
        Case "9"
            HELPDECODE = 8
        Case "a"
```

```
    HELPDECODE = 9
Case "b"
    HELPDECODE = 10
Case "c"
    HELPDECODE = 11
Case "d"
    HELPDECODE = 12
Case "e"
    HELPDECODE = 13
Case "f"
    HELPDECODE = 14
Case "g"
    HELPDECODE = 15
Case "h"
    HELPDECODE = 16
Case "i"
    HELPDECODE = 17
Case "j"
    HELPDECODE = 18
Case "k"
    HELPDECODE = 19
Case "l"
    HELPDECODE = 20
Case "m"
    HELPDECODE = 21
Case "n"
    HELPDECODE = 22
Case "p"
    HELPDECODE = 23
Case "q"
    HELPDECODE = 24
Case "r"
    HELPDECODE = 25
Case "s"
    HELPDECODE = 26
Case "t"
    HELPDECODE = 27
Case "u"
    HELPDECODE = 28
Case "v"
    HELPDECODE = 29
Case "w"
```

```
            HELPDECODE = 30
        Case "x"
            HELPDECODE = 31
        Case "y"
            HELPDECODE = 32
        Case "z"
            HELPDECODE = 33
    End Select

End Function

Sub TEST_ENCODE()

    MsgBox (ENCODE(InputBox("Gib eine Zahl zwischen 0 und 39303 ein")))

End Sub

Sub TEST_DECODE()

    MsgBox (DECODE(InputBox("Gib 3 Zeichen mit den Ziffern 1-9 und den Buchstaben a-z ohne o ein ein")))

End Sub
```

REPORTING

Allgemein

```
Public Sub AUS()

' Schaltet Bildschirmaktualisierung und Events ab

    Application.ScreenUpdating = False
    Application.EnableEvents = False
    ' Application.Calculation = xlCalculationManual

End Sub
```

```
Public Sub EIN()

' Schaltet Bildschirmaktualisierung und Events ein

   Application.ScreenUpdating = True
   Application.EnableEvents = True
   ' Application.Calculation = xlCalculationAutomatic

End Sub


Public Function LETZTEZELLE(TABELLENBLATT As String) As Range
' Neue Version 2015

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
```

```
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function
```

Daten Importieren über Zwischenablage

```
Sub Zwischenablage_einfuegen()

' Fügt die Zwischenablage ein - jedoch über eine ausgeblendete Zwischentabelle "Zwischenablage"
' diese Zwischentabelle darf nicht jedes Mal neu erzeugt werden (Sheets.add), da dadurch die Zwischenablage geleert wird
' daher: sie wird beim ersten Mal angelegt, wenn es sie noch nicht geben sollte - und dann immer ausgeblendet


Dim ERSTE_QUELLZEILE As Integer ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
Dim ERSTE_QUELLSPALTE As Integer ' was ist die erste Spalte von der Zwischenablage
Dim ERSTE_ZIELZEILE As Integer ' was ist die erste Zeile in der empfangenden Zieltabelle, ab der die Daten eingefügt
werden sollen
Dim ERSTE_ZIELSPALTE As Integer ' was ist die erste Spalte in der empfangenden Zieltabelle, ab der die Daten eingefügt
werden sollen
Dim AUFTRENNUNG_TRENNZEICHEN As Integer ' wurde eine TXT-Datei eingefügt, die mit Trennzeichen getrennte Daten enthält -
1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, 0=keine Trennung

Dim AKTUELLE_TABELLE As String ' Für den Rücksprung in die aktuelle Tabelle



' ****************************************************************************************************
' -----------------
' Variablen setzen
' -----------------
```

```
' Hier bitte die Einstellungen für die Prozedur machen

    ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
    ERSTE_QUELLZEILE = 1

    ' was ist die erste Spalte von der Zwischenablage, die kopiert werden soll - in der Regel will man immer alle
Spalten, daher ab Spalte 1
    ERSTE_QUELLSPALTE = 1

    ' was ist die erste Zeile in der empfangenden Zielzeile, ab der die Daten eingefügt werden sollen
    ERSTE_ZIELZEILE = 2

    ' was ist die erste Spalte in der empfangenden Zielzeile, ab der die Daten eingefügt werden soll - in der Regel ab
Spalte 1
    ERSTE_ZIELSPALTE = 1

    ' sollen TXT-Daten aus der Zwischenablage noch aufgetrennt werden ?
    ' BSP: 0;100002377;20150629;4000 ... soll in 4 eigenständige Spalten getrennt werden
    ' 0= keine Auftrennung - 1=Semikolon - 2=Beistrich - 3=Tabulator - 4=Leerzeichen
    AUFTRENNUNG_TRENNZEICHEN = 1

' ************************************************************************************************************

    Application.ScreenUpdating = False

' merken des Namens der aktuellen Tabelle
    AKTUELLE_TABELLE = ActiveSheet.Name

' Falls nachfolgender Code nicht geht, weil die Tabelle Zwischenablage noch nicht vorhanden ist, soll Code zum
Tabellenanlegecode verzweigen
    On Error GoTo ZWISCHENABLAGE_ANLEGEN

' Zwischenablage_Tabelle einblenden
    Worksheets("Zwischenablage").Visible = True

' Eigentliche Fehlerabfangroutine, wenn Zwischenablage leer ist
    On Error GoTo FEHLER

' Einfügen der Daten aus der Zwischenablage in der Interimstabelle ZWISCHENABLAGE
    Worksheets("Zwischenablage").Activate
```

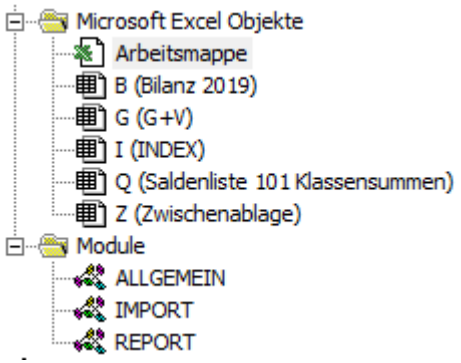
```
    Range("A1").Select ' wir fügen die Daten immer ab der ersten Zeile in Spalte A ein - eine eventuell andere erste
Zelle in der Zieltabelle wird unten eingestellt
    ActiveSheet.Paste

' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten)
    Sheets(AKTUELLE_TABELLE).Activate
    Sheets(AKTUELLE_TABELLE).Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE),
Cells(Sheets(AKTUELLE_TABELLE).UsedRange.Rows.Count, Sheets(AKTUELLE_TABELLE).UsedRange.Columns.Count)).ClearContents

' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, ERSTE_QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row,
LETZTEZELLE(ActiveSheet.Name).Column)).Copy

' alternatives Einfügen, wenn der Code der Hilfsprozedur LETZTEZELLE nicht vorhanden wäre
    ' Range(Cells(1, 1), Cells(ActiveSheet.Name.UsedRange.Rows.Count, ActiveSheet.Name.UsedRange.Columns.Count)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' Falls aus Zwischenablage keine Excelzellen-Daten kommen, sondern aus zB dem Editor die Werte einer TXT-Datei
' die noch aufgetrennt werden müssen - zB 0;100002377;20150629;4000 ...
' Hier gemäß dem Trennzeichen erfolgt die Auftrennung: 1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, (0=keine
Trennung)

    If AUFTRENNUNG_TRENNZEICHEN > 0 Then ' falls Auftrennung gewünscht, markiere die Spalte, in der alle Daten sind
        Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE), Cells(ActiveSheet.UsedRange.Rows.Count, ERSTE_ZIELSPALTE)).Select
    End If

    Select Case AUFTRENNUNG_TRENNZEICHEN
        Case 1 ' Semikolon
            Selection.TextToColumns DataType:=xlDelimited, Semicolon:=True
        Case 2 ' Beistrich
            Selection.TextToColumns DataType:=xlDelimited, Comma:=True
        Case 3 ' Tabulator
            Selection.TextToColumns DataType:=xlDelimited, Tab:=True
        Case 4 ' Leerzeichen
            Selection.TextToColumns DataType:=xlDelimited, Space:=True
```

```
    End Select

' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
    Worksheets("Zwischenablage").Activate
    ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
    Worksheets("Zwischenablage").Visible = False

' Rückkehr zur aktuellen Tabelle mit den eingefügten Daten
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_QUELLZEILE, ERSTE_QUELLSPALTE).Select

' Hinweismeldung

    Application.ScreenUpdating = True

    If Range("A2") <> "Firma" Or Range("C2") <> "Kontoklasse" Or Range("D2") <> "KontoNr" Then
            MsgBox "Die Daten wurden eingefügt - aber Sie scheinen nicht im richtigen Format zu sein. Gebraucht wird eine
Saldenliste 101 Jahr/Monat mit Klassensummen"
        Else
            MsgBox "Die Daten wurden eingefügt"
    End If
' ------------------------
' Ende des normalen Codes
' ------------------------

Exit Sub

' ------------------------------------------------
' Fehlerroutine 1, wenn keine Daten in Zwischenablage
' ------------------------------------------------

FEHLER:
    Sheets(AKTUELLE_TABELLE).Activate
    Application.ScreenUpdating = True
    MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte erneut die Daten kopieren"
    Exit Sub

' ------------------------------------------------
' Fehlerroutine 2, wenn Tabelle Zwischenablage fehlt
' ------------------------------------------------
```

```
ZWISCHENABLAGE_ANLEGEN:

    Worksheets.Add.Move After:=Worksheets(Worksheets.Count): ' Neues Tabellenblatt am Ende anlegen
    ActiveSheet.Name = "Zwischenablage": ' Neues Blatt umbenennen
    MsgBox "Die Hilfstabelle 'Zwischenablage' war noch nicht vorhanden und musste erst erzeugt werden." & vbCrLf & vbCrLf
& _
    "Dadurch wurden leider die Daten in der Zwischenablage geleert. Bitte wiederholen Sie noch einmal das Kopieren der
Daten in die Zwischenablage und klicken Sie dann erneut auf den Button hier für das Einfügen der Zwischenablage - danke."
    Sheets(AKTUELLE_TABELLE).Activate

End Sub
```

Bilanzkonten 1-X,GuV-Konten X

So sieht unsere Mappe aus: Tabelle Q enthält die Quelldaten aus NTCS - eine 101-Saldenliste mit Klassensummen
In Tabelle B kommen die Werte 1-X, in Tabelle G nur der Umsatz des Monats X



In der B und G - Tabelle gibt es ab Spalte AA die Index-Hinterlegung (Spalte AC enthält Spalte AA & AB, weil dies in der Indextabelle mit DATEN ÜBERPRÜFEN als erlaubte Datenliste angeboten wird):

Hier die B(ilanz)-Tabelle (GuV-Tabelle sieht genauso aus, hat aber Indexnummern ab 50)

AA	AB	AC
INDEX		
NUMME		
1	BANKEN UND POSTKONTO	1 BANKEN UND POSTKONTO
2	KASSE	2 KASSE
3	FORDERUNGEN	3 FORDERUNGEN
4	FORDERUNGEN IM STREITVERFAHREN	4 FORDERUNGEN IM STREITVERFAHREN

Hier die Index-Tabelle:

A	B	C
Kto.Nr.	Kontoname	INDEX NUMMER
150	Sonstige Bauten auf fremden Grundstück	13 MATERIALFESTLEGUNGEN
160	Bauliche Invest. in fremden Betriebsge	
400	Lagereinrichtung	
410	Allgemeine Betriebsausstattung	
420	Büroeinrichtung	
460	Datenverarbeitungsanlagen	
490	Geringwertige Wirtschaftsgüter	13 MATERIALFESTLEGUNGEN
500	Lastkraftwagen	13 MATERIALFESTLEGUNGEN

1 BANKEN UND POSTKONTO
2 KASSE

3 FORDERUNGEN
4 FORDERUNGEN IM STREITVERFAHREN
5 FINANZKREDITE
6 SONSTIGE FORDERUNGEN

In beiden Zieltabellen B und G gibt es ganz unten eine Zeile NICHT AUSWERTEN mit der Indexnummer 49 bzw 99

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Z	AA	AB	AC
VERMÖGENSBESTAND															INDEX		
															NUMME		
	Jan 18	Feb 18	Mrz 18	Apr 18	Mai 18	Jun 18	Jul 18	Aug 18	Sep 18	Okt 18	Nov 18	Dez 18					
NICHT AUSGEWERTET															49	NICHT AUSWERTEN	49 NICHT AUSWERTEN

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Z	AA	AB	
VERMÖGENSBESTAND															INDEX		
															NUMME		
	Jan 18	Feb 18	Mrz 18	Apr 18	Mai 18	Jun 18	Jul 18	Aug 18	Sep 18	Okt 18	Nov 18	Dez 18					
NICHT AUSGEWERTET															99	NICHT AUSWERTEN	99 NICHT AUSWERTEN

Und hier der Code für das Daten_Aufbereiten

```
Sub START()

' ---------------------------------------
' Dies ist der Code für den Start-Knopf
' ---------------------------------------

' --- Variablen-Definition ---

' Allgemeine Variablen-Definition

Dim EZ ' Erste Zeile
Dim LZ ' Letztezeile
Dim MONAT As Integer ' Nummer des aktuellen Monats
Dim MONATSTEXT As String

Dim ZIELSPALTE ' Spalte in der Controllingtabelle mit dem richtigen Monat
Dim BETRAG ' der zu übertragende Betrag

Dim KONTO
Dim INDEXZEILE
Dim KONTONAME as String

Dim Z ' Laufvariable durch die Zeilen
Dim S ' Laufvariable durch die 12 Monatsspalten - in der Zieltabelle greift S direkt auf die Monatsspalte zu - in der
Quelltabelle sind die Daten um eine gewisse Anzahl von Spalten (in unserem Beispiel um 6) weiter rechts

' Variablen für Suche

   Dim SUCHKRITERIUM        ' Danach wird gesucht
   Dim FUNDZELLE As Range   ' hier wurde es gefunden
   Dim INDEX                ' Die Indexzeilennummer des Kontos

' Vorbereitung

    AUS ' Keine Bildschirm-Aktualisierung - keine VBA-Events


' Auslesen des Monats (In F2 steht zB Gesamt 1 - 8/19 oder Gesamt 1 - 10/19)

    On Error GoTo ABFRAGE_MONAT
```

```
    MONATSTEXT = Mid(Range("F2"), InStr(Range("F2"), "/") - 2, 2)
    If Left(MONATSTEXT, 1) = " " Then MONATSTEXT = Right(MONATSTEXT, 1)
    MONAT = Val(MONATSTEXT)
    On Error GoTo 0

MONAT_OK:

' ------------------------------------------------------------------------------------------------------------------------
' Leeren der aktuellen Monatsspalte in den beiden Controllingtabellen Bilanz und GuV in den Zeilen, die eine Indexnummer
in Spalte AA haben
' ------------------------------------------------------------------------------------------------------------------------

    B.Activate ' Bilanztabelle
    For Z = 4 To 500 ' Schleife durch alle Zeilen der Zieltabelle
        If Cells(Z, 27) <> "" Then Cells(Z, MONAT + 1).ClearContents ' Wenn die Indexspalte (27/AA) eine Indexnummer
hat=>Zelle in Monatsspalte (Jan beginnt in Spalte 2, daher +1) wird geleert
    Next Z

    G.Activate ' Bilanztabelle
    For Z = 4 To 500 ' Schleife durch alle Zeilen der Zieltabelle
        If Cells(Z, 27) <> "" Then Cells(Z, MONAT + 2).ClearContents ' Wenn die Indexspalte (27/AA) eine Indexnummer
hat=>Zelle in Monatsspalte (Jan beginnt in Spalte 3, daher +2) wird geleert
    Next Z

' ------------------------------------------------
' Übertragen der Daten
' ------------------------------------------------

    ' Springe in Quelldatentabelle
    Q.Activate

    ' Suchen der ersten Zeile mit Daten
    For Z = 1 To 99
        If Cells(Z, 4) = "KontoNr" Then EZ = Z + 1: GoTo EZ_GEFUNDEN
    Next Z
    MsgBox "Die erste Zeile mit Daten wurde nicht gefunden - offensichtlich haben sich die Quelldaten vom Satzaufbau
verändert"
    End

EZ_GEFUNDEN:
```

```
    ' --- Schleife durch alle Zeilen der Quelltabelle ---

    LZ = LETZTEZELLE(Q.Name).Row

    For Z = EZ To LZ

        Q.Activate ' bei jedem Schleifendurchlauf verlässlich zurückspringen in Quelltabelle

        ' beende die Schleife, wenn Schlusszeile erreicht (hier nicht notwendig, da die letzte Zelle ausreichend in LZ
definiert ist)
        ' If Cells(Z, 1) = "" Then Exit For

        ' Ausschlusskriterium falsche Kontenklasse (hier nicht notwendig, da wir alle Konten übertragen)
        ' If Cells(Z, 5) < 1000 Then ' wenn Kontonummer von Klasse 0, dann nächste Zeile
        '    GoTo WEITER
        ' End If

        ' Summenzeilen werden auch nicht übernommen
        If Cells(Z, 4) = "" Then ' wenn Summenzeile (kein Wert in Spalte D mit Kontonummer)
            GoTo WEITER ' dann nächste Zeile
        End If

        ' ab hier der eigentliche Übertrag der Beträge

            SUCHKRITERIUM = Cells(Z, 4) ' Auslesen Kontonummer
            KONTO = SUCHKRITERIUM ' Nur für Fehlermeldung wichtig
            KONTONAME = Cells(Z, 5) ' Auslesen des Kontonamens für Neuanlagen^

            ' Springe in Indextabelle
            I.Activate

            ' Suche die Zeile mit der Kontonummer
            Range("A1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
            Set FUNDZELLE = Columns("A").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole)
            If FUNDZELLE Is Nothing Then ' wenn betreffendes Konto nicht gefunden wurde
                EIN
                MsgBox "Achtung - das Konto " & SUCHKRITERIUM & " von der Quelldaten-Tabelle wurde in der Indextabelle
nicht gefunden. Bitte legen Sie es dort an und wiederholen den Vorgang."
                I.Activate
                Z = LETZTEZELLE(ActiveSheet.Name).Row + 1 ' nächste leere Zeile
                Cells(Z, 1) = KONTO
```

```
                Cells(Z, 2) = KONTONAME
                Cells(Z, 3).Activate
                End
            End If

            ' Lies die Indexnummer aus
            If Cells(FUNDZELLE.Row, 3) = "" Then
                Cells(FUNDZELLE.Row, 3).Select
                EIN
                MsgBox "Achtung - das Konto " & KONTO & " hat in der Indextabelle in Zeile " & FUNDZELLE.Row & " keine
Indexnummer. Bitte legen Sie diese dort in Spalte C an und wiederholen den Vorgang." & vbCrLf & vbCrLf & _
                "(Sie haben die Möglichkeit das Konto auch auszugliedern, sodass es gar nicht ausgewertet wird - dazu
wählen Sie in der INDEX-Tabelle beim Konto in der Spalte C in der Dropdownliste den untersten Eintrag 'NICHT AUSWERTEN')"
                I.Activate
                End
            End If
            INDEX = Left(Cells(FUNDZELLE.Row, 3), InStr(Cells(FUNDZELLE.Row, 3), " ") - 1)

            SUCHKRITERIUM = INDEX
            INDEXZEILE = FUNDZELLE.Row ' Nur für Fehlermeldung wichtig

            If Val(INDEX) < 50 Then ' wenn ein Bestandskonto (Indexnummer bis 50)

                ' Springe in Bilanz-Tabelle
                B.Activate

                ' Suche die Zeile mit der Indexnummer
                Range("AA1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
                Set FUNDZELLE = Columns("AA").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues,
LookAt:=xlWhole)

                If FUNDZELLE Is Nothing Then ' wenn betreffendes Konto nicht gefunden wurde
                    EIN
                    MsgBox "Achtung - die Indexnummer " & INDEX & " des Kontos " & KONTO & " wurde in der Indextabelle
nicht gefunden. Bitte tragen Sie sie dort in der Spate AA ein und wiederholen den Vorgang."
                    I.Activate
                    End
                End If

                ' Buchungsbetrag in der aktuellen Monatsspalte übertragen / Bestandskonten haben in der Quelltabelle
(NTCS-Saldenliste) in Spalte F den Monat 1-X-Wert
```

```
                If Q.Cells(Z, 6) <> "" And Q.Cells(Z, 6) <> 0 Then  ' Übertragen nur, wenn es Werte gibt

                    If Q.Cells(Z, 6) < 0 Then
                        ' wenn Betrag/Verbindlichkeit mit Minus => wir wollen die im Controlling mit MINUS haben
                        If Cells(FUNDZELLE.Row, MONAT + 1) = "" Then
                            Cells(FUNDZELLE.Row, MONAT + 1).Formula = "=-" & Abs(Val(Q.Cells(Z, 6)))
                        Else
                            Cells(FUNDZELLE.Row, MONAT + 1).Formula = Cells(FUNDZELLE.Row, MONAT + 1).Formula & "-" &
Abs(Val(Q.Cells(Z, 6)))
                        End If
                    Else
                        ' Wenn Betrag/Forderung mit Plus - wir wollen die mit Plus
                        If Cells(FUNDZELLE.Row, MONAT + 1) = "" Then
                            Cells(FUNDZELLE.Row, MONAT + 1).Formula = "=" & Val(Q.Cells(Z, 6))
                        Else
                            Cells(FUNDZELLE.Row, MONAT + 1).Formula = Cells(FUNDZELLE.Row, MONAT + 1).Formula & "+" &
Val(Q.Cells(Z, 6))
                        End If
                    End If

                End If

            Else ' GuV-Konto


                ' Springe in GuV-Tabelle
                G.Activate

                ' Suche die Zeile mit der Indexnummer
                Range("AA1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
                Set FUNDZELLE = Columns("AA").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues,
LookAt:=xlWhole)

                If FUNDZELLE Is Nothing Then ' wenn betreffendes Konto nicht gefunden wurde
                    EIN
                    MsgBox "Achtung - die Indexnummer " & INDEX & " des Kontos " & KONTO & " wurde in der Indextabelle
nicht gefunden. Bitte tragen Sie sie dort in der Spate AA ein und wiederholen den Vorgang."
                    I.Activate
                    End
                End If
```

```
                ' Buchungsbetrag in der aktuellen Monatsspalte übertragen / GuV-Konten haben in der Quelltabelle (NTCS-
Saldenliste) in Spalte H den Monat X-Wert

                If Q.Cells(Z, 8) <> "" And Q.Cells(Z, 8) <> 0 Then  ' Übertragen nur, wenn es Werte gibt

                    If Q.Cells(Z, 8) < 0 Then
                        ' wenn Betrag (Aufwand oder Erlösminderung) mit Minus => wir wollen ihn im Controlling mit PLUS
haben
                        If Cells(FUNDZELLE.Row, MONAT + 2) = "" Then
                            Cells(FUNDZELLE.Row, MONAT + 2).Formula = "=" & Abs(Val(Q.Cells(Z, 8)))
                           Else
                            Cells(FUNDZELLE.Row, MONAT + 2).Formula = Cells(FUNDZELLE.Row, MONAT + 2).Formula & "+" &
Abs(Val(Q.Cells(Z, 8)))
                        End If
                      Else
                        ' Wenn eine Einnahme (oder Ausgabengutschrift) mit Plus - wir wollen sie mit Minus
                        If Cells(FUNDZELLE.Row, MONAT + 2) = "" Then
                            Cells(FUNDZELLE.Row, MONAT + 2).Formula = "=-" & Val(Q.Cells(Z, 8))
                           Else
                            Cells(FUNDZELLE.Row, MONAT + 2).Formula = Cells(FUNDZELLE.Row, MONAT + 2).Formula & "-" &
Val(Q.Cells(Z, 8))
                        End If
                    End If

                End If


            End If
WEITER:
    Next Z



' ------------------------------------------------
' Abschluss
' ------------------------------------------------


EIN

B.Activate
```

```
End


' ----------------------------------------
' --- Abfrage des Auswertungsmonats
' ----------------------------------------
' Fehlerroutine, falls kein Monat ermittelt werden konnte

ABFRAGE_MONAT:

    MONAT = Month(Date) - 1 ' Default Auswertungsmonat ist das Vormonat
    If MONAT = 0 Then MONAT = 12
    MONAT = InputBox("Welches Monat möchten Sie auswerten ?: ", MONAT, "Auswertungsmonat")

    GoTo MONAT_OK ' Rückkehr zum Hauptcode

End Sub
```

Periodensaldenliste mittels INDEX-Tabelle in Controlling übertragen

```
Sub START()

' ------------------------------------
' Dies ist der Code für den Start-Knopf
' ------------------------------------

' --- Variablen-Definition ---

' Allgemeine Variablen-Definition

Dim EZ ' Erste Zeile
Dim LZ ' Letztezeile
Dim MONAT As Integer ' Nummer des aktuellen Monats

Dim ZIELSPALTE ' Spalte in der Controllingtabelle mit dem richtigen Monat
Dim BETRAG ' der zu übertragende Betrag
```

```
Dim KONTO
Dim INDEXZEILE

Dim Z ' Laufvariable durch die Zeilen
Dim S ' Laufvariable durch die 12 Monatsspalten - in der Zieltabelle greift S direkt auf die Monatsspalte zu - in der Quelltabelle sind die Daten um eine gewisse Anzahl von Spalten (in unserem Beispiel um 6) weiter rechts

' Variablen für Suche

   Dim SUCHKRITERIUM        ' Danach wird gesucht
   Dim FUNDZELLE As Range   ' hier wurde es gefunden
   Dim INDEX                ' Die Indexzeilennummer des Kontos

' Vorbereitung

    AUS ' Keine Bildschirm-Aktualisierung - keine VBA-Events

' Abfrage des Auswertungsmonats

    MONAT = Month(Date) - 1 ' Default Auswertungsmonat ist das Vormonat
    If MONAT = 0 Then MONAT = 12
    MONAT = InputBox("Welches Monat möchten Sie auswerten ?: ", MONAT, "Auswertungsmonat")

' ------------------------------------------------
' Leeren der Indexzeilen in der Controllingtabelle
' ------------------------------------------------

    T5.Activate ' Zieltabelle leeren

    For Z = 2 To 500 ' Schleife durch alle Zeilen der Zieltabelle
        If Cells(Z, 53) <> "" Then Range(Cells(Z, 2), Cells(Z, 13)).ClearContents ' Wenn die Indexspalte (53/BA) eine Indexnummer hat=>sie wird geleert
    Next Z

' ------------------------------------------------
' Übertragen der Daten
' ------------------------------------------------

    ' Springe in Quelldatentabelle
    T3.Activate

    ' Suchen der ersten Zeile mit Daten
```

```
    For Z = 1 To 99
        If Cells(Z, 1) = "Konto" Then GoTo EZ_GEFUNDEN
    Next Z
    MsgBox "Die erste Zeile mit Daten wurde nicht gefunden - offensichtlich haben sich die Quelldaten vom Satzaufbau
verändert"
    End

EZ_GEFUNDEN:

    ' --- Schleife durch alle Zeilen der Quelltabelle ---

    LZ = LETZTEZELLE(T3.Name).Row

    For Z = EZ To LZ

        T3.Activate ' bei jedem Schleifendurchlauf verlässlich zurückspringen in Quelltabelle

        ' beende die Schleife, wenn Schlusszeile erreicht
        If Cells(Z, 1) = "" Then Exit For

        ' Ausschlusskriterium falsche Kontenklasse
        If Cells(Z, 5) < 1000 Then ' wenn Kontonummer von Klasse 0, dann nächste Zeile
            GoTo WEITER
        End If

        ' Summenzeilen werden auch nicht übernommen
        If Cells(Z, 9) = "" Then ' wenn Summenzeile
            GoTo WEITER ' dann nächste Zeile
        End If

        ' ab hier der eigentliche Übertrag der Beträge

            SUCHKRITERIUM = Cells(Z, 5) ' Auslesen Kontonummer
            KONTO = SUCHKRITERIUM ' Nur für Fehlermeldung wichtig

            ' Springe in Indextabelle
            T2.Activate

            ' Suche die Zeile mit der Kontonummer
            Range("C1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
            Set FUNDZELLE = Columns("C").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole)
            If FUNDZELLE Is Nothing Then ' wenn betreffendes Konto nicht gefunden wurde
```

```
                EIN
                MsgBox "Achtung - das Konto " & SUCHKRITERIUM & " von der Quelldaten-Tabelle wurde in der Indextabelle
nicht gefunden. Bitte legen Sie es dort an und wiederholen den Vorgang."
                T2.Activate
                End
            End If

            ' Lies die Indexnummer aus
            INDEX = Cells(FUNDZELLE.Row, 5)
            If INDEX = "" Then
                Cells(FUNDZELLE.Row, 5).Select
                EIN
                MsgBox "Achtung - das Konto " & KONTO & " hat in der Indextabelle in Zeile " & FUNDZELLE.Row & " keine
Indexnummer. Bitte legen Sie diese dort in Spalte E an und wiederholen den Vorgang."
                T2.Activate
                End
            End If

            SUCHKRITERIUM = INDEX
            INDEXZEILE = FUNDZELLE.Row ' Nur für Fehlermeldung wichtig

            ' Springe in Controllingtabelle
            T5.Activate

            ' Suche die Zeile mit der Indexnummer
            Range("BA1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
            Set FUNDZELLE = Columns("BA").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole)

            If FUNDZELLE Is Nothing Then ' wenn betreffendes Konto nicht gefunden wurde
                EIN
                MsgBox "Achtung - die Indexnummer " & INDEX & " des Kontos " & KONTO & " wurde in der Indextabelle nicht
gefunden. Bitte tragen Sie sie dort in der Spate BA ein und wiederholen den Vorgang."
                T2.Activate
                End
            End If

            ' Buchungsbetrag übertragen
            For S = 2 To 13 ' Schleife durch die max. 12 Monatsspalten (
                If S > MONAT + 1 Then Exit For ' Aussteigen wenn schon nicht mehr im aktuellen Monat

                If T3.Cells(Z, S + 6) <> "" And T3.Cells(Z, S + 6) <> 0 Then  ' Übertragen nur, wenn es Werte gibt
```

```
                    If T3.Cells(Z, S + 6) < 0 Then
                        ' wenn eine Ausgabe mit Minus => wir wollen die im Controlling mit PLUS haben
                        If Cells(FUNDZELLE.Row, S) = "" Then
                            Cells(FUNDZELLE.Row, S).Formula = "=" & Abs(Val(T3.Cells(Z, S + 6)))
                           Else
                            Cells(FUNDZELLE.Row, S).Formula = Cells(FUNDZELLE.Row, S).Formula & "+" & Abs(Val(T3.Cells(Z,
S + 6)))
                        End If
                      Else
                        ' Wenn eine Einnahme (oder Ausgabengutschrift) mit Plus - wir wollen sie mit Minus
                        If Cells(FUNDZELLE.Row, S) = "" Then
                            Cells(FUNDZELLE.Row, S).Formula = "=-" & Val(T3.Cells(Z, S + 6))
                           Else
                            Cells(FUNDZELLE.Row, S).Formula = Cells(FUNDZELLE.Row, S).Formula & "-" & Val(T3.Cells(Z, S +
6))
                        End If
                    End If

                End If

            Next S

WEITER:
    Next Z



' ------------------------------------------------
' Abschluss
' ------------------------------------------------


EIN

T5.Activate

End Sub

Public Sub AUS()

' Schaltet Bildschirmaktualisierung und Events ab
```

```
    Application.ScreenUpdating = False
    Application.EnableEvents = False
    ' Application.Calculation = xlCalculationManual

End Sub
Public Sub EIN()

' Schaltet Bildschirmaktualisierung und Events ein

   Application.ScreenUpdating = True
   Application.EnableEvents = True
   ' Application.Calculation = xlCalculationAutomatic

End Sub
```

SCHLEIFEN + WENN

For To Schleifen mit dynamischem Endwert

For Z=1 to LZ - Schleifen funktionieren solange gut, als sich die Endvariable LZ nicht ändert. Füge ich neue Zeilen ein (LZ=LZ+1) oder lösche Zeilen (LZ=LZ-1) hätte ich erwartet, dass Excel den Schleifedurchlauf dynamisch anpassen kann.

Das ist aber nicht so - Excel liest beim Betreten der FOR - Schleife einmal die Variable LZ aus und so oft läuft er dann automatisch durch, auch wenn sich die Variable LZ durch den Code in der Schleife ändert.

Lösungsmöglichkeiten:

Eine Exit-Bedingung einbauen,

1.) - wenn zB ein Tabellenbereich so tief unten erreicht wird, der in einer Spalte, die immer einen Wert enthält, plötzlich keinen Inhalt mehr hat.

2.) - oder durch die einfache Exit-Bedingung in der Schleife If Z>=LZ then Exit For

3.) oder man arbeitet mit einer Schleifenfunktion, die die LZ-Variable aktiv ausliest:

3a.) While Wend

```
Z=0
LZ=99
WHILE Z <= LZ
        Z=Z+1 ' als Ersatz für das automatische Erhöhen der For-Next-Schleife
        ....
        LZ=LZ-1
WEND
```

3b.) Do Loop

```
Do
        ...
        If Z>=LZ Then Exit Do

Loop
```

3c.) Do Until Loop

```
Do Until Z>=LZ
...
Loop
```

ALLGEMEINES

For-Schleifen

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-2Einfache For-Schleifen

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-3Einfache For-Schleife zum Eintragen von Zahlen in eine Tabelle

In die erste Spalte des aktiven Arbeitsblattes werden die Ziffern 1 bis 100 eingetragen:

```
Sub EintragenZahlen()
   Dim intRow As Integer
   For intRow = 1 To 100
      Cells(intRow, 1) = intRow
   Next intRow
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-4

Einfache For-Schleife zum Eintragen von Wochentagen in eine Tabelle

Als einzige Veränderung zum obigen wird in diesem Beispiel über die Zählvariable der Wochentag, beginnend beim Sonntag, eingetragen.

```
Sub EintragenWochenTage()
   Dim intTag as Integer
   For intTag = 2 To 8
      Cells(intTag, 1) = Format(intTag, "dddd")
   Next intTag
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-5

Einfache For-Schleife mit variablem Ende

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-6

For-Schleife zum Eintragen einer zu ermittelnden Anzahl von Tagen

Start oder Ende einer Schleife liegen nicht immer fest und müssen möglicherweise bestimmt werden. Hier wird über die DateSerial-Funktion aus VBA der letzte Tag des aktuellen Monats bestimmt, um, beginnend bei Zelle E1, die Datumseintragungen des aktuellen Monats vorzunehmen.

```
Sub EintragenMonatTage()
   Dim intTag As Integer
   For intTag = 1 To Day(DateSerial(Year(Date), Month(Date) + 1, 0))
      Cells(intTag, 5) = DateSerial(Year(Date), Month(Date), intTag)
   Next intTag
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-7

Verschachtelte For-Schleife

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-8

Verschachtelte For-Schleife zum Eintragen des aktuellen Kalenderjahres

Die Variablen für Jahr, Monat und Tag werden dimensioniert. Das aktuelle Jahr wird an die Jahresvariable übergeben. Die äussere Schleife führt über die 12 Monate, wobei in Zeile 1 der jeweilige Monatsname eingetragen wird. Die innere Schleife führt über die Anzahl der Tage des jeweiligen Monats und trägt das jeweilige Datum in die Zellen ein. Zu beachten ist, dass Zeilen- und Schleifenzähler unterschiedliche Werte haben können. Im Beispiel werden die Tage erst ab Zeile 2 eingetragen, also wird der Zeilen- gegenüber dem Schleifenzähler um 1 hochgesetzt.

```
Sub EintragenJahr()
   Dim intYear As Integer, intMonat As Integer, intTag As Integer
   intYear = Year(Date)
   For intMonat = 1 To 12
      Cells(1, intMonat) = Format(DateSerial(1, intMonat, 1), "mmmm")
      For intTag = 1 To Day(DateSerial(intYear, intMonat + 1, 0))
         Cells(intTag + 1, intMonat) = DateSerial(Year(Date), intMonat, intTag)
      Next intTag
   Next intMonat
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-9# Do-Schleifen

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-10## Do-Schleifen

Do-Schleifen, ähnlich wie While-Schleifen, wiederholen sich beliebig oft. Die Schleife wird erst durch die Anweisung "Exit Do" beendet, die innherhalb der Do-Schleife z.B.(?) in einer If-Abfrage umgesetzt wird.

In dieser Do-Schleife wird eine Zufallszahl ermittelt. Wenn diese dem Index des aktuellen Monats entspricht, erfolgt eine Ausgabe in einer MsgBox.

```
Sub Zufall()
   Dim intCounter As Integer, intMonth As Integer
   Randomize
   Do
      intCounter = intCounter + 1
      intMonth = Int((12 * Rnd) + 1)
      If intMonth = Month(Date) Then
         MsgBox "Der aktuelle Monat " & _
         Format(DateSerial(1, intMonth, 1), "mmmm") & _
            " wurde im " & intCounter & _
            ". Versuch gefunden!"
         Exit Do
      End If
   Loop
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-11## Do-While-Schleifen

In dieser Do-While-Schleife, startend in Zelle A1, werden die Zellen abwärts geprüft, ob ein Suchbegriff darin vorkommt. Ist die Fundstelle erreicht, wird die Schleife verlassen und eine Meldung ausgegeben

```
Sub SuchenBegriff()
   Dim intRow As Integer
   intRow = 1
   Do While Left(Cells(intRow, 1), 7) <> "Zeile 7"
      intRow = intRow + 1
   Loop
   MsgBox "Suchbegriff wurde in Zelle " & _
      Cells(intRow, 1).Address & " gefunden!"
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-12## Do-Until-Schleifen

In dieser Do-Until-Schleife wird eine Zählvariable hochgezählt, bis der aktuelle Monat erreicht wird. Die Ausgabe erfolgt in einer Messagebox.

```
Sub PruefenWerte()
   Dim intCounter As Integer
   intCounter = 1
```

```
   Do Until Month(DateSerial(Year(Date), intCounter, 1)) = _
      Month(Date)
      intCounter = intCounter + 1
   Loop
   MsgBox "Der aktuelle Monat ist:" & vbLf & _
      Format(DateSerial(Year(Date), intCounter, 1), "mmmm")
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-13

For-Each-Schleifen

Es wird eine Objektvariable für ein Arbeitsblatt angelegt und alle Arbeitsblätter einer Arbeitsmappe werden durchgezählt. Das Ergebnis wird in einer MsgBox ausgegeben.

```
Sub ZaehlenBlaetter()
    Dim wks As Worksheet
    Dim intCounter As Integer
    For Each wks In Worksheets
        intCounter = intCounter + 1
    Next wks
    If intCounter = 1 Then
        MsgBox "Die aktive Arbeitsmappe hat 1 Arbeitsblatt!"
    Else
        MsgBox "Die aktive Arbeitsmappe hat " & _
            intCounter & " Arbeitsblätter!"
    End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen&action=edit&section=T-14

While-Schleifen

Beispiel ohne "echte" Funktion, dient lediglich zur Veranschaulichung der While-Schleife. Die Schleife zählt so lange hoch (nach jedem Schritt wird das neue Ergebnis ausgegeben) bis die While-Bed. erfüllt ist. Im Gegensatz zur Do-While-Schleife muss die While-Schleife mit "Wend" (steht für "While-Schleifen Ende") beendet werden! (Siehe auch Unterschied While-Wend / Do-While-Loop)

```
Sub WhileBsp()
    Dim i As Integer
    i = 0
    While i <> 3
        MsgBox "While-Schleife: " & i
        i = i + 1
    Wend
End Sub
```

Alle Zellen in einer Tabelle mit For Each

```
Dim ZELLE As Range

For Each ZELLE In ActiveSheet.UsedRange.Cells

   If IsDate(ZELLE) = True And ZELLE.Column <> 1 Then ZELLE = ZELLE + 1462 ' Repariert in allen Zellen außer in Spalte A das Datum nach Umstellung 1.1.1904=>31.12.1899

Next ZELLE
```

Alle Zellen in einer Spalte mit For Each

Im Gegensatz zur Usedrange-Funktion, die nicht auf das Ende der aktuellen Spalte eingeht, sondern das Ende ALLER Spalten nimmt, geht End(xlUp) zur letzten Zelle in der aktuellen Spalte mit Zell-INHALT.

```
Dim Zelle As Range
Dim Bereich As Range

Range("A2:A" & Range("A65536").End(xlUp).Row).Select  ' alle verwendeten Zellen in einer Spalte (hier A und ab Zeile 2), die einen Inhalt haben

For Each Zelle In Selection
   Zelle = Zelle * 10
Next Zelle
```

Alle Zellen in einer Zeile mit For Each

```
Dim Zelle As Range
Dim Bereich As Range

Range(Cells(2, 1), Cells(2, Range("IV2").End(xlToLeft).Column)).Select  ' alle verwendeten Zellen in Zeile 2 mit Zellinhalt (hier ab Spalte A)

For Each Zelle In Selection
   Zelle = Zelle * 10
Next Zelle
```

Alle Zellen in einem Bereich mit For Each

LÖSUNG 1:

```
Dim ZELLE As Range
```

```
Range("A2:C10").Select

For Each ZELLE In Selection
   ZELLE = ZELLE * 10
Next ZELLE
```

LÖSUNG 2:

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each ZELLE In BEREICH
   ZELLE = ZELLE * 10
Next ZELLE
```

Alle Spalten in einem Bereich mit For Each

```
Dim SPALTE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each SPALTE In BEREICH.Columns
   MsgBox SPALTE.Column
Next SPALTE
```

Alle Zeilen in einem Bereich mit For Each

```
Dim ZEILE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each ZEILE In BEREICH.Rows
   MsgBox ZEILE.Row
Next ZEILE
```

Alle Bereiche in einer Markierung mit For Each

Da man mit gehaltener STRG-Taste mehrere unabhängige Zellbereiche gleichzeitig markieren kann, braucht man folgenden Code, um durch alle markierten Bereiche zu wandern

```
Dim MARKIERUNG As Range
Dim BEREICH As Range

Set MARKIERUNG = Application.Selection

For Each BEREICH In MARKIERUNG.Areas
   MsgBox BEREICH.Address
Next BEREICH
```

Alle Zellen mit Formeln in einem Bereich mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("B1:C99")
'   Set BEREICH = Application.Selection ' Alternative, die die aktuelle Auswahl nimmt

'  Arbeite nur mit Zellen, die eine Formel enthalten
   For Each ZELLE In BEREICH.SpecialCells(xlFormulas, 23)
      MsgBox ZELLE.Address
   Next ZELLE
```

Alle Zellen mit Formel-Fehlerwerten in einem Bereich mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("B1:C99").SpecialCells(xlCellTypeFormulas, xlErrors)

For Each ZELLE In BEREICH.Cells
   MsgBox ZELLE.Address
Next ZELLE
```

Alle Zellen mit bestimmten Formel-Ergebnissen (Wert, Text,) mit For Each

Formeln, die Text ergeben:

```
Dim ZELLE As Range
Dim BEREICH As Range
```

```
    Set BEREICH = Range("A1:C99")

'   Arbeite nur mit Zellen, die eine Formel enthalten und die als Ergebnis einen Text haben
    For Each ZELLE In BEREICH.SpecialCells(xlCellTypeFormulas, xlTextValues)
       MsgBox ZELLE.Address
    Next ZELLE
```

Formeln, die Zahlen ergeben:

```
    Dim ZELLE As Range
    Dim BEREICH As Range

    Set BEREICH = Range("A1:C99")

'   Arbeite nur mit Zellen, die eine Formel enthalten und die als Ergebnis einen Text haben
    For Each ZELLE In BEREICH.SpecialCells(xlCellTypeFormulas, xlNumbers)
       MsgBox ZELLE.Address
    Next ZELLE
```

Alle Zellen, die leer sind mit For Each

```
    Dim ZELLE As Range
    Dim BEREICH As Range

    Set BEREICH = Range("A1:C99")

'   Arbeite nur mit Zellen, die eine Formel enthalten und die leer sind
    For Each ZELLE In BEREICH.SpecialCells(xlCellTypeBlanks)
       MsgBox ZELLE.Address
    Next ZELLE
```

Alle Zellen mit einem Kommentar mit For Each

```
    Dim ZELLE As Range
    Dim BEREICH As Range

    Set BEREICH = Range("D1:D99")

'   Arbeite nur mit Zellen, die einen Kommentar enthalten
    For Each ZELLE In BEREICH.SpecialCells(xlCellTypeComments, 23)
       MsgBox ZELLE.Address
    Next ZELLE
```

Alle sichtbaren Zellen mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("D1:D99")

'  Arbeite nur mit Zellen, die sichtbar und nicht ausgeblendet sind
For Each ZELLE In BEREICH.SpecialCells(xlCellTypeVisible, 23)
   MsgBox ZELLE.Address
Next ZELLE
```

CASE SELECT benutzen

```
A = Inputbox ("Zahl eingeben zwischen 1 und 9")
Select Case A
        Case 0
                msgbox "Sie haben Null eingegeben"
        Case 1 to 5
                msgbox "Sie haben eine Zahl zwischen 1 und 5 eingegeben"
        Case 6,8
                msgbox "Sie haben entweder 6 oder 8 eingegeben"
        Case 7,9
                msgbox " Sie haben entweder 7 oder 9 eingegeben"
        Case else
                msgbox "Sie haben was größer als 9 eingegeben")
End select

Wichtig: Case arbeitet auch mit Ziffern

A= Inputbox ("Zeichen a oder b eingeben")
Select Case A
        Case "a"
                msgbox "Sie haben a eingegeben"
        Case "b"
                msgbox "Sie haben a eingegeben"
End Select
```

Routinen verlassen / beenden

Mit EXIT SUB innerhalb einer Prozedur verlässt man Prozedur hier - wurde die betreffende Prozedur von einer anderen Prozedur aufgerufen, geht der Programmcode normal in der

aufrufenden Prozedur weiter

Mit END beendet man die Ausführung von Programmcode absolut.

SCHLEIFEN schneller durchführen

Performance (1) - Vorwort + Schleifen die keiner braucht

Autor: Peter Haserodt - Erstellt: -- - Letzte Revision: --

Gruppenthema: 4 Folgen **1** 2 3 4 Sie sind in Folge:1

Performance - Schnelligkeit - Kein Problem für Excel, aber ...
oder:
Schleifen, die kein Mensch braucht

Der erste Teil hier ist nur mal als Einstieg gedacht und soll auch ein bisserl zum Schmunzeln sein (wenn es nicht so traurig wäre)

Die Rechner werden immer schneller aber auch die zu bearbeitenden Datenmengen werden immer größer.
Nicht selten hört man Leute klagen, wie langsam Excel oder VBA doch ist.

Tatsächlich sitzt in 99 % aller Fälle die Bremse vor dem Computer.

Undurchdachte Codes, Nichtausnutzung von eingebauten Excelfunktionalitäten, Makrorekorder produzierter Müll und vieles mehr sind mögliche Ursachen für Geschwindigkeitsprobleme.
Mit allem möglichen Unfug wird versucht, die Geschwindigkeit zu erhöhen, anstatt an der Wurzel zu arbeiten.

Erschwert wird für den VBA Einsteiger das ganze aber noch durch völlig perverse Beispiele in manchen Excelbüchern oder in irgendwelchen Artikeln.

Wollen wir uns ein paar solcher negativ Beispiele anschauen.
(Die HorrorBeispiele stammen alle aus einem VBA-Buch eines bekannten VBA Autoren, die er dort durchaus ernst gemeint veröffentlicht hat. Ich nutze dieses Machwerk gerne, wenn ich nach schlechtem Code suche für Beispiele, wie man es nicht machen soll)
(Ich habe die Beispiele zum Teil gekürzt - ohne aber den Sinn zu entstellen - weil es wirklich weh tut, soetwas abzuschreiben)

Fangen wir mit einem wunderschönen Beispiel an: Hier geht es darum, ausgeblendete Zeilen im aktiven Tabellenblatt wieder einzublenden

```
Sub ZeilenEinblendenHorror()
```

```
        Dim zeile As Object
        For Each zeile In ActiveSheet.UsedRange.Rows
                zeile.Hidden = False
        Next zeile
End Sub

Sub ZeilenEinblendenAnders()
        ActiveSheet.UsedRange.Rows.Hidden = False
End Sub
```

Um die beiden Codes in ihrer Geschwindigkeit miteinander zu vergleichen, füllen Sie einfach mal eine Spalte mit möglichst vielen Werten, damit möglichst viele Zeilen im UsedRange sind.
Während Sie die HorrorProzedur dann ausführen, gehen Sie ruhig einen Kaffee trinken.

Hier ist die Performancebremse, dass jede einzelne Zeile angesprochen und ihr hidden Status auf false gesetzt wird.
Dabei stellt mir der UsedRange das Rows Objekt zur Verfügung, mit welchem ich direkt alle Zeilen einblenden kann.
Wenn es nur irgendwie zu vermeiden geht, niemals einzele Objekte ansprechen sondern Objektgruppen.

Das nächste Beispiel zeigt die völllig sinnlose Verwendung einer Schleife, anstatt Excel Bordmittel zu benutzen.
(Dazu kommt noch, dass der Code von der Deklaration her völlig unsinnig ist)

Thema ist es, in einem selektierten Bereich den Durchschnittswert zu berechnen.

```
Sub DurchSchnittHorror()
        Dim Zelle As Range
        Dim Avg As Long ' Falsche Deklaration sollte double sein
        Dim i As Integer ' falsche Deklaration sollte long sein
        Avg = 0 ' Nicht notwendig
        i = Selection.Cells.Count ' und ab hier die völlig sinnlose Schleife
        For Each Zelle In Selection
                Avg = Avg + Zelle.Value
        Next Zelle
        Avg = Avg / i ' Aufgrund falscher Deklaration falsche Ergebnisse (z.B. 3/4 wird zu 1)
        MsgBox Avg
End Sub

Sub DurchschnittAnders()
        MsgBox WorksheetFunction.Average(Selection)
End Sub
```

Das Horrorbeispiel entspricht etwa folgendem Szenario:
Ein Anwender will in Excel von einem Bereich den Durchschnitt ermitteln und ich sage ihm:
Nimm einen Taschenrechner und zähle alle Zellen zusammen. Ermittle die Anzahl der Zellen und teile die Summe dann durch diese ermittelte Zahl.

Anstatt ganz einfach die Funktion Mittelwert zu benutzen.
(Ganz davon abgesehen, dass in dem Beispiel auch fehlerhaft berechnet wird, sobald eine Kommazahl als Durchschnitt herauskommt!)

Zuerst immer prüfen ob nicht eine Excelfunktion die Arbeit für mich erledigen kann.
Dieses ist immer schneller.

Das nächste Beispiel ist mein Favorit.
Hier zeigt sich das völlige Unverständnis für vom Makrorekorder produzierten Code.
Ziel dieses Meisterwerkes ist es, in einem selektierten Bereich Umlaute zu ersetzen (ich habe dies auf ä reduziert)

```
Sub UmlautHorror()
          Dim Zelle As Range
          For Each Zelle In Selection
                    With Selection
                              .Replace What:="ä", Replacement:="ae", LookAt:=xlPart
                    End With
          Next Zelle
End Sub
```

Man muss sich dies wirklich in Ruhe zu Gemüte führen.

Hier wird nun eine ganz und gar völlig verblödete Schleife produziert.
Tatsächlich ist die Arbeit schon nach dem ersten Schleifendurchlauf getan 
Mit der Anweisung:

With Selection
.Replace What:="ä", Replacement:="ae", LookAt:=xlPart
End With

erledige ich die Ersetzung schon für den ganzen Bereich.
Aber weil es so schön ist, machen wir das halt so oft, soviele Zellen in dem Bereich sind.
Die Schleife kann ich einfach weglassen!

Man sollte seinen Code immer daraufhin überprüfen ob er überhaupt Sinn macht!

Dieser erste Teil soll überhaupt erstmal das Gefühl vermitteln, dass man an Geschwindigkeit arbeiten kann.

Weiterhin soll er aufzeigen, dass ich mit Beispielen, egal woher ich sie beziehe, sehr vorsichtig sein muss.

Traurig in diesem Fall ist aber, dass ein Einsteiger in Excel VBA solchen geschriebenen Mist für bare Münze nimmt und dafür auch noch viel Geld bezahlt.
Wie soll ein Anfänger je das Gefühl für das Excel Objekt bekommen, wenn er schwarz auf weiß lesen muss, wie man es eigentlich nicht macht.

Statusleiste – Schleifen-Durchlauf-Prozent von 1-100

Wenn eine Schleife lange dauert, weil sie z.B. durch viele Zeilen durchwandert, kann man die Prozent der erledigten Schleifendurchläufe in der Statuszeile anzeigen. (Dabei ist Application.Screenupdating = True NICHT notwendig :o)

```
' ----------------------------------------
' ---  Such- Schleife durch alle Zeilen ---
' ----------------------------------------

Application.ScreenUpdating = False

LETZTEZEILE = LETZTEZELLE(ActiveSheet.Name).Row

For Z = 11 To LETZTEZEILE
' …. der Code

   ' Schleifenprozent
   SCHLEIFENPROZENT = (Z - 10) / (LETZTEZEILE - 10) * 100  ' Da unsere Schleife erst in Zeile 11 beginnt, müssen wir die ersten 10 Zeilen immer abziehen)
   If Round(SCHLEIFENPROZENT, 0) > PROZENT Then
      PROZENT = Round(SCHLEIFENPROZENT, 0)
      Application.StatusBar = PROZENT
   End If

Next Z

Application.ScreenUpdating = True
Application.StatusBar = ""
```

--- http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Druckversion&action=edit&section=10 Wenn-Abfragen ---

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-1 Einfache Verzweigung (If … Then)

Wenn es sich beim aktuellen Tag um einen Sonntag handelt, wird eine entsprechende Meldung ausgegeben, wenn nicht, erfolgt keine Aktion.

```
Sub WennSonntagMsg()
   If Weekday(Date) = 1 Then MsgBox "Heute ist Sonntag"
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-2 Wenn/Dann/Sonst-Verzweigung (If … Then … Else)

In der Regel werden Wenn-/Dann-Abfragen erstellt, um Verzweigungen zu ermöglichen. In Beispiel 2.2 wird bei WAHR die Sonntagsmeldung, bei FALSCH der aktuelle Wochentag ausgegeben.

```
Sub WennSonntagOderMsg()
   If Weekday(Date) = 1 Then
      MsgBox "Heute ist Sonntag"
   Else
      MsgBox "Heute ist " & Format(Weekday(Date), "dddd")
   End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-3 Wenn-Dann-SonstWenn-Verzweigung (If..Then..ElseIf.. ..Else..)

Über ElseIf können weitere Bedingungen mit entsprechenden Verzweigungen angefügt werden.

```
Sub WennSonntagSonstMsg()
   If Weekday(Date) = 1 Then
      MsgBox "Heute ist Sonntag"
   ElseIf Weekday(Date) = 7 Then
      MsgBox "Heute ist Samstag"
   Else
      MsgBox "Heute ist " & Format(Weekday(Date), "dddd")
   End If
End Sub
```

Zweckmäßig ist diese Struktur auch bei der Fehlerprüfung, wenn völlig unterschiedliche Bedingungen geprüft werden sollen:

```
Public Function DiscoEinlass(GeburtsTag As Date) As Boolean
    DiscoEinlass = False

    If DateSerial(Year(GeburtsTag) + 18, Month(GeburtsTag), Day(GeburtsTag)) > Date Then
        MsgBox "Sie sind leider noch nicht volljährig"
    ElseIf Year(Date) - Year(GeburtsTag) > 65 Then
        MsgBox "Rentner dürfen hier nicht rein!"
    ElseIf Weekday(GeburtsTag, vbSunday) <> 1 Then
        MsgBox "Sie sind kein Sonntagskind und können keine Elfen sehen"
    Else
        DiscoEinlass = True
    End If
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-4

Select-Case-Verzweigung

Bei mehr als zwei Bedingungen empfiehlt sich meist - wenn möglich - die Select-Case-Prüfung einzusetzen. Der vorliegende Fall wird eingelesen und danach schrittweise auf seinen Wahrheitsgehalt geprüft.

```
Sub PruefeFallMsg()
   Select Case Weekday(Date)
      Case 1, 7: MsgBox "Heute ist kein Arbeitstag"
      Case 2: MsgBox "Heute ist Montag"
      Case 3: MsgBox "Heute ist Dienstag"
      Case 4: MsgBox "Heute ist Mittwoch"
      Case 5: MsgBox "Heute ist Donnerstag"
      Case 6: MsgBox "Heute ist Freitag"
   End Select
End Sub
```

Sehr zweckmäßig ist die Select Anweisung auch, wenn man Optionsfelder in einem Formular (hier mit Objektbezeichner Me angesprochen) auswerten möchte. Dazu dreht man die Vergleichsbedingung um, so dass der konstante Teil des Vergleichs (hier True) hinter der Select Case Anweisung steht:

```
Sub ZeigeOption()
    Select Case True
        Case Me.Option1.Value: MsgBox "Option 1 gewählt"
        Case Me.Option2.Value: MsgBox "Option 2 gewählt"
        Case Me.Option3.Value: MsgBox "Option 3 gewählt"
        Case Me.Option4.Value: MsgBox "Option 4 gewählt"
        Case Else:             MsgBox "Nichts gewählt"
    End Select
End Sub
```

Grundsätzlich sollte der häufigste Fall für eine Verzweigung mit der ersten CASE-Anweisung abgefangen werden, um die Laufzeit bei häufigen Aufrufen zu reduzieren.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-5

Inline Verzweigungen mit IIf()

Für besonders einfache Fälle gibt es auch die Möglichkeit, Verzweigungen in einer Zeile zu erstellen. Die IIf() Funktion ist dabei das Pendant zur IF..Then..Else..End If Struktur. Die folgende Funktion baut einen Text mit einer IIf()-Funktion zusammen:

```
Public Function GeradeOderUngerade(Zahl As Long) As String
    GeradeOderUngerade = "Die Zahl ist eine " & IIf(Zahl Mod 2 = 0, "gerade", "ungerade") & " Zahl"
End Function
```

Diese Form der Verzweigung hat zwei besondere Merkmale:

- Es muss für beide Antwortmöglichkeiten ein Ergebnis angegeben werden
- Die beiden Teile werden unabhängig vom Ergebnis des Vergleichs immer beide ausgeführt. Dies ist zu beachten, falls Funktionen aufgerufen werden.

Das folgende Beispiel illustriert das Problem:

```
Public Function Division(Dividend As Double, Divisor As Double) As Double
    Division = IIf(Divisor = 0, 0, Dividend / Divisor)
End Function
```

Eigentlich sollte man im vorhergehenden Beispiel davon ausgehen, dass im Falle einer Division durch 0 (z.B. bei Aufruf von =Division(2,0) in einem Tabellenblatt) in dieser speziellen Funktion eine 0 zurückgegeben wird, statt dass ein Fehler die Ausführung unterbricht. Da aber stets alle Teile der IIf()-Verzweigung ausgeführt werden, probiert VBA auch die Division durch 0 und die ganze Funktion bricht mit einem Fehler ab.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-6

Inline Verzweigungen mit Choose()

Das Inline Pendant zur Select Case Struktur ist die Choose() Funktion. Das folgende Beispiel zeigt, wie man in einer Zeile dem Datum einen Wochentag zuordnet:

```
Public Function Wochentag(Datum As Date) As String
    Wochentag = Choose(Weekday(Datum, vbMonday), "Mo", "Di", "Mi", "Do", "Fr", "Sa", "So")
End Function
```

Hier gilt wie bei IIf(), dass alle Ausdrücke von VBA ausgeführt werden, egal wie das Ergebnis des Vergleichs ist.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Wenn-Abfragen&action=edit&section=T-7

Wann sollte welche Verzweigung gewählt werden?

Die vermutlich größte Schwierigkeit besteht, falls die Wahl zwischen IF..Then..ElseIf und Select Case besteht:

- Select Case setzt voraus, dass ein Ausdruck eines Vergleiches mit allen anderen verglichen wird, und der sollte in der Zeile mit Select Case auftauchen. Damit eignet es sich beispielsweise zur Abfrage von Optionsfeldern (siehe Beispiel oben), zur Abfrage von Bereichen oder wenn eine Funktion wie MsgBox mehr als zwei verschiedene Rückgabewerte hat.
- If..Then..ElseIf erlaubt es, völlig unterschiedliche Vergleiche auszuführen. If..Then..ElseIf eignet sich beispielsweise für Plausibilitätsabfragen am Anfang einer Funktion. Hier werden die Eingabedaten auf oft völlig unterschiedliche Kriterien geprüft, aber wenn nur eines erfüllt ist, gibt es eine spezielle Fehlermeldung.

SOUND UND KLÄNGE

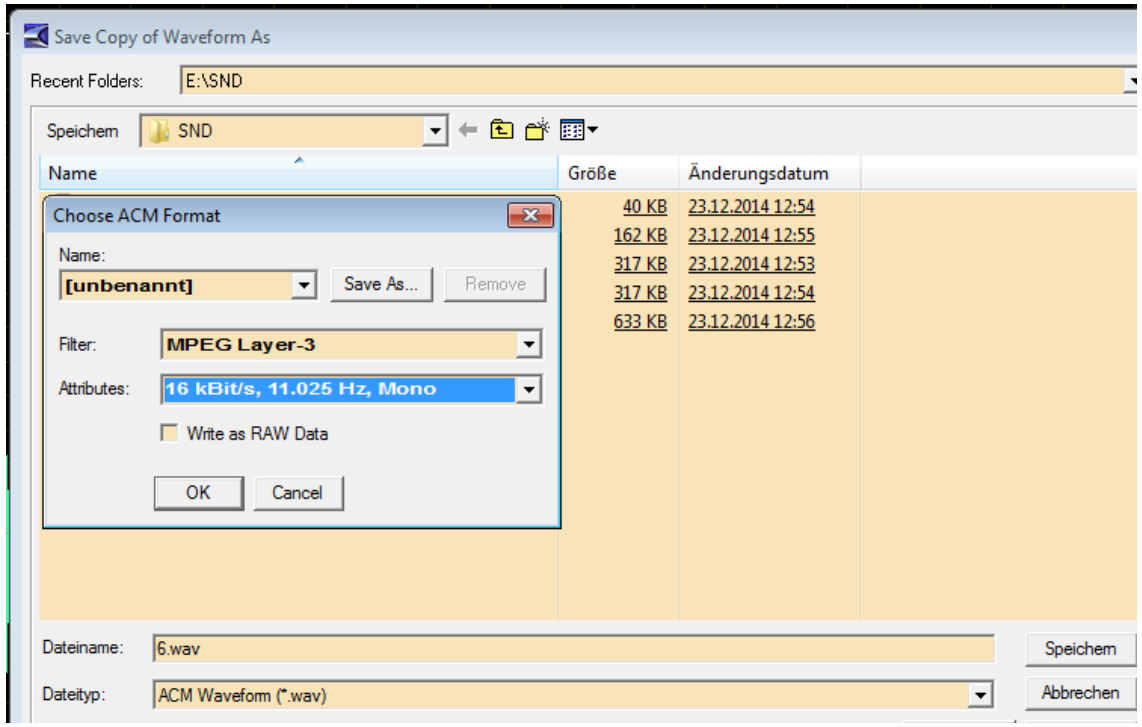
Abspielen von eingefügten Wave-Sounddateien

WEG 1 indem man die WAV-Datei direkt in Exceldatei integriert

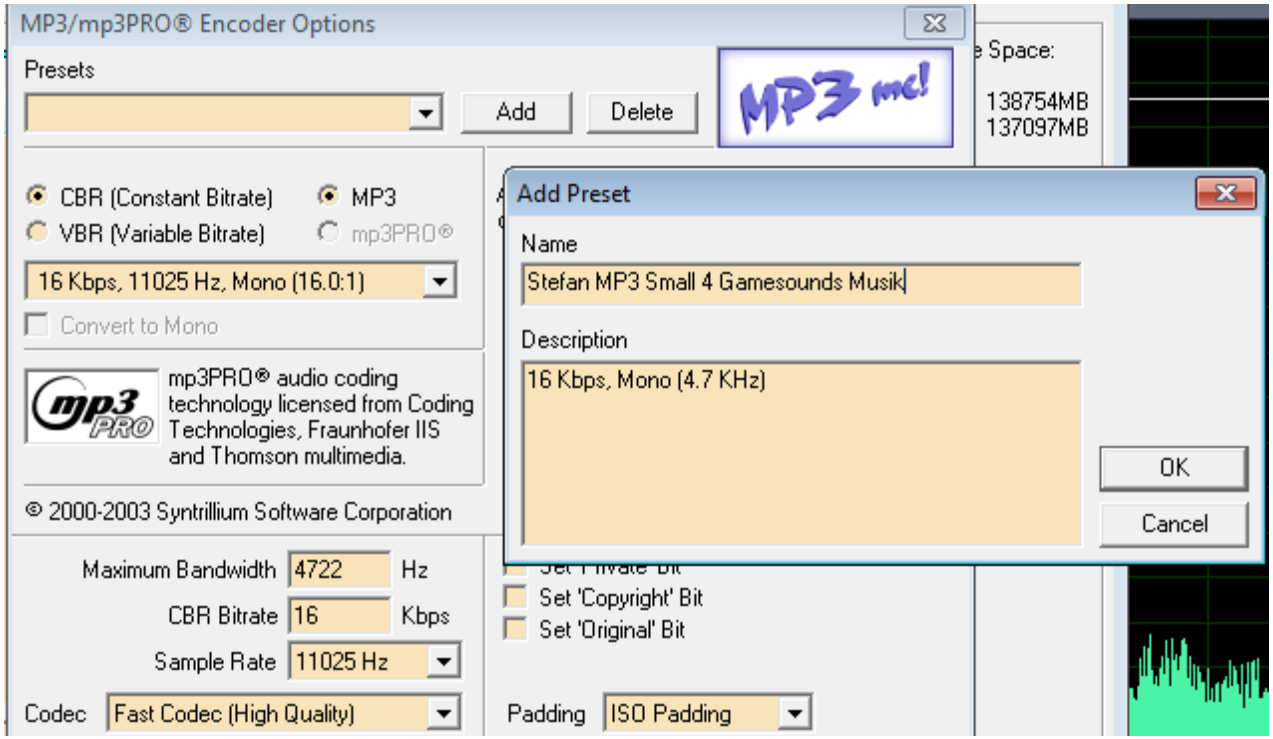
Wichtig: die Audiodatei erst noch klein machen

Was nicht geht: 2 WAV-Dateien gleichzeitig abzuspielen

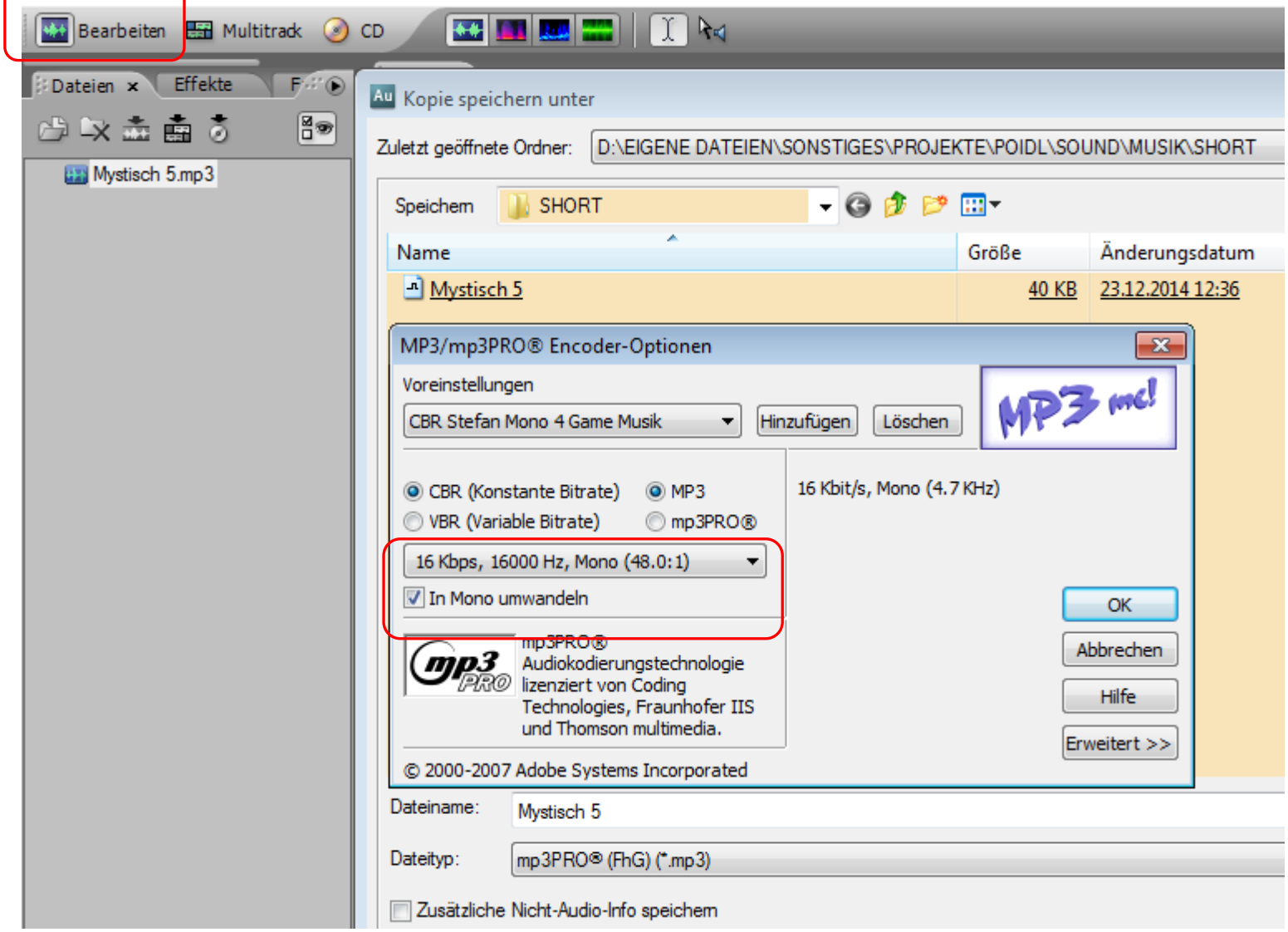
Achtung: mancher nachfolgender Code funktioniert nur mit WAV-Dateien => daher WAV-Dateien erzeugen – geht z.B. sehr gut mit Cool EDIT (ist ja der Vorgänger vom hier ebenfalls beschriebenen ADOBE AUDIOTON 3 und kann es genauso gut) – ich wähle bevorzugt dieses Format ACM mit MP3-Filter , weil es gleich kleine Dateien erzeugt wie MP3:



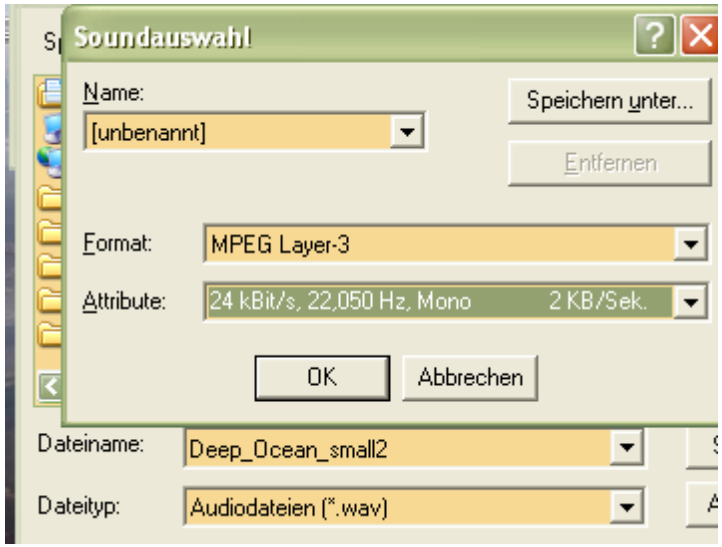
MP3 Beste Qualität gab es mit Cool Edit: MONO-MP3



Oder mit Adobe Auditon - Ich nahm MONO MP3:

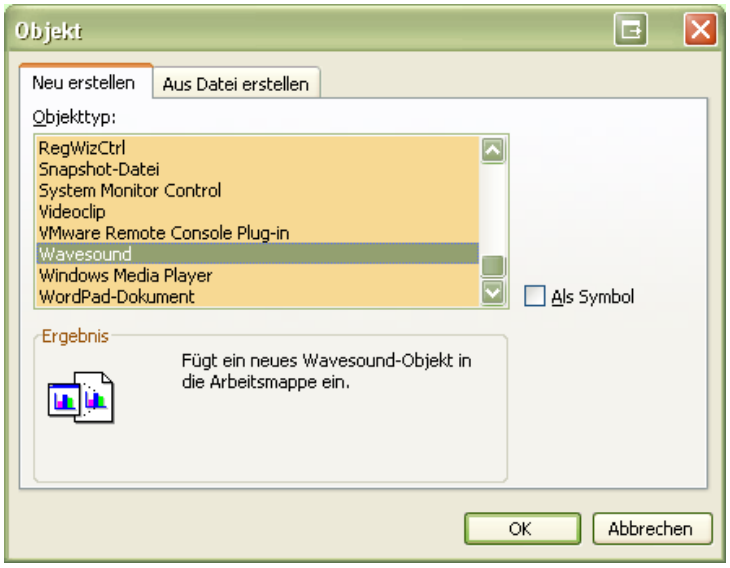


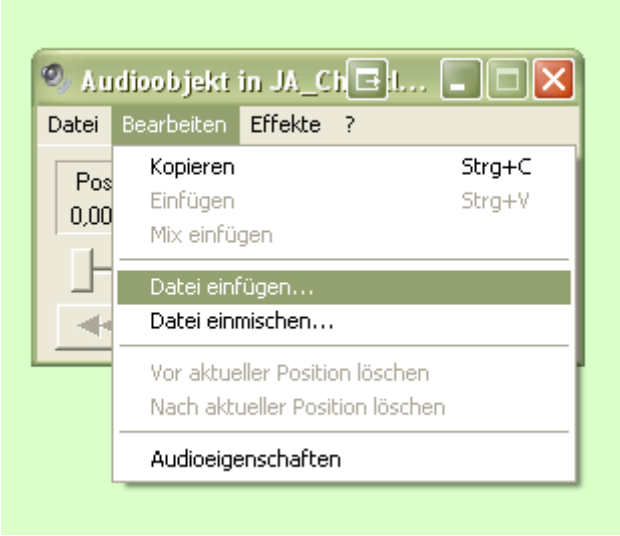
Oder mit Windowseigenen Soundrecorder (Menü ZUBEHÖR / UNTERHALTUNGSMEDIEN / AUDIORECORDER) und dort bei den Eigenschaften (Menü Datei) den Dateityp checken - richtig wäre:



Ein 15 Sekundenclip darf höchstens 50 KB haben !

Dann im Menü Einfügen - Objekt wählen

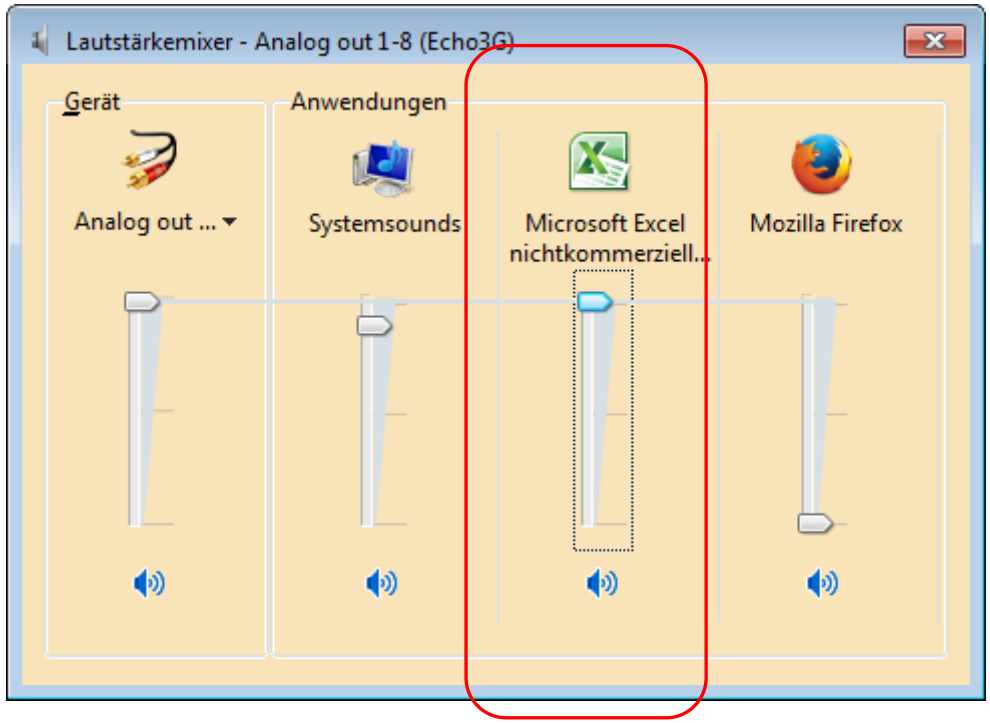




Abspielen geht dann mit

```
ActiveSheet.Shapes("Object 916").Select ' wenn Wave auf aktueller Tabelle ist
Selection.Verb Verb:=xlPrimary
```

Wenn du mal nichts hörst … - schau nach, ob die Lautstärke des Excel-VBA-Schiebers nicht auf NULL ist:



Abspielen von externen Wave-Sounddateien

Wie man möglichst kleine WAV-Dateien erzeugt ist im vorigen Unterkapitel "Abspielen von eingefügten Wave-Sounddateien" beschrieben

MEIN CODE

```
' ---------------------------------
' Variablen und Funktionendefinition
' ---------------------------------

Private Declare Function waveOutSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags
As Long) As Long
```

```
Private Type intVolumeType
    Left As Integer
    Right As Integer
End Type

Private Type lngVolumeType
    Value As Long
End Type

Const SND_ASYNC = &H1
Const SND_NODEFAULT = &H2

' ----------------------------------
' DEMO-PROZEDUR
' ----------------------------------

Sub Test()

' Demomodul zum Soundabspielen

' im Hauptordner der Excelarbeitsmappe gibt es den Unterordner SOUNDS mit weiteren Unterordnern, in denen die WAV-Dateien
sind
' wir müssen nur den Unterordner im Unterordner SOUNDS übergeben, den Rest ergänzen die Abspielroutinen Sound1 und
Soundloop#

' Lautstärke auf 100% setzen
    SetVolume_100

' Einen Sound nur einmal abspielen
    Sound1 ("ALLG\Short1.wav")
    MsgBox "Sound wird nur einmal abgespielt"

' Einen Sound unbegrenzt abspielen
    Soundloop ("ALLG\DeepOcean.wav")
    MsgBox "Sound wird endlos abgespielt"

' Lautstärke von 100 auf 0 faden
    MsgBox "Lautstärke wird nun ausgefadet"
    SetVolume_100_to_0

' Lautstärke von 100 auf 0 faden
```

```
    MsgBox "Lautstärke wird nun wieder eingefadet"
    SetVolume_0_to_100

' Einen Endlossound abdrehen
    MsgBox "Endlosschleife wird nun abgedreht"
    Soundstill

End Sub


' ---------------------------------
' HILFS-PROZEDUREN
' ---------------------------------

Sub Sound1(SoundName As String)

' --- Der übergebene Sound wird nur einmal abgespielt ---

    ' Der übergebene Unterordner wird um den restlichen Arbeitsmappenpfad ergänzt

    SoundName = ActiveWorkbook.Path & "\SOUNDS\" & SoundName

    Call sndPlaySound(SoundName, SND_ASYNC)    ' dieser Code spielt Sound nur einmal ab

End Sub

Sub Soundloop(SoundName As String)

' --- Der übergebene Sound wird unbegrenzt abgespielt ---

    ' Der übergebene Unterordner wird um den restlichen Arbeitsmappenpfad ergänzt

    SoundName = ActiveWorkbook.Path & "\SOUNDS\" & SoundName

    Call sndPlaySound(SoundName, SND_ASYNC Or &H8) ' ' dieser Code spielt es als Loop

End Sub

Sub Soundstill()

' Da ich noch keinen Code gefunden habe um endlos abspielende Sounds zu stoppen, spiele ich einfach einen leeren Sound
kurz ab
```

```
    SoundName = ActiveWorkbook.Path & "\SOUNDS\ALLG\Still.wav"

    Call sndPlaySound(SoundName, SND_ASYNC) ' dieser Code spielt das leere Wavefile nur einmal ab, um das Sound abspielen zu stoppen


End Sub

Sub SetVolume_100()

' setzt die Lautstärke auf 100

    Dim intVol As intVolumeType
    Dim lngVol As lngVolumeType

    intVol.Left = 32000
    intVol.Right = 32000
    LSet lngVol = intVol
    Call waveOutSetVolume(0, lngVol.Value)

End Sub

Sub SetVolume_0()

' setzt die Lautstärke auf Null

    Dim intVol As intVolumeType
    Dim lngVol As lngVolumeType

    intVol.Left = 0
    intVol.Right = 0
    LSet lngVol = intVol
    Call waveOutSetVolume(0, lngVol.Value)

End Sub

Sub SetVolume_0_to_100()

' erhöht die Lautstärke von Null auf 100%

    Dim intVol As intVolumeType
```

```
    Dim lngVol As lngVolumeType

    Schrittweite = 7

    For T = 1 To 32000 Step Schrittweite
        intVol.Left = T
        intVol.Right = T
        LSet lngVol = intVol
        Call waveOutSetVolume(0, lngVol.Value)
    Next T

End Sub
Sub SetVolume_100_to_0()

' erniedrigt die Lautstärke von 100% auf Null

    Dim intVol As intVolumeType
    Dim lngVol As lngVolumeType

    Schrittweite = -7

    For T = 32000 To 1 Step Schrittweite
        intVol.Left = T
        intVol.Right = T
        LSet lngVol = intVol
        Call waveOutSetVolume(0, lngVol.Value)
    Next T

End Sub
```

DETAILS ZUM MEINEM CODE

An die winmm.dll kann man ja verschiedene Werte übergeben:

```
Public Const SND_SYNC = &H0             '  play synchronously (default) - Exel wartet bis fertig
Public Const SND_ASYNC = &H1            '  play asynchronously - Excel wartet nicht
Public Const SND_NODEFAULT = &H2        '  silence not default, if sound not found (kein Beep)
Public Const SND_MEMORY = &H4           '  lpszSoundName points to a memory file (Datei ist im Arbeitsspeicher)
```

```
Public Const SND_ALIAS = &H10000         '  name is a WIN.INI [sounds] entry (ein Windows - Standardsound)
Public Const SND_FILENAME = &H20000      '  name is a file name (???)
Public Const SND_RESOURCE = &H40004      '  name is a resource name or atom (Sound in VB - Resourcendatei)
Public Const SND_ALIAS_ID = &H110000     '  name is a WIN.INI [sounds] entry identifier (in Verbindung mit SND_ALIAS zu benutzen)
Public Const SND_ALIAS_START = 0         '  must be > 4096 to keep strings in same section of resource file (in Verbindung mit
SND_ALIAS zu benutzen
Public Const SND_LOOP = &H8              '  loop the sound until next sndPlaySound (endlos abspielen)
Public Const SND_NOSTOP = &H10           '  don't stop any currently playing sound (!!!)
Public Const SND_VALID = &H1F            '  valid flags          / ;Internal / (???)
Public Const SND_NOWAIT = &H2000         '  don't wait if the driver is busy (spielt ohne Verzögerung beim wechsel der Datei)
Public Const SND_VALIDFLAGS = &H17201F   '  Set of valid flag bits. Anything outside this range will raise an error (???)
Public Const SND_RESERVED = &HFF000000   '  In particular these flags are reserved (???)
Public Const SND_TYPE_MASK = &H170007 (???)
```

Ein Beispiel für endlos:

Mach dir ein Userform mit vier Commandbuttons und folgendem Code:

```
Option Explicit

Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" ( _
   ByVal lpszSoundName As String, _
   ByVal uFlags As Long) As Long
Private Declare Function waveOutGetNumDevs Lib "winmm.dll" () As Long

Private Const SND_ASYNC = &H1
Private Const SND_NODEFAULT = &H2
Private Const SND_LOOP = &H8

Private bolHaveSoundcard As Boolean, bolSound_plays As Boolean

Private Sub CommandButton1_Click()
   Call prcPlay("F:\Eigene MusikCD's\Nico_a\Nico_a14.wav")
End Sub

Private Sub CommandButton2_Click()
   Call prcPlay("F:\Eigene MusikCD's\Nico_a\Nico_a15.wav")
End Sub
```

```
Private Sub CommandButton3_Click()
   If bolSound_plays Then sndPlaySound "NULL", SND_ASYNC
   bolSound_plays = False
End Sub

Private Sub CommandButton4_Click()
   Unload Me
End Sub

Private Sub UserForm_Activate()
   bolHaveSoundcard = waveOutGetNumDevs > 0
End Sub

Private Sub UserForm_Terminate()
   If bolSound_plays Then sndPlaySound "NULL", SND_ASYNC
End Sub

Private Sub prcPlay(strFilename As String)
   If bolHaveSoundcard Then
      If bolSound_plays Then sndPlaySound "NULL", SND_NODEFAULT
      If Dir(strFilename) <> "" Then
         sndPlaySound strFilename, SND_ASYNC Or SND_LOOP
         bolSound_plays = True
      Else
         bolSound_plays = False
      End If
   End If
End Sub
```

Button1 = Sound 1
Button2 = Sound 2
Button3 = Stop
Button4 = beenden

WEG 0

Declare Function sndPlaySound32 Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

```
Sub Klang()
Call sndPlaySound32("D:\Programme\ICQ\Connect.wav", 0)
End Sub
```

Die 1 steht für asynchrone Wiedergabe.
0 wäre synchron, also eine Zwangspause für Excel.

WEG 0-B

```
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Sub WavDateiAbspielen()
    Call sndPlaySound("C:/WINDOWS/Media/tada.wav", 1)
   'Call sndPlaySound32("C:/WINDOWS/Media/Windows XP-Anmeldesound.wav", 1)
End Sub
```

Die 1 steht für asynchrone Wiedergabe.
0 wäre synchron, also eine Zwangspause für Excel.

WEG 2 - Sounddatei liegt extern in einem Ordner

```
'   API Funktion für Sound abspielen
Declare Function sndPlaySound32 Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName  As String, ByVal uFlags As Long) As Long

Sub WAValsStart()
    Call sndPlaySound32("c:\Windows\media\Windows XP-Herunterfahren.wav", 1)
End Sub
```

WEG 3

```
Sub Auto_Open()
Worksheets("Sheet1").OnCalculate = "PlayIt"
End Sub

Sub PlayIt()
If Range("A1").Value > 5 Then
ExecuteExcel4Macro ("SOUND.PLAY(,
```

```
""C:\Windows\Media\Tada.wav"")")
End If
End Sub
```

WEG 4

```
Call PlaySound("d:\\Soundname.wav", 0, SND_FILENAME Or SND_ASYNC)
```

Und im Modul:

```
Declare Function PlaySound Lib "winmm.dll" _
Alias "PlaySoundA" _
(ByVal lpszName As String, _
ByVal hModule As Long, _
ByVal dwFlags As Long) As Long

Const SND_ASYNC As Long = &H1
Const SND_FILENAME As Long = &H20000
Const SND_PURGE As Long = &H40
Const SND_LOOP As Long = &H8
```

"SND_FILENAME" muss immer angegeben werden, da Sie eine Datei abspielen wollen. "SND_ASYNC" bedeutet, dass die Klangwiedergabe quasi im Hintergrund abläuft und Sie nicht erst warten müssen, bis der Sound oder das ganze Musikstück beendet ist. "

WEG 5 - MIDI ABSPIELEN

```
Sub PlayMIDI()
Dim Player
Player = Shell("C:\Progra~1\Window~1\mplayer2.exe
C:\Songs\regypti.mid", 6)
AppActivate Player
End Sub
```

Lautstärke von Wave-Dateien einstellen

MEINE VERSION

```
Private Declare Function waveOutSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Private Type intVolumeType
   Left As Integer
   Right As Integer
End Type

Private Type lngVolumeType
   Value As Long
End Type

Const SND_ASYNC = &H1
Const SND_NODEFAULT = &H2



Sub Sound()

  Dim SoundName As String
  SoundName = "C:\bach.wav"
  Call sndPlaySound(SoundName$, SND_ASYNC Or &H8) ' SND_NODEFAULT)

End Sub


Sub SetVolume()
   Dim intVol As intVolumeType
   Dim lngVol As lngVolumeType

   For t = 1 To 32000
   intVol.Left = t
   intVol.Right = t
   LSet lngVol = intVol
   Call waveOutSetVolume(0, lngVol.Value)
   Next t

   For t = 32000 To 1 Step -1
   intVol.Left = t
```

```
   intVol.Right = t
   LSet lngVol = intVol
   Call waveOutSetVolume(0, lngVol.Value)
   Next t

End Sub
```

ORIGINAL

```
Option Explicit

Private Declare Function waveOutSetVolume Lib "winmm.dll" ( _
   ByVal uDeviceID As Long, _
   ByVal dwVolume As Long) As Long
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" ( _
   ByVal lpszSoundName As String, _
   ByVal uFlags As Long) As Long

Private Type intVolumeType
   Left As Integer
   Right As Integer
End Type

Private Type lngVolumeType
   Value As Long
End Type

Const SND_ASYNC = &H1
Const SND_NODEFAULT = &H2

Private Sub CheckBox1_Click()
   If CheckBox1.Value Then ScrollBar2.Value = ScrollBar1.Value
```

```
End Sub

Private Sub CommandButton1_Click()
  Dim SoundName As String
  SoundName = "C:\Windows\Media\tada.wav"    ' Pick any wave file.
  Call sndPlaySound(SoundName$, SND_ASYNC Or SND_NODEFAULT)
End Sub

Private Sub ScrollBar1_Change()
   Call SetVolume
   If CheckBox1.Value Then ScrollBar2.Value = ScrollBar1.Value
End Sub

Private Sub ScrollBar1_Scroll()
   Call SetVolume
   If CheckBox1.Value Then ScrollBar2.Value = ScrollBar1.Value
End Sub

Private Sub ScrollBar2_Change()
   Call SetVolume
   If CheckBox1.Value Then ScrollBar1.Value = ScrollBar2.Value
End Sub

Private Sub ScrollBar2_Scroll()
   Call SetVolume
   If CheckBox1.Value Then ScrollBar1.Value = ScrollBar2.Value
End Sub

Private Sub UserForm_Activate()
   ScrollBar1.Value = 5000
   ScrollBar2.Value = 5000
End Sub

Function SignedInteger(ByVal UnsignedInteger As Long) As Integer
   UnsignedInteger = UnsignedInteger And 65535
   If (UnsignedInteger And 32768) Then
      SignedInteger = UnsignedInteger - 65536
   Else
      SignedInteger = UnsignedInteger
   End If
End Function
```

```
Sub SetVolume()
   Dim intVol As intVolumeType
   Dim lngVol As lngVolumeType
   intVol.Left = SignedInteger(ScrollBar1.Value * 6.5535)
   intVol.Right = SignedInteger(ScrollBar2.Value * 6.5535)
   LSet lngVol = intVol
   Call waveOutSetVolume(0, lngVol.Value)
End Sub
```

MP3-Dateien (extern) abspielen

MP3 abspielen - Teil 1

```
Option Explicit
'Abspielen einer mp3-Datei ohne Player
'Quelle - http://www.vbarchiv.net


 'zunächst die benötigte API-Deklaration
Private Declare Function mciSendString Lib "winmm.dll" _
Alias "mciSendStringA" (ByVal lpszCommand As String, _
ByVal lpszReturnString As String, _
ByVal cchReturnLength As Long, _
ByVal hwndCallback As Long) As Long

'Abspielen einer MP3-Datei
Sub playMP3()
Dim sFile As String

sFile = "F:\Dein Pfad\Deine Datei.mp3"

' MCI öffnen
If mciSendString("open """ & sFile & _
""" type MPEGVideo alias MyMP3", 0, 0, 0) = 0 Then
' MP3 abspielen
mciSendString "play MyMP3 from 0", 0, 0, 0
End If
End Sub
```

```
Sub StopMP3()
'Wiedergabe beenden und MCI schließen
mciSendString "stop MyMP3", 0, 0, 0
mciSendString "close MyMP3", 0, 0, 0
End Sub
```

MP3 Abspielen Version 2

Voraussetzung ist aber, dass du auf deinem Rechner DirectX 7.0 oder höher hast. Sollte aber Standard ab Windows98 sein.

Setzte im VBA - Editor unter Extras - Verweise einen Verweis auf "ActiveMovie control type library".

Und ein Codebeispiel:

```
Option Explicit

Public bolPlaying As Boolean
Public objDirectX As FilgraphManager
Public bolPause As Boolean

Public Sub PlayMusic()
   Set objDirectX = New FilgraphManager
   objDirectX.RenderFile "D:\Eigene Dateien\Eigene Präsentationen\Holland - Niederlande.mp3"
   objDirectX.Run
   bolPlaying = True
   bolPause = False
End Sub

Public Sub PauseMusic()
   If bolPlaying Then
      If bolPause Then
         objDirectX.Run
         bolPause = False
      Else
         objDirectX.Pause
         bolPause = True
      End If
   End If
End Sub
```

```
Public Sub StopMusic()
   bolPlaying = False
   bolPause = False
   objDirectX.Stop
   Set objDirectX = Nothing
End Sub
```

VERSION 3

```
'Verweis auf "ActiveMovie control type library" setzen !!!

Private objDirectX As QuartzTypeLib.FilgraphManager
Private blnBreak As Boolean, blnPlay As Boolean

Public Sub PlayMusic(strFile As String)
   Set objDirectX = New QuartzTypeLib.FilgraphManager
   objDirectX.RenderFile strFile
   objDirectX.Run
   blnPlay = True
   blnBreak = False
End Sub

Public Sub BreakMusic()
   If blnPlay Then
      If blnBreak Then objDirectX.Run Else objDirectX.Pause
      blnBreak = Not blnBreak
   End If
End Sub

Public Sub StopMusic()
   blnPlay = False
   blnBreak = False
   objDirectX.Stop
   Set objDirectX = Nothing
End Sub

Sub test1()
   Call PlayMusic("D:\Eigene Dateien\Eigene Präsentationen\Holland - Niederlande.mp3")
End Sub
```

VERSION 4

```
Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" ( _
   ByVal lpszCommand As String, _
   ByVal lpszReturnString As String, _
   ByVal cchReturnLength As Long, _
   ByVal hwndCallback As Long) As Long

Public Sub playMP3()
   Dim strFile As String
   strFile = "D:\Eigene Dateien\Eigene Präsentationen\Holland - Niederlande.mp3"
   If mciSendString("open """ & strFile & _
      """ type MPEGVideo alias MyMP3", vbNullString, 0, 0) = 0 Then
      mciSendString "play MyMP3 from 0", vbNullString, 0, 0
   End If
End Sub

Public Sub StopMP3()
   mciSendString "stop MyMP3", vbNullString, 0, 0
   mciSendString "close MyMP3", vbNullString, 0, 0
End Sub
```

Sprechen

VARIANTE 1

```
Application.Speech.Speak "Hallo mein Lieber"
```

VARIANTE 2 - ergibt denselben Sound wie Variante 1

```
Public Function Sprich(strText)
Dim sabbeln As Object
Set sabbeln = CreateObject("SAPI.SpVoice")
sabbeln.Speak strText
Set sabbeln = Nothing
End Function

Sub Test

a=SPRICH("Hello")

End Sub
```

WAVE, MIDI, VIDEO abspielen

Für Wave-Files abspielen sieht auch hier die ersten beiden Einträge "Abspielen von …"

```
' ------------------------------------------------------------------------
'
'  Title: Plays Wave files, Midi files, AVI Video files..
'


' ------------------------------------------------------------------------

' Declarations

'   Multimedia Class
'   ~~~~~~~~~~~~~~~~
'
'   Declare As follows:
'
'       Dim MMPlayer As New clsMultimedia
'
'
'   Properties:
'
'       Filename            (String       The media file currently open)
'       Wait                (True/False   If the code should pause until until the file has finished playing)
'       Length              (Integer      The length (in seconds?) of the media file)
'       Position            (Integer      The position of the 'playback head')
'       Status              (String       What's happening to the media file...playing, stopped etc.)
'
'
'
'
'
'   Methods:
'
'       mmOpen(Filename)    (Open a media file specified by "Filename")
'       mmClose()           (Close the currently open media file)
'       mmPlay              (Plays the currently open media file)
```

```
'       mmPause()           (Pause the currently playing media file)
'       mmStop              (Stop the currently playing media file)
'       mmSeek(Time)        (Move the playback head to a position specified by "Time")

Option Explicit

Private sAlias As String

Private sFilename As String
Private nLength As Single

Private nPosition As Single
Private sStatus As Single
Private bWait As Boolean

Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpstrCommand As String, ByVal
lpstrReturnString As String, ByVal uReturnLength As Long, ByVal hwndCallback As Long) As Long


' ---------------
' Code
' ---------------

Public Sub mmOpen(ByVal sTheFile As String)
    Dim nReturn As Long
    Dim sType As String

    If sAlias <> "" Then
        mmClose
    End If

    Select Case UCase$(Right$(sTheFile, 3))
        Case "WAV"
            sType = "Waveaudio"
        Case "AVI"
            sType = "AviVideo"
        Case "MID"
            sType = "Sequencer"
        Case "MP3"
            sType = "MPegVideo"
        Case Else
            Exit Sub
```

```
    End Select

    Randomize
    sAlias = Right$(sTheFile, 3) & Minute(Now) & Second(Now) & Int(1000 * Rnd + 1)

    If InStr(sTheFile, " ") Then sTheFile = Chr(34) & sTheFile & Chr(34)
    nReturn = mciSendString("Open " & sTheFile & " ALIAS " & sAlias & " TYPE " & sType & " wait", "", 0, 0)

End Sub

Public Sub mmClose()
    Dim nReturn As Long

    If sAlias = "" Then Exit Sub

    nReturn = mciSendString("Close " & sAlias, "", 0, 0)
    sAlias = ""
    sFilename = ""
End Sub

Public Sub mmPause()
    Dim nReturn As Long

    If sAlias = "" Then Exit Sub

    nReturn = mciSendString("Pause " & sAlias, "", 0, 0)
End Sub

Public Sub mmPlay()
    Dim nReturn As Long

    If sAlias = "" Then Exit Sub

    If bWait Then
        nReturn = mciSendString("Play " & sAlias & " wait", "", 0, 0)
    Else
        nReturn = mciSendString("Play " & sAlias, "", 0, 0)
    End If
End Sub

Public Sub mmStop()
    Dim nReturn As Long
```

```
    If sAlias = "" Then Exit Sub

    nReturn = mciSendString("Stop " & sAlias, "", 0, 0)
End Sub

Public Sub mmSeek(ByVal nPosition As Single)
    Dim nReturn As Long

    nReturn = mciSendString("seek " & sAlias & " to " & nPosition, "", 0, 0)
End Sub

Property Get FileName() As String
    FileName = sFilename
End Property

Property Let FileName(ByVal sTheFile As String)
    mmOpen sTheFile
End Property

Property Get Wait() As Boolean
    Wait = bWait
End Property

Property Let Wait(bWaitValue As Boolean)
    bWait = bWaitValue
End Property

Property Get Length() As Single
    Dim nReturn As Long, nLength As Integer

    Dim sLength As String * 255

    If sAlias = "" Then
        Length = 0
        Exit Property
    End If

    nReturn = mciSendString("Status " & sAlias & " length", sLength, 255, 0)
    nLength = InStr(sLength, Chr$(0))
    Length = Val(Left$(sLength, nLength - 1))
End Property
```

```
Property Let Position(ByVal nPosition As Single)
    mmSeek nPosition
End Property

Property Get Position() As Single
    Dim nReturn As Integer, nLength As Integer

    Dim sPosition As String * 255

    If sAlias = "" Then Exit Property

    nReturn = mciSendString("Status " & sAlias & " position", sPosition, 255, 0)
    nLength = InStr(sPosition, Chr$(0))
    Position = Val(Left$(sPosition, nLength - 1))
End Property

Property Get Status() As String
    Dim nReturn As Integer, nLength As Integer

    Dim sStatus As String * 255

    If sAlias = "" Then Exit Property

    nReturn = mciSendString("Status " & sAlias & " mode", sStatus, 255, 0)

    nLength = InStr(sStatus, Chr$(0))
    Status = Left$(sStatus, nLength - 1)
End Property
```

Videos abspielen

```
'  Title: Use MS Word's Spell Checker in Visual Basic
'


'-----------------------------------------------------------------------

'----------
' Function
```

```
' ----------

Public Function SpellCheck(ByVal IncorrectText$) As String
      Dim Word As Object, retText$

      On Error Resume Next

      ' Create the Object and open Word
      Set Word = CreateObject("Word.Basic")

      ' Change the active window to Word,
      'and insert the text from Text1 into Word.
      Word.AppShow
      Word.FileNew

      Word.Insert IncorrectText

      ' Runs the Speller Corrector
      Word.ToolsSpelling
      Word.EditSelectAll

      ' Trim the trailing character from the returned text.
      retText = Word.Selection$()
      SpellCheck = Left$(retText, Len(retText) - 1)

      ' Close the Document and return to Visual Basic.
      Word.FileClose 2
      Show

      ' Set the word object to nothing to liberate the'occupied memory
      Set Word = Nothing
End Function
```

SYSTEM - SYSTEMORDNER

Bildschirmauflösung auslesen

```
Declare Function GetSystemMetrics Lib "user32" (ByVal nIndex As
Long) As Long
Const SM_CYSCREEN As Long = 1
Const SM_CXSCREEN As Long = 0
Sub GetScreenDimensions()
Dim lWidth As Long
Dim lHeight As Long
lWidth = GetSystemMetrics(SM_CXSCREEN)
lHeight = GetSystemMetrics(SM_CYSCREEN)
MsgBox "Screen Width = " & lWidth & vbCrLf & "Screen
Height = " &
lHeight
End Sub
Declare Function GetDeviceCaps Lib "gdi32" (ByVal hdc As Long,
ByVal nIndex As Long) As Long
Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, ByVal hdc
As Long) As Long
Const HORZRES = 8
Const VERTRES = 10
Function ScreenResolution()
Dim lRval As Long
Dim lDc As Long
Dim lHSize As Long
Dim lVSize As Long
lDc = GetDC(0&)
lHSize = GetDeviceCaps(lDc, HORZRES)
lVSize = GetDeviceCaps(lDc, VERTRES)
lRval = ReleaseDC(0, lDc)
ScreenResolution = lHSize & "x" & lVSize
End Function
Sub GetScreenSize()
Debug.Print ScreenResolution()
End Sub
```

VERSION 2

```
'  Title: Display the Resolution of the Monitor
'
```

```
' ------------------------------------------------------------------------

' Declarations

Private Declare Function GetSystemMetrics Lib "User32" (ByVal index As Long) As Long
Dim X As Long, Y As Long

' Code

Private Sub Command1_Click()
   x = GetSystemMetrics(0)
   y = GetSystemMetrics(1)

   Msgbox "Resolutions is : " & CStr(x) & " X "  & CStr(y),vbExclamation,"Monitor"
End Sub
```

Bildschirmauflösung ändern

```
Private Declare Function EnumDisplaySettings Lib "user32" Alias
"EnumDisplaySettingsA" (ByVal lpszDeviceName As Long, ByVal iModeNum As
Long, lpDevMode As Any) As Boolean

Private Declare Function ChangeDisplaySettings Lib "user32" Alias
"ChangeDisplaySettingsA" (lpDevMode As Any, ByVal dwflags As Long) As
Long

Const CCDEVICENAME = 32
Const CCFORMNAME = 32
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000

Type DEVMODE
dmDeviceName As String * CCDEVICENAME
dmSpecVersion As Integer
dmDriverVersion As Integer
dmSize As Integer
dmDriverExtra As Integer
dmFields As Long
dmOrientation As Integer
dmPaperSize As Integer
dmPaperLength As Integer
dmPaperWidth As Integer
dmScale As Integer
dmCopies As Integer
dmDefaultSource As Integer
dmPrintQuality As Integer
dmColor As Integer
```

```
dmDuplex As Integer
dmYResolution As Integer
dmTTOption As Integer
dmCollate As Integer
dmFormName As String * CCFORMNAME
dmUnusedPadding As Integer
dmBitsPerPel As Integer
dmPelsWidth As Long
dmPelsHeight As Long
dmDisplayFlags As Long
dmDisplayFrequency As Long
End Type

Dim DevM As DEVMODE

'------------------------------------------------------------
'Comments : Allows changing of screen resolution in Win95
' Example: Call ChangeScreenResolution(800,600)
'Parameters: iWidth, iheight: integer values of resolution
'Sets : Requested screen resolution or if screen is
' already at resolution returns true
'Returns : None
'Created by: Bridgett M. Cole, Saltware Computer Services
'Created : 12/1/97 8:15:58 PM
'------------------------------------------------------------
Private Sub ChangeScreenResolution(iWidth As Single, iHeight As Single)

Dim a As Boolean
Dim i&
Dim b&

i = 0

Do
a = EnumDisplaySettings(0&, i&, DevM)
i = i + 1
Loop Until (a = False)

DevM.dmFields = DM_PELSWIDTH Or DM_PELSHEIGHT
DevM.dmPelsWidth = iWidth
DevM.dmPelsHeight = iHeight
b = ChangeDisplaySettings(DevM, 0)
End Sub

Sub ChangeTo800()

Call ChangeScreenResolution(800, 600)

End Sub
```

CAPS-Lock einschalten

```
Private Declare Function GetVersionEx Lib "kernel32" _
Alias "GetVersionExA" (lpVersionInformation As OSVERSIONINFO) As Long
Private Declare Sub keybd_event Lib "user32" _
(ByVal bVk As Byte, ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo
As Long)
Private Declare Function GetKeyboardState Lib "user32" _
(pbKeyState As Byte) As Long
Private Declare Function SetKeyboardState Lib "user32" _
(lppbKeyState As Byte) As Long

Private Type OSVERSIONINFO
dwOSVersionInfoSize As Long
dwMajorVersion As Long
dwMinorVersion As Long
dwBuildNumber As Long
dwPlatformId As Long
szCSDVersion As String * 128 ' Maintenance string for PSS usage
End Type

Const VK_CAPITAL = &H14
Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2
Const VER_PLATFORM_WIN32_NT = 2
Const VER_PLATFORM_WIN32_WINDOWS = 1

Dim Keys(0 To 255) As Byte

Sub SetCapsOn()
Dim o As OSVERSIONINFO
Dim NumLockState As Boolean
Dim ScrollLockState As Boolean
Dim CapsLockState As Boolean ' CapsLock handling:
o.dwOSVersionInfoSize = Len(o)
GetVersionEx o
CapsLockState = Keys(VK_CAPITAL)
If CapsLockState <> True Then 'Turn capslock on
If o.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS Then '===== Win95
Keys(VK_CAPITAL) = 1
SetKeyboardState Keys(0)
ElseIf o.dwPlatformId = VER_PLATFORM_WIN32_NT Then '===== WinNT
'Simulate Key Press
keybd_event VK_CAPITAL, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
'Simulate Key Release
keybd_event VK_CAPITAL, &H45, KEYEVENTF_EXTENDEDKEY _
Or KEYEVENTF_KEYUP, 0
End If
End If
End Sub
```

CD-ROM-Funktionen

```
' ---------------------------------------------------------------------------
'  Title: Common CD-ROM functions and features
'
' ---------------------------------------------------------------------------

Option Explicit

' ----------------------------
' Constants & API Declarations
' ----------------------------

'Declare for CD-ROM DriveLetter / Volume Label
Private Declare Function GetLogicalDriveStrings _
  Lib "kernel32" Alias "GetLogicalDriveStringsA" _
  (ByVal nBufferLength As Long, _
  ByVal lpBuffer As String) As Long

Private Declare Function GetDriveType _
  Lib "kernel32" Alias "GetDriveTypeA" _
  (ByVal nDrive As String) As Long

Private Declare Function GetVolumeInformation _
  Lib "kernel32" Alias "GetVolumeInformationA" _
  (ByVal lpRootPathName As String, _
  ByVal lpVolumeNameBuffer 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 for CD-ROM DriveLetter / Volume Label
Private Const DRIVE_CDROM = 5

'Property holders for CD-ROM DriveLetter / Volume Label
Private sCDROMDrive As String
Private sCDROMVolume As String
Private bCDROMExists As Boolean
```

```
'Declare for Eject/Close Feature
Private Declare Function mciSendString Lib "winmm.dll" _
  Alias "mciSendStringA" _
  (ByVal lpstrCommand As String, _
  ByVal lpstrReturnString As String, _
  ByVal uReturnLength As Long, _
  ByVal hwndCallback As Long) As Long

'Const for Eject/Close Feature
Private Const MCI_CDADUIO_DOOR_OPEN = "CDAudio Door Open"
Private Const MCI_CDADUIO_DOOR_CLOSE = "CDAudio Door Close"


' ------
' Code
' ------

Public Sub EjectCD()
   Call mciSendString("Set " & MCI_CDADUIO_DOOR_OPEN & " Wait", 0&, 0&, 0&)
End Sub

Public Sub CloseCD()
   Call mciSendString("Set " & MCI_CDADUIO_DOOR_CLOSE & " Wait", 0&, 0&, 0&)
End Sub

Public Property Get CDDriveLetter() As String
  GetCDROMInformation
  CDDriveLetter = sCDROMDrive
End Property

Public Property Get CDVolumeLabel() As String
  GetCDROMInformation
  CDVolumeLabel = sCDROMVolume
End Property

Public Property Get CDExists() As Boolean
  GetCDROMInformation
  CDExists = bCDROMExists
End Property

Private Sub GetCDROMInformation()
```

```
'get the available drives, determine their type,
'and if CD, get the CD volume label
Dim r As Long
Dim DriveType As Long
Dim allDrives As String
Dim JustOneDrive As String
Dim CDLabel As String
Dim pos As Integer
Dim CDfound As Boolean

'pad the string with spaces
allDrives = Space$(64)
'call the API to get the string containing all drives
r = GetLogicalDriveStrings(Len(allDrives), allDrives)

'trim off any trailing spaces. 'AllDrives'
'now contains all the drive letters.
allDrives = Left$(allDrives, r)
'begin a loop
Do
  'first check that there is a chr$(0) in the string
   pos = InStr(allDrives, Chr$(0))
   'if there's one, then...
   If pos Then
     'extract the drive up to the chr$(0)
      JustOneDrive = Left$(allDrives, pos - 1)
     'and remove that from the Alldrives string,
     'so it won't be checked again
      allDrives = Mid$(allDrives, pos + 1)
     'with the one drive, call the API to
     'determine the drive type
      DriveType = GetDriveType(JustOneDrive)
     'check if its what we want
     If DriveType = DRIVE_CDROM Then
        'got it (or at least the first one,
        'anyway, if more than one), so set
        'the found flag... this part can be modified
        'to continue searching remaining drives for
        'those systems that might have more than
        'one CD installed.
        CDfound = True
        DoEvents
```

```
            CDLabel = GetVolumeLabel(JustOneDrive)
            'we're done for now, so get out
            Exit Do
          End If
        End If
    Loop Until allDrives = "" Or DriveType = DRIVE_CDROM

    'display the appropriate message
    If CDfound Then
      sCDROMDrive = UCase$(JustOneDrive)
      sCDROMVolume = CDLabel
      bCDROMExists = True
    Else
      sCDROMDrive = ""
      sCDROMVolume = ""
      bCDROMExists = False
    End If
End Sub

Private Function GetVolumeLabel(CDPath As String) As String
    'create working variables  'to keep it simple, use dummy variables for info
    'we're not interested in right now
    Dim r As Long
    Dim DrvVolumeName As String
    Dim pos As Integer
    Dim UnusedVal1 As Long
    Dim UnusedVal2 As Long
    Dim UnusedVal3 As Long
    Dim UnusedStr As String

    DrvVolumeName = Space$(14)
    UnusedStr = Space$(32)

    'do what it says
    r = GetVolumeInformation(CDPath, _
      DrvVolumeName, _
      Len(DrvVolumeName), _
      UnusedVal1, UnusedVal2, _
      UnusedVal3, _
      UnusedStr, Len(UnusedStr))

    'error check
```

```
  If r = 0 Then Exit Function

  'the volume label
  pos = InStr(DrvVolumeName, Chr$(0))
  If pos Then DrvVolumeName = Left$(DrvVolumeName, pos - 1)
  If Len(Trim$(DrvVolumeName$)) = 0 Then DrvVolumeName$ = "(no label)"
  GetVolumeLabel = DrvVolumeName$
End Function


Private Sub Class_Initialize()
  'Get the CD-ROM info and initilize the Property valiables
  GetCDROMInformation
End Sub
```

Freier Festplattenspeicherplatz auslesen

Funktion 0

```
'  Title: Get the amount of free disk space for the specified drive
' ----------------------------------------------------------------------------

'-------------------------------------------------------------
' FUNCTION: GetDiskSpaceFree
' Get the amount of free disk space for the specified drive
'
' IN: [strDrive] - drive to check space for
'
' Returns: Amount of free disk space, or -1 if an error occurs
'-------------------------------------------------------------
'
Function GetDiskSpaceFree(ByVal strDrive As String) As Long
    Dim strCurDrive As String
    Dim lDiskFree As Long
```

```
On Error Resume Next

'
'Save the current drive
'
strCurDrive = Left$(CurDir$, 2)

'
'Fixup drive so it includes only a drive letter and a colon
'
If InStr(strDrive, gstrSEP_DRIVE) = 0 Or Len(strDrive) > 2 Then
    strDrive = Left$(strDrive, 1) & gstrSEP_DRIVE
End If

'
'Change to the drive we want to check space for.  The DiskSpaceFree() API
'works on the current drive only.
'
ChDrive strDrive

'
'If we couldn't change to the request drive, it's an error, otherwise return
'the amount of disk space free
'
If Err <> 0 Or (strDrive <> Left$(CurDir$, 2)) Then
    lDiskFree = -1
Else
    Dim lRet As Long
    Dim lBytes As Long, lSect As Long, lClust As Long, lTot As Long

    lRet = GetDiskFreeSpace(vbNullString, lSect, lBytes, lClust, lTot)
    On Error Resume Next
    lDiskFree = (lBytes * lSect) * lClust
    If Err Then lDiskFree = 2147483647
End If

If lDiskFree = -1 Then
    MsgError Error$ & vbLf & vbLf & ResolveResString(resDISKSPCERR) & strDrive, vbExclamation, gstrTitle
End If

GetDiskSpaceFree = lDiskFree
```

```
    '
    'Cleanup by setting the current drive back to the original
    '
    ChDrive strCurDrive

    Err = 0
End Function
```

Funktion 1

```
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias
"GetDiskFreeSpaceA" (ByVal lpRootPathName As String,
lpSectorsPerCluster As Long, lpBytesPerSector As Long, _
lpNumberOfFreeClusters As Long, _
lpTotalNumberOfClusters As Long) As Long
Function GetFreeSpace(ByVal Drive$) As Double
Dim SecPerCluster&, BytesPerSector&, NumFreeClusters&,
NumClusters&
Dim lRet&
Dim dVal#
lRet& = GetDiskFreeSpace(Drive$, SecPerCluster&, BytesPerSector&,
NumFreeClusters&, NumClusters&)
dVal# = SecPerCluster& * BytesPerSector&
dVal# = dVal# * NumFreeClusters&
GetFreeSpace = dVal#
End Function
```

Funktion 2

```
' ----------------------------
' Constants & API Declarations
' ----------------------------


Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA" (ByVal lpRootPathName As String,
lpSectorsPerCluster As Long, lpBytesPerSector As Long, lpNumberOfFreeClusters As Long, lpTtoalNumberOfClusters As Long)
As Long

Public Type DiskInformation
    lpSectorsPerCluster As Long
    lpBytesPerSector As Long
    lpNumberOfFreeClusters As Long
```

```
    lpTotalNumberOfClusters As Long
End Type


' ----------
' Function
' ----------

Function FreeDiskSpace()
    Dim info As DiskInformation
    Dim lAnswer As Long
    Dim lpRootPathName As String
    Dim lpSectorsPerCluster As Long
    Dim lpBytesPerSector As Long
    Dim lpNumberOfFreeClusters As Long
    Dim lpTotalNumberOfClusters As Long
    Dim lBytesPerCluster As Long
    Dim lNumFreeBytes As Double
    Dim sString As String

    lpRootPathName = "c:\"
    lAnswer = GetDiskFreeSpace(lpRootPathName, lpSectorsPerCluster, lpBytesPerSector, lpNumberOfFreeClusters,
lpTotalNumberOfClusters)
    lBytesPerCluster = lpSectorsPerCluster * lpBytesPerSector
    lNumFreeBytes = lBytesPerCluster * lpNumberOfFreeClusters

    ' sString = "Number of Free Bytes : " & lNumFreeBytes & vbCr & vbLf
    ' sString = sString & "Number of Free Kilobytes: " & (lNumFreeBytes / 1024) & "K" & vbCr & vbLf
    ' sString = sString & "Number of Free Megabytes: " & Format(((lNumFreeBytes / 1024) / 1024), "0.00") & "MB"
End Function
```

IP-Adresse auslesen

```
' ------------------------------------------------------------------------
'  Title: Find Your IP Address
' ------------------------------------------------------------------------

' ---------------------------
' Constants & API Declarations
' ---------------------------
```

```
' Winsock2.bas
Option Explicit

Public Const MAX_WSADescription = 256
Public Const MAX_WSASYSStatus = 128
Public Const ERROR_SUCCESS       As Long = 0
Public Const WS_VERSION_REQD     As Long = &H101
Public Const WS_VERSION_MAJOR    As Long = WS_VERSION_REQD \ &H100 And &HFF&
Public Const WS_VERSION_MINOR    As Long = WS_VERSION_REQD And &HFF&
Public Const MIN_SOCKETS_REQD    As Long = 1
Public Const SOCKET_ERROR        As Long = -1

Public Type HOSTENT
   hName      As Long
   hAliases   As Long
   hAddrType  As Integer
   hLen       As Integer
   hAddrList  As Long
End Type

Public Type WSADATA
   wVersion      As Integer
   wHighVersion  As Integer
   szDescription(0 To MAX_WSADescription)   As Byte
   szSystemStatus(0 To MAX_WSASYSStatus)    As Byte
   wMaxSockets   As Integer
   wMaxUDPDG     As Integer
   dwVendorInfo  As Long
End Type


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

Public Declare Function WSAStartup Lib "WSOCK32.DLL" _
   (ByVal wVersionRequired As Long, lpWSADATA As WSADATA) As Long

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

Public Declare Function gethostname Lib "WSOCK32.DLL" _
   (ByVal szHost As String, ByVal dwHostLen As Long) As Long

Public Declare Function gethostbyname Lib "WSOCK32.DLL" _
   (ByVal szHost As String) As Long

Public Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
   (hpvDest As Any, ByVal hpvSource As Long, ByVal cbCopy As Long)


' ---------
' Code
' ---------
Public Function GetIPAddress() As String
   Dim sHostName    As String * 256
```

```
   Dim lpHost    As Long
   Dim HOST      As HOSTENT
   Dim dwIPAddr  As Long
   Dim tmpIPAddr() As Byte
   Dim i         As Integer
   Dim sIPAddr  As String

   If Not SocketsInitialize() Then
      GetIPAddress = ""
      Exit Function
   End If
   If gethostname(sHostName, 256) = SOCKET_ERROR Then
      GetIPAddress = ""
      MsgBox "Windows Sockets error " & Str$(WSAGetLastError()) & _
              " has occurred. Unable to successfully get Host Name."
      SocketsCleanup
      Exit Function
   End If
   sHostName = Trim$(sHostName)
   lpHost = gethostbyname(sHostName)

   If lpHost = 0 Then
      GetIPAddress = ""
      MsgBox "Windows Sockets are not responding. " & _
              "Unable to successfully get Host Name."
      SocketsCleanup
      Exit Function
   End If

   CopyMemory HOST, lpHost, Len(HOST)
   CopyMemory dwIPAddr, HOST.hAddrList, 4
   ReDim tmpIPAddr(1 To HOST.hLen)
   CopyMemory tmpIPAddr(1), dwIPAddr, HOST.hLen
   For i = 1 To HOST.hLen
      sIPAddr = sIPAddr & tmpIPAddr(i) & "."
   Next

   GetIPAddress = Mid$(sIPAddr, 1, Len(sIPAddr) - 1)

   SocketsCleanup
End Function

' ================================================================== '

Public Function HiByte(ByVal wParam As Integer)
    HiByte = wParam \ &H100 And &HFF&
End Function

' ================================================================== '

Public Function LoByte(ByVal wParam As Integer)
    LoByte = wParam And &HFF&
End Function
```

```
' ================================================================== '

Public Sub SocketsCleanup()
    If WSACleanup() <> ERROR_SUCCESS Then
        MsgBox "Socket error occurred in Cleanup."
    End If
End Sub

' ================================================================== '

Public Function SocketsInitialize() As Boolean
   Dim WSAD As WSADATA
   Dim sLoByte As String
   Dim sHiByte As String

   If WSAStartup(WS_VERSION_REQD, WSAD) <> ERROR_SUCCESS Then
      MsgBox "The 32-bit Windows Socket is not responding."
      SocketsInitialize = False
      Exit Function
   End If


   If WSAD.wMaxSockets < MIN_SOCKETS_REQD Then
        MsgBox "This application requires a minimum of " & _
                CStr(MIN_SOCKETS_REQD) & " supported sockets."

        SocketsInitialize = False
        Exit Function
   End If


   If LoByte(WSAD.wVersion) < WS_VERSION_MAJOR Or _
     (LoByte(WSAD.wVersion) = WS_VERSION_MAJOR And _
      HiByte(WSAD.wVersion) < WS_VERSION_MINOR) Then

      sHiByte = CStr(HiByte(WSAD.wVersion))
      sLoByte = CStr(LoByte(WSAD.wVersion))

      MsgBox "Sockets version " & sLoByte & "." & sHiByte & _
             " is not supported by 32-bit Windows Sockets."

      SocketsInitialize = False
      Exit Function

   End If
    SocketsInitialize = True
End Function
```

IP-Adresse anpingen

```
' ------------------------------------------------------------------------
'  Title: How to ping an IP address using VB
'
' ------------------------------------------------------------------------

' ----------------------------
' Constants & API Declarations
' ----------------------------

Option Explicit

Public Const IP_STATUS_BASE = 11000
Public Const IP_SUCCESS = 0
Public Const IP_BUF_TOO_SMALL = (11000 + 1)
Public Const IP_DEST_NET_UNREACHABLE = (11000 + 2)
Public Const IP_DEST_HOST_UNREACHABLE = (11000 + 3)
Public Const IP_DEST_PROT_UNREACHABLE = (11000 + 4)
Public Const IP_DEST_PORT_UNREACHABLE = (11000 + 5)
Public Const IP_NO_RESOURCES = (11000 + 6)
Public Const IP_BAD_OPTION = (11000 + 7)
Public Const IP_HW_ERROR = (11000 + 8)
Public Const IP_PACKET_TOO_BIG = (11000 + 9)
Public Const IP_REQ_TIMED_OUT = (11000 + 10)
Public Const IP_BAD_REQ = (11000 + 11)
Public Const IP_BAD_ROUTE = (11000 + 12)
Public Const IP_TTL_EXPIRED_TRANSIT = (11000 + 13)
Public Const IP_TTL_EXPIRED_REASSEM = (11000 + 14)
Public Const IP_PARAM_PROBLEM = (11000 + 15)
Public Const IP_SOURCE_QUENCH = (11000 + 16)
Public Const IP_OPTION_TOO_BIG = (11000 + 17)
Public Const IP_BAD_DESTINATION = (11000 + 18)
Public Const IP_ADDR_DELETED = (11000 + 19)
Public Const IP_SPEC_MTU_CHANGE = (11000 + 20)
Public Const IP_MTU_CHANGE = (11000 + 21)
Public Const IP_UNLOAD = (11000 + 22)
Public Const IP_ADDR_ADDED = (11000 + 23)
Public Const IP_GENERAL_FAILURE = (11000 + 50)
Public Const MAX_IP_STATUS = 11000 + 50
Public Const IP_PENDING = (11000 + 255)
Public Const PING_TIMEOUT = 200
Public Const WS_VERSION_REQD = &H101
Public Const WS_VERSION_MAJOR = WS_VERSION_REQD \ &H100 And &HFF&
Public Const WS_VERSION_MINOR = WS_VERSION_REQD And &HFF&
Public Const MIN_SOCKETS_REQD = 1
```

```
Public Const SOCKET_ERROR = -1

Public Const MAX_WSADescription = 256
Public Const MAX_WSASYSStatus = 128

Public Type ICMP_OPTIONS
   Ttl            As Byte
   Tos            As Byte
   Flags          As Byte
   OptionsSize    As Byte
   OptionsData    As Long
End Type

Dim ICMPOPT As ICMP_OPTIONS

Public Type ICMP_ECHO_REPLY
   Address        As Long
   status         As Long
   RoundTripTime  As Long
   DataSize       As Integer
   Reserved       As Integer
   DataPointer    As Long
   Options        As ICMP_OPTIONS
   Data           As String * 250
End Type

Public Type HOSTENT
   hName As Long
   hAliases As Long
   hAddrType As Integer
   hLen As Integer
   hAddrList As Long
End Type

Public Type WSADATA
   wVersion As Integer
   wHighVersion As Integer
   szDescription(0 To MAX_WSADescription) As Byte
   szSystemStatus(0 To MAX_WSASYSStatus) As Byte
   wMaxSockets As Integer
   wMaxUDPDG As Integer
   dwVendorInfo As Long
End Type
```

```
Public Declare Function IcmpCreateFile Lib "icmp.dll" () As Long

Public Declare Function IcmpCloseHandle Lib "icmp.dll" _
  (ByVal IcmpHandle As Long) As Long

Public Declare Function IcmpSendEcho Lib "icmp.dll" _
  (ByVal IcmpHandle As Long, _
   ByVal DestinationAddress As Long, _
   ByVal RequestData As String, _
   ByVal RequestSize As Integer, _
   ByVal RequestOptions As Long, _
   ReplyBuffer As ICMP_ECHO_REPLY, _
   ByVal ReplySize As Long, _
   ByVal Timeout As Long) As Long

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

Public Declare Function WSAStartup Lib "WSOCK32.DLL" _
  (ByVal wVersionRequired As Long, _
   lpWSADATA As WSADATA) As Long

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

Public Declare Function gethostname Lib "WSOCK32.DLL" _
  (ByVal szHost As String, _
   ByVal dwHostLen As Long) As Long

Public Declare Function gethostbyname Lib "WSOCK32.DLL" _
  (ByVal szHost As String) As Long

Public Declare Sub RtlMoveMemory Lib "kernel32" _
  (hpvDest As Any, _
   ByVal hpvSource As Long, _
   ByVal cbCopy As Long)



' -----------------
' Code
' -----------------
```

```
Public Function GetStatusCode(status As Long) As String
  Dim msg As String

  Select Case status
    Case IP_SUCCESS:            msg = "ip success"
    Case IP_BUF_TOO_SMALL:      msg = "ip buf too_small"
    Case IP_DEST_NET_UNREACHABLE:  msg = "ip dest net unreachable"
    Case IP_DEST_HOST_UNREACHABLE: msg = "ip dest host unreachable"
    Case IP_DEST_PROT_UNREACHABLE: msg = "ip dest prot unreachable"
    Case IP_DEST_PORT_UNREACHABLE: msg = "ip dest port unreachable"
    Case IP_NO_RESOURCES:       msg = "ip no resources"
    Case IP_BAD_OPTION:         msg = "ip bad option"
    Case IP_HW_ERROR:           msg = "ip hw_error"
    Case IP_PACKET_TOO_BIG:     msg = "ip packet too_big"
    Case IP_REQ_TIMED_OUT:      msg = "ip req timed out"
    Case IP_BAD_REQ:            msg = "ip bad req"
    Case IP_BAD_ROUTE:          msg = "ip bad route"
    Case IP_TTL_EXPIRED_TRANSIT:   msg = "ip ttl expired transit"
    Case IP_TTL_EXPIRED_REASSEM:   msg = "ip ttl expired reassem"
    Case IP_PARAM_PROBLEM:      msg = "ip param_problem"
    Case IP_SOURCE_QUENCH:      msg = "ip source quench"
    Case IP_OPTION_TOO_BIG:     msg = "ip option too_big"
    Case IP_BAD_DESTINATION:    msg = "ip bad destination"
    Case IP_ADDR_DELETED:       msg = "ip addr deleted"
    Case IP_SPEC_MTU_CHANGE:    msg = "ip spec mtu change"
    Case IP_MTU_CHANGE:         msg = "ip mtu_change"
    Case IP_UNLOAD:             msg = "ip unload"
    Case IP_ADDR_ADDED:         msg = "ip addr added"
    Case IP_GENERAL_FAILURE:    msg = "ip general failure"
    Case IP_PENDING:            msg = "ip pending"
    Case PING_TIMEOUT:          msg = "ping timeout"
    Case Else:                  msg = "unknown  msg returned"
  End Select

  GetStatusCode = CStr(status) & "   [ " & msg & " ]"
End Function

' ================================================================ '

Public Function HiByte(ByVal wParam As Integer)
   HiByte = wParam \ &H100 And &HFF&
End Function

' ================================================================ '
```

```
Public Function LoByte(ByVal wParam As Integer)
   LoByte = wParam And &HFF&
End Function

' ================================================================== '

Public Function Ping(szAddress As String, ECHO As ICMP_ECHO_REPLY) As Long
  Dim hPort As Long
  Dim dwAddress As Long
  Dim sDataToSend As String
  Dim iOpt As Long

  sDataToSend = "Echo This"
  dwAddress = AddressStringToLong(szAddress)

  Call SocketsInitialize
  hPort = IcmpCreateFile()

  If IcmpSendEcho(hPort, _
               dwAddress, _
               sDataToSend, _
               Len(sDataToSend), _
               0, _
               ECHO, _
               Len(ECHO), _
               PING_TIMEOUT) Then

      'the ping succeeded,
      '.Status will be 0
      '.RoundTripTime is the time in ms for
      '             the ping to complete,
      '.Data is the data returned (NULL terminated)
      '.Address is the Ip address that actually replied
      '.DataSize is the size of the string in .Data
       Ping = ECHO.RoundTripTime
  Else: Ping = ECHO.status * -1
  End If

  Call IcmpCloseHandle(hPort)
  Call SocketsCleanup

End Function
```

```
' ================================================================ '

Function AddressStringToLong(ByVal tmp As String) As Long

   Dim i As Integer
   Dim parts(1 To 4) As String

   i = 0

  'we have to extract each part of the
  '123.456.789.123 string, delimited by
  'a period
   While InStr(tmp, ".") > 0
      i = i + 1
      parts(i) = Mid(tmp, 1, InStr(tmp, ".") - 1)
      tmp = Mid(tmp, InStr(tmp, ".") + 1)
   Wend

   i = i + 1
   parts(i) = tmp

   If i <> 4 Then
      AddressStringToLong = 0
      Exit Function
   End If

  'build the long value out of the
  'hex of the extracted strings
   AddressStringToLong = Val("&H" & Right("00" & Hex(parts(4)), 2) & _
                         Right("00" & Hex(parts(3)), 2) & _
                         Right("00" & Hex(parts(2)), 2) & _
                         Right("00" & Hex(parts(1)), 2))

End Function

' ================================================================ '

Public Function SocketsCleanup() As Boolean

    Dim X As Long

    X = WSACleanup()

    If X <> 0 Then
```

```
        MsgBox "Windows Sockets error " & Trim$(Str$(X)) & _
              " occurred in Cleanup.", vbExclamation
        SocketsCleanup = False
    Else
        SocketsCleanup = True
    End If

End Function

' ==================================================================== '

Public Function SocketsInitialize() As Boolean
    Dim WSAD As WSADATA
    Dim X As Integer
    Dim szLoByte As String, szHiByte As String, szBuf As String

    X = WSAStartup(WS_VERSION_REQD, WSAD)

    If X <> 0 Then
        MsgBox "Windows Sockets for 32 bit Windows " & _
              "environments is not successfully responding."
        SocketsInitialize = False
        Exit Function
    End If

    If LoByte(WSAD.wVersion) < WS_VERSION_MAJOR Or _
       (LoByte(WSAD.wVersion) = WS_VERSION_MAJOR And _
        HiByte(WSAD.wVersion) < WS_VERSION_MINOR) Then

        szHiByte = Trim$(Str$(HiByte(WSAD.wVersion)))
        szLoByte = Trim$(Str$(LoByte(WSAD.wVersion)))
        szBuf = "Windows Sockets Version " & szLoByte & "." & szHiByte
        szBuf = szBuf & " is not supported by Windows " & _
                        "Sockets for 32 bit Windows environments."
        MsgBox szBuf, vbExclamation
        SocketsInitialize = False
        Exit Function

    End If

    If WSAD.wMaxSockets < MIN_SOCKETS_REQD Then
        szBuf = "This application requires a minimum of " & _
                Trim$(Str$(MIN_SOCKETS_REQD)) & " supported sockets."
        MsgBox szBuf, vbExclamation
```

```
    SocketsInitialize = False
    Exit Function
  End If

  SocketsInitialize = True
End Function
```

Laufwerks-Typ auslesen

Die folgende Funktion meldet den Laufwerkstyp. Der Aufruf kann innerhalb einer Zelle in der Form =LaufwerkTyp("G:\") erfolgen.

```
Declare Function GetDriveType Lib "kernel32" Alias _
"GetDriveTypeA" (ByVal nDrive As String) As Long

Function LaufwerkTyp(strLaufwerk As String) As String
  Dim lngRWert As Long

  lngRWert = GetDriveType(strLaufwerk)
  Select Case lngRWert
  Case 2
    LaufwerkTyp = "Diskette/Wechselplatte"
  Case 3
    LaufwerkTyp = "Festplatte"
  Case 4
    LaufwerkTyp = "Netzlaufwerk"
  Case 5
    LaufwerkTyp = "CD-ROM"
  Case 6
    LaufwerkTyp = "RAM-Disk"
  Case Else
    LaufwerkTyp = ""
  End Select
End Function
```

Die folgende Prozedur setzt auf die Funktion LaufwerkTyp() auf und gibt alle vohandenen Laufwerke in eine Messagebox aus:

```
Sub LaufwerkeAuflisten()
  Dim intI As Integer
  Dim strLaufwerk As String
  Dim strTyp As String

  For intI = 65 To 90
    strLaufwerk = Chr$(intI) & ":"
    strTyp = LaufwerkTyp(strLaufwerk)
    If strTyp > "" Then
      MsgBox strLaufwerk & " = " & strTyp
    End If
  Next
End Sub
```

FUNKTION 2

```
'-------------------------------------------------------
' FUNCTION: GetDriveType
' Determine whether a disk is fixed, removable, etc. by
' calling Windows GetDriveType()
'-------------------------------------------------------
'
Function GetDriveType(ByVal intDriveNum As Integer) As Integer
   '
   ' This function expects an integer drive number in Win16 or a string in Win32
   '
   Dim strDriveName As String

   strDriveName = Chr$(Asc("A") + intDriveNum) & gstrSEP_DRIVE & gstrSEP_DIR
   GetDriveType = CInt(GetDriveType32(strDriveName))
End Function
```

Laufwerk und Pfad in UNC-Pfad umwandeln

```
Option Explicit

Declare Function WNetGetConnection& Lib "mpr.dll" Alias _
    "WNetGetConnectionA" (ByVal lpszLocalName As String, _
    ByVal lpszRemoteName As String, cbRemoteName As Long)

Function PfadNachUnc(ByVal Pfadname As String) As String
Dim dummy, UncLaufwerk$, Laufwerk$, Pfad$
On Error GoTo Fehlerbehandlung
    Laufwerk = Left(Pfadname, 2)
    Pfad = Right(Pfadname, Len(Pfadname) - 2)
    If InStr(1, Laufwerk, ":") = 2 Then
        UncLaufwerk = String(1001, 0)
        dummy = WNetGetConnection&(Laufwerk, _
            UncLaufwerk, 1000)
        If dummy <> 0 Then UncLaufwerk = Pfadname: GoTo _
            Fehlerbehandlung
        UncLaufwerk = Left(UncLaufwerk, InStr(1, UncLaufwerk, _
            Chr(0)) - 1) & Pfad
    Else
        UncLaufwerk = Pfadname
    End If
Fehlerbehandlung:
PfadNachUnc = UncLaufwerk
End Function
```

```
Sub Test()
Dim Pfad$
    Pfad = "n:\test"
    Pfad = PfadNachUnc(Pfad)
End Sub
```

Laufwerksbuchstaben festlegen

```
'  Title: Sets the volume label for a drive
'
' ----------------------------------------------------------------------------

' ---------------------------
' Constants & API Declarations
' ---------------------------

Declare Function SetVolumeLabel Lib "kernel32" Alias "SetVolumeLabelA" (ByVal lpRootPathName As String, ByVal lpVolumeName As String) As Long


' ----------
' Function
' ----------

Public Function SetLabel(RootName As String, NewLabel As String)
      If RootName = "" Then
         Exit Function
      End If

      Call SetVolumeLabel(RootName,NewLabel)
End Function
```

Netzlaufwerke und Shares feststellen

```
Option Explicit
Option Base 1
Private Declare Function WNetGetConnection _
Lib "User" (ByVal LocalName As String, _
ByVal RemoteName As String, _
RetLength As Integer) As Integer
' 32 Bit version of above
Private Declare Function WNetGetConnectionA _
Lib "MPR.DLL" (ByVal LocalName As String, _
```

```
ByVal RemoteName As String, _
RetLength As Long) As Long
Sub Netzlaufwerke()
Dim strServerNames() As String
Dim intNumServers As Integer
Dim i As Integer
' Initialise string array
ReDim strServerNames(2, 23) As String
' Execute function
intNumServers = pfGetConnection(strServerNames)
' Shrink array to get rid of empty elements
ReDim Preserve strServerNames(2, intNumServers)
' Display results
For i = 1 To intNumServers
MsgBox strServerNames(1, i) & " = " & _
strServerNames(2, i)
Next
End Sub
Function pfGetConnection(ByRef strServers() As String) _
As Integer
Dim lngMaxLen As Long
Dim strLocalName As String
Dim strRemoteName As String
Dim intCount As Integer, lngGetConRet As Long
'Loop through drive letters D to Z
For intCount = 65 To 90
' Length of fixed string pointer
' for API call
lngMaxLen = 255
' Drive letter with trailing colon
strLocalName = Chr(intCount) & ":"
' initialise string pointer
strRemoteName = Space(lngMaxLen)
' Feed drive letter into API function
If Not Application.OperatingSystem Like _
"*32*" Then
' 16 bit version
lngGetConRet = WNetGetConnection _
(strLocalName, strRemoteName, _
CInt(lngMaxLen))
Else
' 32 bit version
lngGetConRet = WNetGetConnectionA _
(strLocalName, strRemoteName, _
lngMaxLen)
End If
' Strip out terminating Null character
' and trailing spaces
strRemoteName = Left(strRemoteName, _
InStr(strRemoteName, Chr(0)) - 1)
If Not Len(strRemoteName) = 0 Then
' Load drive letter into referenced array
pfGetConnection = pfGetConnection + 1
```

```
strServers(1, pfGetConnection) = strLocalName
strServers(2, pfGetConnection) = strRemoteName
End If
Next intCount
End Function
```

NUM-Lock ein/ausschalten

```
Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal
bScan As Byte, ByVal dwFlags As Long, _
ByVal dwExtraInfo As Long)
Public Const VK_NUMLOCK = &H90
Sub Num_Lock_On()
keybd_event VK_NUMLOCK, 1, 0, 0
End Sub
Sub Num_Lock_Off()
keybd_event VK_NUMLOCK, 0, 0, 0
End Sub
```

Prüfen ob Rechner mit Internet verbunden ist

```
' ------------------------------------------------------------------------
'  Title: Determine if a computer is connected to the Internet
' ------------------------------------------------------------------------

' ---------------------------
' Constants & API Declarations
' ---------------------------

Public Declare Function RasEnumConnections Lib "RasApi32.dll" Alias "RasEnumConnectionsA" (lpRasCon As Any, lpcb As Long, lpcConnections As Long) As
Long
Public Declare Function RasGetConnectStatus Lib "RasApi32.dll" Alias "RasGetConnectStatusA" (ByVal hRasCon As Long, lpStatus As Any) As Long

Public Const RAS95_MaxEntryName = 256
Public Const RAS95_MaxDeviceType = 16
Public Const RAS95_MaxDeviceName = 32

Public Type RASCONN95
   dwSize As Long
   hRasCon As Long
   szEntryName(RAS95_MaxEntryName) As Byte
```

```
    szDeviceType(RAS95_MaxDeviceType) As Byte
    szDeviceName(RAS95_MaxDeviceName) As Byte
End Type
'
Public Type RASCONNSTATUS95
    dwSize As Long
    RasConnState As Long
    dwError As Long
    szDeviceType(RAS95_MaxDeviceType) As Byte
    szDeviceName(RAS95_MaxDeviceName) As Byte
End Type


' --------
' Code
' --------

'A call to the function IsConnected returns true if the computer has established a connection to the internet.

Public Function IsConnected() As Boolean
    Dim TRasCon(255) As RASCONN95
    Dim lg As Long
    Dim lpcon As Long
    Dim RetVal As Long
    Dim Tstatus As RASCONNSTATUS95

    TRasCon(0).dwSize = 412
    lg = 256 * TRasCon(0).dwSize

    RetVal = RasEnumConnections(TRasCon(0), lg, lpcon)
    If RetVal <> 0 Then
        MsgBox "Error!"
        Exit Function
    End If

    Tstatus.dwSize = 160
    RetVal = RasGetConnectStatus(TRasCon(0).hRasCon, Tstatus)
    If Tstatus.RasConnState = &H2000 Then
      IsConnected = True
    Else
        IsConnected = False
    End If
End Function
```

--- VERSION 2 ---

```
'  Title: Is Connected, Online
' ----------------------------------------------------------------------------

Option Explicit

' ---------------------------
' Constants & API Declarations
' ---------------------------
Public Declare Function InternetGetConnectedState _Lib "wininet.dll"(ByRef lpdwFlags As Long, _
    ByVal dwReserved As Long) As Long

    Public Const INTERNET_CONNECTION_MODEM_BUSY As Long = &H8
    Public Const INTERNET_RAS_INSTALLED As Long = &H10
    Public Const INTERNET_CONNECTION_OFFLINE As Long = &H20
    Public Const INTERNET_CONNECTION_CONFIGURED As Long = &H40

' ---------------
' Function
' ---------------
Public Function IsNetConnectOnline() As Boolean
    IsNetConnectOnline = InternetGetConnectedState(0&, 0&)
End Function
```

--- Version 3 ---

```
' ----------------------------------------------------------------------------
'  Title: Various Methods to Detect or Test Internet Connection
'
' ----------------------------------------------------------------------------

' ---------------
' Put in Module
' ---------------

Option Explicit
```

```
Public Declare Function InternetGetConnectedState _
    Lib "wininet.dll" (ByRef lpdwFlags As Long, _
    ByVal dwReserved As Long) As Long

    ' Local system uses a modem to connect to the Internet
    Public Const INTERNET_CONNECTION_MODEM As Long = &H1

    ' Local system uses a LAN to connect to the Internet.
    Public Const INTERNET_CONNECTION_LAN As Long = &H2

    ' Local system uses a proxy server to connect to the Internet.
    Public Const INTERNET_CONNECTION_PROXY As Long = &H4

    ' No longer used
    Public Const INTERNET_CONNECTION_MODEM_BUSY As Long = &H8
    Public Const INTERNET_RAS_INSTALLED As Long = &H10
    Public Const INTERNET_CONNECTION_OFFLINE As Long = &H20
    Public Const INTERNET_CONNECTION_CONFIGURED As Long = &H40


' ------------------------------------------
' InternetGetConnectedState wrapper functions
' ------------------------------------------

Public Function IsNetConnectViaLAN() As Boolean
    Dim dwflags As Long
    'pass an empty varialbe into which the API will
    'return the flags associated with the connection
    Call InternetGetConnectedState(dwflags, 0&)

    'return True if the flags indicate a LAN connection
    IsNetConnectViaLAN = dwflags And INTERNET_CONNECTION_LAN
End Function


Public Function IsNetConnectViaModem() As Boolean
    Dim dwflags As Long
    'pass an empty varialbe into which the API will
    'return the flags associated with the connection
    Call InternetGetConnectedState(dwflags, 0&)

    'return True if the flags indicate a modem connection
    IsNetConnectViaModem = dwflags And INTERNET_CONNECTION_MODEM
End Function
```

```
Public Function IsNetConnectViaProxy() As Boolean
    Dim dwflags As Long
    'pass an empty varialbe into which the API will
    'return the flags associated with the connection
    Call InternetGetConnectedState(dwflags, 0&)

    'return True if the flags indicate a proxy connection
    IsNetConnectViaProxy = dwflags And INTERNET_CONNECTION_PROXY
End Function


Public Function IsNetConnectOnline() As Boolean
    'no flags needed here - the API returns True
    'if there is a connection of any type
    IsNetConnectOnline = InternetGetConnectedState(0&, 0&)
End Function


Public Function IsNetRASInstalled() As Boolean
    Dim dwflags As Long
    'pass an empty varialbe into which the API will
    'return the flags associated with the connection
    Call InternetGetConnectedState(dwflags, 0&)

    'return True if the falgs include RAS installed
    IsNetRASInstalled = dwflags And INTERNET_RAS_INSTALLED
End Function


Public Function GetNetConnectString() As String
    Dim dwflags As Long
    Dim msg As String

    'build a string for display
    If InternetGetConnectedState(dwflags, 0&) Then
        If dwflags And INTERNET_CONNECTION_CONFIGURED Then
            msg = msg & "You have a network connection configured." & vbCrLf
        End If

        If dwflags And INTERNET_CONNECTION_LAN Then
            msg = msg & "The local system connects to the Internet via a LAN"
        End If
```

```
        If dwflags And INTERNET_CONNECTION_PROXY Then
           msg = msg & ", and uses a proxy server. "
        Else: msg = msg & "."
        End If

        If dwflags And INTERNET_CONNECTION_MODEM Then
           msg = msg & "The local system uses a modem to connect to the Internet. "
        End If

        If dwflags And INTERNET_CONNECTION_OFFLINE Then
           msg = msg & "The connection is currently offline. "
        End If

        If dwflags And INTERNET_CONNECTION_MODEM_BUSY Then
           msg = msg & "The local system's modem is busy With a non-Internet connection. "
        End If

        If dwflags And INTERNET_RAS_INSTALLED Then
           msg = msg & "Remote Access Services are installed On this system."
        End If
     Else
        msg = "Not connected to the internet now."
     End If

     GetNetConnectString = msg
End Function
```

Rechner neu starten - herunterfahren

```
'  Title: Reboot the computer
'
' ----------------------------------------------------------------------------

' ---------------------------
' Constants & API Declarations
' ---------------------------

Private Const EWX_REBOOT As Long = 2
Private Declare Function ExitWindowsEx Lib "user32" (ByVal dwOptions As Long, ByVal dwReserved As Long) As Long
```

```
' ----------------------------
' Function
' ----------------------------

Sub Reboot_Computer()
   Dim lngResult As Long
   lngResult = ExitWindowsEx(EWX_REBOOT, 0&)
End Sub
```

' Title: Shut down System

```
'

' ------------------------------------------------------------------------------

Option Explicit

' ----------------------------
' Constants & API Declarations
' ----------------------------

Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As
Long, ByVal cy As Long, ByVal wFlags As Long) As Long
Private Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long, lpvParam As Any,
ByVal fuWinIni As Long) As Long
Private Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long) As Long

Private Const EWX_SHUTDOWN = 1

' ---------------
' Function
' ---------------

Private sub Shutdown()
      Dim ret As Integer
      Dim pOld As Boolean

      ret = SystemParametersInfo(97, False, pOld, 0)
End Sub
```

```
'  Title: Shut down the computer
'

' ----------------------------------------------------------------------------

' ----------------------------
' Constants & API Declarations
' ----------------------------

Private Const EWX_SHUTDOWN As Long = 1
Private Declare Function ExitWindowsEx Lib "user32" (ByVal dwOptions As Long, ByVal dwReserved As Long) As Long


' ---------
' Function
' ---------

Sub Shutdown_Computer()
    Dim lngResult As Long
    lngResult = ExitWindowsEx(EWX_SHUTDOWN, 0&)
End Sub
```

Systemverzeichnisse auslesen

Version ENVIRON

Bei der Verwendung von Makros kann es sinnvoll sein auf die vorhandenen Windows-Umgebungsvariablen zurückzugreifen.

So läßt sich zum Beispiel feststellen wie der bei Windows angemeldete User heisst. Wird ein Makro auf verschiedenen Rechnern ausgeführt könnte man damit zum Beispiel nachvollziehen wer welche Änderungen vorgenommen hat.

Oder es lassen sich diverse Pfade die vom System verwendet werden auslesen.

Hier ein paar Beispiele:

```
Desktoppfad = Environ("USERPROFILE") & "\Desktop"
```

Environ("USERNAME") = Name des angemeldeten Users
Environ("USERPROFILE") = Pfad des Nutzerprofils
Environ("ALLUSERSPROFILE") = Pfad für Benutzerprofile (ALLE)
Environ("APPDATA") = Pfad Anwendungsdaten
Environ("CommonProgramFiles") = Ordner für Gemeinsame Dateien
Environ("COMPUTERNAME") = Name des Computers
Environ("HOMEDRIVE") = Standardlaufwerk des Benutzers
Environ("OS") = Betriebssystemversion
Environ("Path") = Pfadangaben für Anwendungen
Environ("ProgramFiles") = Pfad zum Programme-Ordner
Environ("SystemDrive") = Laufwerk des Betriebssystems
Environ("SystemRoot") = Pfad des Betriebssystems
Environ("TEMP") = Pfad temporärer Ordner

Es gibt noch einige Umgebungsvariablen mehr. Um diese rauszufinden kann man statt der Bezeichnung auch eine Zahl zwischen 1 und 100 in die Klammern schreiben. Probiert habe ich die Zahlen 1 bis 100.

Environ(39) = USERNAME=Michel

Environ("USERNAME") = Michel

ANDERE VERSIONEN

Debug.Print objFolderItem.Path &" (x86)"

Set objShell = CreateObject("WScript.Shell")
strProgramFiles = objShell.ExpandEnvironmentStrings("%ProgramFiles%")
MsgBox strProgramFiles
strProgramFiles = objShell.ExpandEnvironmentStrings("%ProgramFiles(X86)%")
MsgBox strProgramFiles

```
Public Function GetProgramFilesFolder() As String

       Const PROGRAM_FILES = &H26&

       Dim objShell As Object

       Dim objFolder As Object

       Dim objFolderItem As Object


       Set objShell = CreateObject("Shell.Application")

       Set objFolder = objShell.Namespace(PROGRAM_FILES)

       Set objFolderItem = objFolder.Self


      Debug.Print objFolderItem.Path

End Function
```

Hier die Werte (ausgelesen mit nachfolgendem Code)

	A	B	C	D	E
1	ENVIRON(X)	Name	Value	iPos	
2	1	ALLUSERSPROFILE	C:\ProgramData	16	
3	2	APPDATA	C:\Users\Stefan\AppData\Roaming	8	
4	3	CommonProgramFiles	C:\Program Files (x86)\Common Files	19	
5	4	CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files	24	
6	5	CommonProgramW6432	C:\Program Files\Common Files	19	
7	6	COMPUTERNAME	STEFAN-HP	13	
8	7	ComSpec	C:\Windows\system32\cmd.exe	8	
9	8	FP_NO_HOST_CHECK	NO	17	
10	9	HOMEDRIVE	C:	10	
11	10	HOMEPATH	\Users\Stefan	9	
12	11	LOCALAPPDATA	C:\Users\Stefan\AppData\Local	13	
13	12	LOGONSERVER	\\STEFAN-HP	12	
14	13	NUMBER_OF_PROCESSORS	4	21	
15	14	OnlineServices	Online Services	15	
16	15	OS	Windows_NT	3	
17	16	Path	C:\Program Files (x86)\Microsoft Office\OFFICE11\;C:\Program Fil	5	
18	17	PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC	8	
19	18	PCBRAND	Pavilion	8	
20	19	Platform	MCD	9	
21	20	PROCESSOR_ARCHITECTURE	x86	23	
22	21	PROCESSOR_ARCHITEW6432	AMD64	23	
23	22	PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 42 Stepping 7, GenuineIntel	21	
24	23	PROCESSOR_LEVEL	6	16	
25	24	PROCESSOR_REVISION	2a07	19	
26	25	ProgramData	C:\ProgramData	12	
27	26	ProgramFiles	C:\Program Files (x86)	13	
28	27	ProgramFiles(x86)	C:\Program Files (x86)	18	
29	28	ProgramW6432	C:\Program Files	13	
30	29	PSModulePath	C:\Windows\system32\WindowsPowerShell\v1.0\Modules\	13	
31	30	PUBLIC	C:\Users\Public	7	
32	31	SESSIONNAME	Console	12	
33	32	SystemDrive	C:	12	
34	33	SystemRoot	C:\Windows	11	
35	34	TEMP	C:\Users\Stefan\AppData\Local\Temp	5	
36	35	TMP	C:\Users\Stefan\AppData\Local\Temp	4	
37	36	USERDOMAIN	Stefan-HP	11	
38	37	USERNAME	Stefan	9	
39	38	USERPROFILE	C:\Users\Stefan	12	
40	39	WecVersionForRosebud.CDC	2	25	
41	40	windir	C:\Windows	7	
42	41	windows_tracing_flags	3	22	
43	42	windows_tracing_logfile	C:\BVTBin\Tests\installpackage\csilogfile.log	24	
44					

hier der Code dazu

```
Sub Umgebungsvarriablen_auslesen()
Dim iIndex As Integer
Dim sResult As String
Dim sKey As String
Dim sValue As String
Dim iPos As Integer

' Environ() durchlaufen
' Beginnen mit 1
iIndex = 1
Do
' Umgebungsvariable auslesen
sResult = Environ(iIndex)

' Wird ein Leerstring zurückgegeben, so ist das
' Ende der Liste erreicht
If sResult <> "" Then

' = suchen
iPos = InStr(sResult, "=")
sKey = Left$(sResult, iPos - 1)
sValue = Mid$(sResult, iPos + 1)


'der Tabelle hinzufügen
Cells(iIndex, 1).Value = iIndex
Cells(iIndex, 2).Value = sKey
Cells(iIndex, 3).Value = sValue
Cells(iIndex, 4).Value = iPos
End If

' Index um 1 erhöhen
iIndex = iIndex + 1
Loop Until sResult = ""
End Sub
```

Gruß Nicklos

tom_r - 08. Aug 2005, 08:28 hat folgendes geschrieben:

Moin,

hier noch eine Version, die die Umgebungsvariablen in einem Tabellenblatt auflistet (auch irgendwo gefunden)

Code:

```
Sub Umgebungsvarriablen auslesen()
Dim iIndex As Integer
Dim sResult As String
Dim sKey As String
Dim sValue As String
```

```
Dim iPos As Integer

' Environ() durchlaufen
' Beginnen mit 1
iIndex = 1
Do
' Umgebungsvariable auslesen
sResult = Environ(iIndex)

' Wird ein Leerstring zurückgegeben, so ist das
' Ende der Liste erreicht
If sResult <> "" Then

' = suchen
iPos = InStr(sResult, "=")
sKey = Left$(sResult, iPos - 1)
sValue = Mid$(sResult, iPos + 1)


'der Tabelle hinzufügen
Cells(iIndex, 1).Value = sKey
Cells(iIndex, 2).Value = sValue
Cells(iIndex, 3).Value = iPos
End If

' Index um 1 erhöhen
iIndex = iIndex + 1
Loop Until sResult = ""
End Sub
```

1. Version mit fixem Verzeichnis hinterlegt in Funktion

```
' Options For specical folders
 '    AllUsersDesktop
 '    AllUsersStartMenu
 '    AllUsersPrograms
 '    AllUsersStartup
 '    Desktop
 '    Favorites
 '    Fonts
 '    MyDocuments
 '    NetHood
 '    PrintHood
 '    Programs
 '    Recent
 '    SendTo
 '    StartMenu
 '    Startup
 '     Templates
 Function SpecialFolderPath() As String
```

```
    Dim objWSHShell As Object
    Dim strSpecialFolderPath

     'On  Error GoTo ErrorHandler
     ' Create a shell object
    Set objWSHShell = CreateObject("WScript.Shell")
     '  Find out the path to the passed special folder,
     '  just change the "Desktop" for one of the other options
    SpecialFolderPath = objWSHShell.SpecialFolders("Desktop")
     ' Clean up
    Set objWSHShell = Nothing
    Exit Function
ErrorHandler:

     MsgBox "Error finding " & strSpecialFolder, vbCritical + vbOKOnly, "Error"
End Function
```

2. Version mit Übergabe des Ordners an die Funktion

```
Function SpecialFolderPath(ORDNER) As String

' Options For specical folders
 '    AllUsersDesktop
 '    AllUsersStartMenu
 '    AllUsersPrograms
 '    AllUsersStartup
 '    Desktop
 '    Favorites
 '    Fonts
 '    MyDocuments
 '    NetHood
 '    PrintHood
 '    Programs
 '    Recent
 '    SendTo
 '    StartMenu
 '    Startup
 '    Templates


    Dim objWSHShell As Object
    Dim strSpecialFolderPath

     'On  Error GoTo ErrorHandler
```

```
    ' Create a shell object
    Set objWSHShell = CreateObject("WScript.Shell")
    '  Find out the path to the passed special folder,

    ' ************************************************************
    '  just change the "Desktop" for one of the other options
      SpecialFolderPath = objWSHShell.SpecialFolders(ORDNER)
    ' ************************************************************

    ' Clean up
    Set objWSHShell = Nothing
    Exit Function
ErrorHandler:

    MsgBox "Error finding " & strSpecialFolder, vbCritical + vbOKOnly, "Error"
End Function


' ****************************************
' 1. BESONDERE VERZEICHNISSE
' ****************************************

Sub STARTUP_ORDNER()

MsgBox SpecialFolderPath("Desktop")
MsgBox SpecialFolderPath("Startup")

End Sub
```

3. Version

```
Private Const CSIDL_ADMINTOOLS          As Long = &H30   '{user}\Start Menu _                                    '\Programs\Administrative Tools
Private Const CSIDL_COMMON_ADMINTOOLS   As Long = &H2F   '(all users)\Start Menu\Programs\Administrative Tools
Private Const CSIDL_APPDATA             As Long = &H1A   '{user}\Application Data
Private Const CSIDL_COMMON_APPDATA      As Long = &H23   '(all users)\Application Data
Private Const CSIDL_COMMON_DOCUMENTS    As Long = &H2E   '(all users)\Documents
Private Const CSIDL_COOKIES             As Long = &H21
Private Const CSIDL_HISTORY             As Long = &H22
Private Const CSIDL_INTERNET_CACHE      As Long = &H20   'Internet Cache folder
Private Const CSIDL_LOCAL_APPDATA       As Long = &H1C   '{user}\Local Settings\Application Data (non roaming)
```

```
Private Const CSIDL_MYPICTURES          As Long = &H27   'C:\Program Files\My Pictures
Private Const CSIDL_PERSONAL            As Long = &H5    'My Documents
Private Const CSIDL_PROGRAM_FILES       As Long = &H26   'Program Files folder
Private Const CSIDL_PROGRAM_FILES_COMMON As Long = &H2B  'Program Files\Common
Private Const CSIDL_SYSTEM              As Long = &H25   'system folder
Private Const CSIDL_WINDOWS             As Long = &H24   'Windows directory or SYSROOT()
Private Const CSIDL_FLAG_CREATE = &H8000&               'combine with CSIDL_ value to force
Private Const MAX_PATH = 260
Private Const SHGFP_TYPE_CURRENT = &H0
Private Const S_OK = 0

Private Declare Function SHGetFolderPath Lib "shfolder" _
       Alias "SHGetFolderPathA" (ByVal hwndOwner As Long, _
       ByVal nFolder As Long, ByVal hToken As Long, _
       ByVal dwFlags As Long, ByVal pszPath As String) As Long

Function ActualApplicationDataFolder() As String

   Dim lngCSIDL   As Long
   Dim strPath    As String
   Dim lngReturn  As Long

   lngCSIDL = CSIDL_APPDATA
   strPath = String(MAX_PATH, 0)

   'Get the folder's path.
   'If the "Create" flag is used, the folder will be created if it does not exist.

   lngReturn = SHGetFolderPath(0, lngCSIDL, 0, SHGFP_TYPE_CURRENT, strPath)
'    lngReturn = SHGetFolderPath(0, lngCSIDL Or CSIDL_FLAG_CREATE, 0, SHGFP_TYPE_CURRENT, strPath)

   Select Case lngReturn
   Case S_OK
      ActualApplicationDataFolder = Left$(strPath, InStr(1, strPath, Chr(0)) - 1)

   Case Else
      ActualApplicationDataFolder = ""

   End Select

End Function

Function ActualProgramFilesFolder() As String
```

```
    Dim lngCSIDL   As Long
    Dim strPath    As String
    Dim lngReturn  As Long

    lngCSIDL = CSIDL_PROGRAM_FILES
    strPath = String(MAX_PATH, 0)

    'Get the folder's path.
    'If the "Create" flag is used, the folder will be created if it does not exist.

    lngReturn = SHGetFolderPath(0, lngCSIDL, 0, SHGFP_TYPE_CURRENT, strPath)
'    lngReturn = SHGetFolderPath(0, lngCSIDL Or CSIDL_FLAG_CREATE, 0, SHGFP_TYPE_CURRENT, strPath)

    Select Case lngReturn
    Case S_OK
       ActualProgramFilesFolder = Left$(strPath, InStr(1, strPath, Chr(0)) - 1)

    Case Else
       ActualProgramFilesFolder = ""

    End Select

End Function

Sub SPEZIAL_ORDNER2()

MsgBox ActualApplicationDataFolder ' hier ruft man die obigen Funktionen auf


End Sub
```

Umgebungsvariablen direkt im Explorer / Windowsscript

Windows XP	Windows Vista/7	Bemerkung
umente und Einstellungen\All Users	C:\ProgramData	
umente und Einstellungen\{username} ndungsdaten	C:\Users\{username}\AppData\Roaming	
ıtername}		
gramme\Gemeinsame Dateien	C:\Program Files\Common Files	
gramme (x86)\Gemeinsame Dateien	C:\Program Files (x86)\Common Files	
emRoot%\System32\cmd.exe		Pfad zum Kommandozeilen-Interpreter %SystemRoot% entsprechend expandiert auch DOS
	C:\	
nente und Einstellungen\{username}	\Users\{username}	
	C:\Users\{username}\AppData\Local	
ain_logon_server}		
ws_NT	Windows_NT	
emRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem		PATH %SystemRoot% entsprechend expandiert auch DOS
.EXE, .BAT, .CMD, .VBS, .VBE, .JS, .WSF,	.com, .exe, .bat, .cmd, .vbs, .vbe, .js, .jse, .wsf, .wsh, .msc	
gramme	C:\Program Files	
gramme (x86)	C:\Program Files (x86)	
		auch DOS
	C:\	
dows (Windows 2000 und davor: INT)	C:\Windows	
dows\TEMP		TEMP
omain}		
ame}		Der Name des aktuellen Benutzers
emDrive%\Dokumente und lungen\{username}	C:\Users\{username}	
dows	C:\Windows	Historische Form (16-Bit-Windows), zur Kompatibilität immer identisch mit %SystemRoot% sein.
	C:\Users\Public	
	%SystemRoot%\system32	

Systemvariablen auslesen

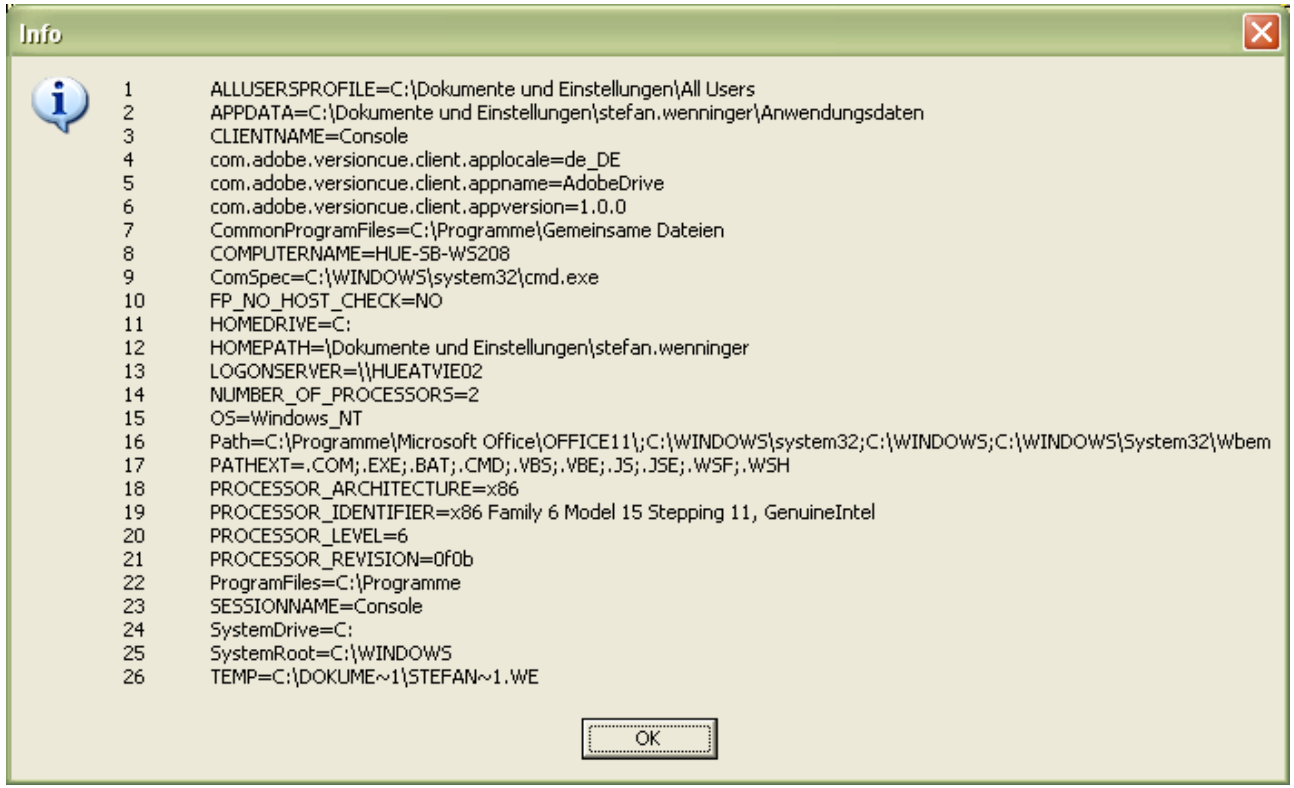
Es gibt ja auch noch den anderen Weg - siehe BESONDERE VERZEICHNISSE AUSLESEN

Aber auch mit nachfolgendem Code erhält man viele wichtige Infos

Wichtig: die Environ-Variable ist nicht immer gleich befüllt - also nicht immer sind die Infos an der selben Stelle - daher wenn man eine bestimmte sucht, dann durch alle 100 durchgehen und die ersten Buchstaben der Environ-Variable auf den gewünschten Wert untersuchen

```
Sub SYSTEMVARIABLEN()
Dim I As Integer
Dim strMTEXT As String
For I = 1 To 100
   If Environ(I) <> "" Then

   strMTEXT = strMTEXT & I & vbTab & Environ(I) & vbCr
   End If
Next I
MsgBox strMTEXT, vbInformation, "Info"
End Sub
```

Das kann dann so aussehen:



VERSION 2

```
' ------------------------------------------------------------------------
'
'  Title: Retrieve various System and Windows Info
'


' ------------------------------------------------------------------------

Option Explicit

'Windows System Information Constants
```

```
Private Const SQL_SUCCESS As Long = 0
Private Const SQL_FETCH_NEXT = 1
Private Const SQL_FETCH_FIRST_SYSTEM = 32

Private Const SECS_PER_DAY As Long = 86000
Private Const MINS_PER_DAY As Long = 1400

'Enumerated Types
Public Enum ControlPanelTypes
   cplSystem = 1
   cplInternet = 2
   cplModem = 3
   cplSoftware = 4
   cplHardware = 5
   cplSounds = 6
   cplNetwork = 7
   cplMouse = 8
   cplKeyboard = 9
   cplDateTime = 10
   cplRegional = 11
   cplPassword = 12
   cplDisplay = 13
End Enum

Public Enum DIR_TYPE
   dirWINDOWS
   dirSYSTEM
   dirTEMP
End Enum

Public Enum DRIVE_TYPE
   DRIVE_DOESNT_EXIST = 1
   DRIVE_REMOVABLE = 2
   DRIVE_FIXED = 3
   DRIVE_REMOTE = 4
   DRIVE_CDROM = 5
   DRIVE_RAMDISK = 6
End Enum

Public Enum enStockIcons
   IDI_APPLICATION = 32512&
   IDI_ASTERISK = 32516&
   IDI_EXCLAMATION = 32515&
   IDI_HAND = 32513&   'This is the STOP icon
```

```
    IDI_QUESTION = 32514&
End Enum

Public Enum OS_VERSION
    OS_WINDOWS_UNKNOWN
    OS_WINDOWS_3X
    OS_WINDOWS_95
    OS_WINDOWS_98
    OS_WINDOWS_NT3X
    OS_WINDOWS_NT40
    OS_WINDOWS_2000
End Enum

Private Type OSVERSIONINFOEX
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type

Private Type MEMORYSTATUS
    dwLength As Long
    dwMemoryLoad As Long
    dwTotalPhys As Long
    dwAvailPhys As Long
    dwTotalPageFile As Long
    dwAvailPageFile As Long
    dwTotalVirtual As Long
    dwAvailVirtual As Long
End Type

Public Enum ShutdownType
    Logoff = 0
    ByeBye = 1
    Reboot = 2
End Enum

Private Const VER_PLATFORM_WIN32s = 0
Private Const VER_PLATFORM_WIN32_WINDOWS = 1
Private Const VER_PLATFORM_WIN32_NT = 2

Private Const BITSPIXEL = 12
```

```
Private Const PLANES = 14

' API Declarations for the Locale Methods
Private Declare Sub GlobalMemoryStatus Lib "kernel32" (lpBuffer As MEMORYSTATUS)
'API Functions
Private Declare Function DrawIconApi Lib "user32" Alias "DrawIcon" (ByVal HDC As Long, ByVal x As Long, ByVal Y As Long, ByVal hIcon As Long) As Long
Private Declare Function ExitWindowsEx Lib "user32" (ByVal dwOptions As Long, ByVal dwReserved As Long) As Long
Private Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long) As Long
Private Declare Function GetComputerName Lib "kernel32" Alias "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Private Declare Function GetDeviceCaps Lib "gdi32" (ByVal HDC As Long, ByVal nIndex As Long) As Long
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA" (ByVal lpRootPathName As String, lpSectorsPerCluster As Long,
lpBytesPerSector As Long, lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters As Long) As Long
Private Declare Function GetDriveType Lib "kernel32" Alias "GetDriveTypeA" (ByVal nDrive As String) As Long
Private Declare Function GetProcAddress Lib "kernel32" (ByVal hModule As Long, ByVal lpProcName As String) As Long
Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Declare Function GetTickCount Lib "kernel32.dll" () As Long
Private Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
Private Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Private Declare Function GetVersionEx Lib "kernel32" Alias "GetVersionExA" (lpVersionInformation As OSVERSIONINFOEX) As Long
Private Declare Function GetVolumeInformation Lib "kernel32" Alias "GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal lpVolumeNameBuffer
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
Private Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Declare Function InternetGetConnectedState Lib "wininet.dll" (ByRef lpSFlags As Long, ByVal dwReserved As Long) As Long
Private Declare Function LoadIconApi Lib "user32" Alias "LoadIconA" (ByVal hInstance As Long, ByVal lpIconName As Long) As Long
Private Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal lpLibFileName As String) As Long
Private Declare Function ShellAbout Lib "shell32.dll" Alias "ShellAboutA" (ByVal hWnd As Long, ByVal szApp As String, ByVal szOtherStuff As String, ByVal
hIcon As Long) As Long
Private Declare Function SQLAllocEnv Lib "odbc32.dll" (phenv As Long) As Integer
Private Declare Function SQLDataSources Lib "odbc32.dll" (ByVal hEnv As Long, ByVal fDirection As Integer, ByVal szDSN$, ByVal cbDSNMax%, pcbDSN As
Integer, ByVal szDescription As String, ByVal cbDescriptionMax As Integer, pcbDescription As Integer) As Integer
Private Declare Function SQLFreeEnv Lib "odbc32.dll" (ByVal hEnv As Long) As Integer
Private Declare Function waveOutGetNumDevs Lib "winmm.dll" () As Long

Private pUdtOSVersion As OSVERSIONINFOEX
Private pUdtMemStatus As MEMORYSTATUS

Private plMajorVersion  As Long
Private plMinorVersion As Long
Private plPlatformID As Long

Private psComputerName As String
Private plLastDllError As Long
```

```
'-----------------
'Public Properties
'-----------------
Public Property Get Name() As String
    Dim sBuffer As String
    Dim lAns As Long

    plLastDllError = 0
    sBuffer = Space$(255)
    lAns = GetComputerName(sBuffer, 255)
    If lAns <> 0 Then
        'read from beginning of string to null-terminator
        Name = Left$(sBuffer, InStr(sBuffer, Chr(0)) - 1)
    Else
        plLastDllError = Err.LastDllError
    End If
End Property

Public Property Get CurrentUser() As String
    Dim l As Long
    Dim sUser As String

    plLastDllError = 0
    sUser = Space(255)
    l = GetUserName(sUser, 255)
    'strip null terminator
    If l <> 0 Then
        CurrentUser = Left(sUser, InStr(sUser, Chr(0)) - 1)
    Else
        plLastDllError = Err.LastDllError
    End If
End Property

Public Property Get MaxScreenColors() As Double
    'Returns the maximum number of colors supported
    'by the system - e.g.,  256, 16,777,216
    Dim lngBits As Long
    Dim lngPlanes As Long
    Dim lwndHandle As Long
    Dim dblAns As Double
    plLastDllError = 0
    lwndHandle = Form1.HDC
    'bits per pixel
    lngBits = GetDeviceCaps(lwndHandle, BITSPIXEL)
```

```
    'number of color planes
    lngPlanes = GetDeviceCaps(lwndHandle, PLANES)
    'maximum colors available
    MaxScreenColors = (2 ^ (lngBits * lngPlanes))
    plLastDllError = Err.LastDllError
End Property

Public Property Get OSVersion() As OS_VERSION
    On Error GoTo ErrorHandler

    plLastDllError = 0
    Select Case plMajorVersion
        Case 5: OSVersion = OS_WINDOWS_2000 'UNTESTED
        Case 4
            If plPlatformID = VER_PLATFORM_WIN32_NT Then
                OSVersion = OS_WINDOWS_NT40
            Else
                OSVersion = IIf(plMinorVersion = 0, OS_WINDOWS_95, OS_WINDOWS_98)
            End If
        Case 3
            If plPlatformID = VER_PLATFORM_WIN32s Then
                OSVersion = OS_WINDOWS_3X
            ElseIf plPlatformID = VER_PLATFORM_WIN32_NT Then
                OSVersion = OS_WINDOWS_NT40
            End If
        Case Else:  OSVersion = OS_WINDOWS_UNKNOWN
    End Select
Exit Property
ErrorHandler:
    OSVersion = OS_WINDOWS_UNKNOWN
    plLastDllError = Err.LastDllError
End Property

Public Property Get ScreenPixelWidth() As Integer
    plLastDllError = 0
    ScreenPixelWidth = Screen.Width \ Screen.TwipsPerPixelX
End Property

Public Property Get ScreenPixelHeight() As Integer
    plLastDllError = 0
    ScreenPixelHeight = Screen.Height \ Screen.TwipsPerPixelY
End Property

Public Property Get ScreenResolution() As String
```

```
    plLastDllError = 0
    ScreenResolution = ScreenPixelWidth & "x" & ScreenPixelHeight
End Property

Public Property Get SystemErrorCode() As Long
    SystemErrorCode = plLastDllError
End Property

Public Property Get SoundCard() As Boolean
     SoundCard = waveOutGetNumDevs > 0
End Property
'----------------
'Public Functions
'----------------
Public Function AboutBox(Optional hWndA As Long = -1, Optional Copyright As String, Optional ProductInfo As String, Optional Icon As Long = 0)
    If Len(Copyright) = 0 Then Copyright = Common.Version() & "#" & App.ProductName
    If Len(ProductInfo) = 0 Then
        If Len(App.LegalTrademarks) > 0 And Len(App.CompanyName) > 0 Then
            ProductInfo = App.LegalTrademarks
            ProductInfo = ProductInfo & " are legal trademarks of "
            ProductInfo = ProductInfo & App.CompanyName
        Else
            ProductInfo = ProjectName
            ProductInfo = ProductInfo & " is a legal trademark of "
            ProductInfo = ProductInfo & IIf(Len(App.CompanyName) > 0, App.CompanyName, "FailSafe Systems")
        End If
    End If
    Call ShellAbout(hWndA, Copyright, ProductInfo, Icon)
End Function

Public Function APIFunctionPresent(ByVal FunctionName As String, ByVal DLLName As String) As Boolean
    'USAGE: Dim bAvail as boolean
    '       bAvail = APIFunctionPresent("GetDiskFreeSpaceExA", "kernel32")
    Dim lHandle As Long
    Dim lAddr  As Long
    lHandle = LoadLibrary(DLLName)
    If lHandle <> 0 Then
        lAddr = GetProcAddress(lHandle, FunctionName)
        FreeLibrary lHandle
    End If
    APIFunctionPresent = (lAddr <> 0)
End Function

Public Sub CascadeWindows()
```

```
' TODO:
'    Dim objShell As New Shell32.Shell
'    objShell.CascadeWindows
'    Set objShell = Nothing
End Sub

Public Function ControlPanel(Setting As ControlPanelTypes) As Long
   Dim lPanel As String
   Select Case Setting
      Case cplSoftware:   lPanel = "appwiz.cpl,,1"
      Case cplHardware:   lPanel = "sysdm.cpl @1"
      Case cplInternet:   lPanel = "inetcpl.cpl,,0"
      Case cplKeyboard:   lPanel = "main.cpl @1"
      Case cplModem:      lPanel = "modem.cpl"
      Case cplMouse:      lPanel = "main.cpl @0"
      Case cplNetwork:    lPanel = "netcpl.cpl"
      Case cplSounds:     lPanel = "mmsys.cpl @1"
      Case cplSystem:     lPanel = "sysdm.cpl,,0"
      Case cplDisplay:    lPanel = "desk.cpl,,0"
      Case cplPassword:   lPanel = "password.cpl"
      Case cplRegional:   lPanel = "intl.cpl,,0"
      Case cplDateTime:   lPanel = "timedate.cpl"
   End Select
   If Len(lPanel) > 0 Then ControlPanel = Shell("rundll32.exe shell32.dll,Control_RunDLL " & lPanel, 5)
   plLastDllError = Err.LastDllError
End Function

Public Function Directory(WhichDir As DIR_TYPE) As String
   Dim Temp As String
   Dim Ret As Long
   Const MAX_LENGTH = 255

   Temp = String$(MAX_LENGTH, 0)
   Select Case WhichDir
      Case dirWINDOWS:    Ret = GetWindowsDirectory(Temp, MAX_LENGTH)
      Case dirSYSTEM:     Ret = GetSystemDirectory(Temp, MAX_LENGTH)
      Case dirTEMP:       Ret = GetTempPath(Len(Temp), Temp)
      Case Else:          Ret = 0: Temp = ""
   End Select
   Directory = PathCheck(Left$(Temp, Ret))
End Function

Public Function DriveMBFree(Optional Drive As String = "C:\") As Double
   'some time in the future disk may be to large to calculate
```

```
    'like this so resume next on any errors
    On Error Resume Next

    Dim lAns As Long
    Dim lSectorsPerCluster As Long
    Dim lBytesPerSector As Long

    Dim lFreeClusters As Long
    Dim lTotalClusters As Long
    Dim lBytesPerCluster As Long
    Dim lFreeBytes As Double

    'fix bad parameter values
    If Len(Drive) = 1 Then Drive = Drive & ":\"
    If Len(Drive) = 2 And Right$(Drive, 1) = ":" Then Drive = Drive & "\"

    lAns = GetDiskFreeSpace(Drive, lSectorsPerCluster, lBytesPerSector, lFreeClusters, lTotalClusters)
    lBytesPerCluster = lSectorsPerCluster * lBytesPerSector

    DriveMBFree = ((lBytesPerCluster / 1024) / 1024) * lFreeClusters
    DriveMBFree = Format(DriveMBFree, "###,###,##0.00")
End Function

Public Function DriveMBSize(Optional Drive As String = "C:\") As Double
    'some time in the future disk may be to large to calculate
    'like this so resume next on any errors
    On Error Resume Next

    Dim lAns As Long
    Dim lSectorsPerCluster As Long
    Dim lBytesPerSector As Long

    Dim lFreeClusters As Long
    Dim lTotalClusters As Long
    Dim lBytesPerCluster As Long
    Dim lTotalBytes As Double

    'fix bad parameter values
    If Len(Drive) = 1 Then Drive = Drive & ":\"
    If Len(Drive) = 2 And Right$(Drive, 1) = ":" Then Drive = Drive & "\"

    lAns = GetDiskFreeSpace(Drive, lSectorsPerCluster, lBytesPerSector, lFreeClusters, lTotalClusters)
    lBytesPerCluster = lSectorsPerCluster * lBytesPerSector
```

```
    DriveMBSize = ((lBytesPerCluster / 1024) / 1024) * lTotalClusters
    DriveMBSize = Format(DriveMBSize, "###,###,##0.00")
End Function

Public Function DriveName(Optional Drive As String = "C:\")
    Dim sBuffer As String

    plLastDllError = 0
    sBuffer = Space$(255)
    'fix bad parameter values
    If Len(Drive) = 1 Then Drive = Drive & ":\"
    If Len(Drive) = 2 And Right$(Drive, 1) = ":" Then Drive = Drive & "\"
    If GetVolumeInformation(Drive, sBuffer, Len(sBuffer), 0, 0, 0, Space$(255), 255) = 0 Then
        plLastDllError = Err.LastDllError
    Else
        DriveName = Left$(sBuffer, InStr(sBuffer, Chr$(0)) - 1)
    End If
End Function

Public Function DriveType(Drive As String) As DRIVE_TYPE
    'fix bad parameter values
    DriveType = DRIVE_DOESNT_EXIST
    plLastDllError = 0
    Drive = IIf(Len(Drive) = 1, Drive & ":", Drive)
    If Len(Drive) = 1 Then Drive = Drive & ":\"
    If Len(Drive) = 2 And Right$(Drive, 1) = ":" Then Drive = Drive & "\"
    Drive = PathCheck(Drive)
    DriveType = GetDriveType(Drive)
    plLastDllError = Err.LastDllError
End Function

Public Sub DSNs(DSNArray() As String)
    'POPULATES DSNARRAY WITH ALL DSNs installed on the system,
    'in the form of "DSN | DRIVER"
    'USAGE:
    'Dim asDSNArray() As String
    'Dim iCtr As Integer
    'SystemDSNs asDSNArray
    'For iCtr = 0 To UBound(asDSNArray)
    '    Debug.Print asDSNArray(iCtr)
    'Next
    Dim iRet As Integer
    Dim sDSN As String
    Dim sDriver As String
```

```
    Dim iDSNLen As Integer
    Dim iDriverLen As Integer
    ReDim DSNArray(0) As String
    Dim lEnvHandle As Long

    iRet = SQLAllocEnv(lEnvHandle)
    sDSN = Space(1024)
    sDriver = Space(1024)
    iRet = SQLDataSources(lEnvHandle, SQL_FETCH_FIRST_SYSTEM, sDSN, 1024, iDSNLen, sDriver, 1024, iDriverLen)

    If iRet = SQL_SUCCESS Then
       sDSN = Mid(sDSN, 1, iDSNLen)
       sDriver = Mid(sDriver, 1, iDriverLen)
       DSNArray(0) = sDSN & " | " & sDriver
       Do Until iRet <> SQL_SUCCESS
          sDSN = Space(1024)
          sDriver = Space(1024)
          iRet = SQLDataSources(lEnvHandle, SQL_FETCH_NEXT, sDSN, 1024, iDSNLen, sDriver, 1024, iDriverLen)
          If Trim(sDSN) <> "" Then
             sDSN = Mid(sDSN, 1, iDSNLen)
             sDriver = Mid(sDriver, 1, iDriverLen)
             ReDim Preserve DSNArray(UBound(DSNArray) + 1)
             DSNArray(UBound(DSNArray)) = sDSN & " | " & sDriver
          End If
      Loop
    End If
    iRet = SQLFreeEnv(lEnvHandle)
End Sub

Public Function IsConnectedToInternet(Optional ConnectMode As Integer) As Boolean
    Dim flags As Long
    IsConnectedToInternet = InternetGetConnectedState(flags, 0) ' this ASPI function does it all
    ConnectMode = flags                                         ' return the flag through the optional argument
End Function

Public Function MemoryAvailable() As Double
    'Return Value in Megabytes
     MemoryAvailable = MemoryAvailablePhysical + MemoryPageFile
End Function

Public Function MemoryAvailablePhysical() As Double
    'Return Value in Megabytes
    Dim dblAns As Double
    plLastDllError = 0
```

```
    GlobalMemoryStatus pUdtMemStatus
    dblAns = pUdtMemStatus.dwAvailPhys
    MemoryAvailablePhysical = BytesToMegabytes(dblAns)
    plLastDllError = Err.LastDllError
End Function

Public Function MemoryInTotal() As Double
    'Return Value in Megabytes
    MemoryInTotal = MemoryPageFileSize + MemoryTotalPhysical
End Function

Public Function MemoryPageFile() As Double
    'Return Value in Megabytes
    Dim dblAns As Double

    plLastDllError = 0
    GlobalMemoryStatus pUdtMemStatus
    dblAns = pUdtMemStatus.dwAvailPageFile
    MemoryPageFile = BytesToMegabytes(dblAns)
    plLastDllError = Err.LastDllError
End Function

Public Function MemoryPageFileSize() As Double
    'Return Value in Megabytes
    Dim dblAns As Double

    plLastDllError = 0
    GlobalMemoryStatus pUdtMemStatus
    dblAns = pUdtMemStatus.dwTotalPageFile
    MemoryPageFileSize = BytesToMegabytes(dblAns)
    plLastDllError = Err.LastDllError
End Function

Public Function MemoryPercentFree() As Double
    MemoryPercentFree = Format(MemoryAvailable / MemoryInTotal * 100, "0#")
End Function

Public Function MemoryTotalPhysical() As Double
    'Return Value in Megabytes
    Dim dblAns As Double
    plLastDllError = 0
    GlobalMemoryStatus pUdtMemStatus
    dblAns = pUdtMemStatus.dwTotalPhys
    MemoryTotalPhysical = BytesToMegabytes(dblAns)
```

```
    plLastDllError = Err.LastDllError
End Function

Public Function Running(Optional InSeconds As Boolean = False) As Double
    Running = SECS_PER_DAY * (GetTickCount / 1000 / 60 / 60 / 24)
    Running = IIf(InSeconds, Round(Running, 2), Round(Running / 3600, 2))
End Function

Public Function ShutDown(ExitType As ShutdownType) As Long
    ShutDown = ExitWindowsEx(ExitType, 0&)
End Function

Public Function Started() As Date
    Dim dTicks     As Double
    'Store the number of days the systems has been running
    dTicks = GetTickCount / 1000 / 60 / 60 / 24
    Started = Now() - dTicks
End Function

'-----------------
'Private Functions
'-----------------
Private Function BytesToMegabytes(bytes As Double) As Double
  Dim dblAns As Double
  dblAns = (bytes / 1024) / 1024
  BytesToMegabytes = Format(dblAns, "###,###,##0.00")
End Function

Private Function FreeBytesOnDisk(Drive As String) As Long
    On Error Resume Next
    plLastDllError = 0

    Dim lAns As Long
    Dim lSectorsPerCluster As Long
    Dim lBytesPerSector As Long

    Dim lFreeClusters As Long
    Dim lTotalClusters As Long
    Dim lBytesPerCluster As Long
    Dim lFreeBytes As Double


    lAns = GetDiskFreeSpace(Drive, lSectorsPerCluster, lBytesPerSector, lFreeClusters, lTotalClusters)
    lBytesPerCluster = lSectorsPerCluster * lBytesPerSector
```

```
    lFreeBytes = lBytesPerCluster * lFreeClusters
    FreeBytesOnDisk = lFreeBytes
    plLastDllError = Err.LastDllError
End Function

Private Function PathCheck(ByVal PathName As String, Optional AltDelimiter As String = "") As String
    If Len(PathName) = 0 Then Exit Function
    Dim Delimiter As String
    Delimiter = IIf(InStr(PathName, "/"), "/", "\")
    PathCheck = IIf(Right$(PathName, 1) = Delimiter, PathName, PathName & Delimiter)
    PathCheck = IIf(Len(AltDelimiter) = 0, PathCheck, Replace(PathCheck, Delimiter, AltDelimiter))
End Function

Private Function TotalBytesOnDisk(Drive As String) As Double
    On Error Resume Next
    plLastDllError = 0
    Dim lAns As Long
    Dim lSectorsPerCluster As Long
    Dim lBytesPerSector As Long

    Dim lFreeClusters As Long
    Dim lTotalClusters As Long
    Dim lBytesPerCluster As Long
    Dim lTotalBytes As Double

    lAns = GetDiskFreeSpace(Drive, lSectorsPerCluster, lBytesPerSector, lFreeClusters, lTotalClusters)
    lBytesPerCluster = lSectorsPerCluster * lBytesPerSector
    'dblAns = (Bytes / 1024) / 1024
    TotalBytesOnDisk = lBytesPerCluster * lTotalClusters
    If TotalBytesOnDisk = 0 Then
        TotalBytesOnDisk = ((lBytesPerCluster / 1024) / 1024) * lTotalClusters
    End If
    plLastDllError = Err.LastDllError
End Function

Public Function LoadSystemIcon(ByVal StockIcon As enStockIcons) As Long
    On Error Resume Next
    LoadSystemIcon = LoadIconApi(0, StockIcon)
End Function


Public Sub DrawIcon(ByVal HDC As Long, ByVal xPos As Long, ByVal yPos As Long, ByVal Icon As Long)
    Dim lRet As Long
    lRet = DrawIconApi(HDC, xPos, yPos, Icon)
```

```
    If (Err.LastDllError > 0) Or (lRet = 0) Then Debug.Print "DrawIcon failed"
End Sub
'use:

'---------------------
' Class Initialization
'---------------------
Private Sub Class_Initialize()
    pUdtOSVersion.dwOSVersionInfoSize = Len(pUdtOSVersion)
    GetVersionEx pUdtOSVersion
    plMajorVersion = pUdtOSVersion.dwMajorVersion
    plMinorVersion = pUdtOSVersion.dwMinorVersion
    plPlatformID = pUdtOSVersion.dwPlatformId
End Sub

Private Sub Class_Terminate()
    '
End Sub
```

Username / Benutzername auslesen

0. Datum, Usernamen und letzte Belegnummer ausgeben

```
    ' Speichern des letzten Datums, des letzten Users und der letzten Belegnummer

    E.Range("U1") = "am " & Left(Now, 10) & " um " & Right(Now, 8) & vbCrLf & _
    "von " & Application.UserName & vbCrLf & _
    "mit letzter Belegnummer: " & Cells(Range("B65536").End(xlUp).Row, 2)
```

1.Möglichkeit Application.Username

aus unerfindlichen Gründen klappte dieser Befehl bei den meisten Thin Clients und Stand-PCs, aber nicht bei einen Mitarbeiter, das klappte nur die 2 Variante durch Auslesen mittels API (siehe unten)

Msgbox Application.UserName

Möchte man Vornamen und Nachnamen auftrennen und großschreiben:

```
Sub MITARBEITER_NAME()

Dim BENUTZERNAME
Dim VORNAME
Dim NACHNAME

MITARBEITERNAME = Application.UserName
VORNAME = UCase(Left(MITARBEITERNAME, 1)) & Mid(MITARBEITERNAME, 2, InStr(1, MITARBEITERNAME, ".") - 2)
NACHNAME = UCase(Mid(MITARBEITERNAME, InStr(1, MITARBEITERNAME, ".") + 1, 1)) & Mid(MITARBEITERNAME, InStr(1, MITARBEITERNAME, ".") + 2)
Range("I7") = VORNAME & " " & NACHNAME

End Sub
```

Zweite Möglichkeit zum Auslesen mittels API:

```
Option Explicit

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

Function fOSUserName() As String
    Dim lngLen As Long, lngX As Long
    Dim strUserName As String

    strUserName = String$(254, 0)
    lngLen = 255
    lngX = apiGetUserName(strUserName, lngLen)
    If lngX <> 0 Then
       fOSUserName = Left$(strUserName, lngLen - 1)
      Else
       fOSUserName = ""
    End If
End Function

Sub DEMO
   MsgBox "Zur Zeit angemeldet ist: " & fOSUserName
End Sub
```

3. Möglichkeit

Wer bin ich ?

Sollte man eigentlich wissen,,.. in einem Netzwerk aber nicht immer selbstverständlich ;-)

```
'Eigenes Kürzel definieren
Private Declare Function GCN Lib "kernel32" Alias "GetComputerNameA" (ByVal myPara As String, myLen As Long) As Long
Private Declare Function GUN Lib "advapi32.dll" Alias "GetUserNameA" (ByVal myPara As String, myLen As Long) As Long
'Standardeklarationen lauten sonst im allgemeinen
'Private Declare Function GetComputerNameA Lib "kernel32" (ByVal lpBuffer As String, nSize As Long) As Long
'Private Declare Function GetUserNameA Lib "advapi32.dll" (ByVal lpBuffer As String, nSize As Long) As Long

'Prozeduren:
Public Function ActiveUserName() As String
'Benutzernamen auslesen
Dim AUN As String * 100
Dim AunLen As Byte
'100 Zeichen reichen in den meisten Fällen aus
AunLen = 100
If GUN(AUN, Len(AUN)) Then
   'Siehe Hinweis *
   ActiveUserName = Left(AUN, AunLen)
Else
   ActiveUserName = "User can not be Identified"
End If
End Function


Public Function ActiveComputerName() As String
'Benutzernamen auslesen
Dim ACN As String * 100
Dim AcnLen As Byte
AcnLen = 100
If GCN(ACN, Len(ACN)) Then
   'Siehe Hinweis*
   ActiveComputerName = Left(ACN, AcnLen)
Else
   ActiveComputerName = "User can not be Identified"
End If
End Function


Sub wer_und_was_bin_ich()
Dim Qe As Byte
MsgBox ("Mein Rechner heisst" & ActiveComputerName)
MsgBox ("Aktuell angemeldeter User ist: " & ActiveUserName)
```

```
End Sub
```

Hinweis

Das Problem das nun auftritt, ist, dass eine Überprüfung des Benutzernamens mit grosser Wahrscheinlichkeit/Sicherheit fehlschlägt. Ihr Benutzer heisst "Uwe" und eine Überprüfung in der Art

If ActiveUserName = "Uwe" Then

schlägt fehl, obwohl im Debug.Fenster der richtige Name angezeigt wird. Das Problem liegt in der Dimensionierung der Variablen "AUN As String * 100" Nach der Rückgabe aus der API-Funktion *"GetUserNameA"* wird die Variable mit sogenannten 0-Zeichen (ASCII 0) aufgefüllt bis der String eben die definierten 100 Zeichen enthält.

Testen können Sie dies, indem Sie am Punkt "If GUN...:" einen Haltepunkt setzen und mit dem Mauszeiger über die Variable AUN fahren. EXCEL zeigt Ihnen nun den Inhalt der Variablen "AUN" in einem Kommentarfeld an. Dieses wird in der Regel dann in etwa so aussehen:

"USERNAME]]]]]]]]]]]].....

Um diese *0*-Zeichen zu eliminieren können Sie diese Funktion verwenden

Left(AUN, InStr(AUN, vbNullChar) - 1)

oder

Left(ACN, InStr(ACN, vbNullChar) - 1)

Mit "InStr" suchen Sie innerhalb des Strings AUN nach der Position des ersten Zeichen das diesem ASCII 0 Zeichen entspricht. Dieses Zeichen können sie mit "vbNullChar" identifizieren.

Von dieser Positionsnummer subtrahieren Sie 1 und haben den "reinen" Benutzernamen, bzw. den "reinen" Computernamen.

Alternativ geht's auch mit einer, etwas umständlichen ;-), eigenen Funktion der Sie die Variable "AUN" übergeben

```
Function PureName(tmpName As String)
Dim i
For i = 1 To 100
   Select Case UCase(Mid(tmpName, i, 1))
      Case vbNullChar
         'Hier wird die Position des
         'Zeichens ebenfalls gefunden
         Debug.Print "Pos: " & i
         PureName = Left(tmpName, i - 1)
         Exit For
   End Select
Next i
End Function
Sub wer_und_was_bin_ich()
MsgBox ("Mein Rechner ist" & PureName(ActiveComputerName) & PureName(ActiveUserName))
End Sub
```

Dann klappt's auch mit dem Vergleich ;-)

4. Version

```
Declare Function GetUserName Lib "advapi32.dll" Alias
```

```
"GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Sub ShowUserName()
Dim Buffer As String * 100
Dim BuffLen As Long
BuffLen = 100
GetUserName Buffer, BuffLen
MsgBox Left(Buffer, BuffLen - 1)
End Sub
Function NetUserName()
Dim Buffer As String * 100
Dim BuffLen As Long
BuffLen = 100
GetUserName Buffer, BuffLen
NetUserName = Left(Buffer, BuffLen - 1)
End Function
```

Windows-Systemsteuerungsfenster öffnen

```
' ------------------------------------------------------------------------
'
'  Title: Display various Windows and Control Panel dialogs
'


' ------------------------------------------------------------------------

' -------
' Code
' -------

Sub CP_Dialog()
    ' Launch Control Panel Dialog

    Dim dblReturn As Double

    ' Depending on the dialog, replace "rundll32.exe shell..." with the correspondant command line below
    dblReturn = Shell("rundll32.exe shell32.dll,Control_RunDLL access.cpl,,1", 1)
End Sub



'Control Panel    ( CONTROL.EXE )
'-------------------------------
'Control Panel:
```

```
'    rundll32.exe shell32.dll,Control_RunDLL
'
'Add/Remove Programs    ( APPWIZ.CPL )
'-----------------------------------
'Add/Remove Programs Properties (Install/Uninstall):
'    rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,1
'Add/Remove Programs Properties (Windows Setup):
'    rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,2
'Add/Remove Programs Properties (Startup Disk):
'    rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,3
'
'Display Options    ( DESK.CPL )
'-------------------------------
'Display Properties (Background):
'    rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,0
'Display Properties (Screen Saver):
'    rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,1
'Display Properties (Appearance):
'    rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,2
'Display Properties (Settings):
'    rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,3
'
'Regional Settings    ( INTL.CPL )
'---------------------------------
'Regional Settings Properties (Regional Settings):
'    rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,0
'Regional Settings Properties (Number):
'    rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,1
'Regional Settings Properties (Currency):
'    rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,2
'Regional Settings Properties (Time):
'    rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,3
'Regional Settings Properties (Date):
'    rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,4
'
'Joystick Options    ( JOY.CPL )
'-------------------------------
'Joystick Properties (Joystick):
'    rundll32.exe shell32.dll,Control_RunDLL joy.cpl
'
'Mouse/Keyboard/Printers/Fonts Options    ( MAIN.CPL )
'-----------------------------------------------------
'Mouse Properties:
'    rundll32.exe shell32.dll,Control_RunDLL main.cpl @0
```

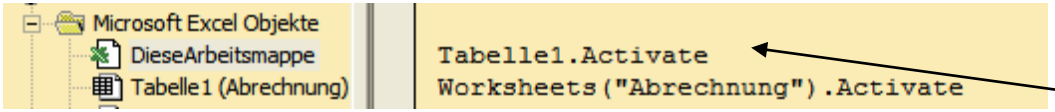
```
'Keyboard Properties:
'    rundll32.exe shell32.dll,Control_RunDLL main.cpl @1
'Printers:
'    rundll32.exe shell32.dll,Control_RunDLL main.cpl @2
'Fonts:
'    rundll32.exe shell32.dll,Control_RunDLL main.cpl @3
'
'Mail and Fax Options    ( MLCFG32.CPL )
'-------------------------------------
'Microsoft Exchange Profiles (General):
'    rundll32.exe shell32.dll,Control_RunDLL mlcfg32.cpl
'
'Multimedia/Sounds Options    ( MMSYS.CPL )
'----------------------------------------
'Multimedia Properties (Audio):
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,0
'Multimedia Properties (Viedo):
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,1
'Multimedia Properties (MIDI):
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,2
'Multimedia Properties (CD Music):
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,3
'Multimedia Properties (Advanced):
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,4
' = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
'Sounds Properties:
'    rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl @1
'
'Modem Options    ( MODEM.CPL )
'-----------------------------
'Modem Properties (General):
'    rundll32.exe shell32.dll,Control_RunDLL modem.cpl
'
'Network Options    ( NETCPL.CPL )
'--------------------------------
'Network (Configuration):
'    rundll32.exe shell32.dll,Control_RunDLL netcpl.cpl
'
'Password Options    ( PASSWORD.CPL )
'-----------------------------------
'Password Properties (Change Passwords):
'    rundll32.exe shell32.dll,Control_RunDLL password.cpl
'
'System/Add New Hardware Options    ( SYSDM.CPL )
```

```
'-----------------------------------------------
'System Properties (General):
'    rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,0
'System Properties (Device Manager):
'    rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,1
'System Properties (Hardware Profiles):
'    rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,2
'System Properties (Performance):
'    rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,3
' = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
'Add New Hardware Wizard:
'    rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl @1
'
'Date and Time Options    ( TIMEDATE.CPL )
'---------------------------------------
'Date/Time Properties:
'    rundll32.exe shell32.dll,Control_RunDLL timedate.cpl
'
'Microsoft Mail Postoffice Options    ( WGPOCPL.CPL )
'-------------------------------------------------
'Microsoft Workgroup Postoffice Admin:
```

TABELLENBLÄTTER

Grundlagen Tabellen-Blattname und -Codename

Es gibt ja immer den Tabellenblattnamen und den Codenamen einer Tabelle - sichtbar z.B. hier:



Der Codename der ersten Tabelle ist Tabelle1 und entsprechend können wir sie auch direkt ansprechen mit Tabelle1.Activate

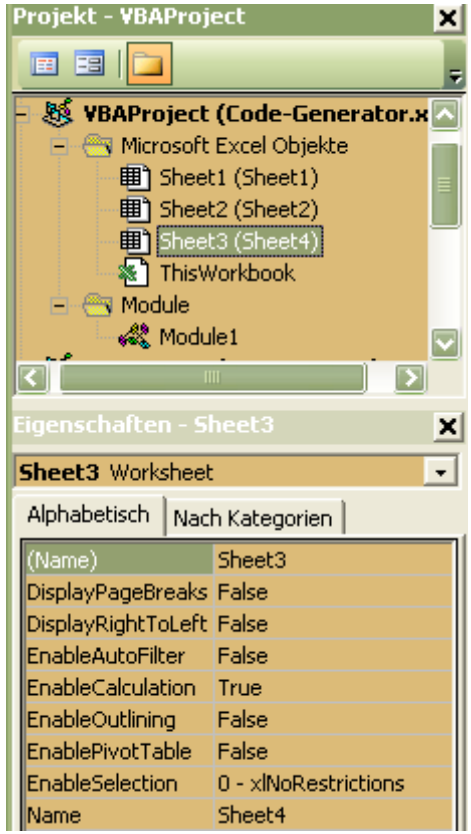
Der Tabellenblattname ist "Abrechnung" und darum können wir alternativ auch Worksheets("Abrechnung").Activate verwenden.

Wenn man möchte, dass User den Tabellenblattnamen ändern dürfen, aber der VBA-Code trotzdem immer auf das richtige Tabellenblatt zugreift, verwendert man im VBA-Code nicht den Worksheet"Tabellenblattname")-Befehl, sondern arbeitet gleich mit dem Codenamen der Tabelle - also z.B.

Tabelle1.Range("C6" ….

Siehe auch "Tabellenblattnamen-Änderung zurücksetzen"

Dass der Tabellenblattnamen nicht gleich dem Codenamen einer Tabelle ist sieht man hier schön: der Codenamen des 3. Blattes ist Sheet3 - und in Excel in der Arbeitsmappe heißt das Tabellenblatt Sheet4



Normalerweise ist der Tabellenblattnamen (Name in Klammer) immer gleich mit dem Tabellen-Codenamen – aber sobald der User (oder VBA) den Tabellenblattnamen ändern, lautet der Tabellenblattname anders als der Code-Namen.

Angesprochen mit normalem VBA-Code wird das Blatt natürlich mit dem Tabellenblatt-Namen (Sheet4) - zB

Worksheets("Sheet4").Activate

Wenn wir jedoch den VBA-Code des Tabellenblattes ändern wollen, müssen wir die Tabelle als Sheet3 ansprechen – da VBA-Code-Änderungen immer mit dem Tabellen-Codenamen arbeiten.

Code um Codenamen und Tabellenblattnamen anzuzeigen

```
    MsgBox "Codename " & Workbooks("Countrates_Overview.xls").Charts(1).CodeName & " – Tabellenblattname " &
Workbooks("Countrates_Overview.xls").Charts(1).Name
```

```
MsgBox "Codename " & Workbooks("Countrates_Overview.xls").Sheets(1).CodeName & " – Tabellenblattname " &
Workbooks("Countrates_Overview.xls").Sheets(1).Name
```

Oder eine Schleife um alle Chart-Codenamen anzuzeigen

```
Sub TEST_7()

    For xx = 1 To Workbooks("Countrates_Overview.xls").Charts.Count

        Code_Name = Workbooks("Countrates_Overview.xls").Charts(xx).CodeName
        MsgBox ActiveWorkbook.VBProject.VBComponents(Code_Name).Properties(5).value

    Next xx

End Sub
```

Code, um den Codenamen eines einzelnen Tabellenblattes umzubenennen

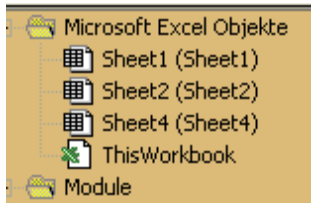
```
ThisWorkbook.VBProject.VBComponents("Tabelle1").Properties(5).Value = "KRola"
```

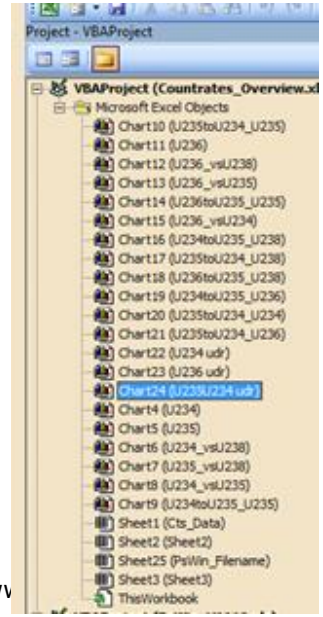
Code um alle Tabellen-Codenamen anzupassen an den Tabellen-Blattnamen:

Nachfolgender Code (aus dem Excel-VBA Codebook als PDF) wandelt die Tabellenblatt-Codenamen (hier sheet3) auf die Tabellenblattnamen (hier sheet4) um:

```
Sub ChangeAllCodeNames()
  Dim objSh As Object

  On Error Resume Next
  For Each objSh In ThisWorkbook.Application.Sheets
    With ThisWorkbook.VBProject.VBComponents(objSh.CodeName)
      .Properties("_CodeName").Value = objSh.Name
    End With
  Next
End Sub
```

Ergebnis:





Das Ganze gibts auch bei Diagrammen / Charts

Array mit allen Tabellenblattnamen

```
Public Sub SheetArray()

Private Sub Build_Worksheet_Array()
ArraySize = 0
For j = 1 To Sheets.Count              ' Provides a count of all worksheets
   If TypeName(Sheets(j)) = "Worksheet" Then   ' Gives the type of worksheet
      MyWorkSheetName = Sheets(j).Name              ' Provides the name for the worksheet
      ArraySize = ArraySize + 1
      ReDim Preserve SubsheetArray(ArraySize)
      SubsheetArray(ArraySize) = MyWorkSheetName
   End If
Next
End Sub
```

Alle Tabellen Blattschutz aufheben / setzen

Siehe bitte unter ARBEITSMAPPEN Arbeitsmappe & Tabellenblatt schützen

Alle Tabellenblätter als eigene Datei speichern wie Arbeitsmappe

```
Sub TabelleAlsDatei()
Dim wks As Worksheet
Dim wkb As Workbook
Dim wkb2 As Workbook
Dim Pfad

Set wkb = ThisWorkbook
Pfad = wkb.Path
```

```
For Each wks In wkb.Worksheets
   Workbooks.Add
   Set wkb2 = ActiveWorkbook
   wks.Copy Before:=wkb2.Sheets(1)
   'wkb2.SaveAs wks.Name & ".xls"
   wkb2.SaveAs Pfad & "\" & wks.Name & ".xls"
   wkb2.Close
   MsgBox wks.Name
Next
End Sub
```

Aktuelle Tabelle als neue Arbeitsmappe kopieren

Activesheet.Copy

' möchte man der neuen Mappe auch gleich einen Namen geben (nur den Anzeigenamen im Excelwindow könnte man ändern, aber das zieht nicht zur Speicherung) geht am besten mit Activeworkbook.Saveas

Aktuelle Tabelle merken und zurückspringen

```
   Dim AKTUELLETABELLE As Worksheet

   Set AKTUELLETABELLE = ActiveSheet

….. eigentlicher Code inkl Springen in andere Tabellen

   AKTUELLETABELLE.Select
```

Aktuelles oder anderes Tabellenblatt

Ist man im aktuellen Tabellenblatt könnte man vor einen Cell- oder Range-Befehl immer ein ActiveSheet. setzen - also zB ActiveSheet.Range("J1").Value

Lässt man dies aber weg, weiß Excel bescheid, dass man das aktuelle Tabellenblatt meint.

Zellwerte anderer Tabellenblätter ruft man auf mit Sheets("NAME") auf.

Sheets("NAME").Cells.Copy
Sheets("NAME2").Select *wählt dieses Blatt aus - ähnlich dem Activate*

Sheets("NAME3").Activate aktiviert dieses Blatt (wie Mausclick auf Blattregister)

Aktuelles Tabellenblatt sichtbare Zellen in neue Arbeitsmappe kopieren

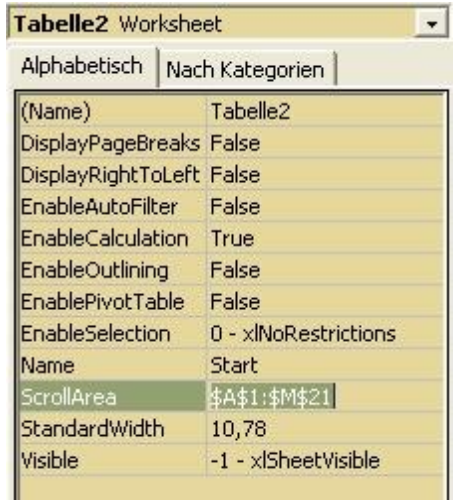
```
Sub EXCEL_EXPORT()

' Dieser Button dient zum Export der aktuell gefilterten Datenzeilen in eine neue Arbeitsmappe (ab Zeile 10)

Dim AKTUELLEMAPPE As String
Dim NEUEMAPPE As String
Dim SICHTBARESPALTEN As Integer


AKTUELLEMAPPE = ActiveWorkbook.Name

Application.ScreenUpdating = False

' Auslesen wieviele Spalten sichtbar sind
   For S = 1 To 26
      If Columns(S).Hidden = False Then SICHTBARESPALTEN = SICHTBARESPALTEN + 1
   Next S

' Neue Arbeitsmappe anlegen
   Workbooks.Add

   NEUEMAPPE = ActiveWorkbook.Name

' Spalten in der Zieltabelle optimal breit machen
   Columns(SICHTBARESPALTEN).ColumnWidth = 40 ' Anmerkungenspalte
   Columns(SICHTBARESPALTEN - 1).ColumnWidth = 30 ' Keywords found
   Columns(SICHTBARESPALTEN - 2).ColumnWidth = 8 ' Nr
   Columns(SICHTBARESPALTEN - 3).ColumnWidth = 40 ' Kommentarspalte
   Columns(SICHTBARESPALTEN - 4).ColumnWidth = 30 ' Keywords found
   Columns(SICHTBARESPALTEN - 5).ColumnWidth = 8 ' Nr

' Die restlichen Spalten optimieren
   For S = 1 To SICHTBARESPALTEN - 6
      Columns(S).ColumnWidth = 25
```

```
    Next S

' Zurückgehen in originale Arbeitsmappe
    Workbooks(AKTUELLEMAPPE).Activate

' Kopieren der sichtbaren Zeilen inkl. Überschrift
    Range("A10:Z" & Cells(Rows.Count, 1).End(xlUp).Row).SpecialCells(xlCellTypeVisible).Copy

' Wechseln in neue Arbeitsmappe
    Workbooks(NEUEMAPPE).Activate
    Range("A1").Select
    ActiveSheet.Paste

    Application.CutCopyMode = False ' Kopiermarkierung aufheben
    Workbooks(AKTUELLEMAPPE).Activate
    Range("W6").Select
    Workbooks(NEUEMAPPE).Activate
    Range("A1").Select
    Application.ScreenUpdating = True


End Sub
```

Arbeitsbereich einschränken

DEN BEARBEITENBAREN TABELLENBEREICH EINSCHRÄNKEN

möchte man zB, dass man nur im Tabellenbereich A1:M21 arbeiten kann und der User nicht darüber hinaus navigieren / klicken / ändern kann, so tut man das in der Scroll-Area-Eigenschaft der betreffenden Tabelle:



Leider geht dies bei jedem neuen Öffnen der Arbeitsmappe verloren - aber man kann dies ja beim Öffnen der Vorlage automatisch hinterlegen lassen.

Was natürlich immer geht ist die SelectionChange-Routine zu verwenden und wenn der User in falschen Bereich klickt, dann springt die Markierung automatisch wieder weg von dort.

Aus- und Einblenden von Tabellenblättern

Ich verwende bevorzugt die internen Tabellenblattnamen (also wie die Tabellenblätter in der VBA-Umgebung genannt werden - der Tabellenblattname in Excel selbst kann davon ja abweichen). zB

```
Tabelle15.Visible = xlSheetHidden

Tabelle16.Visible = xlSheetVeryHidden
Tabelle8.Visible = xlSheetVisible
```

Ruft man eine Tabelle mit ihrem echten Tabellenblattnamen auf, lautet der Code anders:

```
Worksheets("VBA-Aktivierung").Visible = False
```

Ausgewählte Tabellenblätter in neue Datei kopieren

```
Sub EXPORT_TABELLENBLAETTER()
```

```
' um mehrere ausgewählte Tabellenblätter in neue Datei zu kopieren

Dim TABELLE As Worksheet
Dim ANZAHLTABELLEN
Dim E(31) ' maximal 31 Tabellenblätter können exportiert werden

For Each TABELLE In ActiveWindow.SelectedSheets
   ANZAHLTABELLEN = ANZAHLTABELLEN + 1
   E(ANZAHLTABELLEN) = TABELLE.name
Next TABELLE

Select Case ANZAHLTABELLEN
 Case 1
   Sheets(Array(E(1))).Copy
 Case 2
   Sheets(Array(E(1), E(2))).Copy
 Case 3
   Sheets(Array(E(1), E(2), E(3))).Copy
 Case 4
   Sheets(Array(E(1), E(2), E(3), E(4))).Copy
 Case 5
   Sheets(Array(E(1), E(2), E(3), E(4), E(5))).Copy
 Case 6
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6))).Copy
 Case 7
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7))).Copy
 Case 8
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8))).Copy
 Case 9
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9))).Copy
 Case 10
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10))).Copy
 Case 11
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11))).Copy
 Case 12
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12))).Copy
 Case 13
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13))).Copy
 Case 14
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14))).Copy
 Case 15
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15))).Copy
 Case 16
   Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16))).Copy
```

```
 Case 17
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17))).Copy
 Case 18
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18))).Copy
 Case 19
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19))).Copy
 Case 20
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20))).Copy
 Case 21
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21))).Copy
 Case 22
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21),
E(22))).Copy
 Case 23
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23))).Copy
 Case 24
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24))).Copy
 Case 25
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25))).Copy
 Case 26
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26))).Copy
 Case 27
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26), E(27))).Copy
 Case 28
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26), E(27), E(28))).Copy
 Case 29
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26), E(27), E(28), E(29))).Copy
 Case 30
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26), E(27), E(28), E(29), E(30))).Copy
 Case 31
    Sheets(Array(E(1), E(2), E(3), E(4), E(5), E(6), E(7), E(8), E(9), E(10), E(11), E(12), E(13), E(14), E(15), E(16), E(17), E(18), E(19), E(20), E(21), E(22),
E(23), E(24), E(25), E(26), E(27), E(28), E(29), E(30), E(31))).Copy
End Select
```

End Sub

Autofilter wieder zurücksetzen

Hat man einen Autofilter gesetzt, vergisst man bisweilen einzelne Auswahlfilter wieder zurückzusetzen, um alles zu sehen

Manuell geht man ins Menü DATEN _ FILTER _ ALLE ANZEIGEN

VBA:

ActiveSheet.ShowAllData

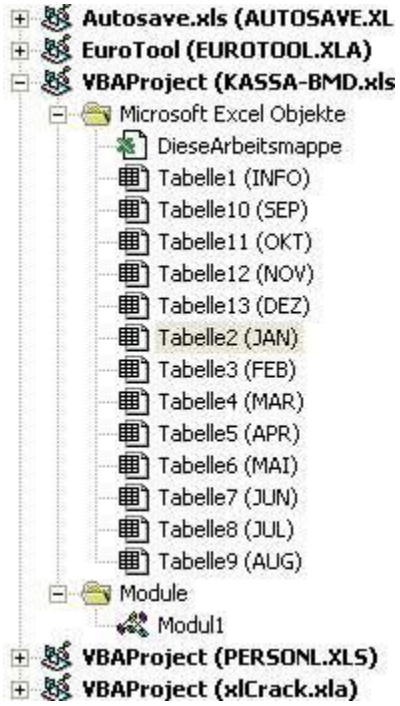
Automatisch mitwachsende Tabelle

Damit eine Tabellenvorlage (zB Kassabuch) nicht 100 leere Zeilen mit immer den gleichen Formeln hat , die dann ohnedies meist nicht angefüllt werden und beim Ausdruck unnötig Papier brauchen, wäre es doch schön, wenn eine Vorlage automatisch mit dem Ausfüllen mitwächst.

Konkret wählte ich den Weg, dass wenn man den Zellcursor in die erste Zelle der letzten Zeile setzt, dass diese Zeile kopiert und anschließend wieder eingefügt wird.

Dazu braucht man erstens ein leeres Feld, das niemand löscht/beschreibt (ich verstecke es gerne unter ein darüberliegendes Bild - im folgenden BSp in G8) - und darin ist die Zeilennummer der aktuell letzten Zeile enthalten. Das Tabellenblatt (und jedes einzelne von ihnen extra) bekommt einen eigenen Code, der abfagt, ob der Schreibcursor in dieser aktuell letzten Zelle ist.

Dazu gibt man folgenden Code in jedes Tabellenblatt (hier JAN; FEB; MAR...) einzeln ein:




```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)

Dim Letztezeile As Integer

' Auslesen der Zeilennummer der zur Zeit letzten Zeile im Feld G8
    Letztezeile = Sheets("JAN").Range("G8").Value

' Wenn nun das Feld "A[Letztezeile] selektiert wird - Zb "A10"
If Target.Address = "$A$" & Letztezeile Then

    ' Schreib in das "Speicher"-Feld G8, dass sich die letzte Zeile eines tiefer verschiebt
    Sheets("JAN").Range("G8").Value = Letztezeile + 1

    Range("A" & Letztezeile).EntireRow.Select ' Markiere die aktuelle Zeile ganz
    Selection.Copy  ' Kopiere diese Zeile
    Selection.Insert Shift:=xlDown  ' Füge sie eins weiter darunter ein

    ' Spring zurück auf Zelle A, der anfangs alles aulösenden (jetzt vorvorletzte) Zeile
    Range("A" & Letztezeile).Activate
```

```
    Application.CutCopyMode = False  ' und lösch die Kopiermarkierung

End If
End Sub
```

Als Variante wählte ich später den Weg, dass ich den obigen Code nicht in die 12 Tabellenbläter gab, sondern direkt in das Hauptblatt "Diese Arbeitsmappe" und dort mit dem folgenden Aufruf startete:

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)
[…]
```

Auslesen ob Status den Tabellenblattes auf Blattschutz

Auslesen ob Status des Tabellenblattes auf Blattschutz - und wenn ja, dann mit Passwort den Schutz aufheben

```
If Worksheets(TABELLENBLATT).ProtectContents = True Then Worksheets(TABELLENBLATT).Unprotect Password:="xxxx"
```

Blattschutz siehe ARBEITSMAPPEN Arbeitsmappe & Tabellenblatt schützen

Druckbereich einer Tabelle automatisch auf letzte Zeile einstellen

siehe gleichnamigen Eintrag im Bereich DRUCKEN + PDF

Erste sichtbare Tabelle aktivieren

Die schnellste Variante geht angeblich nicht, wenn die Tabelle ausgeblendet ist - bei mir hats aber nie Probleme gegeben

1.) Schnellste Variante

```
Sheets(1).Select
```

2.) Langsame Variante

```
Sub GoToFirstSheet()
    Dim i As Long

    For i = 1 To ThisWorkbook.Sheets.Count
        On Error Resume Next
        Sheets(i).Activate
        If Err.Number = 0 Then Exit For
    Next i

    On Error Goto 0

End Sub
```

2.Variante

```
Sub GoToFirstSheet()

    Dim ws As Worksheet 'declare a worksheet variable
    'loop through all the worksheets in the workbook
    For Each ws In ThisWorkbook.Worksheets
        'If the sheet is not hidden
        If ws.Visible = xlSheetVisible Then
            ws.Select 'select it
            Exit For 'exit the loop
        End If
    Next ws

End Sub
```

Variante 3

```
Dim i As Integer
Do
i = i + 1
On Error Resume Next
Worksheets(i).Select
Loop Until Worksheets(i).Visible = True
```

Filter wieder zurücksetzen

Hat man einen Autofilter gesetzt, vergisst man bisweilen einzelne Auswahlfilter wieder zurückzusetzen, um alles zu sehen

Manuell geht man ins Menü DATEN _ FILTER _ ALLE ANZEIGEN

VBA:

ActiveSheet.ShowAllData

F9-Taste Tabellenblatt-Werte aktualisieren

Selection.Calculate ' kalkuliert nur in der aktuellen Tabelle im zuvor gewählten Bereich die Zellen neu (am schnellsten)
Worksheet.Calculate ' kalkuliert nur in der aktuellen Tabelle alle Zellen mit Formeln (immer noch sehr schnell)

Calculate ' kalkuliert alle Tabellenblätter in der aktuellen Arbeitsmappe die Zellen neu (schon langsamer)

alternativ geht auch:
For Each wks In ActiveWorkbook.Worksheets ' kalkuliert alle Tabellenblätter in der aktuellen Arbeitsmappe die Zellen neu (schon langsamer)
    wks.Calculate
Next

Application.Calculate ' Kalkuliert in allen geöffneten Arbeitsmappen die geänderten Zellen mit Formeln neu (noch langsamer)
Application.CalcualteFull ' Kalkuliert in allen geöffneten Arbeitsmappen alle Zellen (noch langsamer)
Application.CalculateFullRebuild ' Kalkuliert in allen Arbeitsmappen alle Zellen mit Formeln und berechnet den Formelabhängigkeitsbaum neu (am langsamsten)

Gruppieren von Tabellenblättern

Version 1 für AUSWÄHLBARE TABELLENBLÄTTER

' Markieren der gewünschten Tabellen
Sheets(Array(DIR_ORGA.Name, DIR_MA.Name, DIR_PK.Name, DIR_PENS.Name)).Select

Version 2 für ALLE TABELLENBLÄTTER

```
Option Explicit
'Wichtig damit die Array Zählung bei 1 startet und nicht wie sonst bei 0
```

```
Option Base 1


Sub Group_Sheets()
'(C) by Ramses
'Gruppiert Sheets für die weitere Bearbeitung
Dim i, n
Dim tbArr() As Variant
n = 1
'Array-Grösse ist Anzahl Tabellenblätter - 1
ReDim tbArr(Worksheets.count - 1)
'Startet die Gruppierung ab Tabelle2
For i = 2 To Worksheets.count
   tbArr(n) = Worksheets(i).name
   n = n + 1
Next i
Sheets(tbArr).Select
End Sub
```

Gruppierung von Tabellenblättern aufheben

Einfach durch select eines Tabellenblattes - zB. Tabelle1.Select

Mehrere Tabellenblätter als PDF exportieren

siehe PDF/MEHRERE TABELLENBLÄTTER ALS PDF EXPORTIEREN

Mehrere Tabellenblätter in neue Datei kopieren

Siehe Tabellenblätter (mehrere) in neue Datei kopieren

Nummer / Index eines Tabellenblattes

Gemäß der Reihenfolge in der Arbeitsmappe hat jedes Tabellenblatt eine Nummer beginnend bei 1 für die linkeste Tabelle

Mit Sheets(1).Activate könnte man so z.B. die "linkeste" Tabelle aktivieren.

Die Nummer der aktuellen Tabelle erhält man mit

Msgbox Activesheet.Index

Die Nummer einer bestimmten Tabelle, deren Namen man kennt, erhält man mit

Msgbox Sheets("Tabelle1").Index

Nächstes Tabellenblatt anspringen

Wenn die Tabellenblätter durchnummeriert benannt sind von 1-99 dann kommt man mit diesem Befehl zum nächsten Tabellenblatt

Worksheets(Worksheets(ActiveSheet.Name).Name + 1).Activate

Besser gehts mit folgendem Code, der unabhängig vom Blattname ist

```
Sub BlattVor()
On Error Resume Next
ActiveSheet.Next.Activate
End Sub

Sub BlattZurück()
On Error Resume Next
ActiveSheet.Previous.Activate
Ende:
End Sub
```

Möchte man auch noch das Arbeitsmappenende und Anfang berücksichtigen

```
Sub Select_Sheet_nachfolgend()
Dim lgIndex As Long
lgIndex = ActiveSheet.Index

If lgIndex = Worksheets.Count Then
    MsgBox "letzes Arbeitsblatt ist schon selektiert"
Else
    Worksheets(lgIndex + 1).Select
End If
End Sub

Sub Select_Sheet_vorher()
Dim lgIndex As Long
lgIndex = ActiveSheet.Index

If lgIndex = 1 Then
    MsgBox "erstes Arbeitsblatt ist schon selektiert"
Else
    Worksheets(lgIndex - 1).Select
End If
End Sub
```

Name des aktuellen Tabellenblattes + Rücksprung

```
Dim AKTUELLETABELLE As String

AKTUELLETABELLE = ActiveSheet.Name   ' merken des Blattnames

Sheets(AKTUELLETABELLE).Select  ' Rücksprung auf voriges Tabellenblatt
```

Neues Tabellenblatt erzeugen, falls noch nicht existiert

dazu wird eine Funktion benötigt - hier "Existiertblatt" genannt - die in nachfolgendem Programmcode aufgerufen wird:

```
' HILFSFUNKTION FÜR 'SUB SEITEKOPIEREN'

Function Existiertblatt(Blattname As String) As Boolean

    On Error Resume Next
    Existiertblatt = CBool(Len(Worksheets(Blattname).Name) > 0)

End Function

' PROGAMMCODE
' Löschen eines ev.exist.Arbeitsblattes "BMD" und anschließend dieses neu erstellen

If (Existiertblatt("BMD")) Then ActiveWorkbook.Worksheets("BMD").Delete
Worksheets.Add.Move after:=Worksheets(Worksheets.Count): ' Neu anlegen
ActiveSheet.Name = "BMD": ' Neues Blatt benennen
```

Neues Tabellenblatt am Ende der Arbeitsmappe einfügen

```
Sub NEUES_BLATT()
    Sheets.Add after:=Worksheets(Worksheets.Count)
End Sub
```

Prüfen ob Tabellenblatt existiert + Anlegen oder Löschen

TEIL 1 - Prüfen ob Tabelle in einer bestimmten Arbeitsmappe existiert

TEIL 2 - Prüfen ob Tabelle in aktueller Arbeitsmappe existiert

1.Weg ohne Funktion
a.) ANLEGEN falls Blatt nicht existiert

```
    If IsError(Evaluate("T1!A1")) Then
        Worksheets.Add
        Worksheets(Worksheets.Count).Name = "T1"
    Else
        MsgBox "T1 schon vorhanden"
    End If
```

Bedingung ist allerdings, dass in der auszuwertenden Zelle A1 keine Formel steht, die einen Fehlerwert ausgibt.

b.) LÖSCHEN falls Blatt existiert

```
    If Not IsError(Evaluate("T1!A1")) Then
        Application.DisplayAlerts = False
        Worksheets("T1").Delete
        Application.DisplayAlerts = True
    Else
        MsgBox "T1 war nicht vorhanden"
    End If
```

2.Weg ohne Funktion

```
Sub test1()
    Dim ws As Worksheet

    On Error Resume Next
       Set ws = ThisWorkbook.Worksheets("Karneval ole")
    On Error GoTo 0

    If ws Is Nothing Then
       MsgBox "Tabelle nicht vorhanden!", vbSystemModal + 16
       Exit Sub
    End If

    'hier gehts weiter, falls die Tabelle existiert!
```

```
    'blablabla
    '...
    Set ws = Nothing
End Sub
```

3.Weg mit Funktion

```
Sub einfuegen()

  If Not SheetExist("T1") Then
    Sheets.Add(after:=Sheets(Sheets.Count)).Name = "T1"
  End If

End Sub


Sub loeschen()

  If SheetExist("T1") Then
    Application.DisplayAlerts = False
    Sheets("T1").Delete
    Application.DisplayAlerts = True
  End If

End Sub


Private Function SheetExist(ByVal sheetName As String, Optional Wb As Workbook) As Boolean
  Dim wks As Worksheet
  On Error GoTo ERRORHANDLER
  If Wb Is Nothing Then Set Wb = ThisWorkbook
  For Each wks In Wb.Worksheets
    If LCase(wks.Name) = LCase(sheetName) Then SheetExist = True: Exit Function
  Next
  ERRORHANDLER:
  SheetExist = False
End Function
```

4. Weg mit schneller Funktion

```
Sub test()
  If WorksheetExists("Tabelle100") Then MsgBox "Tabelle exitiert"
End Sub

Public Function WorksheetExists(ByVal WorksheetName As String) As Boolean
  On Error Resume Next
```

```
 WorksheetExists = (Sheets(WorksheetName).Name <> "")
 On Error GoTo 0
End Function
```

5. Weg mit langsamer Funktion

```
Function WorksheetExists(wsName As String) As Boolean
    Dim ws As Worksheet
    Dim ret As Boolean
    ret = False
    wsName = UCase(wsName)
    For Each ws In ThisWorkbook.Worksheets

        If UCase(ws.Name) = wsName Then
            ret = True
            Exit For
        End If
    Next
    WorksheetExists = ret
End Function
```

Anwendung in einer Prozedur

```
IF WorksheetExists("Tabelle1") THEN
            ' Code falls es existiert
       Else
            ' Code falls nicht existiert
End if
```

Schleife durch alle Tabellenblätter

```
' Alle Tabellenblattnamen anzeigen
For n = 1 To Worksheets.Count
   Msgbox Worksheets(n).name          ' name bleibt klein, aber Befehl passt
Next
```

Schleife durch alle ausgewählten Tabellenblätter

wenn man mit STRG-Taste mehrere Tabellenblätter auswählt:

Meine Version:

```
Dim TABELLE As Worksheet
Dim ANZAHLTABELLEN
Dim E(31) ' maximal 31 Tabellenblätter können exportiert werden
Dim AKTUELLETABELLE

AKTUELLETABELLE = ActiveWorkbook.name

For Each TABELLE In ActiveWindow.SelectedSheets
   ANZAHLTABELLEN = ANZAHLTABELLEN + 1
   E(ANZAHLTABELLEN) = TABELLE.name
Next TABELLE

Select Case ANZAHLTABELLEN
 Case 1
   Sheets(Array(E(1))).Copy
 Case 2
   Sheets(Array(E(1), E(2))).Copy
 Case 3
   Sheets(Array(E(1), E(2), E(3))).Copy

…
```

Allg. Version

```
Sub AusgewählteBlätter()
Dim sh As Worksheet
For Each sh In ActiveWindow.SelectedSheets
Debug.Print sh.Name
Next sh
Debug.Print "Aktiv: " & ActiveSheet.Name
End Sub
```

Schleife alle Tabellen öffnen und sperren

Verwende ich zB für meine Buchungssymbol-Erfassungsvorlagen, um alle 12 Monatstabellen auf und zu zumachen.

```
Sub Blattschutz_Alle_Blaetter_AUF()

   For n = 1 To Worksheets.Count
     Worksheets(n).Unprotect Password:=""
   Next

End Sub

Sub Blattschutz_Alle_Blaetter_ZU()

   For n = 1 To Worksheets.Count
     Worksheets(n).Protect Password:="", DrawingObjects:=True, Contents:=True, Scenarios:=True ' Gesperrte Zellen können nicht gewählt werden
   Next

End Sub
```

Schutz Tabellen siehe ARBEITSMAPPEN Arbeitsmappe & Tabellenblatt schützen

SCROLLEN - siehe ZELLEN / SCROLLEN IN EINE BESTIMMTE ZELLE

Seitenumbruch

Es gibt ihn ja horizontal und vertical

Horzontal:

In diesem Beispiel wird ein manueller Seitenumbruch oberhalb von Zeile 25 in Sheet1 eingefügt.

```
Worksheets("Sheet1").Rows(25).PageBreak = xlPageBreakManual
```

Vertikal:

```
ActiveWindow.SelectedSheets.VPageBreaks.Add before:=Columns("AK")
```

RESET UND VERTIKAL AUF EINE SEITE:

` Zu Anfang setzte ich alle Seitenumbrücke zurück und lege den vertikalen Umbruch an den rechten Rand des Druckbereichs.

```
ActiveSheet.ResetAllPageBreaks
ActiveSheet.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1
```

Umfangreiches Beispiel, das zusammenhängende Bereiche nicht trennt:

Meine Version PM-tool

```
Sub Paint_Phase

.....

ActiveSheet.PageSetup.PrintArea = "$C$2:$EK$" & (MAX_ZEILE)

' Bis zu 5 Phasen: Hochformat - sonst Querformat
With ActiveSheet.PageSetup
    If AnzahlPhasen < 6 Then
        .Orientation = xlPortrait
       Else
        .Orientation = xlLandscape
    End If
End With

    Application.PrintCommunication = False
    With ActiveSheet.PageSetup
        .FitToPagesWide = 1
        .FitToPagesTall = 99
    End With
    Application.PrintCommunication = True

Set_Pagebreaks

' Zoomstufe
'With ActiveSheet.PageSetup
'    If AnzahlPhasen < 6 Then .Zoom = 41
'    If AnzahlPhasen = 6 Then .Zoom = 41
'    If AnzahlPhasen = 7 Then .Zoom = 35
'    If AnzahlPhasen = 8 Then .Zoom = 31
```

```
'    If AnzahlPhasen = 9 Then .Zoom = 28
'    If AnzahlPhasen = 10 Then .Zoom = 24
'    If AnzahlPhasen = 11 Then .Zoom = 22
'    If AnzahlPhasen = 12 Then .Zoom = 18
'    If AnzahlPhasen = 13 Then .Zoom = 15
'    If AnzahlPhasen = 14 Then .Zoom = 13
'    If AnzahlPhasen >= 14 Then .Zoom = 10
'End With

Application.EnableEvents = True
ZU
End

End Sub

Sub Set_Pagebreaks()


Dim objCell As Range
Dim lngPageBreaks As Long

ActiveWindow.View = xlPageBreakPreview

On Error Resume Next

With ActiveSheet

.ResetAllPageBreaks
.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1

For lngPageBreaks = 1 To .HPageBreaks.Count

For Each objCell In .HPageBreaks(lngPageBreaks).Location.Offset(-5).Resize(5)

   If objCell.Height = 15 Then Exit For

Next

If Not objCell Is Nothing Then _
Set .HPageBreaks(lngPageBreaks).Location = objCell.Offset(1)

Next
End With
```

Set objCell = Nothing

End Sub

1.) Problem von Klappspaten

Hi Excel Profis

Ich habe mich daran versucht, Seitenumbrüche per VBA Code festzulegen.

Hintergrund ist ganz einfach, dass ich beim Ausdrucken der Tabellen keine 100, sondern nur die benötigten 10 - 20 Seiten ausdrucken will.

Ich habs mit folgendem Code (nur für ein Tabellenblatt) versucht:

Code:

```
Sub Seitenumbrüche Festlegen()

Dim i As long
Dim wks As Worksheet
Dim Range As Range
Dim Rows As Range
Dim Columns As Range
Dim Cells As Range
Dim Pg As Pages
Dim AlterOrt As Integer
Dim NeuerOrt As Integer
Dim Bereich As Range
Dim p As Integer


ActiveSheet.ResetAllPageBreaks
ActiveSheet.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1

For p = 1 To ActiveSheet.HPageBreaks.Count

AlterOrt = ActiveSheet.HPageBreaks(p).Location.Row

With ActivSheet
Set i = AlterOrt
Set Bereich = Application.Union(Cells(i, 1), Cells(i - 1, 1), Cells(i - 2, 1), Cells(i - 3, 1), Cells(i - 4, 1), Cells(i
- 5, 1), Cells(i - 6, 1), Cells(i - 7, 1), Cells(i - 8, 1), Cells(i - 9, 1), Cells(i - 10, 1), Cells(i - 11, 1), Cells(i
```

```
- 12, 1))
Bereich.Select
For Each Cell In Selection
If Cell.Interior.Color = xlThemeColorDark1 Then
Cell.Row = NeuerOrt
End If
Next Cell
Set ActiveSheet.HPageBreaks(p).Location.Row = NeuerOrt + 1
End With
Next p
End Sub
```

Der Code scheitert mit Fehlercode 91 beim Festlegen des Bereichs. Und ich hab keinen Plan, warum.

2.) Nachfrage von Nepomuk

Hallo,

ungeachtet gravierender Fehler, erschließen sich mir deine Gedanken welche du beim erstellen des Codes hegtest in keinster Weise.

Was bitte soll denn das werden? Bzw. was dachtest du dass das werden könnte?

Set Bereich = Application.Union(Cells(i, 1), Cells(i - 1, 1), Cells(i - 2, 1), Cells(i - 3, 1), Cells(i - 4, 1), Cells(i - 5, 1), Cells(i - 6, 1), Cells(i - 7, 1), Cells(i - 8, 1), Cells(i - 9, 1), Cells(i - 10, 1), Cells(i - 11, 1), Cells(i - 12, 1))

So wie ich das sehe nimmst du den 1. horizontalen Seitenumbruch und bildest von der Zeile aus einen Bereich aus den 12 Zeilen darüber. Das kannst du auch billiger haben, da das ein zusammenhängender Bereich ist.

Besser du erzählst mal was es werden sollte.

3.) Erklärung von Klappspaten

Hi Nepomuk

Zu Anfang setzte ich alle Seitenumbrücke zurück und lege den vertikalen Umbruch an den rechten Rand des Druckbereichs.

Mein Tabellenblatt ist eingeteilt in Blöcke zu 13 Zeilen, welche beim Ausdruck nicht durch Seitenumbrüche geteilt werden sollen.

Die automatischen Seitenumbrüche liegen in der Regel mitten in den Blöcken, jedoch maximal 12 Zeilen unter der Zeile, in welcher ich einen Seitenumbruch festlegen will.

Ich stelle fest, wo sich die Seitenumbrüche Nummer 1 bis (p) befinden (For p = 1 To ActiveSheet.HPageBreaks.Count).

Ich suche den ersten Umbruch (p = 1). Dessen Zeilenwert speichere ich in "AlterOrt" als Integer. Ich wähle den Bereich der 12 über dem Umbruch liegenden Zeilen aus, nur Spalte A. In diesem Bereich suche ich nach einer Zelle, welche als Innenfarbe "xlThemeColorDark1" hat. Wenn das so ist, speichere ich die Zeilennummer dieser Zelle als "NeuerOrt". Solche Zellen kann es (weil die Blöcke 13 Zeilen umfassen) im durchsuchten Bereich nur eine geben.

Dann verschiebe ich den Seitenumbruch(p) zur Zeile "NeuerOrt +1", denn die Blöcke, welche am Stück bleiben sollen, enden immer mit einer Zelle mit der Innenfarbe "xlThemeColorDark1".

Dann gehe ich zum nächsten Seitenumbruch(p), und so fort, bis der Wert (p) die Anzahl der Seitenumbrüche erreicht hat.

Der Druckbereich ist definiert von Zeile 1 bis 910.

Das ist im wesentlichen, was ich dabei tun will. Und Du hast sicher recht, dass ich einen zusammenhängenden Bereich einfacher auswählen kann (.Range(xy).select zum Beispiel). Bloss hat es auch noch andere Arbeitsblätter in der Mappe, bei denen der zu prüfende Bereich nicht zusammenhängt. Dort werde ich dann wohl oder übel die Union-Methode anwenden müssen. Weshalb also nicht hier? Ich will ja zuletzt dahin gelangen, die Methode für alle Worksheets in der Arbeitsmappe anwenden zu können.

Ich hoffe, diese Zusatzangaben helfen ein wenig, mein Code-Kauderwelsch ein wenig verständlicher zu machen.

Gruss, Peter

4.) Eigene neue Lösung von Klappspaten

Ich hab's mal ein wenig anders versucht (anderes Tabellenblatt mit Blöcken zu 18 Zeilen):

Code:

```
Sub Seitenumbrüche Festlegen()

Dim i As
Dim wks As Worksheet
Dim Range As Range
Dim Rows As Range
Dim Columns As Range
Dim AlterOrt As Long
Dim NeuerOrt As Long
Dim Bereich As Range
Dim p As Integer

Set wks = ActiveSheet

ActiveSheet.ResetAllPageBreaks
ActiveSheet.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1

With ActivSheet

For p = 1 To 30
```

```
On Error Resume Next
AlterOrt = ActiveSheet.HPageBreaks(p).Location.Row
wks.Range(Cells(AlterOrt - 1, 1), Cells(AlterOrt - 18, 1)).Select
For Each Cell In Selection
If Cell.Interior.Color = 7566195 Then
Cell.Activate
NeuerOrt = ActiveCell.Row
End If
Next Cell

Set ActiveSheet.HPageBreaks(p).Location = ActiveSheet.Cells(NeuerOrt + 1, 1)

Next p
End With
End Sub
```

Das haut so ziemlich gut hin. Bloss, die Auswahl wird eben jetzt über "Range(...).Select" getroffen. Das ist nach wie vor unbefriedigend. Es wäre schön, die Auswahl mit der Union-Methode treffen zu können. Hättest Du dazu einen Hinweis?

5.) Nepomuk Lösung 1

Hallo,

ein bisschen objektorientierter:

Code:

```
Public Sub Set Pagebreaks()

Dim objCell As Range
Dim lngPageBreaks As Long

ActiveWindow.View = xlPageBreakPreview

With ActiveSheet

.ResetAllPageBreaks
.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1

For lngPageBreaks = 1 To .HPageBreaks.Count

For Each objCell In .HPageBreaks(lngPageBreaks).Location.Offset(-12).Resize(12)
If objCell.Interior.Color = 7566195 Then Exit For
Next
```

```
If Not objCell Is Nothing Then
Set .HPageBreaks(lngPageBreaks).Location = objCell.Offset(1)

Next
End With

Set objCell = Nothing

End Sub
```

6.) Klappspaten fragt nach

Hi Nepomuk

Danke für den Code. Einiges eleganter. Und danke für die Ergänzung bezüglich der Umbruchvorschau als Ansicht.

Den kann ich sogar verstehen. Mit einer kleinen Ausnahme:

Code:

```
HPageBreaks(lngPageBreaks).Location.Offset(-12).Resize(12)
```

Den Befehl Resize in V.m. Offset zur Beschreibung eines Bereichs habe ich noch nie gesehen. Was tut <Resize>? Und zum Schluss die Frage, wie sähe die Syntax aus, wenn ich in mehr als einer Spalte suchen wollte. Ginge das damit auch?

Freundliche Grüsse, Peter

7.) Nepomuk antwortet

Hallo,

mit Offset(-12) geht ich, ausgehend von der Zeile mit dem Seitenumbruch, 12 Zeilen nach oben. Ist z.B. der Umbruch in Zeile 70, befinde ich mich damit in Zeile 58.

Mit Resize(12) erweitere ich diesen um 12 Zeilen. Um beim Beispiel zu bleiben, erstreckt sich dadurch der Bereich von Zeile 58 - 69.

Natürlich kannst du mit Resize auch die Anzahl der Spalten erweitern. Mit z.B. Resize(12, 5) würde sich der Bereich von A59 - E69 erstrecken.

8.) Nachfrage Klappspaten

Hi Nepomuk

Ungefähr so hab ich mir das gedacht. Schöne, kurze Methode. Thx für die Auskunft.

Bin noch über eine Kleinigkeit gestolpert: Wenn ich mit

Code:

```
For lngPageBreaks = 1 To .HPageBreaks.Count
```

die Variable bestimme, geht dem Ding der Saft aus. Die Seitenumbrüche werden alle nach oben verschoben. Dass heisst, die Anzahl der Seitenumbrüche verändert sich dynamisch und wird grösser. Das führt dazu, dass die For ... Next-Anweisung, weil die Zählung ja zu Beginn stattfindet und konstant bleibt, mitten in der Tabelle aufhört, obwohl da noch weitere Seitenumbrüche kommen.

Einfach zu lösen, indem man anstelle von .HPageBreaks.Count einen sehr hohen Wert eingibt, aber eben nicht mehr ganz so sophisticated. Siehst Du eine Möglichkeit, die Umbrüche nach Erreichen von .HPageBreaks.Count neuerlich zu zählen und das ganze für die weiteren Umbrüche weiter laufen zu lassen, z.B. mit einer Variablen lngPageBreaks2?

Gruss, Peter

9.) Finaler Code von Nepomuk der bei Klappspaten perfekt life

Hallo,

versuch es mal so (ungetestet). Setz vorsichtshalber einen Haltepunkt auf die Zeile mit Loop. Nicht das du in eine Endlosschleife läufst.

Code:

```
Public Sub Set Pagebreaks()

Dim objCell As Range
Dim lngPageBreaks As Long, lngStart As Long

ActiveWindow.View = xlPageBreakPreview

With ActiveSheet

.ResetAllPageBreaks
.VPageBreaks(1).DragOff Direction:=xlToRight, RegionIndex:=1
```

```
lngStart = 1

Do

For lngPageBreaks = lngStart To .HPageBreaks.Count

For Each objCell In .HPageBreaks(lngPageBreaks).Location.Offset(-12).Resize(12)
If objCell.Interior.Color = 7566195 Then Exit For
Next

If Not objCell Is Nothing Then
Set .HPageBreaks(lngPageBreaks).Location = objCell.Offset(1)

Next

lngStart = lngPageBreaks - 1

Loop While .HPageBreaks.Count > lngStart

End With

Set objCell = Nothing

End Sub
```

Seitenzahl eines Tabellenblattes auslesen

Lösung 1 für aktuelle Tabelle

```
MsgBox ActiveSheet.PageSetup.Pages.Count
```

Lösung 2 mit Funktion für variable Tabelle

```
Public Function Seitenanzahl(Tabelle As Worksheet)
Application.Volatile
Seitenanzahl = Tabelle.HPageBreaks.Count + 1
End Function
```

```
Sub test2()

MsgBox Seitenanzahl(K4_MA) ' K4_MA ist der VBA-Name einer Tabelle

End Sub
```

Lösung 3 - ein ganzes Inhaltsverzeichnis

```
Public Function Seitenanzahl(Tabelle As Worksheet)
Application.Volatile
Seitenanzahl = Tabelle.HPageBreaks.Count + 1
End Function

Sub INHALTSVERZEICHNIS_AKTUALISIEREN()

Dim SZ As Integer ' Seitenzahl

SZ = 1 + Seitenanzahl(Titel)
Range("H22") = "S." & SZ: SZ = SZ + Seitenanzahl(DIR_ORGA)
Range("H23") = "S." & SZ: SZ = SZ + Seitenanzahl(DIR_MA)
Range("H24") = "S." & SZ: SZ = SZ + Seitenanzahl(DIR_PK)
Range("H25") = "S." & SZ: SZ = SZ + Seitenanzahl(DIR_PENS)

Range("H27") = "S." & SZ: SZ = SZ + Seitenanzahl(K2_ORGA)
Range("H28") = "S." & SZ: SZ = SZ + Seitenanzahl(K2_MA)
Range("H29") = "S." & SZ: SZ = SZ + Seitenanzahl(K2_PK)
Range("H30") = "S." & SZ: SZ = SZ + Seitenanzahl(K2_PENS)

Range("H32") = "S." & SZ: SZ = SZ + Seitenanzahl(K3_ORGA)
Range("H33") = "S." & SZ: SZ = SZ + Seitenanzahl(K3_MA)
Range("H34") = "S." & SZ: SZ = SZ + Seitenanzahl(K3_PK)
Range("H35") = "S." & SZ: SZ = SZ + Seitenanzahl(K3_PENS)

Range("H37") = "S." & SZ: SZ = SZ + Seitenanzahl(K4_ORGA)
Range("H38") = "S." & SZ: SZ = SZ + Seitenanzahl(K4_MA)
Range("H39") = "S." & SZ: SZ = SZ + Seitenanzahl(K4_PK)
Range("H40") = "S." & SZ: SZ = SZ + Seitenanzahl(K4_PENS)

Range("H42") = "S." & SZ: SZ = SZ + Seitenanzahl(K5_ORGA)
Range("H43") = "S." & SZ: SZ = SZ + Seitenanzahl(K5_MA)
Range("H44") = "S." & SZ: SZ = SZ + Seitenanzahl(K5_PK)
Range("H45") = "S." & SZ: SZ = SZ + Seitenanzahl(K5_PENS)
```

End Sub

Lösung 4

```
Public Sub GetNumberOfPagesWithVBA()
  Dim nHBreaks As Integer
  Dim nVBreaks As Integer
  Dim nHPages As Integer
  Dim nVPages As Integer
  Dim nPagesTot As Integer

  If TypeName(ActiveWorkbook.ActiveSheet) = "Worksheet" Then
    With ActiveWorkbook.ActiveSheet
      nHBreaks = .HPageBreaks.Count
      nHPages = nHBreaks + 1
      nVBreaks = .VPageBreaks.Count
      nVPages = nVBreaks + 1
      nPagesTot = nHPages * nVPages
    End With
    MsgBox nPagesTot
  Else
    MsgBox "Das aktive Blatt ist kein Tabellenblatt!", _
          vbOKOnly + vbInformation
  End If
End Sub
```

Seitenzahl einer ganzen Mappe auslesen

```
Public Function NumberOfPagesInWB(ByVal WB As Workbook) _
     As Integer
  Dim nPagesTot As Integer
  Dim nPagesSht As Integer

  Dim sh As Worksheet

  Dim strWBName As String
  Dim strSheetName As String

  strWBName = "[" & WB.Name & "]"

  nPagesTot = 0
  For Each sh In WB.Worksheets
    strSheetName = strWBName & sh.Name
    nPagesSht = ExecuteExcel4Macro("Get.Document(50," & """" & strSheetName & """" & ")")
    nPagesTot = nPagesTot + nPagesSht
  Next sh
  NumberOfPagesInWB = nPagesTot
End Function

Public Sub GetNumberOfPagesInWB()

  MsgBox NumberOfPagesInWB(ActiveWorkbook)

End Sub
```

Seitenzahl in Zelle einfügen

```
Sub SeitenNr()
Dim Trennzeile As Variant
Dim AlteZeile As Integer
Dim Trennspalte As Variant
Dim AlteSpalte As Integer
Dim V_Seitenanzahl As Integer
Dim H_Seitenanzahl As Integer
Dim V_Seite As Integer
Dim H_Seite As Integer
V_Seitenanzahl = 0
```

```
V_Seite = 0
AlteZeile = 0
AlteSpalte = 0
Do
V_Seitenanzahl = V_Seitenanzahl + 1
Trennzeile = ExecuteExcel4Macro("INDEX(GET.DOCUMENT(64)," &
V_Seitenanzahl & ")")
If IsError(Trennzeile) Then Exit Do
If Trennzeile <= AlteZeile Then Exit Do
AlteZeile = Trennzeile
If Trennzeile >= ActiveCell.Row And V_Seite = 0 Then
V_Seite = V_Seitenanzahl
End If
Loop
V_Seitenanzahl = V_Seitenanzahl - 1
H_Seitenanzahl = 0
H_Seite = 0
Do
H_Seitenanzahl = H_Seitenanzahl + 1
Trennspalte = ExecuteExcel4Macro("INDEX(GET.DOCUMENT(65)," _
& H_Seitenanzahl & ")")
If IsError(Trennspalte) Then Exit Do
If Trennspalte <= AlteSpalte Then Exit Do
AlteSpalte = Trennspalte
If Trennspalte >= ActiveCell.Column And H_Seite = 0 Then
H_Seite = H_Seitenanzahl
End If
Loop
H_Seitenanzahl = H_Seitenanzahl - 1
If ActiveSheet.PageSetup.Order = xlOverThenDown Then
ActiveCell.Formula = "Seite " & (V_Seite - 1) * H_Seitenanzahl +
H_Seite & " von " & H_Seitenanzahl * V_Seitenanzahl
Else
ActiveCell.Formula = "Seite " & (H_Seite - 1) * V_Seitenanzahl +
V_Seite & " von " & H_Seitenanzahl * V_Seitenanzahl
End If
```

End Sub

Seitenzahl einer bestimmten Zelle ausgeben

Gesucht werden soll eine Zelle in der das Wort "Hallo" vorkommt und von ihr soll die Seitennummer ausgegeben werden

```
Sub seitennummer()
    Dim raZelle As Range
    Dim loHUmbruch As Long
    Dim loVUmbruch As Long
    Dim loSeite As Long
```

```
    Set raZelle = ActiveSheet.UsedRange.Find("Hallo")
    For loHUmbruch = 1 To ActiveSheet.HPageBreaks.Count
        If ActiveSheet.HPageBreaks(loHUmbruch).Location.Row > raZelle.Row Then Exit For
    Next loHUmbruch
    For loVUmbruch = 1 To ActiveSheet.VPageBreaks.Count
        If ActiveSheet.VPageBreaks(loVUmbruch).Location.Column > raZelle.Column Then Exit For
    Next loVUmbruch
    If ActiveSheet.PageSetup.Order = xlDownThenOver Then
        If loVUmbruch = 1 Or loHUmbruch = 1 Then
            If loVUmbruch = 1 Then
                loSeite = loHUmbruch
            Else
                loSeite = (ActiveSheet.HPageBreaks.Count + 1) * (loVUmbruch - 1) + loHUmbruch
            End If
        Else
            If loHUmbruch = 1 Then
                loSeite = loVUmbruch
            Else
                loSeite = (ActiveSheet.HPageBreaks.Count + 1) * (loVUmbruch - 1) + loHUmbruch
            End If
        End If
    Else
        If loVUmbruch = 1 Or loHUmbruch = 1 Then
            If loHUmbruch = 1 Then
                loSeite = loVUmbruch
            Else
                loSeite = (ActiveSheet.VPageBreaks.Count + 1) * (loHUmbruch - 1) + loVUmbruch
            End If
        Else
            If loVUmbruch = 1 Then
                loSeite = loHUmbruch
            Else
                loSeite = (ActiveSheet.VPageBreaks.Count + 1) * (loHUmbruch - 1) + loVUmbruch
            End If
        End If
    End If
    MsgBox "Seite " & loSeite
End Sub
```

SHEET_CHANGE etc deaktivieren

Ich habe früher das rekursive Aufrufen der Selection_Change bzw. Worksheet_Change Prozeduren immer mit einer Variable VBA-CODE ausgetrickst, wenn mein VBA-Code selbst eine Auswahl oder Zelle änderte, aber ich nicht wollte, dass dadurch die betreffenden Selection_Change bzw. Worksheet_Change Prozeduren aufgerufen wurden, indem ich an deren Beginn immer gleich abrief: If VBA-CODE=1 then exit sub.

Das manuelle Aussteigen aus Worksheet-Events ist gar nicht nötig, denn man kann das Ausführen aller automatischen vorübergehend Worksheet-Events deaktivieren mit

Application.EnableEvents = False
Danach bitte wieder Aktiv schalten mit
Application.EnableEvents = True

Wichtig: der EnableEvents-Parameter bleibt auch nach Ende des VBA-Codes erhalten ! Daher immer nach dem Ausschalten wieder ans Einschalten denken - vor allem auch wenn innerhalb der Prozedur ein END oder EXIT SUB-Befehl kommt !

Eventuell möchte man die Excel-Ereignisse deaktivieren - also, dass etwa das BEFORE_SAVE oder das WORKSHEET_CHANGE oder andere Ereignisse NICHT erfolgen (also der betreffende VBA-Code dieser Ereignisse nicht ausgeführt wird)

```
Application.EnableEvents = True ' aktiviert
Application.EnableEvents = False ' deaktiviert
```

In diesem Beispiel werden Ereignisse vor dem Speichern einer Datei deaktiviert, sodass das Ereignis **BeforeSave** nicht auftritt.
*Application.**EnableEvents** = False*
ActiveWorkbook.Save
*Application.**EnableEvents** = True*

SHEETCHANGE / SELECTION CHANGE

SHEETCHANGE

WICHTIG:

1.) arbeitet man in "Diese Arbeitsmappe", dann gilt der code dort für ALLE Seiten
entsprechend lautet der Aufruf auch Private Sub Workbook_SheetChange

2.) arbeitet man jedoch mit Code nur in einem bestimmten Tabellenblatt, heißt der Code
Private Sub Worksheet_Change(ByVal Target As Range)

WEITERS: man arbeitet vor allem mit zwei Abfragen:

a.) **ActiveCell:** mit ActiveCell.Row oder ActiveCell.Column liest man die Daten der aktuellen Zelle aus - dies ist aber abhängig davon, wohin Excel nach dem drücken der ENTER-Taste springt - also ob nach rechts oder nach unten - daher besser:

b.) **Target.Address / Target.Row / Target.Column (Ergebnis ist eine Zahl) / Target.Value**: hier drin wird gespeichert, welche Zelle gerade selektiert (also wichtig für SELECTIONCHANGE) wurde bzw geändert (also wichtig für SHEETCHANGE) wurde

Wichtig: ich hatte immer wieder Probleme, wenn ich Abfragen machte wie If Target.Value <> 0
Lösung: die Abfragen liefen problemlos durch, wenn man stattdessen " If cells(Target.row, Target.column) <>0

Man kann hier nur nach Spalten (aber auch nach Zeilen) fragen:

```
If Left(Target.Address, 2) = "$H" Or Left(Target.Address, 2) = "$J" Then ' wenn ein Wert in der Spalte H oder J geändert wurde
```

Oder nach konkreten Zellen:

```
    If Target.Address = "$H$11" Or Target.Address = "$J$11" Then ' wenn H11 oder J11 geändert wurde
```

TIPP:
MsgBox Target.Row ' direkt die Zeile der geänderten Zelle
MsgBox Target.Value ' direkt den Zellinhalt der geänderten Zelle anzeigen

TIPP2: möchte man bei Auswahl mehrer Zellen den Selection_Change-Code abbrechen lassen (weil er nur laufen soll, wenn man nur eine Zelle markiert hat), dann fügt man diesen Befehl ein

```
If Selection.Columns.Count > 1 Or Selection.Rows.Count > 1 Then Exit Sub
```

ad 1.) GESAMTE ARBEITSMAPPE

BSP SIMPLEFIBU

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)

' Überprüfen, ob Betragseingaben in Spalte C und D auch wirklich einen Betrag darstellen

    If Target.Address = "$C$" & ActiveCell.Row Or Target.Address = "$D$" & ActiveCell.Row Then

      If Cells(ActiveCell.Row, 3).Value <> "" And CDbl(Cells(ActiveCell.Row, 3).Value) = 0 Or Cells(ActiveCell.Row, 4).Value <> "" And CDbl(Cells(ActiveCell.Row, 4).Value) = 0
Then
```

BSP STUNDENLISTE Uhrzeit 800 oder 8 auf 8:00 umwandeln

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)

Dim A As String ' Target.Adress

If Val(Left(ActiveSheet.Name, 1)) > 0 Then ' wenn aktuelles Tabellenblatt mit einer Zahl beginnt - daher eines der 12 Monatstabellen ist

   ' Einschränkung auf bestimmte Spalten
   A = Mid(Target.Address, 2, 1) ' die Spalte
   If A = "E" Or A = "F" Or A = "G" Or A = "H" Or A = "K" Or A = "L" Or A = "M" Then ' wenn ein Wert in den Spalten E-H oder K-M geändert wurde

      ' Einschränkung auf bestimmte Zellen
      A = Right(Target.Address, 2) ' Zeilennummern
      If Val(A) > 13 And Val(A) < 35 Then ' wenn ein Wert zwischen den Zeilen 13 und 35 geändert wurde
         If Target.Value > 1 Then ' wenn eine Zeit ohne Doppelpunkt eingegeben wird (Doppelpunktzeiten werden ja in Teilen eines Tages gespeichert und 24:00 wäre 1)
            If Len(Target.Value) = 4 Then
               Target.Value = Left(Target.Value, 2) & ":" & Right(Target.Value, 2)
            End If
            If Len(Target.Value) = 3 Then
               Target.Value = Left(Target.Value, 1) & ":" & Right(Target.Value, 2)
            End If
            If Len(Target.Value) = 2 Then
               Target.Value = Left(Target.Value, 2) & ":" & "00"
            End If
            If Len(Target.Value) = 1 Then
               Target.Value = Left(Target.Value, 1) & ":" & "00"
            End If
         End If
      End If
   End If
End If
End Sub
```

ad 2.) NUR IM AKTUELLEN TABELLENBLATT

Arbeitet man mit nachfolgendem ActiveCell.Row, so wird die Adresse der aktuellen Zelle NACH der Zelländerung ausgelesen. Sprang Excel nach einer Zelländerung eine Zeile tiefer, dann ist die aktuelle Zelle eins tiefer als die zu überwachende Zelle. Besser geht es daher mit der nachfolgenden Variante mit Target.Address

```
Private Sub Worksheet_Change(ByVal Target As Range)

   If ActiveCell.Row = 11 or ActiveCell.Row = 12 Then ' Änderung in Zeile 11 führen bei Zellsprungrichtung nach unten zu einem Wert 12
      If Left(Target.Address, 2) = "$H" Or Left(Target.Address, 2) = "$J" Then ' wenn die Zelle H11 oder J11 geändert wurde
         MsgBox Target.Address ' zeigt an H11 oder J11
```

```
    End If
  End If

End Sub
```

BSP zwei (schöner)

```
Private Sub Worksheet_Change(ByVal Target As Range)

    If Target.Address = "$H$11" Or Target.Address = "$J$11" Then ' wenn die Zelle H11 oder J11 geändert wurde
      MAIN = Range("H11")
      REVBOOKS = Range("J11")
      MsgBox MAIN & REVBOOKS
    End If
  End If

End Sub
```

oder einen Spaltenbereich abfragen:

```
  If Left(Target.Address, 2) = "$T" And Right(Target.Address, 2) > 13 And Right(Target.Address, 2) < 45 Then ' wenn T14-T44 geändert wurde
      MsgBox Target.Address
  End If
```

SHEETSELECTIONCHANGE

1.) GESAMTE ARBEITSMAPPE

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)

' 0.) das Beschreiben der Datumszellen C4 und E4 nicht erlauben
    If ActiveCell.Address = "$C$4" Or ActiveCell.Address = "$E$4" Then


  ' 1.) ZEILENVORSCHUB: Weiterspringen am Ende einer Eingabezeile, wenn aktiviert
  If Sheets("EINSTELLUNGEN").Range("J4") <> "" Then
    If ActiveCell.Row > 9 Then
      If ActiveCell.Column = 11 And Columns("N:N").EntireColumn.Hidden = True Then
        ActiveCell.Offset(1, -10).Select

  ' 2.) MITWACHSEN DER TABELLE
  LETZTEZEILE = Range("F8").Value
  If Target.Address = "$A$" & LETZTEZEILE And Range("F7") <> "ZU" Then
```

```
' 2b) Springen bei keiner Eingabe der Steuer
    If ActiveCell.Row > 9 And LASTCELL = 8 And Target.Address = "$I$" & ActiveCell.Row And Cells(ActiveCell.Row, 8) <> "88" Then
```

ad 2.) NUR IM AKTUELLEN TABELLENBLATT

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

TIPP2: möchte man bei Auswahl mehrer Zellen den Selection_Change-Code abbrechen lassen (weil er nur laufen soll, wenn man nur eine Zelle markiert hat), dann fügt man diesen Befehl ein

```
' wenn mehr als eine Zelle markiert ist
If Selection.Columns.Count > 1 Or Selection.Rows.Count > 1 Then
   Exit Sub
End If
```

SHEETCALCULATE

Wie beim SheetChange und SelectionChange gibt es auch SheetCalculate sowohl 1.) für die gesamte Arbeitsmappe und 2.) auch für eine einzelne Tabelle.

SheetCalculate wird immer dann ausgeführt, wenn das betreffende - also die gesamte Arbeitsmappe oder die aktuelle Tabelle - von Excel neu berechnet wird.

Wichtig: Excel berechnet eine Tabelle / Mappe nur, wenn durch eine Eingabe

- eine Zelle mit einer Formel umgeändert wird
- oder eine Zelle geändert wird auf die mittels Formel in einer anderen Zelle zugegriffen wird.

Daher: wird in eine Zelle ein Wert (keine Formel) eingegeben, auf die sich keine Formel in einer anderen Zelle bezieht, dann wird SheetCalculate nicht ausgefüllt.

1.) Arbeitsmappe:

Private Sub Workbook_SheetCalculate(ByVal Sh As Object)

End Sub

2.) Tabellenblatt:

```
Private Sub Worksheet_Calculate()
   If Range("A1") = 1 Then MsgBox "JA"
End Sub
```

Im obigen Beispiel kann die Zelle A1 eine ganz normale Zelle sein ohne Formel und auch ohne dass sich eine andere Zelle mittels Formel auf diese Zelle bezieht.

Sortieren der Tabellenblätter in Arbeitsmappe (nach Name, Farbe)

0.) SORTIEREN NACH FARBE UND DANN NACH NAME (Meine Prozedur für Seibersdorf)

Man kann ja entweder Worksheets sortieren oder Sheets - Worksheets sortiert dann nur echte Tabellenblätter - Sheets (wie sie hier zum Einsatz kommen) sortieren alle Registerkarten in der Arbeitsmappe, sowohl echte Arbeitsblätter als auch Diagramme, die man als eigenständige Registerkarte angelegt hat (Diagramme müssen ja nicht zwingend eingebettet werden als Objekt in ein Tabellenblatt)

```
Sub SortWorksheetsByColorAndName()

    Dim i          As Long
    Dim j          As Long
    Dim ShC        As Long
    Dim lngSU      As Long

' --- saving state of Screenupdating

    With Application
       lngSU = .ScreenUpdating
       .ScreenUpdating = False
    End With

' --- adding color-number to sheetname

With ThisWorkbook

    For i = 1 To .Sheets.Count
       ShC = .Sheets(i).Tab.ColorIndex
       .Sheets(i).Name = CStr(ShC) & .Sheets(i).Name
    Next i

End With

' --- sorting by name

    For i = 1 To Sheets.Count
```

```
      For j = 1 To Sheets.Count - 1
         If UCase$(Sheets(j).Name) > UCase$(Sheets(j + 1).Name) Then ' replace > by < if sorting should be backwards
            Sheets(j).Move after:=Sheets(j + 1)
         End If
      Next j
   Next i

' --- removing color-number from sheetname

With ThisWorkbook

   For i = 1 To .Sheets.Count
      ShC = .Sheets(i).Tab.ColorIndex
      .Sheets(i).Name = Right(.Sheets(i).Name, Len(.Sheets(i).Name) - Len(CStr(ShC)))
   Next i

End With


' resetting Screenupdating
   Application.ScreenUpdating = lngSU
End Sub
```

1.) SORTIEREN NACH NAME

```
Sub SortSheets()
Dim i As Integer, j As Integer
For i = 1 To Sheets.Count
For j = 1 To Sheets.Count - 1
If UCase$(Sheets(j).Name) < UCase$(Sheets(j + 1).Name) Then
Sheets(j).Move after:=Sheets(j + 1)
End If
Next j
Next i
End Sub
```

Durch Änderung des < - Zeichens in ein > - Zeichen kann eine absteigende Sortierung erreicht werden.

2.) SORTIEREN NACH FARBE

BSP 1a: echte Tabellenblätter aber auch Blätter nur mit Diagrammen sortieren

Sub SortWorksheetsByColorAndName(Optional ByVal SortByAsc As Boolean = True)

```
    Dim i           As Long
    Dim j           As Long
    Dim ShtC()      As Long
    Dim ShtN()      As String
    Dim t, n        As Long
    Dim lngSU       As Long

' --- saving state of Screenupdating
    With Application
        lngSU = .ScreenUpdating
        .ScreenUpdating = False
    End With

' --- sorting
    If Val(Application.Version) >= 10 Then
        With ThisWorkbook
            For i = 1 To .Sheets.Count


                If .Sheets(i).Visible = -1 Then
                    n = n + 1
                    ReDim Preserve ShtC(1 To n)
                    ReDim Preserve ShtN(1 To n)
                    ShtC(n) = .Sheets(i).Tab.ColorIndex
                    ' MsgBox ShtC(n)
                    ShtN(n) = .Sheets(i).Name
                End If
            Next
            For i = 1 To n
                For j = i To n
                    If ShtC(j) < ShtC(i) Then
                        t = ShtN(i)
                        ShtN(i) = ShtN(j)
                        ShtN(j) = t
                        t = ShtC(i)
                        ShtC(i) = ShtC(j)
                        ShtC(j) = t
                    End If
                Next
            Next
            If SortByAsc Then
                For i = n To 1 Step -1
                    .Sheets(CStr(ShtN(i))).Move Before:=.Sheets(1)
```

```
            Next
         Else
            For i = n To 1 Step -1
               .Sheets(CStr(ShtN(i))).Move after:=.Sheets(.Sheets.Count)
            Next
         End If
      End With
   End If

' resetting Screenupdating
   Application.ScreenUpdating = lngSU
End Sub
```

BSP 1b: nur echte Tabellenblätter (Worksheets) sortieren

```
Sub SortWorksheetsByColor(Optional ByVal SortByAsc As Boolean = True)

    Dim i           As Long
    Dim j           As Long
    Dim ShtC()      As Long
    Dim ShtN()      As String
    Dim t, n        As Long
    Dim lngSU       As Long


    'Works XL 2002 or later(Tested only 2007)

    With Application
        lngSU = .ScreenUpdating
        .ScreenUpdating = False
    End With

    If Val(Application.Version) >= 10 Then
        With ThisWorkbook
            For i = 1 To .Worksheets.Count
                If .Worksheets(i).Visible = -1 Then
                    n = n + 1
                    ReDim Preserve ShtC(1 To n)
                    ReDim Preserve ShtN(1 To n)
                    ShtC(n) = .Worksheets(i).Tab.Color
                    ShtN(n) = .Worksheets(i).Name
                End If
            Next
            For i = 1 To n
                For j = i To n
                    If ShtC(j) < ShtC(i) Then
                        t = ShtN(i)
                        ShtN(i) = ShtN(j)
                        ShtN(j) = t
```

```
                        t = ShtC(i)
                        ShtC(i) = ShtC(j)
                        ShtC(j) = t
                    End If
                Next
            Next
            If SortByAsc Then
                For i = n To 1 Step -1
                    .Worksheets(CStr(ShtN(i))).Move before:=.Worksheets(1)
                Next
            Else
                For i = n To 1 Step -1
                    .Worksheets(CStr(ShtN(i))).Move after:=.Worksheets(.Worksheets.Count)
                Next
            End If
        End With
    End If

    Application.ScreenUpdating = lngSU
End Sub
```

Call the routine like

For ascending

```
SortWorksheetsByColor True
```

for descending

```
SortWorksheetsByColor False
```

BSP 2

```
Sub SortWorksheetsByColor()
Dim i As Long
Dim j As Long
Dim ShtC() As Long
Dim ShtN() As String
Dim t, n As Long
Dim lngSU As Long
With Application
```

```
lngSU = .ScreenUpdating
.ScreenUpdating = False
End With
If Val(Application.Version) >= 10 Then
With ThisWorkbook
For i = 1 To .Worksheets.Count
If .Worksheets(i).Visible = -1 Then
n = n + 1
ReDim Preserve ShtC(1 To n)
ReDim Preserve ShtN(1 To n)
ShtC(n) = .Worksheets(i).Tab.Color
ShtN(n) = .Worksheets(i).Name
End If
Next
For i = 1 To n
For j = i To n
If ShtC(j) < ShtC(i) Then
t = ShtN(i)
ShtN(i) = ShtN(j)
ShtN(j) = t
t = ShtC(i)
ShtC(i) = ShtC(j)
ShtC(j) = t
End If
Next
Next
If SortByAsc Then
For i = n To 1 Step -1
.Worksheets(CStr(ShtN(i))).Move before:=.Worksheets(1)
Next
Else
For i = n To 1 Step -1
.Worksheets(CStr(ShtN(i))).Move after:=.Worksheets(.Worksheets.Count)
```

```
Next
End If
End With
End If
Application.ScreenUpdating = lngSU
End Sub
```

BSP 3

```
Sub GroupSheetsByColor()


Dim lCount As Long, lCounted As Long
Dim lShtLast As Long
lShtLast = Sheets.Count
For lCount = 1 To lShtLast
For lCounted = lCount To lShtLast
If Sheets(lCounted).Tab.ColorIndex = Sheets(lCount).Tab.ColorIndex Then
Sheets(lCounted).Move Before:=Sheets(lCount)
End If
Next lCounted
Next lCount
End Sub
```

Tabellenschutz siehe ARBEITSMAPPEN Arbeitsmappe & Tabellenblatt schützen

Tabellenblatt aus-/ein-blenden

```
Tabelle7.Visible = xlSheetVisible
Tabelle7.Visible = xlHidden
Tabelle7.Visible = xlVeryHidden
```

Tabellenblatt in anderer Exceldatei ansprechen

Wenn man den Tabellennamen verlässlich weiß und dieser nicht vom User geändert werden konnte, geht es direkt so

```
Workbooks("Mappe1").Activate
Worksheets("Tabelle2").Activate
```

Oder auch direkt:

```
Workbooks("Mappe1").Worksheets("Tabelle2").Activate
```

Möchte man jedoch das Tabellenblatt mit seinem VBA-Code-Namen (zB VBANAME) ansprechen, muss dieser mit der folgenden Funktion zuerst herausgefunden werden

```
Function TabellenIndex(ByRef wkb As Workbook, ByVal strCodename As String) As Integer

    Dim wks As Worksheet
    For Each wks In wkb.Worksheets
      If wks.CodeName = strCodename Then
        TabellenIndex = wks.Index
        Exit Function
      End If
    Next wks

End Function

Sub test()

       Workbooks("Mappe1").Worksheets(TabellenIndex(Workbooks("Mappe1"), "VBANAME")).Activate

End Sub
```

Oder auch

```
' --- Variablen definieren ---

   Dim DM As Workbook
   Dim QM As Workbook

' --- Allgemeine Variablen setzen ---

   Set DM = ThisWorkbook ' Diese Arbeits-Mappe
   Set QM = ActiveWorkbook ' Quell-Arbeits-Mappe mit den ER-Buchdaten

' Kopieren der Buchungs-Daten in der Lieferantentabelle (VBA-Name TL)

   QM.Worksheets(TabellenIndex(QM, "TL")).Activate
   Range("C10:Z2000").Copy
```

Tabellenblatt in neu zu erzeugende Exceldatei kopieren

Das geht erstaunlich einfach nur mit dem Copy-Befehl.

```
Sheets("P&L PFDC").Copy
```

Tabellenblätter (mehrere) in neu zu erzeugende Exceldatei kopieren

VARIANTE 1

Worksheets(Array("Tabelle1", "Tabelle2")).Copy

es gehen auch die VBA-Tabellen-Namen

Worksheets(Array(DIR_ORGA.Name, DIR_MA.Name, DIR_PK.Name)).Copy

VARIANTE 2

```
Dim strSh As String, strDel As String, i As Long
strDel = "**##,"                              ' Eindeutiges Trennzeichen

For i = 3 To ThisWorkbook.Sheets.Count            ' Schleife über Blätter drei bis "ENDE"
   strSh = strSh & Sheets(i).Name & strDel        ' Blattnamen mit Trennzeichen aneinander reihen
Next
strSh = Left(strSh, Len(strSh) - Len(strDel))     ' letzte Trennzeichen entfernen

ThisWorkbook.Sheets(Split(strSh, strDel)).Copy    ' Die Blätter in eine leere Datei kopieren
```

Variante 3

```
With ThisWorkbook
   .Sheets("Tabelle1").Copy
   .Sheets("Tabelle2").Copy After:=ActiveWorkbook.Sheets(Sheets.Count)
End With
```

Tabellenblatt leeren (nur verwendeter Bereich)

```
' den gesamten genutzten Tabellenbereich ab A1 leeren
' dies geht nur, wenn das Tabellenblatt auch aktiviert ist

ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
```

Tabellenblatt speichern als Arbeitsmappe

Excel hat keinen direkten Befehl dafür - man muss das Blatt als eine neue Arbeitsmappe kopieren und danach speichern und wieder schließen.

--- VERSION NEU 2019 ---

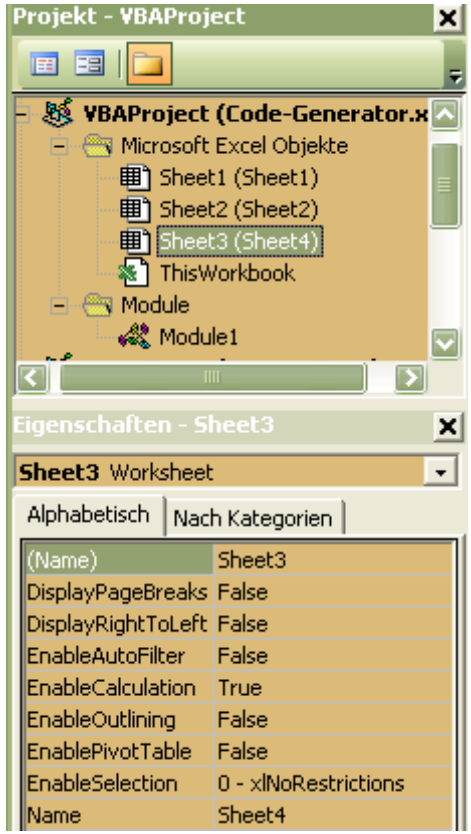
```
Sub Export()

' Der Exportbutton in den Filialtabellen, der die aktuelle Tabelle als Exceldatei speichert

Dim SPEICHERPFAD As String
Dim FORM As Shape ' Schaltflächen



' Auslesen Variablen
   SPEICHERPFAD = ActiveWorkbook.Path & "\" & ActiveSheet.Name & " " & Range("H1") & ".xlsx"
   MsgBox SPEICHERPFAD

' Keine Bildschirmaktualisierung (User bekommt kein Flackern mit)
   Application.ScreenUpdating = False

' Tabellenblatt als neue Arbeitsmappe kopieren
   ActiveSheet.Copy ' oder Sheets("Daten").Copy oder PARAMETER.Copy (VBA-Name)

' Löschen von eventuell vorhandenen Schaltflächen
   For Each FORM In ActiveSheet.Shapes ' Alle Formen (Bilder, Optionsschaltflächen, Kontrollkästchen, Wordart) löschen
         'If WORDART.Type = msoTextEffect Then ' wenn aktiviert, dann werden nur die Wordarts gelöscht, sonst alle Formen
            FORM.Delete
         'End If
```

```
    Next FORM '

' Arbeitsmappe speichern
    Application.DisplayAlerts = False ' nicht nachfragen nach Überschreiben / Ersetzen
    ActiveWorkbook.SaveAs Filename:=SPEICHERPFAD
    ActiveWorkbook.Close
    Application.DisplayAlerts = True

' Bildschirmaktualisierung wieder einschalten
    Application.ScreenUpdating = True


End Sub

--- VERSION ALT ---

Sub Druckliste_Exportieren()


' -------------------------------------------
'
' Erstellt 23.2.2014 - Simplesoft.at
'
' Funktion: Tabellenblatt mit Druckliste
'           als eigene Arbeitsmappe speichern
'
' -------------------------------------------

Dim Speicherpfad As String
Dim Dateiname As String
Dim Gesamtpfad As String


' Auslesen Variablen
    Speicherpfad = Tabelle7.Range("G5") & "\" & Tabelle7.Range("G20") & "\"
    Dateiname = Tabelle7.Range("G27")
    Gesamtpfad = Speicherpfad & Dateiname & ".xlsx"

' Keine Bildschirmaktualisierung (User bekommt kein Flackern mit)
    Application.ScreenUpdating = False

' Tabellenblatt als neue Arbeitsmappe kopieren
    Tabelle20.Copy
```

```
' Arbeitsmappe speichern
    Application.DisplayAlerts = False ' nicht nachfragen nach Überschreiben / Ersetzen
    ActiveWorkbook.SaveAs Filename:=Gesamtpfad
    ActiveWorkbook.Close
    Application.DisplayAlerts = True

' Bildschirmaktualisierung wieder einschalten
    Application.ScreenUpdating = True

 End Sub
```

Tabellenblattname soll Tabellencodenamen werden

Dass der Tabellenblattnamen nicht gleich dem Codenamen einer Tabelle ist sieht man hier schön: der Codenamen des 3. Blattes ist Sheet3 - und in Excel in der Arbeitsmappe heißt das Tabellenblatt Sheet4



Angesprochen wird das Blatt natürlich mit Sheet4 - zB

Worksheets("Sheet4").Activate

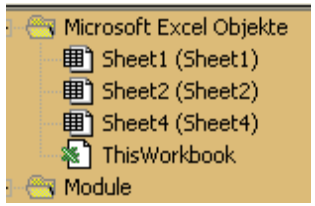
Wenn wir jedoch den Code des Tabellenblattes ändern wollen, müssen wir die Tabelle als Sheet3 ansprechen

Nachfolgender Code (aus dem Excel-VBA Codebook als PDF) wandelt die Tabellenblatt-Codenamen (hier sheet3) auf die Tabellenblattnamen (hier sheet4) um:

```
Sub ChangeAllCodeNames()
  Dim objSh As Object

  On Error Resume Next
  For Each objSh In ThisWorkbook.Application.Sheets
    With ThisWorkbook.VBProject.VBComponents(objSh.CodeName)
```

```
        .Properties("_CodeName").Value = objSh.Name
      End With
    Next
End Sub
```

Ergebnis:



Code, um den Codenamen eines einzelnen Tabellenblattes umzubenennen:

```
ThisWorkbook.VBProject.VBComponents("Tabelle1").Properties(5).Value = "KRola"
```

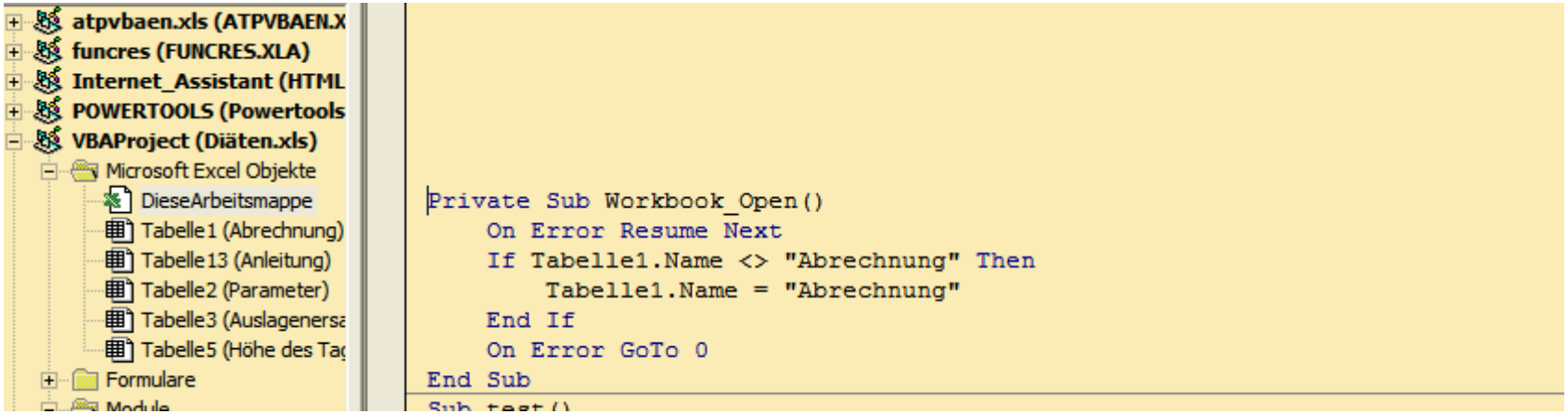
Tabellenblattname aus Zahl berechnen

WICHTIG: wenn man den Tabellenblattnamen aus einer Zahl berechnen möchte (zB es gibt für jede Kalenderwoche ein Tabellenblatt von 1 - 53), dann aufpassen, dass die STR-Funktion immer ein beginnendes Leerzeichen übergibt, das man wegschneiden muss

```
' Schreibschutz setzen

    Dim BLATTNAME As String
    For t = 1 To 53
        BLATTNAME = Mid(Str(t), 2, 2)
        Worksheets(BLATTNAME).Protect DrawingObjects:=True, Contents:=True, Scenarios:=True
    Next

' Schreibschutz entfernen

    Dim BLATTNAME As String
    For t = 1 To 53
        BLATTNAME = Mid(Str(t), 2, 2)
        Worksheets(BLATTNAME).Unprotect
    Next
```

Tabellenblattnamen-Änderung zurücksetzen

Wenn wir wollen, dass ein Tabellenblatt beim Öffnen der Datei immer zurückgesetzt wird auf den ursprünglichen Namen, dann machen wir das so:



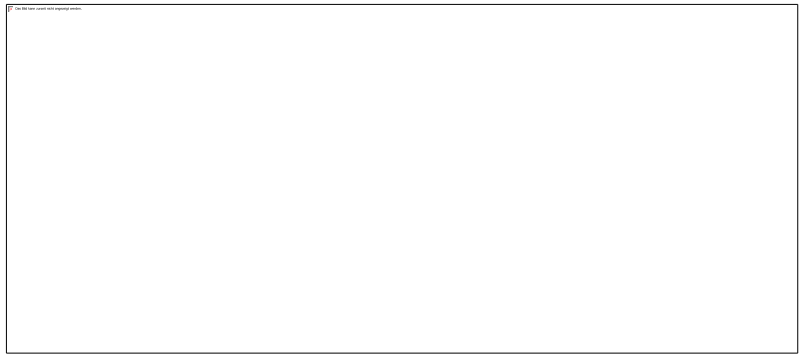
Ich habe das noch nicht unter englischem Excel getestet - aber ich denke, wenn eine Arbeitsmappe unter deutschem Excel erzeugt wird, dass die Tabellenblätte dann auch bei anschließendem englischen Excel immer Tabelle1 heißen und nicht sheet1. (Ansonsten einfach den Code für beide Varianten auslegen oder die Excelsprachversion auslesen)

Sicherheitswarnung beim Kopieren in Excel 2007 umgehen

When you manual copy a worksheet or worksheets from a **xlsm** workbook (with code) with macro's disabled.

1) Right click on a sheet tab
2) Choose "Move or Copy"
3) Change the "To Book" to (new book)
4) Check the "Create a Copy" check box
5) OK

You will see this security dialog



Manual this is no big problem but what if you have code in your personal.xlsb or add-in that copy a sheet or sheets to a new workbook.

The dialog only popup when you try to copy a sheet or sheets from a xlsm workbook (with code in it) with macro's disabled.Because there is no way to check if macros are enabled or not we must always use a workaround.

Here is a basic example to save the ActiveSheet in a new workbook with a date/time stamp in your Application.DefaultFilePath. This test macro exit the sub if your answer is NO in the security dialog

See the Tip below the macro about the SendKeys line.

```
Sub Copy_Sheet_Sheets_To_New_Workbook()
'Working in 97-2007
    Dim FileExtStr As String
    Dim FileFormatNum As Long
    Dim Sourcewb As Workbook
    Dim Destwb As Workbook
    Dim TempFilePath As String
    Dim TempFileName As String

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With
```

```
Set Sourcewb = ActiveWorkbook

'Copy sheet to a new workbook
ActiveSheet.Copy
Set Destwb = ActiveWorkbook

'Determine the Excel version and file extension/format
With Destwb
    If Val(Application.Version) < 12 Then
        'You use Excel 97-2003
        FileExtStr = ".xls": FileFormatNum = -4143
    Else
        'You use Excel 2007
        'We exit the sub when your answer is NO in the security dialog that you
        'see when you copy a sheet from a xlsm file with macro's disabled.
        If Sourcewb.Name = .Name Then
            With Application
                .ScreenUpdating = True
                .EnableEvents = True
            End With
            MsgBox "Your answer is NO in the security dialog"
            Exit Sub
        Else
            ' Note: if you are not sure if the sheet have VBA code then always
            ' save in xlsm or xlsb. it will delete the code if you save as xlsx
            FileExtStr = ".xlsm": FileFormatNum = 52
        End If
    End If
End With

'    'Change all cells in the worksheet to values if you want
'    With Destwb.Sheets(1).UsedRange
'        .Cells.Copy
'        .Cells.PasteSpecial xlPasteValues
'        .Cells(1).Select
'    End With
'    Application.CutCopyMode = False


'Save the new workbook and close it
TempFilePath = Application.DefaultFilePath & "/"
```

```
    TempFileName = "Part of " & Sourcewb.Name & " " & Format(Now, "dd-mmm-yy h-mm-ss")

    With Destwb
        .SaveAs TempFilePath & TempFileName & FileExtStr, FileFormat:=FileFormatNum
        .Close False
    End With

    MsgBox "You can find the file in " & TempFilePath & TempFileName & FileExtStr

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Tip:
Remember that SendKeys is not always reliable but Above **ActiveSheet.Copy**
you can try to add this code line **Application.SendKeys "y"**
It will close the Security Notice dialog with "yes".

Save in a different format

You can save also in a different format if you want
In the 97-2003 and the 2007 part you can use also this to save as txt, prn, csv

FileExtStr = ".txt": FileFormatNum = -4158
FileExtStr = ".prn": FileFormatNum = 36
FileExtStr = ".csv": FileFormatNum = 6

In the 2007 part you can also use

FileExtStr = ".xlsx": FileFormatNum = 51
FileExtStr = ".xlsb": FileFormatNum = 50

FileExtStr = ".xls": FileFormatNum = 56

51 = xlOpenXMLWorkbook (without macro's in 2007, xlsx)
52 = xlOpenXMLWorkbookMacroEnabled (with or without macro's in 2007, xlsm)
50 = xlExcel12 (Excel Binary Workbook in 2007 with or without macro's, xlsb)
56 = xlExcel8 (97-2003 format in Excel 2007, xls)

Spaltenbreite und Zeilenhöhe

Columns(ENDSPALTENNUMMER + 1).ColumnWidth = 3
Rows(1).RowHeight = 20

Statusleiste - Zwischenergebnisse andrucken

Wenn ich eine Schleife durch viele Quellzeilen habe und prozentuell anzeigen will, wie weit wir schon sind, dann geht das mit diesem Befehl vor dem Schleifen-Ende:

Application.StatusBar = Round((QZ - ERSTE_Z) / (LETZTE_Z - ERSTE_Z) * 100, 0)

QZ ist die Schleifenvariable, Erste_Z ist die erste Zeile und Letzte_Z ist die letzte Zeile der Schleife
Am Ende dreht man die Anzeige in der Statuszeile wieder ab mit

Application.StatusBar = False

1.) Besser ist es diese entweder in der Statusbar anzuzeigen

```
Sub Meldung()
 Application.StatusBar = "Hallo ich bin die Meldung in der Statuszeile"
End Sub
```

```
' Meldung deaktivieren
Sub Meldung_aus()
 Application.StatusBar = False
End Sub
```

2.) Eine Liste von Werten in das Direktfenster von VBA reinschreiben (mit STRG-G kann man es im VBA-Fenster öffnen oder im Menü ANSICHT)

Debug.print "Schreib war rein"

Zoomen des aktuellen Tabellenblattes

```
Sub Zoom_in()
   Cells(9, 11).Select
   ActiveWindow.Zoom = ActiveWindow.Zoom + 5
End Sub
Sub Zoom_out()
    Cells(9, 11).Select
    ActiveWindow.Zoom = ActiveWindow.Zoom - 5
End Sub
```

Zoomen für Dropdown-Listfelder und automatisches Öffnen

Die Schrift in Dropdown-Listfeldern kann ja in ihrer Größe nicht eingestellt werden. Abhilfe schafft ein Code, der beim Anwählen einer betreffenden Dropdown-Zelle, die Zoomstufe erhöht. Die aktuelle Zoomstufe muss vorher natürlich Interim-gespeichert werden (in unserem Code in Zelle A3) - und sobald man die Zelle abwählt, wird wieder auf diese ursprüngliche Zoomstufe zurückgezoomt.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)



   ' Automatisches Zoom beim Anwählen einer Zelle in einer Spalte, die ein Dropdownfeld enthält

   If Target.Columns.Count = 1 And Target.Rows.Count = 1 Then ' es soll nur greifen, wenn nur EINE Zelle markiert ist

      If Target.Column = 11 Then ' wenn man in Spalte J klickt

         If Sheets("Optionen").Range("C8") = True Then ' wenn das Zoomen aktiviert ist
            Range("A3") = ActiveWindow.Zoom ' speichern der aktuellen Zoomstufe in ausgeblendeter Zelle A3
            If Sheets("Optionen").Range("F8") > 100 Then
               ActiveWindow.Zoom = Sheets("Optionen").Range("F8")
              Else ' wenn jemand das Feld irrtümlich geleert hat
               ActiveWindow.Zoom = 200
            End If
            'Target.SpecialCells(xlCellTypeAllValidation).Activate ' dieser Code ist nicht notwendig - er würde auch das SelectionChange erneut aufrufen
            SendKeys "%{DOWN}" ' öffnet das Dropdownfeld
```

```
            Exit Sub
        End If
    End If

End If

' Zurücksetzen der Zoomstufe in allen anderen Fällen
If Range("A3") <> "" Then
        On Error Resume Next
        ActiveWindow.Zoom = Range("A3")
        Range("A3") = ""
        On Error GoTo 0
End If
```

T-ELEMENTE (Grafik, Button...)

Allgemein - mögliche Methoden bei Tabellenobjekten herausfinden

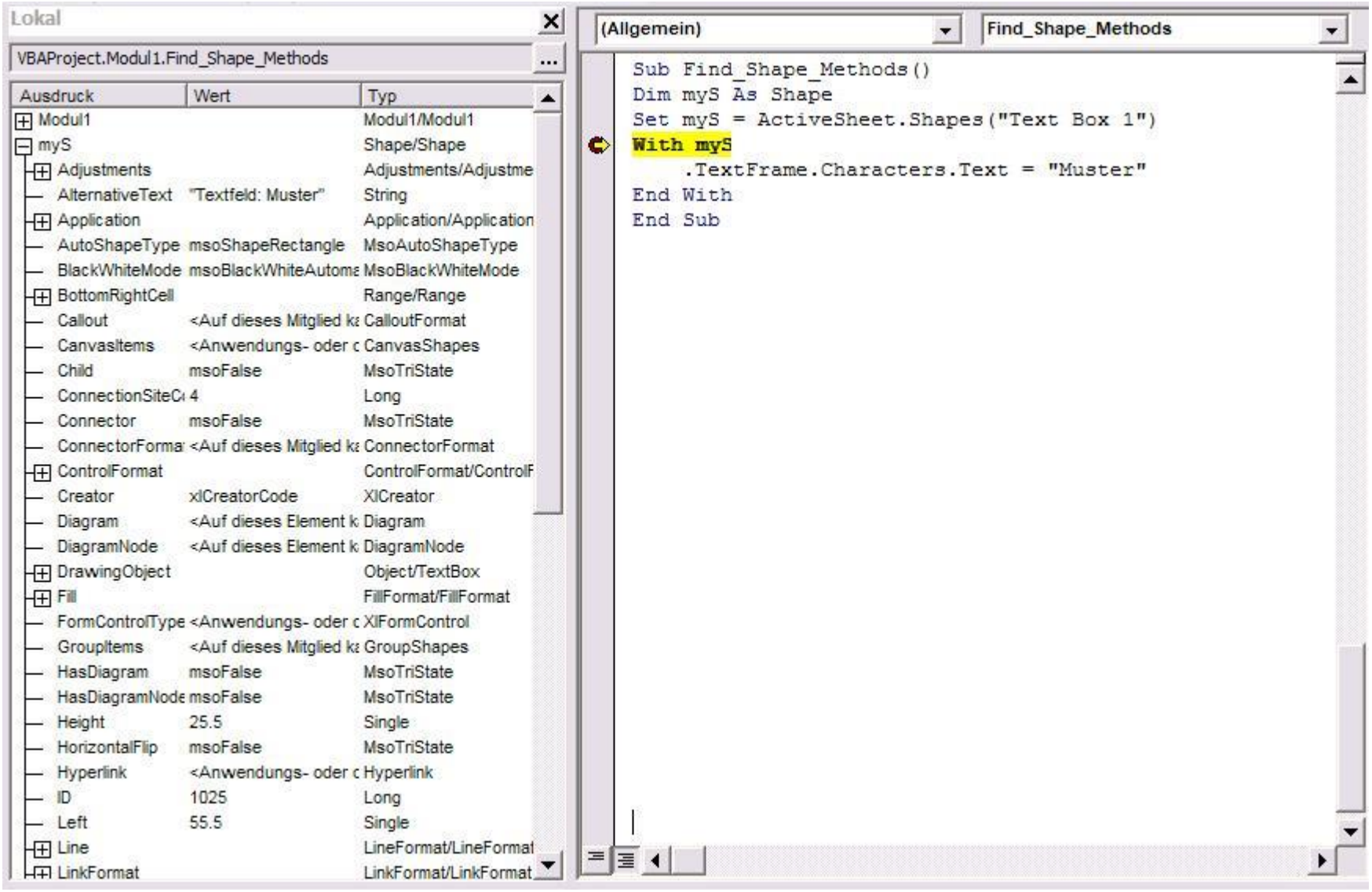
Die beste Möglichkeit herauszufinden welche Methoden auf ein bestimmtes Objekt angewendet werden können, ist diese Variante
Zeichnen Sie zB eine Textbox aus der Symbolleiste "Zeichnen" auf eine leeres Tabellenblatt in einer neuen leeren Mappe.

Fügen Sie dieses Makro in ein Modul, setzen Sie bei "With myS" einen Haltepunkt und blenden Sie anschliessend über "Ansicht-Lokal Fenster" die lokale Überwachung ein.

Starten Sie das Makro

```
Sub Find_Shape_Methods()
Dim myS As Shape
Set myS = ActiveSheet.Shapes("Text Box 1")
With myS
    .TextFrame.Characters.Text = "Muster"
End With
End Sub
```

Im Lokal Fenster können Sie nun alle Methoden ansehen, welche Sie auf das jeweilige Objekt anwenden können. Sie erhalten hier wesentlich mehr Informationen als über den Objekt-Katalog (aufzurufen über F2 im VB-Editor)



Alle Button-Schaltflächen (Autoformen/Shapes) löschen

```
Sub wegdamit()

Dim wks As Worksheet
Dim shp As Shape

Application.ScreenUpdating = False
For Each wks In Worksheets
    For Each shp In wks.Shapes
      shp.Delete
    Next shp
Next
Application.ScreenUpdating = True
End Sub
```

Alle Bilder, alle Wordart, Optionsfelder, Buttons, Kontrollkästchen in Tabelle kopieren, markieren, löschen

```
ActiveSheet.DrawingObjects.Copy ' or .Select or .Delete
```

Oder einzeln:

```
Dim BILD As Picture
Dim FORM As Shape
  For Each BILD In ActiveSheet.Pictures ' Alle Bilder löschen
        BILD.Delete
  Next BILD
  For Each FORM In ActiveSheet.Shapes ' Alle Formen (Bilder, Optionsschaltflächen, Kontrollkästchen, Wordart) löschen
        'If WORDART.Type = msoTextEffect Then ' wenn aktiviert, dann werden nur die Wordarts gelöscht, sonst alle Formen
           FORM.Delete
        'End If
  Next FORM '
```

Alle Bilder / Wordart in einem Tabellenblatt löschen

```
Sub ALLE_BILDER_LOESCHEN()

  Dim BILD As Picture

  For Each BILD In ActiveSheet.Pictures
        BILD.Delete
  Next BILD

End Sub
```

Bestimmtes BILD löschen:

```
   ActiveSheet.Shapes("Picture 1").Delete
```

Alle Wordart in einem Tabellenblatt löschen

```
Dim WORDART As Shape

  For Each WORDART In ActiveSheet.Shapes ' Alle Wordart löschen
        If WORDART.Type = msoTextEffect Then
           WORDART.Delete
        End If
  Next WORDART
```

Auswahl (Autoform/Button) aufheben

Um z.B. eine Auswahl eines Buttons / einer Autoform aufzuheben, reicht es eine andere Zelle zu aktivieren oder man sendet die ESC nach Excel:

Application.SendKeys ("{ESC}")

Autoformen - mein Code

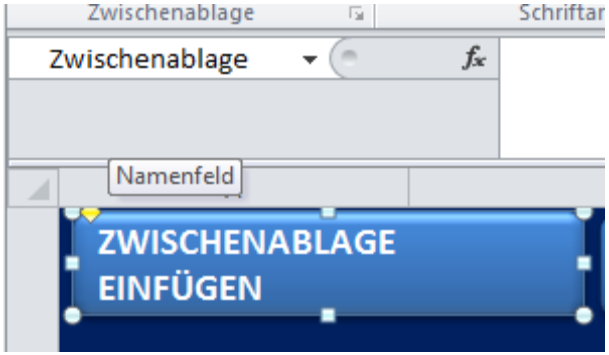
Mein TIPP : Arbeite mit dem Kopieren des Autoformen-Styles und übertrage ihn auf neu gezeichnete Autoformen (die ganze Autoform zu kopieren führt bei vielen Autoform-Kopiervorgängen leider immer in Excel zu nicht behebbaren Fehlern - braucht man z.B. viele Ellipsen: zeichne sie immer neu und kopier den Stil einer ausgeblendeten Vorlagenellipse (siehe hier: Autoformen Format übertragen) - oder zeichne ein weißes Rechteck und verschiebe die nicht benötigten Ellipsen in den Hintergrund und bei Bedarf hol sie wieder nach vorne, wodurch sie sichtbar werden (siehe hier: Autoformen formatieren, verschieben...)

Wichtig: Möchte man eine bereits existierende Autoform, deren Namen man kennt (z.B. Ellipse 1) erneut ansprechen, MUSS zuerst das Sheet-Objekt vorangestellt sein:
ActiveSheet.Shapes("Ellipse 1").ZOrder msoSendToBack oder
Worksheets("Orga").Shapes("Ellipse 1").ZOrder msoBringToFront

Wichtig: Ich zeichnete unter Excel2010 nur ein Viereck ohne Schatten / Spiegelung - und dennoch wurde es in einer speziellen Arbeitsmappe immer mit Schatten und Spiegelung gezeichnet - in allen anderen Vorlagen lief der Code richtig - Lösung: entferne Spiegelung und Schatten und mache dann rechten mausklick auf Autoform und sage "Als Standard speichern" - daher: Standardformatspeicherungen bei einem Typ greifen stärker als der eigentliche VBA-Code (ist leider so)

Autoform mit Text versehen

Angenommen ich habe eine schöne Autoform in Excel gezeichnet und ihr dann oben links den Namen Zwischenablage gegeben



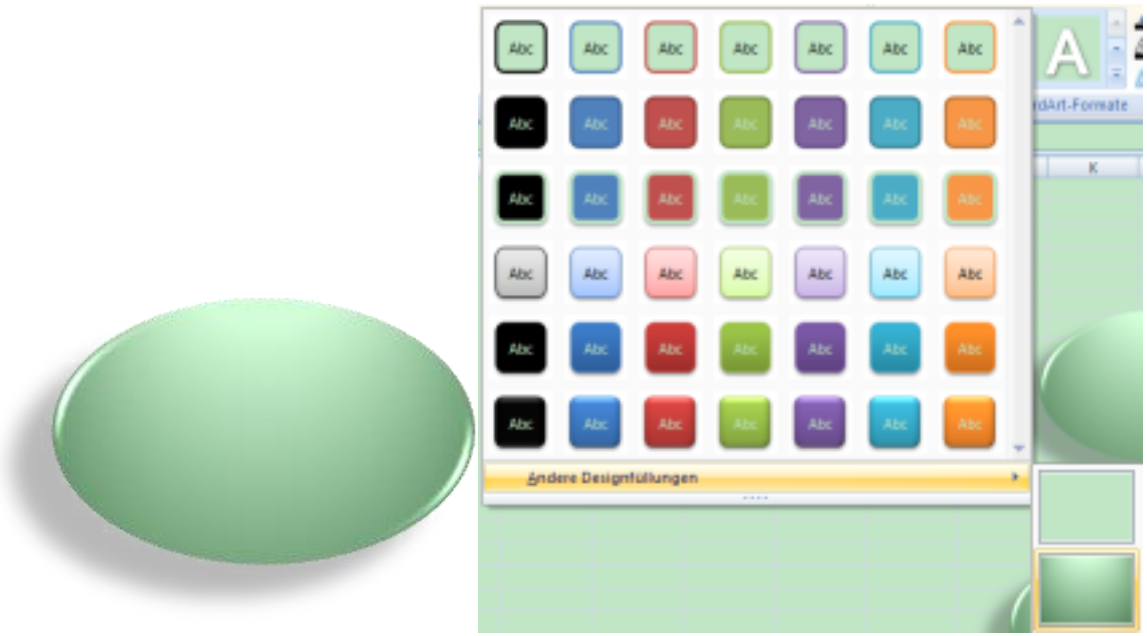

```
Sub text()
 ActiveSheet.Shapes("Zwischenablage").TextFrame2.TextRange.text = "Juhu"
End Sub
```

Weitere Befehle für das Textframe2-Element:

```
        .TextRange.Text = Cells(Z, 6) & "                    " & Cells(Z, 4)
        .MarginBottom = 0
        .MarginLeft = 0
        .MarginRight = 0
        .MarginTop = 0
        .TextRange.Characters.Font.Name = "Calibri"
        .TextRange.Characters.Font.Fill.ForeColor.RGB = RGB(255, 255, 255)
        .TextRange.Characters.Font.Size = 11

ActiveSheet.Shapes("Zwischenablage").TextFrame2.Orientation = msoTextOrientationHorizontal
```

Man kann ja eigentlich fast alles sehr gut mit VBA steuern. Leider geht der VBA-Recorder beim Erzeugen und Einstellen der Autoformen nicht.

Wenn man die Autoformen nicht immer voll per VBA einstellen will, macht man einmal die perfekte Autoform (z.B. Ellipse) in einer ausgeblendeten Tabelle und kopiert diese und fügt sie in der Tabelle an der gewünschten Stelle ein. Dieses Aussehen bekommt man nur versteckt im Excel-Formatmenü ANDERE DESIGNFÜLLUNGEN



Über das Problem mit den sich ändernden Farben (abhängig vom Windows-Desktop-Hintergrundweiß siehe "FARBEN eines Objekts einstellen"

Code für das Kopieren einer versteckten Autoform-Vorlage

```
Sub Test2()
Tabelle2.Shapes(1).Copy
Tabelle1.Range("A1").Select
Tabelle1.Paste
Selection.ShapeRange.IncrementLeft 10 ' verschieben nach rechts
Selection.ShapeRange.IncrementTop 10 ' verschieben nach unten

End Sub
```

Tipp: man kann es auch kombinieren: man erzeugt mit VBA die Autoform - aber hat eine Autoform-Vorlage und deren Format kopiert man mit folgendem Code:

```
Sub Format_Kopieren()
```

```
    Dim MYSHAPE As Shape

    Set MYSHAPE = ActiveSheet.Shapes.AddShape(msoShapeOval, 200, 200, 120, 70)

    Worksheets("Shapes").Shapes(1).PickUp
    MYSHAPE.Apply

End Sub
```

oder

```
Sub FormatSameAsFirst( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    For Each s In ws.Shapes
        ' Get formatting from first shape.
        ws.Shapes(1).PickUp
        ' Apply it to each shape.
        s.Apply
    Next
End Sub
```

Mein Code für Ellipsen-Zeichnen

Tipp: wie immer gibt es beide Möglichkeiten:
- entweder SET MYSHAPE ...SHAPEADD und dann direkt MYSHAPE.-Befehle
- oder einfach SHAPEADD.....SELECT und dann mit Selection.-Befehlen-Arbeiten

Besser ist ersteres, weil beim nachfolgenden Code muss man nur "MYSHAPE." eingeben und bekommt schon alle Möglichkeiten vorgezeigt in VBA (Selection.Shaperange.... hat keine Vorschlags-Anzeige)

```
Sub Test()
```

```
Call PAINT_OVAL(100, 200, 120, 80, "Stakeholder Name")


End Sub

Sub PAINT_OVAL(X, Y, B, H, BEZEICHNUNG)

Dim MYSHAPE As Shape

Set MYSHAPE = ActiveSheet.Shapes.AddShape(msoShapeOval, X, Y, B, H)

MYSHAPE.Select

    ' Verschieben nach rechts
    Selection.ShapeRange.IncrementLeft 84#

    ' Verschieben nach unten
    Selection.ShapeRange.IncrementTop 7.5

    ' Rotation
    Selection.ShapeRange.IncrementRotation 0 ' 30

    ' Füllung vorhanden
    Selection.ShapeRange.Fill.Visible = msoTrue

    ' Füllung einfärbig
    Selection.ShapeRange.Fill.Solid

    ' Füllung-Vordergrundfarbe auswählen aus Farbschema
    Selection.ShapeRange.Fill.ForeColor.SchemeColor = 11

    ' Füllung mit Verlauf
    MYSHAPE.Fill.ForeColor.RGB = RGB(120, 150, 100) ' oder: Selection.ShapeRange.Fill.ForeColor.RGB = RGB(120, 150, 100)
    MYSHAPE.Fill.BackColor.RGB = RGB(200, 230, 170)
    MYSHAPE.Fill.TwoColorGradient msoGradientHorizontal, 1

    ' 3D
    MYSHAPE.ThreeD.Visible = msoTrue
    MYSHAPE.ThreeD.Depth = 50#
    MYSHAPE.ThreeD.ExtrusionColor.RGB = RGB(50, 90, 20)


    ' Füllungtransparenz
```

```
    Selection.ShapeRange.Fill.Transparency = 0.51

    ' Rahmenlinie-Dicke
    Selection.ShapeRange.Line.Weight = 3#

    ' Rahmenlinien-Art
    Selection.ShapeRange.Line.DashStyle = msoLineSolid ' durchgängig
    Selection.ShapeRange.Line.Style = msoLineSingle

    ' Rahmenlinien-Transparenz und Sichtbarkeit
    Selection.ShapeRange.Line.Transparency = 0#
    Selection.ShapeRange.Line.Visible = msoTrue

    ' Rahmenlinien-Farben
    Selection.ShapeRange.Line.ForeColor.SchemeColor = 50
    Selection.ShapeRange.Line.BackColor.RGB = RGB(255, 255, 255)

    ' Bezeichnung-Steuerung1 über Texteffect (Excel 2003)
'    Selection.ShapeRange.TextEffect.FontName = "Arial"
'    Selection.ShapeRange.TextEffect.FontSize = 10
'    Selection.ShapeRange.TextEffect.Text = BEZEICHNUNG

' ---------------------------------------------------------------
    ' Bezeichnung-Steuerung2 über normales Textframe (Excel 2003)
    With MYSHAPE.TextFrame
         .Characters.Text = BEZEICHNUNG
         .Characters.Font.Name = "Calibri"
         .Characters.Font.Size = 12
         .HorizontalAlignment = xlHAlignCenter
         .VerticalAlignment = xlVAlignCenter
    End With

    ' ersten Buchstaben formatieren
     With MYSHAPE.TextFrame.Characters(Start:=1, Length:=1).Font
        .Name = "Arial"
        .FontStyle = "Fett"
        .Size = 40
        .ColorIndex = 40
     End With

     ' Restliche Buchstaben formatieren
     With MYSHAPE.TextFrame.Characters(Start:=2, Length:=7).Font
        .Name = "Arial"
        .Size = 20
```

```
      .ColorIndex = 2
    End With

    ' Text ausrichten
    With MYSHAPE.TextFrame
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
    End With
' ---------------------------------------------------------------

   ' Bezeichnung-Steuerung3 über Textframe2 (ab Excel 2007 und bessere Steuerungsmöglichkeiten als Textframe1)
   With MYSHAPE.TextFrame2
      .TextRange.Text = BEZEICHNUNG
      .MarginBottom = 0
      .MarginLeft = 0
      .MarginRight = 0
      .MarginTop = 0
      .TextRange.Characters.Font.Name = "Calibri"
      .TextRange.Characters.Font.Fill.ForeColor.RGB = RGB(1, 1, 1)
      .TextRange.Characters.Font.Size = 12
   End With
   If MYSHAPE.TextFrame2.HasText Then
      With MYSHAPE.TextFrame2.TextRange.Characters(1, 1).Font ' nur ab 1. Zeichen 1 Zeichen anders färben
         .Name = "Calibri"
         ' Sets color properly - i.e. no rounding like
         ' earlier versions of Excel.:
         .Fill.ForeColor.RGB = RGB(192, 80, 77)
         .Size = 14
         .Bold = True
         .Reflection.Type = msoReflectionType1
         .Shadow.Type = msoShadow14
         .Glow.Color.RGB = RGB(192, 137, 45)
         .Glow.Radius = 5
      End With
   End If

   ' Schatten
   Selection.ShapeRange.Shadow.Visible = msoFalse
   MYSHAPE.Shadow.Visible = msoFalse
   MYSHAPE.Shadow.Type = msoShadow1
   MYSHAPE.Shadow.ForeColor.RGB = RGB(10, 20, 20)

   ' Shapestyle auf Presets einstellen
   For I = 1 To 42
```

```
      MYSHAPE.ShapeStyle = msoShapeStylePreset & I
      MsgBox "Shapestyle " & I
   Next I
End Sub
```

Autoformen - Theorie 1

SEHR GUTE ZUSAMMENFASSUNG

Programming Excel 2007 and Excel 2010 AutoShapes with VBA (Guest Post)

Tuesday, January 12, 2010 by Jon Peltier

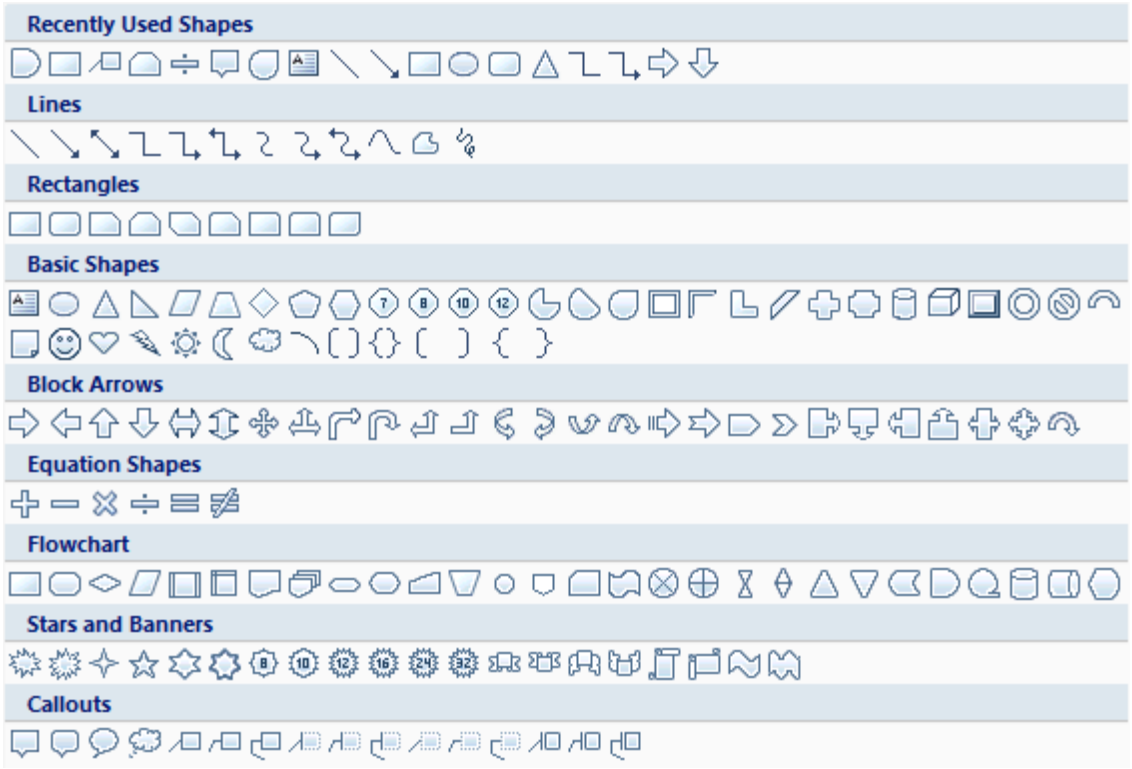
> *Today I am pleased to present a guest post written by my colleague Nicholas Hebb. Nick appears online frequently, commenting on Excel-related blogs and corresponding in the various forums.*

By Nicholas Hebb

With the release of Office 2007, Microsoft rewrote the drawing tools from the ground up. On the plus side, this meant new shapes, new styles, and the addition of SmartArt. On the downside, they didn't have time to incorporate AutoShape operations in their macro recorder prior to release. Thankfully, Excel 2010 has added macro support back, but like all macros the code generated is often bloated and relies heavily on the Selection object, which tends to hide the core objects in use. For the most part, the online help is good, but there is some information that doesn't get explained in detail. This article attempts to provide a basic overview of working with AutoShapes using VBA and touch on some areas that are not covered extensively in the help documentation.

Definitions

Two properties of the Shape object will be used in the code samples below – `Shape.Type` and `Shape.AutoShapeType`. Excel has a broad range of shape Types consisting not only of AutoShapes, but also connectors, lines, pictures, charts, comments, and many other graphical items. For AutoShapes, the AutoShapeType property lets you get/set the type of shape as shown in the gallery image below.



Knowing when to check the `Shape.Type` property versus the `Shape.AutoShapeType` is very useful. For example, if the AutoShapeType value is -2, then for all practical purposes the shape is not an AutoShape. If the value is greater than 1, then the shape is one of the types display in the Shapes gallery. The tricky part comes when the AutoShapeType is 1, which equals the AutoShape constant `msoShapeRectangle`. It could be a Rectangle AutoShape, but it could also be anything shaped like a rectangle, such as a text box, a comment, or even a picture. So if the AutoShapeType evaluates to 1, then you also need to check the Type property.

Callouts are another special type of shape that can cause confusion. They are discussed more in the Miscellaneous Issues section below.

Accessing a Shape Object

Each worksheet contains a Shapes collection consisting of Shape objects. Like other collections in VBA, the Shape object is accessed either via its name or index number, as in:

```
ActiveSheet.Shapes("SHAPE_NAME")
```

or

```
ActiveSheet.Shapes(1)
```

Or, using the For…Each syntax:

```
Dim shp as Shape
For Each shp in ActiveSheet.Shapes
  MsgBox shp.Name
Next
```

Adding an AutoShape

The syntax for adding a shape is:

```
Worksheet.Shapes.AddShape(AutoShapeType, Left, Top, Width, Height)
```

The AutoShapeType is a constant that ranges from 1 to 137 for Excel 2003 and earlier versions. Excel 2007 added shapes 139 through 183. AutoShapeTypes 125-136 are special AutoShapes. The online help file states that they support mouse over and click events, but that only applies when they are used in PowerPoint presentations. You can use them in Excel but they don't have any special properties.

To see what the AutoShapeType constant is for each AutoShape, you can copy and paste the following code into the Excel Visual Basic Editor and run it (or download the sample file and run the macro). Not all the AutoShapes are available in the Shapes gallery, so this will also give you a look at some of the hidden ones.

```
Sub CreateAutoshapes()
  Dim i As Integer
  Dim t As Integer
  Dim shp As Shape

  t = 10
  For i = 1 To 137
    Set shp = ActiveSheet.Shapes.AddShape(i, 100, t, 60, 60)
    shp.TextFrame.Characters.Text = i
    t = t + 70
  Next
  ' skip 138 - not supported
  If CInt(Application.Version) >= 12 Then
    For i = 139 To 183
      Set shp = ActiveSheet.Shapes.AddShape(i, 100, t, 60, 60)
      shp.TextFrame.Characters.Text = i
      t = t + 70
    Next
  End If
```

```
End Sub
```

The Left, Top, Width, and Height parameters of `AddShape()` are specified in points. The origin is the top left corner of the worksheet, with the Left and Top values increasing to the right and down, respectively. Dealing with points on a worksheet isn't intuitive, so if you prefer you can add a shape to a given range address by using code like this:

```
Function AddShapeToRange(ShapeType As MsoAutoShapeType, sAddress As String) As Shape
  With ActiveSheet.Range(sAddress)
    Set AddShapeToRange = ActiveSheet.Shapes.AddShape(ShapeType, _
        .Left, .Top, .Width, .Height)
  End With
End Function
```

Adding Text to an AutoShape

The Shape object has both a TextFrame and TextFrame2 members. The TextFrame2 member was added in Excel 2007 and gives better control over the formatting of the text. Because it is not backward compatible, I would recommend using the TextFrame object, as shown in the following code.

```
Sub AddFormattedTextToShape(oShape As Shape, sText As String)
  If Len(sText) > 0 Then
    With oShape.TextFrame
      .Characters.Text = sText
      .Characters.Font.Name = "Garamond"
      .Characters.Font.Size = 12
      .HorizontalAlignment = xlHAlignCenter
      .VerticalAlignment = xlVAlignCenter
    End With
  End If
End Sub
```

Setting Border and Fill Styles

If you take advantage of the built-in styles for Excel 2007 and Excel 2010, setting the AutoShape formatting is ridiculously easy compared to Excel 2003 and previous versions. Excel 2007 introduced the ShapeStyle property with the 42 preset styles shown below.

The style numbers can be set using a simple line of code:

```
Shape.ShapeStyle = msoShapeStylePresetXX
```

Where *Shape* is the shape object and *XX* is the style number. The style numbers are shown in the image gallery in order from left to right, top to bottom. For example, the red button in the second row `msoShapeStylePreset10`).

Adding Connectors and Lines

Connectors and lines are different objects in Excel. Connectors are special lines that "connect" to shapes, and if the shape is moved the connector stays connected and reroutes accordingly. Connectors cannot connect to other connectors, but they can connect to the end point of a line.

The syntax for adding a line is straightforward:

```
Worksheet.Shapes.AddLine(BeginX, BeginY, EndX, EndY)
```

…with all coordinates as Singles. Adding a connector is a bit more complex, since you typically want it to connect two shapes. The code below calculates the begin and end points, creates the connector, attaches the connector to the two shapes, then finally does a reroute to ensure the shortest path.

```
Function AddConnectorBetweenShapes(ConnectorType As MsoConnectorType, _
    oBeginShape As Shape, oEndShape As Shape) As Shape

  Const TOP_SIDE As Integer = 1
  Const BOTTOM_SIDE As Integer = 3
```

```
  Dim oConnector As Shape
  Dim x1 As Single
  Dim x2 As Single
  Dim y1 As Single
  Dim y2 As Single

  With  oBeginShape
    x1 = .Left + .Width / 2
    y1 = .Top + .Height
  End With
  With oEndShape
    x2 = .Left + .Width / 2
    y2 = .Top
  End With
  If CInt(Application.Version) < 12 Then
    x2 = x2 - x1
    y2 = y2 - y1
  End If

  Set oConnector = ActiveSheet.Shapes.AddConnector(ConnectorType, x1, y1, x2, y2)
  oConnector.ConnectorFormat.BeginConnect oBeginShape, BOTTOM_SIDE
  oConnector.ConnectorFormat.EndConnect oEndShape, TOP_SIDE
  oConnector.RerouteConnections

  Set AddConnectorBetweenShapes = oConnector

  Set oConnector = Nothing
End Function
```

Several points worth mentioning are:

- The ConnectorType can be one of three constants – `msoConnectorCurve`, `msoConnectorElbow`, or `msoConnectorStraight`.
- The calculations for the beginning and ending points are not normally needed. You could put any values in for the `AddConnector()` function because once you call `BeginConnect` and `EndConnect`, the connector is attached to the shapes and the begin and end points get set automatically.

- How the end coordinates are specified is not consistent between Excel versions. Prior to Excel 2007, the end coordinates were relative to the begin coordinates. Starting in Excel 2007, the function now uses absolute coordinates.
- When you route a Connector to an AutoShape, you need to specify the side using a connection site constant. The constants are different for each AutoShape type, but generally they start with the top side = 1 and go counter-clockwise. For example, most rectangular shapes have connection site constants where top = 1, left = 2, bottom = 3, and right = 4.
- When you call the `RerouteConnections()` function, it sets the connection sides automatically in order to create the shortest path between the two shapes. So, unless you want a specific routing, you can get usually just guess at the connection site values then call `RerouteConnections()`.

Formatting Connectors and Lines

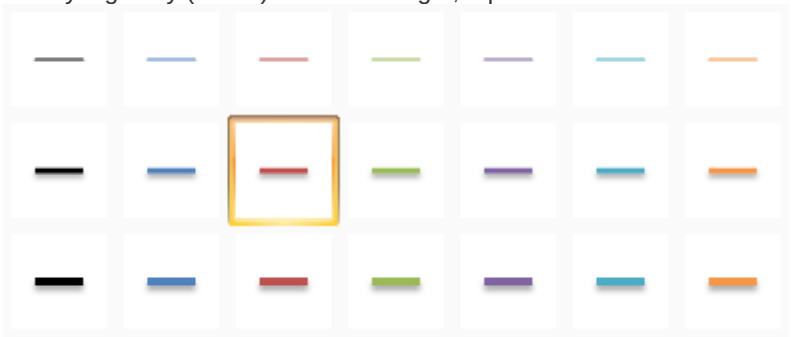
Like AutoShapes, formatting Connectors and Lines is fairly straightforward in Excel 2007 and 2010. Here is a comparison of two formatting routines for older versions of Excel versus the newer versions:

```
Sub FormatConnector2003(oConnector As Shape)
  With oConnector
    If .Connector Or .Type = msoLine Then
      ' rough approximation of the Excel 2007 preset line style #17
      .Line.EndArrowheadStyle = msoArrowheadTriangle
      .Line.Weight = 2
      .Line.ForeColor.RGB = RGB(192, 80, 77)
      .Shadow.Type = msoShadow6
      .Shadow.IncrementOffsetX -4.5
      .Shadow.IncrementOffsetY -4.5
      .Shadow.ForeColor.RGB = RGB(192, 192, 192)
      .Shadow.Transparency = 0.5
      .Visible = msoTrue
    End If
  End With
End Sub

Sub FormatConnector2007(oConnector As Shape)
  With oConnector
    If .Connector Or .Type = msoLine Then
      .Line.EndArrowheadStyle = msoArrowheadTriangle
```

```
      .ShapeStyle = msoLineStylePreset17
    End If
  End With
End Sub
```

The Connector property, used above, returns a Boolean indicating whether the shape is a connector. The `Type=msoLine` statement checks if the shape is a line. In this case the code will format both connectors and lines the same way, but at times you may want handle them separately. *(NB: The Insert Shapes gallery of Excel 2007 only lets you add Connectors, not Lines. So unless you are dealing with legacy files or add Lines via code, testing `Type=msoLine` may never be an issue for you.)*

Like the shape styles, you can format the line style by setting the ShapeStyle to one of the `msoLineStylePreset`*`XX`* values, where *`XX`* matches the order they appear in the style gallery (below) from left to right, top to bottom.



The Line object has several other members worth mentioning. In addition to the EndArrowheadStyle shown above, there is a corresponding BeginArrowheadStyle property, a DashStyle property, and also a Style property that lets you create double lines.

Miscellaneous Issues

Here are a few Excel 2007 issues with AutoShapes that are good to be aware of :

- If you copy a set of shapes, older versions of Office gave the option to Paste Special as editable AutoShapes in other Office applications. This option no longer exists in Office 2007. I haven't tested this in Office 2010 beta yet.
- In Excel 2007 and the 2010 beta, changing the AutoShape type will disconnect any incoming or outgoing Connectors to a shape.
- Some print drivers (including PDF export handlers) do not handle printing thick Connectors well, e.g., elbow connectors may straighten. If this happens, either change the line thickness or try grouping all the shapes together prior to printing.
- The Arc is an AutoShape but needs to be treated as a line when setting the ShapeStyle property.
- Most of the styles are backward compatible except for styles 37-42 (the glossy button look in the bottom row of the style gallery). Styles 31-36 (the second row from bottom) do not render very well in Excel 2000.

- You can add Callouts using the `AddShape()` or `AddCallout()` methods, but the `AddCallout()` method will only let you add four types, two of which are actually the same. Callouts have AutoShapeType values in the range of 105-124. Even though they have AutoShapeType values, the `Shape.Type` property will return `msoCallout` – not `msoAutoShape`. Confused? Wait, there's more. The callouts with AutoShapeTypes from 105-108 actually return a `Shape.Type = msoAutoShape` – not `msoCallout`.

Sample File

The sample file includes three demo sheets. The ShapeDemo sheet contains a macro to add two shapes, format them, then add a connector and format it. The Animation sheet has a simple macro showing how to move a shape around the sheet. The CreateAutoShapes sheet has a macro to create all AutoShapes available in your version of Excel.
The ShapeDemo routine has two function calls that are commented out – `IlluminateShapeText2003()` and `IlluminateShapeText2007()`. These subs add some gaudy formatting to the first letter of each text block, but they serve to highlight some of the differences between Excel 2007 and previous versions. Two parts of the code worth looking at are the HasText property and the formatting properties of TextFrame2. With the old TextFrame object, you would have to try accessing the `TextFrame.Characters.Count` property, which throws an error if no text exists. As for the formatting, the font colors in Excel 2003 and previous were limited to the colors in the pallet. In Excel 2007 and 2010, you can add reflections, drop shadows, and glow as well as set the color to whatever RGB value your heart desires.
Lastly, there is a module MCommon containing a sub that deletes all the AutoShapes on a sheet. In order not to delete the command buttons on the sheet (which are shapes too), it creates a ShapeRange of the AutoShapes and deletes that. The online help file shows the syntax for creating a ShapeRange when you know the names of the shapes at design time, but the syntax is a bit tricky when creating one dynamically. The `DeleteAllAutoShapes()` sub in the sample file shows you how to do this.

VBA-CODE DES SAMPLE-FILES

1.) SHAPE-DEMO

```
Option Explicit

Private m_Worksheet As Worksheet

Private Sub cmdClearShapes_Click()
    DeleteAllAutoShapes 'in module MCommon
End Sub

Private Sub cmdRunDemo_Click()

    Dim oShape1 As Shape
    Dim oShape2 As Shape
    Dim oConnector As Shape

    Set m_Worksheet = ActiveSheet

    If m_Worksheet.Shapes.Count > 2 Then
```

```
        DeleteAllAutoShapes
    End If

    Set oShape1 = AddShapeToRange(msoShapeFlowchartProcess, "B3:D5")
    AddFormattedTextToShape oShape1, "First Shape"

    Set oShape2 = AddShapeToRange(msoShapeFlowchartAlternateProcess, "B9:D11")
    AddFormattedTextToShape oShape2, "Second Shape"

    Set oConnector = AddConnectorBetweenShapes(msoConnectorStraight, oShape1, oShape2)

    If CInt(application.Version) < 12 Then
        FormatShape2003 oShape1
        FormatShape2003 oShape2
        FormatConnector2003 oConnector
        'IlluminateShapeText2003
    Else
        FormatShape2007 oShape1
        FormatShape2007 oShape2
        FormatConnector2007 oConnector
        'IlluminateShapeText2007
    End If

    Set oShape1 = Nothing
    Set oShape2 = Nothing
    Set oConnector = Nothing

    Set m_Worksheet = Nothing

End Sub

Private Function AddShapeToRange(ShapeType As MsoAutoShapeType, _
                                 sAddress As String) As Shape
    With m_Worksheet.Range(sAddress)
        Set AddShapeToRange = m_Worksheet.Shapes.AddShape(ShapeType, .Left, .Top, .Width, .Height)
    End With
End Function

Private Sub AddFormattedTextToShape(oShape As Shape, _
                                    sText As String)
    If Len(sText) > 0 Then
        With oShape.TextFrame
            .Characters.Text = sText
            .Characters.Font.Name = "Garamond"
            .Characters.Font.Size = 12
            .HorizontalAlignment = xlHAlignCenter
            .VerticalAlignment = xlVAlignCenter
        End With
    End If
End Sub

Private Function AddConnectorBetweenShapes(ConnectorType As MsoConnectorType, _
                                           oBeginShape As Shape, _
                                           oEndShape As Shape) As Shape

    ' NOTE: These connection site constants only work for rectangular shapes with
```

```
    '       4 connection points. The call to RerouteConnections below will
    '       automatically reroute the connector to the shortest path between the shapes.
    Const TOP_SIDE As Integer = 1
    Const LEFT_SIDE As Integer = 2
    Const BOTTOM_SIDE As Integer = 3
    Const RIGHT_SIDE As Integer = 4

    Dim oConnector As Shape
    Dim x1 As Single
    Dim x2 As Single
    Dim y1 As Single
    Dim y2 As Single

    With oBeginShape
        x1 = .Left + .Width / 2
        y1 = .Top + .Height
    End With

    With oEndShape
        x2 = .Left + .Width / 2
        y2 = .Top
    End With

    ' Excel 2007 uses absolute coordinates for the second point,
    ' of the AddConnector function. Previous versions of Excel
    ' use relative coordinates. But, ... (continued below)
    If CInt(application.Version) < 12 Then
        x2 = x2 - x1
        y2 = y2 - y1
    End If

    Set oConnector = m_Worksheet.Shapes.AddConnector(ConnectorType, x1, y1, x2, y2)

    ' ... you can use any positive Single values if you connect
    ' the end points with BeginConnect and EndConnect:
    oConnector.ConnectorFormat.BeginConnect oBeginShape, BOTTOM_SIDE
    oConnector.ConnectorFormat.EndConnect oEndShape, TOP_SIDE
    oConnector.RerouteConnections

    Set AddConnectorBetweenShapes = oConnector

    Set oConnector = Nothing

End Function

Private Sub FormatConnector2003(oConnector As Shape)
    If oConnector.Connector Or oConnector.Type = msoLine Then
        ' rough approximation of the Excel 2007 preset line style #17
        With oConnector
            .Line.EndArrowheadStyle = msoArrowheadTriangle
            .Line.Weight = 2
            .Line.ForeColor.RGB = RGB(192, 0, 0)
        End With
    End If
End Sub
```

```
Private Sub FormatConnector2007(oConnector As Shape)
    With oConnector
        If .Connector Or .Type = msoLine Then
            .Line.EndArrowheadStyle = msoArrowheadTriangle
            .ShapeStyle = msoLineStylePreset17
        End If
    End With
End Sub

Private Sub FormatShape2003(oShape As Shape)
    ' rough approximation of the Excel 2007 preset shape style #2
    With oShape
        .Fill.ForeColor.RGB = RGB(255, 255, 255)
        .Line.ForeColor.RGB = RGB(79, 129, 189)
        .Line.Weight = 2
        .Shadow.OffsetX = 0.8
        .Shadow.OffsetY = 0.8
        .Shadow.ForeColor.RGB = RGB(192, 192, 192)
        .Shadow.Transparency = 0.5
        .Shadow.Visible = msoTrue
    End With
End Sub

Private Sub FormatShape2007(oShape As Shape)
    oShape.ShapeStyle = msoShapeStylePreset2
End Sub

Private Sub IlluminateShapeText2003()
    On Error Resume Next

    Dim oShape As Shape
    Dim numChars As Integer

    For Each oShape In m_Worksheet.Shapes
        If oShape.Type = msoAutoShape And oShape.AutoShapeType > 0 Then
            ' Throws error when shape doesn't contain text:
            numChars = oShape.TextFrame.Characters.Count
            If Err.Number <> 0 Then
                Debug.Print oShape.Name
                Err.Clear
            ElseIf numChars > 0 Then
                With oShape.TextFrame.Characters(1, 1).Font
                    .Name = "Garamond"
                    ' The following does not work in Excel 2003 and below.
                    ' It rounds color to nearest preset.
                    .Color = RGB(192, 80, 77)
                    .Size = 24
                    .Bold = True
                End With
            End If
        End If
    Next
End Sub

Private Sub IlluminateShapeText2007()
    Dim oShape As Shape
```

```
    For Each oShape In m_Worksheet.Shapes
        If oShape.Type = msoAutoShape And oShape.AutoShapeType > 0 Then
            ' Does not throw error when shape doesn't contain text:
            If oShape.TextFrame2.HasText Then
                With oShape.TextFrame2.TextRange.Characters(1, 1).Font
                    .Name = "Garamond"
                    ' Sets color properly - i.e. no rounding like
                    ' earlier versions of Excel.:
                    .Fill.ForeColor.RGB = RGB(192, 80, 77)
                    .Size = 24
                    .Bold = True
                    .Reflection.Type = msoReflectionType1
                    .Shadow.Type = msoShadow14
                    .Glow.Color.RGB = RGB(192, 137, 45)
                    .Glow.radius = 5
                End With
            End If

        End If
    Next
End Sub
```

2.) AUTOSHAPE-DEMO

```
Option Explicit

Private Sub cmdCreateAutoshapes_Click()
    Dim i As Integer
    Dim t As Integer
    Dim shp As Shape

    If ActiveSheet.Shapes.Count > 2 Then
        DeleteAllAutoShapes
    End If

    Randomize
    t = 15
    For i = 1 To 137
        Set shp = ActiveSheet.Shapes.AddShape(i, 48, t, 96, 60)
        shp.TextFrame.Characters.Text = i
        If CInt(application.Version) >= 12 Then
            If i = 25 Then
                ' Treat as line
                shp.ShapeStyle = msoLineStylePreset1
            Else
                ' Randomly select a style
                shp.ShapeStyle = Int(Rnd() * 42 + 1)
            End If
        End If
        t = t + 75
    Next
    ' skip 138 - not supported
    If CInt(application.Version) >= 12 Then
```

```
    For i = 139 To 179
        Set shp = ActiveSheet.Shapes.AddShape(i, 48, t, 96, 60)
        shp.TextFrame.Characters.Text = i
        shp.ShapeStyle = Int(Rnd() * 42 + 1)
        t = t + 75
    Next
    ' These shapes don't have TextFrame's
    For i = 180 To 183
        Set shp = ActiveSheet.Shapes.AddShape(i, 48, t, 96, 60)
        t = t + 75
    Next
  End If

End Sub

Private Sub cmdDeleteShapes_Click()
  DeleteAllAutoShapes
End Sub
```

Autoform Beschriftung

Man kann ja mit Textframe hier arbeiten oder mit Textframe2 ab 2007 - letzteres erlaubt mehr Einstellungen

Wichtig: man kann in Textframes einen Zeilenumbruch mit vbCrLf übergeben oder stattdessen fügt man einfach viele Leerzeichen ein

Wie man Linksbündig / Zentriert einstellt, siehe Textbox / Textfeld erzeugen

```
    With Worksheets("Orga").Shapes("Ellipse 252").TextFrame2
        .TextRange.Text = Cells(Z, 6) & "                          " & Cells(Z, 4)
        .MarginBottom = 0
        .MarginLeft = 0
        .MarginRight = 0
        .MarginTop = 0
        .TextRange.Characters.Font.Name = "Calibri"
        .TextRange.Characters.Font.Fill.ForeColor.RGB = RGB(255, 255, 255)
        .TextRange.Characters.Font.Size = 11
    End With
```

Autoformen Format übertragen

```
Set MYSHAPE = ActiveSheet.Shapes.AddShape(msoShapeOval, X, Y, e0_Breite, e0_Hoehe)
' Format übertragen
worksheets("Shapes").Shapes(5).PickUp
MYSHAPE.Apply
```

Autoformen formatieren, verschieben, drehen, in den Hintergrund verschieben

Autoform in den Hintergrund verschieben

Man kann sie nicht direkt nach hinten verschieben, sondern muss sie zuerst auswählen, dann verschieben, dann abwählen

BSP hier für eine Checkbox

```
        ' Blende Häkchenfeld für Kapitel und Vers > 99 ein
        Worksheets("VERS1").Shapes("Vers1_K100").Select
        Selection.ShapeRange.ZOrder msoBringToFront

    Else
        ' Blende Häkchenfeld für Kapitel und Vers > 99 aus
        Worksheets("VERS1").Shapes("Vers1_K100").Select
        Selection.ShapeRange.ZOrder msoSendToBack

' Auswahl deaktivieren
Range("A1").Select
```

oder

```
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    Set s = ws.Shapes.AddLine(100, 100, 200, 200)
    s.Line.ForeColor.RGB = RGB(50, 75, 30)

    s.ZOrder msoSendToBack
```

Einfügen einer Rechteck-Autoform

```
ActiveSheet.Shapes.AddShape(msoShapeRectangle, 204.75, 167.25, 246.75, 132#). _
        Select
    Selection.ShapeRange.IncrementLeft 84#
    Selection.ShapeRange.IncrementTop 7.5
    Selection.ShapeRange.Fill.Visible = msoTrue
    Selection.ShapeRange.Fill.Solid
    Selection.ShapeRange.Fill.ForeColor.SchemeColor = 11
```

```
    Selection.ShapeRange.Fill.Transparency = 0.51
    Selection.ShapeRange.Line.Weight = 3#
    Selection.ShapeRange.Line.DashStyle = msoLineSolid
    Selection.ShapeRange.Line.Style = msoLineSingle
    Selection.ShapeRange.Line.Transparency = 0#
    Selection.ShapeRange.Line.Visible = msoTrue
    Selection.ShapeRange.Line.ForeColor.SchemeColor = 50
    Selection.ShapeRange.Line.BackColor.RGB = RGB(255, 255, 255)
```

Wichtig: Ich zeichnete unter Excel2010 nur ein Viereck ohne Schatten / Spiegelung - und dennoch wurde es in einer speziellen Arbeitsmappe immer mit Schatten und Spiegelung gezeichnet - in allen anderen Vorlagen lief der Code richtig - Lösung: entferne Spiegelung und Schatten und mache dann rechten mausklick auf Autoform und sage "Als Standard speichern" - daher: Standardformatspeicherungen bei einem Typ greifen stärker als der eigentliche VBA-Code (ist leider so)

Autoformen verschieben und drehen

Auswählen einer bestimmten Form

worksheets("Shapes").Shapes(1).Select

In diesem Beispiel wird die erste Form in `myDocument` dupliziert. Das Duplikat wird mit einer Füllung versehen, um 70 Punkt nach rechts und um 50 Punkt nach oben verschoben sowie um 30 Grad im Uhrzeigersinn gedreht.

Visual Basic für Applikationen

```
Set myDocument = Worksheets(1)

With myDocument.Shapes(1).Duplicate

    .Fill.PresetTextured msoTextureGranite

    .IncrementLeft 70

    .IncrementTop -50

    .IncrementRotation 30
```

```
End With
```

Ellipsen hinzufügen

BSP 1

```
Public Sub Bilder_Hochzählen() 'Neu erstellen u. gruppieren
  Dim objShp As Shape
  Dim iZahl As Integer
  With ActiveSheet
    For iZahl = 1 To 2
      Set objShp = .Shapes.AddShape(msoShapeOval, 10 + 10 * iZahl, 15 + 2 * iZahl, 40, 50)
      objShp.Name = "Oval" & iZahl

      With objShp.TextEffect
        .FontName = "Arial"
        .FontSize = 10
        .Text = iZahl
      End With
    Next iZahl
  End With
  Set objShp = Nothing
End Sub
```

BSP 2

```
Public Sub Bilder_Hochzhlen() 'Neu erstellen u. gruppieren
    Dim iZahl As Integer

    With ActiveSheet
       For iZahl = 1 To 5
          .Shapes.AddShape(msoShapeOval, 10 + 30 * iZahl, _
           10 * iZahl, 30, 30).Select
           Selection.Characters.Text = iZahl
          With Selection.ShapeRange.Fill
            .ForeColor.SchemeColor = 48
          End With
          With Selection.Characters.Font
            .Name = "Arial"
            .FontStyle = "Fett"
            .Size = 9
            .Superscript = False
            .ColorIndex = 2
         End With
       Next iZahl
     End With
 End Sub
```

Ändern der Eigenschaften einer Ellipsen-Autoform

BSP1

```
        Activesheet.Shapes(1).Select
        With Selection
           .ShapeRange.Fill.ForeColor.SchemeColor = FarbeLF
           .ShapeRange.line.Visible = msoFalse
           .ShapeRange.Shadow.Visible = msoFalse
           .Font.ColorIndex = FarbeLS
           .PrintObject = False
        End With
```

BSP2

```
Sub format_autoshape(form, hintergrundfarbe)

   activesheet.Shapes(form).Select
   With Selection
      .ShapeRange.Fill.ForeColor.SchemeColor = hintergrundfarbe
      .ShapeRange.line.ForeColor.SchemeColor = 58
      .ShapeRange.line.Visible = msoTrue
      .ShapeRange.Shadow.Visible = msoFalse
      .PrintObject = True
      .ShapeRange.TextFrame.MarginLeft = 0#
      .ShapeRange.TextFrame.MarginRight = 0#
      .ShapeRange.TextFrame.MarginTop = 0#
      .ShapeRange.TextFrame.MarginBottom = 0#
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
   End With
   With Selection.Font
      .Name = "Arial"
      .FontStyle = "Standard"
      .Size = 11
      .Strikethrough = False
      .Superscript = False
      .Subscript = False
      .OutlineFont = False
      .Shadow = False
      .Underline = xlUnderlineStyleNone
      .ColorIndex = 23
```

```
    End With
End Sub
```

Ändern von Text einer Autoform

Hallo,

ich habe in Excel über den Punkt Autoformen / Legenden eine Sprechblase in meine Exceltabelle eingefügt. Über ein VBA Script würde ich in dieser Sprechblase nun gerne einen Text ausgeben.

Leider habe ich dazu bisher nichts finden können, habt ihr vielleicht eine Idee?

```
ThisWorkbook.ActiveSheet.Shapes(1).TextFrame.Characters.Text = "Dies ist ein Text"
```

Solltest du mehrere Shapes haben, musst du den Index dementsprechend anpassen.

Autoform-Linien mit Pfeil machen zwischen Zellen

```
Sub pfeil2()
Dim lZeile As Long
For lZeile = 2 To 20
    If Not Cells(lZeile, 1) = "" Then
    ZelleA = Cells(lZeile, 1).Value
        ZelleB = Cells(lZeile, 2).Value
        Ax = Range(ZelleA).Left + Range(ZelleA).Width / 2
        Ay = Range(ZelleA).Top + Range(ZelleA).Height / 2
        Bx = Range(ZelleB).Left + Range(ZelleB).Width / 2
        By = Range(ZelleB).Top + Range(ZelleB).Height / 2
        ActiveSheet.Shapes.AddLine(Ax, Ay, Bx, By).Select
        Selection.ShapeRange.Line.EndArrowheadStyle = msoArrowheadTriangle
        Selection.ShapeRange.Line.EndArrowheadLength = msoArrowheadLengthMedium
        Selection.ShapeRange.Line.EndArrowheadWidth = msoArrowheadWidthMedium
        Selection.ShapeRange.Line.ForeColor.SchemeColor = Cells(lZeile, 4).Value
        Selection.ShapeRange.Line.Weight = Cells(lZeile, 3).Value
    End If
Next
End Sub
```

Autoform-Rechteck in Word rot färben und ohne Füllung

```
Sub Rahmen_Rot()
'
' Rahmen_Rot Makro

Selection.ShapeRange.Line.ForeColor.RGB = RGB(255, 0, 0)
Selection.ShapeRange.Fill.Visible = msoFalse
Selection.ShapeRange.Line.Weight = 3#


End Sub
```

Autoform-Rechteck in Word einfügen und rot färben und ohne Füllung

```
Sub ShapeEinfuegen_Rechteck()

    ' Fügt direkt ein und bleibt markiert und kann daher weiterverändert werden

    ActiveDocument.Shapes.AddShape(msoShapeRectangle, 100, 100, 300, 200).Select
    Selection.ShapeRange.Line.ForeColor.RGB = RGB(255, 0, 0)    ' rot färben
    Selection.ShapeRange.Fill.Visible = msoFalse                ' keine Füllung
    Selection.ShapeRange.Line.Weight = 3#                       ' Strichstärke

End Sub
```

Autoform-Linie mit Pfeil in Word einfügen und rot färben

```
Sub ShapeEinfuegen_Pfeil()

    ' Fügt Linie mit Pfeil ein

        ActiveDocument.Shapes.AddLine(100, 100, 200, 200).Select
        Selection.ShapeRange.Line.EndArrowheadStyle = msoArrowheadTriangle
        Selection.ShapeRange.Line.EndArrowheadLength = msoArrowheadLong
        Selection.ShapeRange.Line.EndArrowheadWidth = msoArrowheadWide
        Selection.ShapeRange.Line.ForeColor.RGB = RGB(255, 0, 0)    ' rot färben
        Selection.ShapeRange.Line.Weight = 2#


End Sub
```

Autoform-Linien

Alle Linien löschen

```
Sub Linien_loeschen()
Dim Shl As Object
For Each Shl In ActiveSheet.Shapes
  If Shl.Type = 9 Then
     Shl.Delete
  End If
Next
End Sub
```

FÄRBEN:

```
        Set L = ActiveSheet.Shapes.AddLine(ZX + X * NR, ZY + Y * NR, ZX + H * X, ZY + B * Y)
        L.Line.ForeColor.RGB = RGB(W / 360 * 200, 55, 200)
```

Linien zeichnen

Version 1

```
Dim ws As Worksheet, s As Shape
Set ws = ActiveSheet
Set s = ws.Shapes.AddLine(100, 100, 200, 200)
s.Line.ForeColor.RGB = RGB(50, 75, 30)
 s.ZOrder msoSendToBack ' gleich in den Hintergrund verschieben
```

Version 2

```
    ActiveSheet.Shapes.AddLine(153#, 82.5, 418.5, 291.75).Select
    Selection.ShapeRange.Fill.Transparency = 0#
    Selection.ShapeRange.Line.Weight = 0.75
    Selection.ShapeRange.Line.DashStyle = msoLineSolid
    Selection.ShapeRange.Line.Style = msoLineSingle
    Selection.ShapeRange.Line.Transparency = 0#
    Selection.ShapeRange.Line.Visible = msoTrue
    Selection.ShapeRange.Line.ForeColor.SchemeColor = 42
```

```
    Selection.ShapeRange.Line.BackColor.RGB = RGB(255, 255, 255)
    Selection.ShapeRange.Line.BeginArrowheadLength = msoArrowheadLengthMedium
    Selection.ShapeRange.Line.BeginArrowheadWidth = msoArrowheadWidthMedium
    Selection.ShapeRange.Line.BeginArrowheadStyle = msoArrowheadNone
    Selection.ShapeRange.Line.EndArrowheadLength = msoArrowheadLengthMedium
    Selection.ShapeRange.Line.EndArrowheadWidth = msoArrowheadWidthMedium
    Selection.ShapeRange.Line.EndArrowheadStyle = msoArrowheadNone
```

Version 3

```
Sub DrawLine( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Create label (height/width will be set by AutoSize).
    Set s = ws.Shapes.AddLine(100, 100, 200, 200)
    s.Line.DashStyle = msoLineDash
    s.Line.EndArrowheadStyle = msoArrowheadStealth
End Sub
```

Autoform auslesen der Typnummer

Es gibt die AutoshapeType-Bezeichnung und den Shapetype als Nummer

BSP

```
For Each objShape In ActiveSheet.Shapes
    'Alle Shapes des Tabellenblattes durchlaufen
    If objShape.AutoShapeType = msoShapeOval Or objShape.AutoShapeType = 1 Or objShape.Type = 9 Or objShape.AutoShapeType
= msoShapeRoundedRectangle Then ' Type 9=Linie AutoShapeType 1 ist Textfeld
        objShape.Delete
    End If
Next objShape
```

```
MsgBox Selection.ShapeRange.AutoShapeType

Ellipsen sind 9 (msoShapeOval)
Abgerundetes Rechteck ist 5
```

List of Shape Types and Shape Constants in Excel

Shape Type	Shape Constant
msoShapeMixed	-2
msoShapeRectangle	1
msoShapeParallelogram	2
msoShapeTrapezoid	3
msoShapeDiamond	4
msoShapeRoundedRectangle	5
msoShapeOctagon	6
msoShapeIsoscelesTriangle	7
msoShapeRightTriangle	8
msoShapeOval	9
msoShapeHexagon	10
msoShapeCross	11
msoShapeRegularPentagon	12
msoShapeCan	13
msoShapeCube	14
msoShapeBevel	15
msoShapeFoldedCorner	16
msoShapeSmileyFace	17

msoShapeDonut	18
msoShapeNoSymbol	19
msoShapeBlockArc	20
msoShapeHeart	21
msoShapeLightningBolt	22
msoShapeSun	23
msoShapeMoon	24
msoShapeArc	25
msoShapeDoubleBracket	26
msoShapeDoubleBrace	27
msoShapePlaque	28
msoShapeLeftBracket	29
msoShapeRightBracket	30
msoShapeLeftBrace	31
msoShapeRightBrace	32
msoShapeRightArrow	33
msoShapeLeftArrow	34
msoShapeUpArrow	35
msoShapeDownArrow	36
msoShapeLeftRightArrow	37
msoShapeUpDownArrow	38
msoShapeQuadArrow	39
msoShapeLeftRightUpArrow	40
msoShapeBentArrow	41
msoShapeUTurnArrow	42
msoShapeLeftUpArrow	43

msoShapeBentUpArrow	44
msoShapeCurvedRightArrow	45
msoShapeCurvedLeftArrow	46
msoShapeCurvedUpArrow	47
msoShapeCurvedDownArrow	48
msoShapeStripedRightArrow	49
msoShapeNotchedRightArrow	50
msoShapePentagon	51
msoShapeChevron	52
msoShapeRightArrowCallout	53
msoShapeLeftArrowCallout	54
msoShapeUpArrowCallout	55
msoShapeDownArrowCallout	56
msoShapeLeftRightArrowCallout	57
msoShapeUpDownArrowCallout	58
msoShapeQuadArrowCallout	59
msoShapeCircularArrow	60
msoShapeFlowchartProcess	61
msoShapeFlowchartAlternateProcess	62
msoShapeFlowchartDecision	63
msoShapeFlowchartData	64
msoShapeFlowchartPredefinedProcess	65
msoShapeFlowchartInternalStorage	66
msoShapeFlowchartDocument	67
msoShapeFlowchartMultidocument	68
msoShapeFlowchartTerminator	69

msoShapeFlowchartPreparation	70
msoShapeFlowchartManualInput	71
msoShapeFlowchartManualOperation	72
msoShapeFlowchartConnector	73
msoShapeFlowchartOffpageConnector	74
msoShapeFlowchartCard	75
msoShapeFlowchartPunchedTape	76
msoShapeFlowchartSummingJunction	77
msoShapeFlowchartOr	78
msoShapeFlowchartCollate	79
msoShapeFlowchartSort	80
msoShapeFlowchartExtract	81
msoShapeFlowchartMerge	82
msoShapeFlowchartStoredData	83
msoShapeFlowchartDelay	84
msoShapeFlowchartSequentialAccessStorage	85
msoShapeFlowchartMagneticDisk	86
msoShapeFlowchartDirectAccessStorage	87
msoShapeFlowchartDisplay	88
msoShapeExplosion1	89
msoShapeExplosion2	90
msoShape4pointStar	91
msoShape5pointStar	92
msoShape8pointStar	93
msoShape16pointStar	94
msoShape24pointStar	95

msoShape32pointStar	96
msoShapeUpRibbon	97
msoShapeDownRibbon	98
msoShapeCurvedUpRibbon	99
msoShapeCurvedDownRibbon	100
msoShapeVerticalScroll	101
msoShapeHorizontalScroll	102
msoShapeWave	103
msoShapeDoubleWave	104
msoShapeRectangularCallout	105
msoShapeRoundedRectangularCallout	106
msoShapeOvalCallout	107
msoShapeCloudCallout	108
msoShapeLineCallout1	109
msoShapeLineCallout2	110
msoShapeLineCallout3	111
msoShapeLineCallout4	112
msoShapeLineCallout1AccentBar	113
msoShapeLineCallout2AccentBar	114
msoShapeLineCallout3AccentBar	115
msoShapeLineCallout4AccentBar	116
msoShapeLineCallout1NoBorder	117
msoShapeLineCallout2NoBorder	118
msoShapeLineCallout3NoBorder	119
msoShapeLineCallout4NoBorder	120
msoShapeLineCallout1BorderandAccentBar	121

msoShapeLineCallout2BorderandAccentBar	122
msoShapeLineCallout3BorderandAccentBar	123
msoShapeLineCallout4BorderandAccentBar	124
msoShapeActionButtonCustom	125
msoShapeActionButtonHome	126
msoShapeActionButtonHelp	127
msoShapeActionButtonInformation	128
msoShapeActionButtonBackorPrevious	129
msoShapeActionButtonForwardorNext	130
msoShapeActionButtonBeginning	131
msoShapeActionButtonEnd	132
msoShapeActionButtonReturn	133
msoShapeActionButtonDocument	134
msoShapeActionButtonSound	135
msoShapeActionButtonMovie	136
msoShapeBalloon	137
msoShapeNotPrimitive	138

Hier ein Code für alle Shapes des aktuellen Tabellenblattes

irgendwas mit dem Anzeigen und selektieren klappt nicht

```
Sub Object_Types_on_This_Slide()
    'Refers to each object on the current page and returns the Shapes.Type
    'Can be very useful when searching through all objects on a page
    Dim it As String
```

```
Dim i As Integer
Dim Ctr As Integer
''''''''''''''''''
'Read-only  Long
''''''''''''''''''
For i = 1 To ActiveSheet.Shapes.Count
    'No need to select the object in order to use it
    With ActiveSheet.Shapes(i)

    'But it is easier to watch when the object is selected
    'This next line is for demonstration purposes only.
    'It is not necessary
    ActiveSheet.Shapes(i).Select
    ActiveSheet.Shapes(i).incrementleft

    Select Case .Type

        'Type 1
        Case msoAutoShape
            it = "an AutoShape. Type : " & .Type

        'Type 2
        Case msoCallout
            it = "a Callout. Type : " & .Type

        'Type 3
        Case msoChart
            it = "a Chart. Type : " & .Type

        'Type 4
        Case msoComment
            it = "a Comment. Type : " & .Type
```

```
'Type 5
Case msoFreeform
    it = "a Freeform. Type : " & .Type

'Type 6
Case msoGroup
    it = "a Group. Type : " & .Type

' If it's a group them iterate thru
' the items and list them

    it = it & vbCrLf & "Comprised of..."
    For Ctr = 1 To .GroupItems.Count
        it = it & vbCrLf & _
            .GroupItems(Ctr).Name & _
            ". Type:" & .GroupItems(Ctr).Type
    Next Ctr

'Type 7
Case msoEmbeddedOLEObject
    it = "an Embedded OLE Object. Type : " & .Type

'Type 8
Case msoFormControl
    it = "a Form Control. Type : " & .Type

'Type 9
Case msoLine
    it = "a Line. Type : " & .Type

'Type 10
```

```
Case msoLinkedOLEObject
    it = "a Linked OLE Object. Type : " & .Type
    With .LinkFormat
        it = it & vbCrLf & "My Source: " & _
            .SourceFullName
    End With

'Type 11
Case msoLinkedPicture
    it = "a Linked Picture. Type : " & .Type
    With .LinkFormat
        it = it & vbCrLf & "My Source: " & _
            .SourceFullName
    End With

'Type 12
Case msoOLEControlObject
    it = "an OLE Control Object. Type : " & .Type

'Type 13
Case msoPicture
    it = "a embedded picture. Type : " & .Type

'Type 14
Case msoPlaceholder
    it = "a text placeholder (title or regular text--" & _
         "not a standard textbox) object." & _
         "Type : " & .Type

'Type 15
Case msoTextEffect
    it = "a WordArt (Text Effect). Type : " & .Type
```

```
'Type 16
Case msoMedia
    it = "a Media object .. sound, etc. Type : " & .Type
    With .LinkFormat
        it = it & vbCrLf & " My Source: " & _
        .SourceFullName
    End With

'Type 17
Case msoTextBox
    it = "a Text Box."

'Type 18 = msoScriptAnchor, not defined in PPT pre-2000 so we use the numeric value
'Case msoScriptAnchor
Case 18
    it = " a ScriptAnchor. Type : " & .Type

'Type 19 = msoTable, not defined in PPT pre-2000 so we use the numeric value
'Case msoTable
Case 19
    it = " a Table. Type : " & .Type

'Type 19 = msoCanvas, not defined in PPT pre-2000 so we use the numeric value
'Case msoCanvas
Case 20
    it = " a Canvas. Type : " & .Type

'Type 21 = msoDiagram, not defined in PPT pre-2000 so we use the numeric value
'Case msoDiagram
Case 22
    it = " a Diagram. Type : " & .Type
```

```
            'Type 22 = msoInk, not defined in PPT pre-2000 so we use the numeric value
            'Case msoInk
            Case 22
                it = " an Ink shape. Type : " & .Type

            'Type 23 = msoInkComment, not defined in PPT pre-2000 so we use the numeric value
            'Case msoInkComment
            Case 23
                it = " an InkComment. Type : " & .Type


            'Type -2
            Case msoShapeTypeMixed
                it = "a Mixed object (whatever that might be)." & _
                     "Type : " & .Type

            'Just in case
            Case Else
                it = "a mystery!? An undocumented object type?" & _
                        " Haven't found one of these yet!"
        End Select

        MsgBox ("I'm " & it)
        End With
    Next i
End Sub
```

Autoformen löschen (z.B. nur Ellipsen = msoShapeOval)

Bezüglich der Typennummern siehe Eintrag Autoform auslesen der Typennummer

```
For Each objShape In ActiveSheet.Shapes
  ' Alle Shapes des Tabellenblattes durchlaufen

   If objShape.AutoShapeType = msoShapeOval Then ' Ellipse
      objShape.Delete
   End If
   If objShape.AutoShapeType = msoShapeRectangle Then ' Rechteck(ohne Rundung)
      objShape.Delete
   End If
   If objShape.AutoShapeType = 1 Then ' AutoShapeType 1 ist Textfeld
      objShape.Delete
   End If
   If objShape.Type = 9 Then ' Type 9=Linie
      objShape.Delete
   End If
   If objShape.AutoShapeType = msoShapeRoundedRectangle Then ' Abgerundetes Rechteck
      objShape.Delete
   End If
Next objShape
```

Alle Linien löschen

```
Sub Linien loeschen()
Dim Shl As Object
For Each Shl In ActiveSheet.Shapes
  If Shl.Type = 9 Then
     Shl.Delete
  End If
Next
End Sub
```

Alle Ellipsen löschen

```
For Each objShape In ActiveSheet.Shapes
   'Alle Shapes des Tabellenblattes durchlaufen
   If objShape.AutoShapeType = msoShapeOval Then
      ' Shapes vom Tyo msoShapeOVal löschen
      objShape.Delete
   End If
Next objShape
```

Alle Textfelder löschen

```
For Each objShape In ActiveSheet.Shapes
    If objShape.AutoShapeType = 1 Then ' AutoShapeType 1 ist Textfeld
        objShape.Delete
    End If
Next objShape
```

ALLE SHAPES DURCHWANDERN IN ALLEN TABELLENBLÄTTERN UND LÖSCHEN

```
Sub wegdamit()

Dim wks As Worksheet
Dim shp As Shape

Application.ScreenUpdating = False
For Each wks In Worksheets
    For Each shp In wks.Shapes
      shp.Delete
    Next shp
Next
Application.ScreenUpdating = True
End Sub
```

BSP

```
For Each objShape In ActiveSheet.Shapes
    'Alle Shapes des Tabellenblattes durchlaufen
    If objShape.AutoShapeType = msoShapeOval Or objShape.AutoShapeType = 1 Or objShape.Type = 9 Or objShape.AutoShapeType
= msoShapeRoundedRectangle Then ' Type 9=Linie AutoShapeType 1 ist Textfeld
        objShape.Delete
    End If
Next objShape
```

```
MsgBox Selection.ShapeRange.AutoShapeType
```

Ellipsen sind 9 (msoShapeOval)
Abgerundetes Rechteck ist 5

List of Shape Types and Shape Constants in Excel

Shape Type	Shape Constant
msoShapeMixed	-2
msoShapeRectangle	1
msoShapeParallelogram	2
msoShapeTrapezoid	3
msoShapeDiamond	4
msoShapeRoundedRectangle	5
msoShapeOctagon	6
msoShapeIsoscelesTriangle	7
msoShapeRightTriangle	8
msoShapeOval	9
msoShapeHexagon	10
msoShapeCross	11
msoShapeRegularPentagon	12
msoShapeCan	13
msoShapeCube	14
msoShapeBevel	15
msoShapeFoldedCorner	16
msoShapeSmileyFace	17
msoShapeDonut	18

msoShapeNoSymbol	19
msoShapeBlockArc	20
msoShapeHeart	21
msoShapeLightningBolt	22
msoShapeSun	23
msoShapeMoon	24
msoShapeArc	25
msoShapeDoubleBracket	26
msoShapeDoubleBrace	27
msoShapePlaque	28
msoShapeLeftBracket	29
msoShapeRightBracket	30
msoShapeLeftBrace	31
msoShapeRightBrace	32
msoShapeRightArrow	33
msoShapeLeftArrow	34
msoShapeUpArrow	35
msoShapeDownArrow	36
msoShapeLeftRightArrow	37
msoShapeUpDownArrow	38
msoShapeQuadArrow	39
msoShapeLeftRightUpArrow	40
msoShapeBentArrow	41
msoShapeUTurnArrow	42
msoShapeLeftUpArrow	43
msoShapeBentUpArrow	44

msoShapeCurvedRightArrow	45
msoShapeCurvedLeftArrow	46
msoShapeCurvedUpArrow	47
msoShapeCurvedDownArrow	48
msoShapeStripedRightArrow	49
msoShapeNotchedRightArrow	50
msoShapePentagon	51
msoShapeChevron	52
msoShapeRightArrowCallout	53
msoShapeLeftArrowCallout	54
msoShapeUpArrowCallout	55
msoShapeDownArrowCallout	56
msoShapeLeftRightArrowCallout	57
msoShapeUpDownArrowCallout	58
msoShapeQuadArrowCallout	59
msoShapeCircularArrow	60
msoShapeFlowchartProcess	61
msoShapeFlowchartAlternateProcess	62
msoShapeFlowchartDecision	63
msoShapeFlowchartData	64
msoShapeFlowchartPredefinedProcess	65
msoShapeFlowchartInternalStorage	66
msoShapeFlowchartDocument	67
msoShapeFlowchartMultidocument	68
msoShapeFlowchartTerminator	69
msoShapeFlowchartPreparation	70

msoShapeFlowchartManualInput	71
msoShapeFlowchartManualOperation	72
msoShapeFlowchartConnector	73
msoShapeFlowchartOffpageConnector	74
msoShapeFlowchartCard	75
msoShapeFlowchartPunchedTape	76
msoShapeFlowchartSummingJunction	77
msoShapeFlowchartOr	78
msoShapeFlowchartCollate	79
msoShapeFlowchartSort	80
msoShapeFlowchartExtract	81
msoShapeFlowchartMerge	82
msoShapeFlowchartStoredData	83
msoShapeFlowchartDelay	84
msoShapeFlowchartSequentialAccessStorage	85
msoShapeFlowchartMagneticDisk	86
msoShapeFlowchartDirectAccessStorage	87
msoShapeFlowchartDisplay	88
msoShapeExplosion1	89
msoShapeExplosion2	90
msoShape4pointStar	91
msoShape5pointStar	92
msoShape8pointStar	93
msoShape16pointStar	94
msoShape24pointStar	95
msoShape32pointStar	96

msoShapeUpRibbon	97
msoShapeDownRibbon	98
msoShapeCurvedUpRibbon	99
msoShapeCurvedDownRibbon	100
msoShapeVerticalScroll	101
msoShapeHorizontalScroll	102
msoShapeWave	103
msoShapeDoubleWave	104
msoShapeRectangularCallout	105
msoShapeRoundedRectangularCallout	106
msoShapeOvalCallout	107
msoShapeCloudCallout	108
msoShapeLineCallout1	109
msoShapeLineCallout2	110
msoShapeLineCallout3	111
msoShapeLineCallout4	112
msoShapeLineCallout1AccentBar	113
msoShapeLineCallout2AccentBar	114
msoShapeLineCallout3AccentBar	115
msoShapeLineCallout4AccentBar	116
msoShapeLineCallout1NoBorder	117
msoShapeLineCallout2NoBorder	118
msoShapeLineCallout3NoBorder	119
msoShapeLineCallout4NoBorder	120
msoShapeLineCallout1BorderandAccentBar	121
msoShapeLineCallout2BorderandAccentBar	122

msoShapeLineCallout3BorderandAccentBar	123
msoShapeLineCallout4BorderandAccentBar	124
msoShapeActionButtonCustom	125
msoShapeActionButtonHome	126
msoShapeActionButtonHelp	127
msoShapeActionButtonInformation	128
msoShapeActionButtonBackorPrevious	129
msoShapeActionButtonForwardorNext	130
msoShapeActionButtonBeginning	131
msoShapeActionButtonEnd	132
msoShapeActionButtonReturn	133
msoShapeActionButtonDocument	134
msoShapeActionButtonSound	135
msoShapeActionButtonMovie	136
msoShapeBalloon	137
msoShapeNotPrimitive	138

Autoform immer in Mitte des Bildschirms einfügen

ein Shape in der Bildschirmmitte des Monitors einzufügen wird sehr aufwendig, da sich das Bezugsssystem für die Shape-Koordinaten das Excelfenster ist, d.h. der Punkt (0 / 0) ist immer linke obere Ecke der Zelle A1.

Um jetzt den Shape in der Bildschirmmitte zu positioneren, müsstest du erst die Bildschirmposition der Applikation, die Postiion des Aktiven Fensters innerhalb der Applikation sowie den Scrollzustand des Aktiven Fensters ermitteln, um daraus irgendwie die Koordinaten der Bildschirmmitte zu bekommen.

Wesentlich einfacher sieht die Sache aus, wenn du deine Anforderung auf "in die Mitte des aktiven Excel-Fensters" beschränkst.

du kannst mit den Funktionen:

```
ActiveWindow.VisibleRange.top
ActiveWindow.VisibleRange.left
```

die Koordinaten der Linken oberen Ecke des sichtbaren bereiches abfragen, und mit

```
ActiveWindow.VisibleRange.Width
ActiveWindow.VisibleRange.Height
```

die Höhe und die Breite des sichtbaren Bereichs abfragen.

damit solltest du in der Lage sein, die erforderlichen Koordinaten zu berechnen, um dein Shape in der Mitte des sichtbaren Bereichs zu plazieren.

(Berechnungsformel für die X-Position wäre:

```
Links sichtbarer Bereich + (Breite sichtbarerer Bereich  - Breite Shape) /2
```

die Höhe dann ähnlich.

Allerdings gilt das dann nur für den aktuellen Scrollzustand.

Wenn du jetzt im Tabellenblatt herumscrollst, verschiebt sich natürlich auch das Shape, weil das Bezugssysstem ja die Zellen sind

Soll das Shape wie eine Symbolleiste oder eine Userform immer sichtbar bleiben, müsstest du ein Eventgesteuertes Makro schreiben, das das Shape immer wieder neu positioniert.

Da meines Wissen nach für das Scrollen noch kein Event gibt, musst du hier wahrscheinlich auf SelectionChange zurürckgreifen.

Gruß, Daniel

Autoform-Theorie aus meinem Buch Programming Excel with VBA and .Net

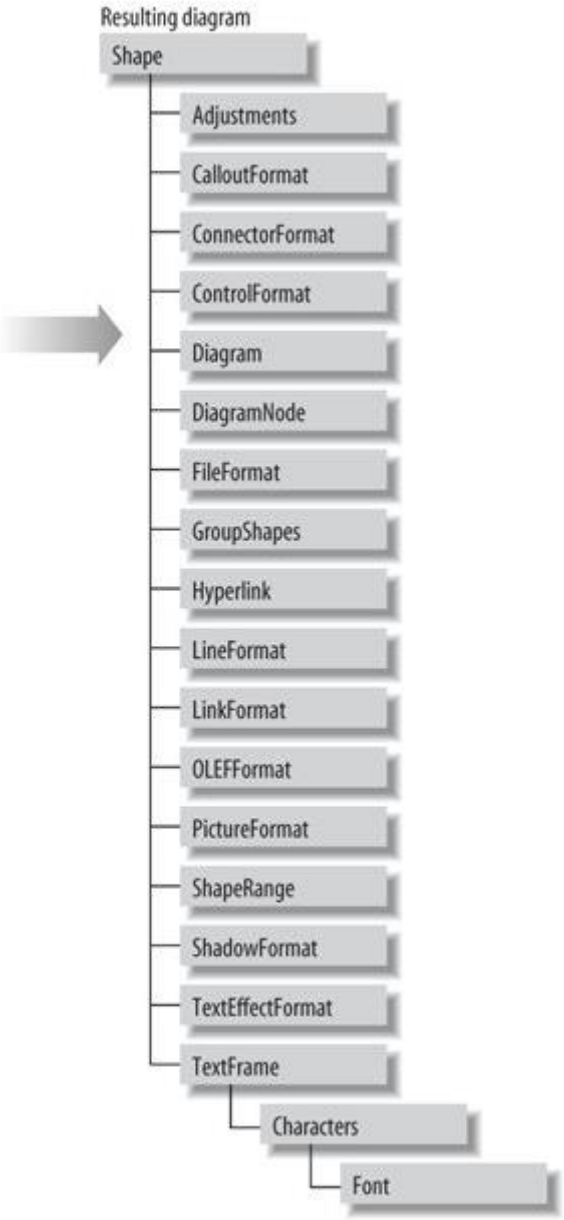
Shape, ShapeRange, and Shapes Members

Use the `Shapes` collection to draw graphics on a worksheet or chart. Use the `Worksheet` or `Chart` object's `Shapes` property to get a reference to this collection. Use the `Shape` object to change the appearance of one shape; use `ShapeRange` to change groups of shapes. `Shapes`, `Shape` and `ShapeRange` have the following members. Key members (shown in bold) are covered in the following reference section:

Figure 18-12. Diagramming the Shape object hierarchy

Source worksheet

	A	B	C	D
1	Shape			
2		Adjustments		
3		CalloutFormat		
4		ConnectorFormat		
5		ControlFormat		
6		Diagram		
7		DiagramNode		
8		FillFormat		
9		GroupShapes		
10		Hyperlink		
11		LineFormat		
12		LinkFormat		
13		OLEFormat		
14		PictureFormat		
15		ShapeRange		
16		ShadowFormat		
17		TextEffectFormat		
18		TextFrame		
19			Characters	
20				Font



AddCallout **Fehler! Linkreferenz ungültig.**	**AddConnector** **Fehler! Linkreferenz ungültig.**	**AddCurve** **Fehler! Linkreferenz ungültig.**
AddDiagram**Fehler! Linkreferenz ungültig.**	AddFormControl**Fehler! Linkreferenz ungültig.**	**AddLabel** **Fehler! Linkreferenz ungültig.**
AddLine **Fehler! Linkreferenz ungültig.**	AddOLEObject**Fehler! Linkreferenz ungültig.**	**AddPicture** **Fehler! Linkreferenz ungültig.**
AddPolyline **Fehler! Linkreferenz ungültig.**	**AddShape** **Fehler! Linkreferenz ungültig.**	**AddTextbox** **Fehler! Linkreferenz ungültig.**
AddTextEffect **Fehler! Linkreferenz ungültig.**	**Adjustments**	**Align** **Fehler! Linkreferenz ungültig.**
AlternativeText	Application**Fehler! Linkreferenz ungültig.**	**Apply**
AutoShapeType	**BlackWhiteMode**	BottomRightCell
BuildFreeform **Fehler! Linkreferenz ungültig.**	**Callout**	Child
ConnectionSiteCount	**Connector**	**ConnectorFormat**
ControlFormat	Copy	CopyPicture
Count**Fehler! Linkreferenz ungültig.**	Creator**Fehler! Linkreferenz ungültig.**	Cut
Delete	Diagram	DiagramNode
Distribute **Fehler! Linkreferenz ungültig.**	**Duplicate**	**Fill**
Flip	**FormControlType**	**Group** **Fehler! Linkreferenz ungültig.**
GroupItems	HasDiagram	HasDiagramNode
Height	**HorizontalFlip**	**Hyperlink**
ID	**IncrementLeft**	**IncrementRotation**

IncrementTop	Item**Fehler! Linkreferenz ungültig.**	Left
Line	**LinkFormat**	**LockAspectRatio**
Locked	Name	Nodes
OLEFormat	OnAction	Parent**Fehler! Linkreferenz ungültig.**
ParentGroup	**PickUp**	**PictureFormat**
Placement	**Range** **Fehler! Linkreferenz ungültig.**	**Regroup** **Fehler! Linkreferenz ungültig.**
RerouteConnections	**Rotation**	ScaleHeight
ScaleWidth	Script	Select
SelectAll **Fehler! Linkreferenz ungültig.**	**SetShapesDefaultProperties**	**Shadow**
TextEffect	**TextFrame**	**ThreeD**
Top	TopLeftCell	**Type**
Ungroup	**VerticalFlip**	**Vertices**
Visible	Width	ZOrder
ZOrderPosition		

[1] Collection only

[2] Object and collection

[3] ShapeRange only

shapes.AddCallout(Type, Left, Top, Width, Height)

Draws a simple callout and returns the callout's `Shape` object.

Argument	Settings
Type	An `msoCalloutType` constant indicating the type of callout to draw. Can be `msoCalloutOne`, `msoCalloutTwo`, `msoCalloutMixed`, `msoCalloutThree`, or `msoCalloutFour`.
Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.
Width	The width of the shape in points.
Height	The height of the shape in points.

shapes.AddConnector(Type, BeginX, BeginY, EndX, EndY)

Draws a connector line and returns the connector's `Shape` object.

Argument	Settings
Type	An `msoConnectorType` constant. Can be `msoConnectorElbow`, `msoConnectorTypeMixed`, `msoConnectorCurve`, or `msoConnectorStraight`.

BeginX	The horizontal coordinate of the start of the connector line.
BeginY	The vertical coordinate of the start of the connector line.
EndX	The horizontal coordinate of the end of the connector line.
EndY	The vertical coordinate of the end of the connector line.

You can set the begin and end coordinates to an arbitrary value, then use the `BeginConnect` and `EndConnect` methods to connect two objects. Using the `RerouteConnections` method creates the shortest path between the objects. The following code demonstrates using those methods to connect two shapes as shown in **Fehler! Linkreferenz ungültig.**:

```
Sub QuickConnect( )
    Dim s1 As Shape, s2 As Shape, conn As Shape
    ' Create a shape
    Set s1 = ActiveSheet.Shapes.AddShape(msoShapeCube, 100, 10, 50, 60)
    ' Create another shape
    Set s2 = ActiveSheet.Shapes.AddShape(msoShapeCan, 50, 100, 50, 60)
    ' Create connector with arbitrary coordinates
    Set conn = ActiveSheet.Shapes.AddConnector(msoConnectorCurve, 1, 1, 1, 1)
    ' Connect shapes
    conn.ConnectorFormat.BeginConnect s1, 1
    conn.ConnectorFormat.EndConnect s2, 1
    ' Connect via shortest path (changes connection sites)
    conn.RerouteConnections
End Sub
```

Figure 18-13. Creating a connection

shapes.AddCurve(SafeArrayOfPoints)

Draws a Bézier curve from an array of coordinate pairs and returns the curve's `Shape` object.

Argument	**Settings**
`SafeArrayOfPoints`	The 2-D array of points containing the vertices and control points of the curve

The following code draws an S-shaped curve that starts at (80,100) and ends at (110,30):

```
Sub DrawCurve( )
    Dim s As Shape, pts( ) As Single
    ' Array of points.
    ReDim pts(3, 1)
    pts(0, 0) = 80
    pts(0, 1) = 100
    pts(1, 0) = 200
    pts(1, 1) = 150
    pts(2, 0) = 15
    pts(2, 1) = 20
    pts(3, 0) = 110
    pts(3, 1) = 30
    ' Draw a curve
    Set s = ActiveSheet.Shapes.AddCurve(pts)
End Sub
```

shapes.AddLabel(Orientation, Left, Top, Width, Height)

Draws a text box without a border and returns the label's `Shape` object.

Argument	Settings
Orientation	An `msoTextOrientation` constant. Can be `msoTextOrientationDownward`, `msoTextOrientationHorizontal`, `msoTextOrientationHorizontalRotatedFarEast`, `msoTextOrientationMixed`, `msoTextOrientationUpward`, `msoTextOrientationVertical`, or `msoTextOrientationVerticalFarEast`.
Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.
Width	The width of the shape in points.
Height	The height of the shape in points.

Use the `TextFrame` property to get or set the text in the label and to set the formatting of the text. The following code draws a label on the active sheet:

```
Sub DrawLabel( )
    Dim s As Shape
    ' Create label (height/width will be set automatically).
    Set s = ActiveSheet.Shapes.AddLabel(msoTextOrientationHorizontal, _
      100, 100, 1, 1)
    s.TextFrame.Characters.text = "This is some label text"
End Sub
```

The *Width* and *Height* arguments in the preceding code are required but arbitrary because the `TextFrame` object's `AutoSize` property is True by default. The label is automatically resized to fit the text.

shapes.AddLine(BeginX, BeginY, EndX, EndY)

Draws a straight line and returns the line's `Shape` object.

Argument	Settings
BeginX	The horizontal coordinate for the origin of the line
BeginY	The vertical coordinate for the origin of the line
EndX	The horizontal coordinate for the end of the line
EndY	The vertical coordinate for the end of the line

Use the `Line` property to set the style and formatting used for the line. The following code draws a dashed line with an arrowhead:

```
Sub DrawLine( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Create label (height/width will be set by AutoSize).
    Set s = ws.Shapes.AddLine(100, 100, 200, 200)
    s.Line.DashStyle = msoLineDash
    s.Line.EndArrowheadStyle = msoArrowheadStealth
End Sub
```

shapes.AddPicture(Filename, LinkToFile, SaveWithDocument, Left, Top, Width, Height)

Adds a picture to a worksheet or chart and the picture's `Shape` object.

Argument	Settings
Filename	The picture file to load.
LinkToFile	True links the shape to the picture file; False copies the image into the file.
SaveWithDocument	True saves the image in the workbook; False saves only link information in the document. If *LinkToFile* is False, *SaveWithDocument* must be True.
Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.
Width	The width of the shape in points.
Height	The height of the shape in points.

Excel scales the image to fit the *Width* and *Height* arguments. To restore the image's actual height and width, use the `ScaleHeight` and `ScaleWidth` methods as shown here:

```
Sub DrawPicture( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Insert the image.
    Set s = ws.Shapes.AddPicture(ThisWorkbook.Path & "\logo.bmp", _
      False, True, 100, 100, 1, 1)
    ' Use picture's height and width.
    s.ScaleHeight 1, msoCTrue
    s.ScaleWidth 1, msoCTrue
End Sub
```

Use the `PictureFormat` property to control a picture's brightness, contrast, and transparency.

shapes.AddPolyline(SafeArrayOfPoints)

Draws a segmented line from an array of coordinate pairs and returns the line's `Shape` object.

Argument	Settings
SafeArrayOfPoints	The 2-D array of points containing the vertices of the line

Use the `Line` property to set the style and formatting used for the line. The following code draws a Z-shaped line that starts at (80,100) and ends at (110,30):

```
Sub DrawZ( )
    Dim s As Shape, pts( ) As Single
    ' Array of points.
    ReDim pts(3, 1)
    pts(0, 0) = 80
    pts(0, 1) = 100
    pts(1, 0) = 200
    pts(1, 1) = 150
    pts(2, 0) = 15
    pts(2, 1) = 20
    pts(3, 0) = 110
    pts(3, 1) = 30
    ' Draw a curve
    Set s = ActiveSheet.Shapes.AddPolyline(pts)
End Sub
```

shapes.AddShape(Type, Left, Top, Width, Height)

Draws an autoshape and returns the autoshape's `Shape` object.

Argument	Settings
Type	An `msoAutoShapeType` constant. Can be any of the settings listed following this table.

Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.
Width	The width of the shape in points.
Height	The height of the shape in points.

The `msoAutoShapeType` constant can be one of:

```
msoShape4pointStar                      msoShape5pointStar
msoShape8pointStar                      msoShape16pointStar
msoShape24pointStar                     msoShape32pointStar
msoShapeActionButtonBackorPrevious      msoShapeActionButtonBeginning
msoShapeActionButtonCustom              msoShapeActionButtonDocument
msoShapeActionButtonEnd                 msoShapeActionButtonForwardorNext
msoShapeActionButtonHelp                msoShapeActionButtonHome
msoShapeActionButtonInformation         msoShapeActionButtonMovie
msoShapeActionButtonReturn              msoShapeActionButtonSound
msoShapeArc                             msoShapeBalloon
msoShapeBentArrow                       msoShapeBentUpArrow
msoShapeBevel                           msoShapeBlockArc
msoShapeCan                             msoShapeChevron
msoShapeCircularArrow                   msoShapeCloudCallout
msoShapeCross                           msoShapeCube
msoShapeCurvedDownArrow                 msoShapeCurvedDownRibbon
msoShapeCurvedLeftArrow                 msoShapeCurvedRightArrow
msoShapeCurvedUpArrow                   msoShapeCurvedUpRibbon
msoShapeDiamond                         msoShapeDonut
msoShapeDoubleBrace                     msoShapeDoubleBracket
msoShapeDoubleWave                      msoShapeDownArrow
msoShapeDownArrowCallout                msoShapeDownRibbon
msoShapeExplosion1                      msoShapeExplosion2
msoShapeFlowchartAlternateProcess       msoShapeFlowchartCard
msoShapeFlowchartCollate                msoShapeFlowchartConnector
msoShapeFlowchartData                   msoShapeFlowchartDecision
msoShapeFlowchartDelay                  msoShapeFlowchartDirectAccessStorage
msoShapeFlowchartDisplay                msoShapeFlowchartDocument
msoShapeFlowchartExtract                msoShapeFlowchartInternalStorage
msoShapeFlowchartMagneticDisk           msoShapeFlowchartManualInput
msoShapeFlowchartManualOperation        msoShapeFlowchartMerge
msoShapeFlowchartMultidocument          msoShapeFlowchartOffpageConnector
msoShapeFlowchartOr                     msoShapeFlowchartPredefinedProcess
msoShapeFlowchartPreparation            msoShapeFlowchartProcess
msoShapeFlowchartPunchedTape            msoShapeFlowchartSequentialAccessStorage
```

```
msoShapeFlowchartSort                    msoShapeFlowchartStoredData
msoShapeFlowchartSummingJunction         msoShapeFlowchartTerminator
msoShapeFoldedCorner                     msoShapeHeart
msoShapeHexagon                          msoShapeHorizontalScroll
msoShapeIsoscelesTriangle                msoShapeLeftArrow
msoShapeLeftArrowCallout                 msoShapeLeftBrace
msoShapeLeftBracket                      msoShapeLeftRightArrow
msoShapeLeftRightArrowCallout            msoShapeLeftRightUpArrow
msoShapeLeftUpArrow                      msoShapeLightningBolt
msoShapeLineCallout1                     msoShapeLineCallout1AccentBar
msoShapeLineCallout1BorderandAccentBar   msoShapeLineCallout1NoBorder
msoShapeLineCallout2                     msoShapeLineCallout2AccentBar
msoShapeLineCallout2BorderandAccentBar   msoShapeLineCallout2NoBorder
msoShapeLineCallout3                     msoShapeLineCallout3AccentBar
msoShapeLineCallout3BorderandAccentBar   msoShapeLineCallout3NoBorder
msoShapeLineCallout4                     msoShapeLineCallout4AccentBar
msoShapeLineCallout4BorderandAccentBar   msoShapeLineCallout4NoBorder
msoShapeMixed                            msoShapeMoon
msoShapeNoSymbol                         msoShapeNotchedRightArrow
msoShapeNotPrimitive                     msoShapeOctagon
msoShapeOval                             msoShapeOvalCallout
msoShapeParallelogram                    msoShapePentagon
msoShapePlaque                           msoShapeQuadArrow
msoShapeQuadArrowCallout                 msoShapeRectangle
msoShapeRectangularCallout               msoShapeRegularPentagon
msoShapeRightArrow                       msoShapeRightArrowCallout
msoShapeRightBrace                       msoShapeRightBracket
msoShapeRightTriangle                    msoShapeRoundedRectangle
msoShapeRoundedRectangularCallout        msoShapeSmileyFace
msoShapeStripedRightArrow                msoShapeSun
msoShapeTrapezoid                        msoShapeUpArrow
msoShapeUpArrowCallout                   msoShapeUpDownArrow
msoShapeUpDownArrowCallout               msoShapeUpRibbon
msoShapeUTurnArrow                       msoShapeVerticalScroll
msoShapeWave
```

Use the `TextFrame` property to add text to an autoshape. Use the `AutoShapeType` property to convert one autoshape into another.

shapes.AddTextbox(Orientation, Left, Top, Width, Height)

Draws a text box surrounded by a rectangular border and returns the text box's `Shape` object.

Argument	Settings
Orientation	An msoTextOrientation constant. Can be msoTextOrientationDownward, msoTextOrientationHorizontal, msoTextOrientationHorizontalRotatedFarEast, msoTextOrientationMixed, msoTextOrientationUpward, msoTextOrientationVertical, or msoTextOrientationVerticalFarEast.
Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.
Width	The width of the shape in points.
Height	The height of the shape in points.

Use the TextFrame property to get or set the text in the shape and to set the formatting of the text. Unlike labels, text boxes do not automatically resize to fit their text. You must set the AutoSize property as shown here:

```
Sub DrawTextbox( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Create label (height/width will be set by AutoSize).
    Set s = ws.Shapes.AddTextbox(msoTextOrientationHorizontal, 100, 100, 1, 1)
    s.TextFrame.Characters.text = "This is some label text"
    ' Resize text box to fit text.
    s.TextFrame.AutoSize = True
End Sub
```

shapes.AddTextEffect(PresetTextEffect, Text, FontName, FontSize, FontBold, FontItalic, Left, Top)

Adds a WordArt embedded object and returns the Shape object for the embedded object.

Argument	Settings
PresetTextEffect	An `MsoPresetTextEffect` constant. Can be `msoTextEffect1` to `msoTextEffect30`.
Text	The text to embed.
FontName	The name of the font to use.
FontSize	The size of the font in points.
FontBold	True uses bold; False uses the normal weight font.
FontItalic	True uses italic font; False uses roman.
Left	The horizontal position of the shape in points.
Top	The vertical position of the shape in points.

Use the `TextEffect` property, not `TextFrame`, to change the text or appearance of the embedded WordArt object. The following code embeds a WordArt object, then changes its text:

```
Sub EmbedWordArt( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Create label (height/width will be set by AutoSize).
    Set s = ws.Shapes.AddTextEffect(msoTextEffect19, "Wombat!", "Arial", 36, _
        True, False, 100, 100)
    ' Change text.
    s.TextEffect.text = "New Text"
End Sub
```

shape.Adjustments

For an autoshape, connector, or WordArt shape, returns the `Adjustments` collection; for other types of shapes, causes an error. Use `Adjustments` to move the adjustment handles on a shape (the equivalent of clicking and dragging on the adjustment handle). **Fehler! Linkreferenz ungültig.** illustrates adjustment handles.

shaperange.Align(AlignCmd, RelativeTo)

Aligns the shapes in a `ShapeRange`.

Argument	Settings
`AlignCmd`	An `msoAlignCmd` constant. Can be `msoAlignCenters`, `msoAlignMiddles`, `msoAlignTops`, `msoAlignBottoms`, `msoAlignLefts`, or `msoAlignRights`.
`RelativeTo`	True aligns shapes relative to the Excel window; False aligns shapes relative to the first shape in the `ShapeRange`.

The following code uses previous examples to draw three shapes, adds them to a ShapeRange, then aligns the shapes relative to the first shape drawn:

```
Sub LeftAlign( )
   Dim ws As Worksheet, sr As ShapeRange
    Set ws = ActiveSheet
    ' Draw three objects (call previous examples)
    DrawRect
    DrawLine
    EmbedWordArt
    ' Create a shape range
    Set sr = ws.Shapes.Range(Array(1, 2, 3))
    ' Left-align three shapes
    sr.align msoAlignLefts, False
End Sub
```

shape.AlternativeText [= setting]

Sets or returns the alternate text used for the shape if the worksheet or chart is saved in HTML format.

shape.Apply( )

Applies formatting that was previously picked up from another shape. The `PickUp` and `Apply` methods are used together to copy formatting from one shape to another. For example, the following code copies the formatting from the first shape on a worksheet to all of the others on the same worksheet:

```
Sub FormatSameAsFirst( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    For Each s In ws.Shapes
        ' Get formatting from first shape.
        ws.Shapes(1).PickUp
        ' Apply it to each shape.
        s.Apply
    Next
End Sub
```

Calling `Apply` clears the formatting being copied, so you must call `PickUp` before each `Apply`.

shape.AutoShapeType [= msoAutoShapeType]

Converts one autoshape to another autoshape. Causes an error for connector, line, picture, OLE object, and WordArt shape types. See the list under "**Fehler! Linkreferenz ungültig.**)," earlier in this chapter, for a list of possible settings.

shape.BlackWhiteMode [= msoBlackWhiteMode]

Sets or returns how the shape appears when viewed in black and white. Can be:

```
    msoBlackWhiteAutomatic            msoBlackWhiteBlack
    msoBlackWhiteBlackTextAndLine     msoBlackWhiteDontShow
```

```
    msoBlackWhiteGrayOutline          msoBlackWhiteGrayScale
    msoBlackWhiteHighContrast         msoBlackWhiteInverseGrayScale
    msoBlackWhiteLightGrayScale       msoBlackWhiteMixed
    msoBlackWhiteWhite
```

shapes.BuildFreeform(EditingType, X1, Y1)

Begins drawing freeform line art and returns a `FreeformBuilder` object used to add elements to the freeform.

Argument	Settings
EditingType	An `msoEditingType` constant. Can be `msoEditingAuto` or `msoEditingCorner`.
X1	The horizontal position of the first element of the shape in points.
Y1	The vertical position of the first element of the shape in points.

Use the `ConvertToShape` method for drawing the freeform and render it as a shape as shown here:

```
Sub DrawFreeform( )
    Dim ws As Worksheet, s As Shape, fb As FreeformBuilder
    Set ws = ActiveSheet
    ' Create freeform
    Set fb = ws.Shapes.BuildFreeform(msoEditingAuto, 380, 230)
    ' Add segments.
    fb.AddNodes msoSegmentCurve, msoEditingCorner, _
        380, 230, 400, 250, 450, 300
    fb.AddNodes msoSegmentCurve, msoEditingAuto, 480, 200
    fb.AddNodes msoSegmentLine, msoEditingAuto, 480, 400
    fb.AddNodes msoSegmentLine, msoEditingAuto, 380, 230
    ' Render drawing.
    fb.ConvertToShape
End Sub
```

shape.Callout

For callout shapes, returns a `CalloutFormat` object used to format the callout. For other shape types, causes an error.

shape.ConnectionSiteCount

Returns the number of connection sites available on the shape.

shape.Connector

Returns True if the shape is a connector, False if it is not.

shape.ConnectorFormat

For connector shapes, returns a `ConnectorFormat` object. For other shape types, causes an error. The following code changes all of the connections on a worksheet to use the curved connector style:

```
Sub ChangeConnectors( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    For Each s In ws.Shapes
        ' If the shape is a connector, change its style.
        If s.Connector Then _
            s.ConnectorFormat.Type = msoConnectorCurve
    Next
End Sub
```

shape.ControlFormat

For Forms 1.0 shapes, returns the `ControlFormat` object used to access the properties and methods of the control. For other shape types, causes an error.

shaperange.Distribute(DistributeCmd, RelativeTo)

Distributes the shapes in a `ShapeRange` vertically or horizontally.

Argument	Settings
DistributeCmd	Can be `msoDistributeHorizontally`or `msoDistributeVertically`.
RelativeTo	Must be False in Excel.

The following code distributes the shapes on a worksheet vertically; this is the equivalent of selecting Draw Align or Distribute Distribute Horizontally on the Drawing toolbar:

```
Sub DistributeVertically( )
   Dim ws As Worksheet, sr As ShapeRange
    Set ws = ActiveSheet
    ' Create a shape range for all shapes on sheet.
    ws.Shapes.SelectAll
    Set sr = Selection.ShapeRange
    ' Distribute shapes
    sr.Distribute msoDistributeVertically, False
End Sub
```

shape.Duplicate( )

Copies a shape and returns a reference to the new `Shape` object. The following code makes a copy of the first shape on a worksheet, then moves the copy to the left:

```
Sub CopyShape( )
```

```
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    Set s = ws.Shapes(1)
    ' Make copy.
    Set s = s.Duplicate
    ' Move copy.
    s.IncrementLeft 100
End Sub
```

shape.Fill

Returns a `FillFormat` object for the shape.

shape.Flip(FlipCmd)

Flips the shape vertically or horizontally.

Argument	Settings
FlipCmd	Can be `msoFlipHorizontal` or `msoFlipVertical`

Most shapes can be flipped, but OLE objects and form controls cannot.

shape.FormControlType

For Form 1.0 controls, returns an `xlFormControl` constant indicating the control type. For other shapes, causes an error.

shaperange.Group( )

Groups the shapes in the `ShapeRange` so that they can be selected, moved, or deleted as a single shape by the user. The following code demonstrates grouping and ungrouping the shapes on a worksheet:

```
Sub GroupUngroup( )
    Dim ws As Worksheet, sr As ShapeRange, s As Shape
    Set ws = ActiveSheet
    Select Case ws.Shapes.Count
        Case Is > 1
            ' Create a shape range for all shapes on sheet.
            ws.Shapes.SelectAll
            Set sr = Selection.ShapeRange
            ' Group all the items
            Set s = sr.Group
            ' Show count of items in group.
            Debug.Print s.GroupItems.Count & " shapes grouped."
        Case 1
            ws.Shapes(1).Ungroup
            Debug.Print "Ungrouped shapes"
        Case 0
            Debug.Print "No shapes to group."
    End Select
End Sub
```

shape.GroupItems

Returns the collection of shapes in a group.

shape.HorizontalFlip

Returns True if the shape has been flipped horizontally, False otherwise.

shape.Hyperlink

Returns a `Hyperlink` object for the shape. See **Fehler! Linkreferenz ungültig.** for more information on the `Hyperlink` object.

shape.ID

Returns a numeric identifier for the shape.

shape.IncrementLeft(Increment)

Moves a shape horizontally.

Argument	Settings
Increment	The number of points to move the shape

shape.IncrementRotation(Increment)

Rotates a shape.

Argument	Settings
Increment	The number of degrees to rotate the shape

The following code draws and rotates a star:

```
Sub Rotate( )
    Dim ws As Worksheet, s As Shape, i As Integer
    Set ws = ActiveSheet
```

```
    ' Draw a star.
    Set s = ws.Shapes.AddShape(msoShape5pointStar, 120, 80, 40, 40)
    ' Rotate it.
    For i = 0 To 6
        Application.Wait Now + 0.00001
        s.IncrementRotation 10
    Next
End Sub
```

shape.IncrementTop(Increment)

Moves a shape vertically.

Argument	Settings
Increment	The number of points to move the shape

shape.Line

For line shapes, returns a `LineFormat` object that controls the appearance of the line. For shape objects with borders, returns a `LineFormat` object that controls the appearance of the border. For other shape types, causes an error.

shape.LinkFormat

For OLE object shapes, returns a `LinkFormat` object used to update the link. For other shape types, causes an error.

shape.LockAspectRatio [= setting]

This property has no effect in Excel.

shape.Locked [= setting]

If the worksheet is protected, True prevents changes to the shape and False enables changes to the shape.

shape.ParentGroup

For shapes that are grouped, returns the group to which the shape belongs. Causes an error if the shape is not part of a group.

shape.PickUp( )

Copies the formatting from a shape. See the **Fehler! Linkreferenz ungültig.** method earlier for a full description and example of copying formatting between shapes.

shape.PictureFormat

For picture and OLE object shapes, returns a `PictureFormat` object used to control the appearance of the shape. For other shape types, causes an error.

shape.Placement [= xlPlacement]

Sets or returns how the shape is related to the cells underneath it. Can be one of these settings:

`xlFreeFloating` (default)

`xlMove`

`xlMoveAndSize`

shapes.Range(Index)

Returns a `ShapeRange` object containing a subset of shapes from the `Shapes` collection.

Argument	**Settings**
Index	An array containing the names or indexes of the shapes to include in the `ShapeRange`

Use `ShapeRange` objects to perform tasks on a group of shapes. Building a `ShapeRange` from an array of items is complex. It is easier to simply select the items you want in the `ShapeRange`, then use the `Selection.ShapeRange` method as shown here:

```
Sub BuildShapeRange( )
    Dim ws As Worksheet, s As Shape, sr As ShapeRange, sList As String, arr
    Set ws = ActiveSheet
    ' Clear selection
    [a1].Select
    ' Find each autoshape on the worksheet and build a list.
    For Each s In ws.Shapes
        If s.Type = msoAutoShape Then s.Select False
    Next
    Set sr = Selection.ShapeRange
    ' Move the ShapeRange.
    sr.IncrementLeft 10
End Sub
```

shaperange.Regroup( )

For shapes within a `ShapeRange` that belonged to a group but were ungrouped, `Regroup` restores those items to their previous group and returns the grouped objects as a single `Shape` object.

shape.RerouteConnections( )

For connector shapes, changes the connection sites so that the connection follows the shortest path. Causes an error for other shape types.

shape.Rotation [= setting]

Returns the rotation of a shape in degrees.

shapes.SelectAll( )

Selects all of the shapes on the worksheet or chart.

shape.SetShapesDefaultProperties( )

Makes the shape's formatting the default formatting for all subsequent shapes. Use the `PickUp` and `Apply` methods to copy formatting from one shape to another. The following code draws a star, sets its fill, and then makes that formatting the default:

```
Sub Defaults( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Draw star
    Set s = ws.Shapes.AddShape(msoShape5pointStar, 50, 50, 40, 40)
    ' Set its fill.
    s.Fill.PresetGradient msoGradientDiagonalUp, 1, msoGradientChrome
    ' Make this the default style.
    s.SetShapesDefaultProperties
```

```
    ' Draw another star.
    Set s = ws.Shapes.AddShape(msoShape5pointStar, 90, 90, 40, 40)
End Sub
```

shape.Shadow

Returns a `ShadowFormat` object used to display and set the appearance of a shape's shadow. The following code draws a star with a shadow:

```
Sub DrawShadow( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Draw star
    Set s = ws.Shapes.AddShape(msoShape5pointStar, 50, 50, 40, 40)
    ' Add a shadow.
    s.Shadow.Type = msoShadow1
End Sub
```

shape.TextEffect

For embedded WordArt shapes, returns a `TextEffectFormat` object used to set the text and change the appearance of the shape. Causes an error for other shape types. Use this property, not `TextFrame`, to change the text displayed in a WordArt shape.

shape.TextFrame

For autoshapes, returns the `TextFrame` object used to set and format text appearing on the shape. Causes an error for most other shape types. The following code draws an oval and adds some text to it:

```
Sub DrawOval( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Draw a shape
    Set s = ws.Shapes.AddShape(msoShapeOval, 60, 30, 1, 1)
    ' Add text.
    s.TextFrame.Characters.text = "Vigorous writing is concise."
    ' Resize shape to fit text.
```

```
    s.TextFrame.AutoSize = True
End Sub
```

shape.ThreeD

For autoshapes, returns a `THReeDFormat` object used to add a 3-D effect to the shape. The following code draws a wave as a wire-frame 3-D figure:

```
Sub DrawThreeD( )
    Dim ws As Worksheet, s As Shape
    Set ws = ActiveSheet
    ' Draw a shape
    Set s = ws.Shapes.AddShape(msoShapeDoubleWave, 50, 50, 40, 40)
    ' Add 3-D effect.
    s.ThreeD.PresetMaterial = msoMaterialWireFrame
End Sub
```

shape.Type

Returns an `msoShapeType` constant identifying the kind of shape. Can be one of these settings:

```
    msoAutoShape            msoCallout
    msoChart                msoComment
    msoDiagram              msoEmbeddedOLEObject
    msoFormControl          msoFreeform
    msoGroup                msoLine
    msoLinkedOLEObject      msoLinkedPicture
    msoOLEControlObject     msoPicture
    msoScriptAnchor         msoShapeTypeMixed
    msoTable                msoTextBox
    msoTextEffect
```

shape.Ungroup( )

Separates a previously grouped object into its individual shapes.

shape.VerticalFlip

Returns True if the shape has been flipped vertically, False otherwise.

shape.Vertices

For a freeform shape, returns a 2-D array containing the coordinate pairs of the shape's vertices.

Bild auswählen + einfügen

siehe GRAFIK AUSWÄHLEN + EINFÜGEN

Button-Arten (Form-Buttons and Active X Command Buttons)

The Difference Between Active X Command Buttons & Form Buttons

By Lindsey Mason, eHow Contributor

Microsoft Excel features two types of controls: Form controls and ActiveX controls. ActiveX controls are also referred to as ActiveX commands. Both Form controls and ActiveX controls allow users to add buttons and other features to Microsoft Excel documents. There are several differences between the two types of controls. Three types of Form control buttons and four types of ActiveX command buttons are available.

Read more: The Difference Between Active X Command Buttons & Form Buttons | eHow.com http://www.ehow.com/info_8703112_difference-command-buttons-form-buttons.html#ixzz2BHcyfup6

- Form Controls

- The original controls for Microsoft Excel are called Form controls. These controls, which are compatible with some earlier versions of the software, do not use Visual Basic for Applications (VBA) code. Form controls can run macros, but cannot be used on

UserForms, Webpages or for controlling events. The major advantages of these controls are their simplicity and compatibility. However, Form controls suffer a major disadvantage due to their limited use.

- ActiveX Controls

- ActiveX controls are often used in conjunction with VBA code, but the code is not required for worksheet forms. Like Form controls, ActiveX controls can be used on worksheets. However, they can also be used on UserForms and to control events. ActiveX controls can run macros as well, but only through event control. The major advantages of these controls are their flexibility and customization capacity. The major disadvantage of these controls is their complexity.

Read more: The Difference Between Active X Command Buttons & Form Buttons | eHow.com http://www.ehow.com/info_8703112_difference-command-buttons-form-buttons.html#ixzz2BHd1bc1a

- Types of Control Buttons

- There are three types of Form control buttons: push buttons, option buttons and spin buttons. ActiveX also has these button types, as well as one additional button type: the toggle button. A standard button is called a push button. When a push button is clicked, it runs a macro that performs an action. An option button, also called a radio button, gives the user one choice from a limited set of options. A spin button allows for the increase or decrease of value. A spin button features an up-and-down arrow and usually allows the direct entry of a value in addition to incremental change. A toggle button allows the user to alternate a setting or mode between enabled and disabled.

- ActiveX Controls vs. Form Controls

- The biggest difference between ActiveX buttons and Form buttons is where these buttons can be used. Form buttons cannot be used on UserForms or Webpages but ActiveX buttons can. Another difference is that ActiveX buttons can use VBA code and Form Buttons do not use VBA code. ActiveX controls also feature one additional button type: the toggle button. Regardless of button type, ActiveX buttons can be customized to a greater degree than Form buttons. However, form buttons are easier to use.

Read more: The Difference Between Active X Command Buttons & Form Buttons | eHow.com http://www.ehow.com/info_8703112_difference-command-buttons-form-buttons.html#ixzz2BHcew700

Button erzeugen

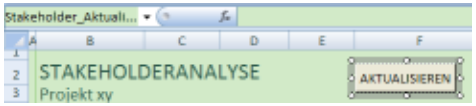
```
Sub setup()
Call createButton("Add", "E2", "addNewKeyTopic")
Call createButton("Delete", "E5", "deleteKeyTopic")
End Sub
Sub createButton(sLabel As String, sPos As String, sFunctionName As String)
' sPos is the location for the button

Sheets("Tabelle 1").Shapes.AddShape(msoShapeRectangle, _
Range(sPos).Left, _
Range(sPos).Top, _
Range(sPos).Width * 2, _
Range(sPos).Height).Select

'Sheets("Input").Shapes.AddShape(msoShapeRoundedRectangle, _
'Range(sPos).Left, _
'Range(sPos).Top, _
'Range(sPos).Width * 2, _
'Range(sPos).Height).Select

Selection.ShapeRange(1).TextFrame.Characters.Text = sLabel

Selection.ShapeRange(1).TextFrame.Characters.Font.Name = "Arial" ' Change the font name
Selection.ShapeRange(1).TextFrame.Characters.Font.Size = 12 ' Change the font size
Selection.ShapeRange(1).TextFrame.Characters.Font.Bold = True
Selection.ShapeRange(1).TextFrame.Characters.Font.Color = RGB(0, 120, 156) 'Change font color

Selection.ShapeRange(1).Fill.ForeColor.RGB = RGB(255, 153, 0) 'Change the background color of the Shape
Selection.ShapeRange(1).Line.ForeColor.RGB = RGB(0, 120, 156) 'Change the border line color of the Shape

Selection.ShapeRange(1).Line.Weight = 1.5 ' Set Line weight
Selection.ShapeRange(1).Line.Style = msoLineSingle

Selection.ShapeRange(1).TextFrame.HorizontalAlignment = xlCenter ' Align the text
Selection.ShapeRange(1).TextFrame.VerticalAlignment = xlTop ' Align the text vertically

Selection.ShapeRange(1).OnAction = """ & sFunctionName & """

End Sub
```

Button: Beschriftung ändern

Mein Code

Wir sprechen von Schaltflächen, die über die Formular-Symbolleiste erzeugt wird !

Zuerst eine Schaltfläche markieren und ihr dann im Adress-Bezeichner einen Namen eingeben:



Nun kann man den Button auch so ansprechen

```
ActiveSheet.Shapes("Stakeholder_Aktualisieren").Select
' Achtung: folgender Code ändert die Beschriftung des Buttons von aktuell AKTUALISIEREN auf OK
Selection.Characters.Text = "OK"
' Achtung: folgender Code ändert den Namen von aktuell "Stakeholder_Aktualisieren" auf Demobutton
Selection.Name = "Demobutton"
' mit nachfolgendem Befehl hebt man die Markierung auf
ActiveSheet.Range("A1").Activate ' Wichtig: ActiveSheet MUSS mit übergeben werden - nur Range.... bringt Fehler
```

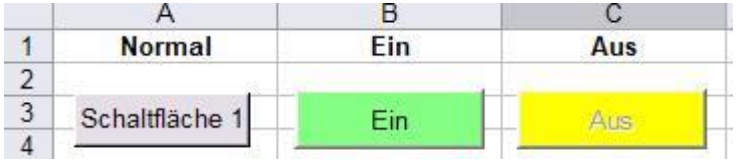
Wichtig: man kann nicht direkt folgenden Befehl verwenden:
Sheets("Tabelle 1").Shapes("Stakeholder_Aktualisieren").Select
Man muss zuerst das Tabellenblatt auswählen und dann auf dem ActiveSheet arbeiten:

```
Sheets("Stakeholder").Activate
ACTIVESHEET.Shapes("Stakeholder_Aktualisieren").Select
Selection.Characters.Text = Sheets("Sprachen").Range("E19")
```

Code 2

Mit diesem Code können Sie auf Benutzereingaben reagieren und via die Beschriftung und Farbe den Benutzer über einen bestimmten Zustand in der Tabelle informieren

```
Private Sub btnStart_Click()
'Name des Buttons ist "btnStart"
With ActiveSheet
   If .OLEObjects("btnStart").Object.Caption = "Aus" Then
      .OLEObjects("btnStart").Object.Caption = "Ein"
      .OLEObjects("btnStart").Object.BackColor = &H80FF80
   Else
      .OLEObjects("btnStart").Object.Caption = "Aus"
      .OLEObjects("btnStart").Object.BackColor = &HFFFF&
      .OLEObjects("btnStart").Enabled = False
   End If
End With
End Sub
```



Button mit eigener VBA-Prozedur generieren

Wie man einen Button generiert und mit VBA-Code hinterlegt => VBA-CODE GENERIEREN

Checkbox siehe Kontrollkästchen

Dropdownlisten zoomen + automatisch öffnen

Dropdownlisten kann man in ihrer Schrift-Größe ja nicht verändern

Folgender Code ändert automatisch beim Anwählen einer Zelle mit Dropdown-Menü die Zoomstufe und öffnet automatisch die Dropdownliste

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

' Automatisches Zoom beim Anwählen einer Zelle in einer Spalte, die ein Dropdownfeld enthält

```
If Target.Columns.Count = 1 And Target.Rows.Count = 1 Then ' es soll nur greifen, wenn nur EINE Zelle markiert ist

   If Target.Column = 11 Then ' wenn man in Spalte J klickt

      If Sheets("Optionen").Range("C8") = True Then ' wenn das Zoomen aktiviert ist
         Range("A3") = ActiveWindow.Zoom ' speichern der aktuellen Zoomstufe in ausgeblendeter Zelle A3
         If Sheets("Optionen").Range("F8") > 100 Then
            ActiveWindow.Zoom = Sheets("Optionen").Range("F8")
           Else ' wenn jemand das Feld irrtümlich geleert hat
            ActiveWindow.Zoom = 200
         End If
         'Target.SpecialCells(xlCellTypeAllValidation).Activate ' dieser Code ist nicht notwendig - er würde auch das SelectionChange erneut aufrufen
         SendKeys "%{DOWN}" ' öffnet das Dropdownfeld
         Exit Sub
      End If
   End If

End If

' Zurücksetzen der Zoomstufe in allen anderen Fällen
If Range("A3") <> "" Then
      On Error Resume Next
      ActiveWindow.Zoom = Range("A3")
      Range("A3") = ""
      On Error GoTo 0
End If
```

Entfernen/ausblenden von Shape-Objekten und -Steuerungselementen

Ron de Bruin (last update 29-July-2007)
Go back to the Excel tips page

Shapes collection

Members of the Shapes collection are:

1. ActiveX controls (Control Toolbox) or a linked or embedded OLE objects
2. Controls from the Forms toolbar
3. Controls from the Drawing toolbar
4. Pictures, charts, ...

You see that all objects/controls are a member of the Shapes collection.

Below you find examples to delete or hide the members of this collection.

Tip: if you only want to hide all shapes for a moment then you can use the
toggle shortcut Ctrl 6 (This is for the whole workbook)

Manual

Excel 97-2003

If you want to delete all objects/controls on a worksheet you can do it manual like this in 97-2003:

1. Press F5
2. Click on Special
3. Choose Objects
4. OK
5. Press the Delete button

Note: for Activex(control toolbox) controls you must be in "Design Mode"
Use the first button on the Control toolbox toolbar to toggle this mode.

You can do the same for Comments.

Excel 2007

In Excel 2007 there is no way to select all shapes.
F5>Special Objects will not select ActiveX and forms controls in Excel 2007.

With VBA code

Delete all shapes

Use this macro to delete all shapes on the worksheet
Working in all Excel versions.

```
Sub Shapes1()
'Delete all Objects except Comments
    On Error Resume Next
    ActiveSheet.DrawingObjects.Visible = True
    ActiveSheet.DrawingObjects.Delete
    On Error GoTo 0
End Sub
```

Use this to delete comments

```
Sub Comments()
'This will delete all comments
```

```
    ActiveSheet.Cells.ClearComments
End Sub
```

Warning :
Not use code like below because it is possible that

It will delete the AutoFilter dropdowns
It will delete the Data>Validation(List option) dropdowns
Excel crash if there are comments on the sheet

Note: Not every Excel versions have all problems.

```
Sub NotUseThisMacro()
'Loop through the Shapes collection
    Dim myshape As Shape
    For Each myshape In ActiveSheet.Shapes
        myshape.Delete
    Next myshape
End Sub
```

Delete only specific shapes

What if you only want to delete control toolbox controls, Pictures or forms controls.
You can loop through the collection and check the Type of the control.

12 = ActiveX control (control toolbox) or a linked or embedded OLE object.
13 = Picture
 8 = Forms controls

For Type 8 we use another macro to avoid the problem of losing AutoFilter and
Data Validation dropdowns on your worksheet.
See the example in this section "Delete only Forms controls"

```
Sub Shapes2()
'Loop through the Shapes collection and use the Type number of the control
    Dim myshape As Shape
    For Each myshape In ActiveSheet.Shapes

        ' ActiveX control (control toolbox) or a linked or embedded OLE object.
        If myshape.Type = 12 Then myshape.Delete
        ' You can also use  myshape.Visible = False

    Next myshape
End Sub
```

If you want to know all the Type numbers of all controls on your worksheet you can run this macro
to add a new worksheet with the names and Type numbers of all objects on your worksheet.
You can find the number then that you must use to delete the objects you want.

```
Sub ListAllObjectsActiveSheet()
    Dim NewSheet As Worksheet
```

```
    Dim MySheet As Worksheet
    Dim myshape As Shape
    Dim I As Long

    Set MySheet = ActiveSheet
    Set NewSheet = Worksheets.Add

    With NewSheet
        .Range("A1").Value = "Name"
        .Range("B1").Value = "Visible(-1) or Not Visible(0)"
        .Range("C1").Value = "Shape type"
        I = 2

        For Each myshape In MySheet.Shapes
            .Cells(I, 1).Value = myshape.Name
            .Cells(I, 2).Value = myshape.Visible
            .Cells(I, 3).Value = myshape.Type
            I = I + 1
        Next myshape

        .Range("A1:C1").Font.Bold = True
        .Columns.AutoFit
        .Range("A1:C" & Rows.Count).Sort Key1:=Range("C1"), _
                    Order1:=xlAscending, Header:=xlYes
    End With

End Sub
```

Delete only Forms controls

This example avoid the problem of losing AutoFilter and
Data Validation dropdowns on your worksheet when you use Type 8.

```
Sub Shapes4()
'Dave Peterson and Bob Phillips
'Example only for the Forms controls
    Dim shp As Shape
    Dim testStr As String

    For Each shp In ActiveSheet.Shapes

        If shp.Type = 8 Then
            If shp.FormControlType = 2 Then
                testStr = ""
                On Error Resume Next
```

```
            testStr = shp.TopLeftCell.Address
            On Error GoTo 0
            If testStr <> "" Then shp.Delete
         Else
            shp.Delete
         End If
      End If

   Next shp
End Sub
```

In the workaround macro above we use FormControlType = 2 in the loop (xlDropDown).
AutoFilter and Data Validation dropdowns do not have TopLeftCell.Address and the macro will not delete this DropDowns.

Other FormControl constants are:
(only for the Forms controls)

xlButtonControl = 0
xlCheckBox = 1
xlDropDown = 2
xlEditBox = 3
xlGroupBox = 4
xlLabel = 5
xlListBox = 6
xlOptionButton = 7
xlScrollBar = 8
xlSpinner = 9

Delete one shape

Because all objects/controls are a member of the shapes collection we can use this to delete one button, picture or ?

```
Sub Delete_One_Shape()
   ActiveSheet.Shapes("YourShapeName").Delete
End Sub

Sub Hide_One_Shape()
   ActiveSheet.Shapes("YourShapeName").Visible = False
End Sub
```

Specific examples for Activex(control toolbox) or Forms controls

For most things the macros in the first section of this page are Ok but if you only want to delete Forms buttons or ActiveX buttons then look here for a few examples.

ActiveX controls (Control Toolbox) or linked or embedded OLE objects

```
Sub OLEObjects1()
'Hide all ActiveX controls(Control Toolbox)or linked or embedded OLE objects
    On Error Resume Next
    ActiveSheet.OLEObjects.Visible = False
    On Error GoTo 0
End Sub

Sub OLEObjects2()
'Delete all ActiveX controls(Control Toolbox)or linked or embedded OLE objects
    On Error Resume Next
    ActiveSheet.OLEObjects.Visible = True
    ActiveSheet.OLEObjects.Delete
    On Error GoTo 0
End Sub

Sub OLEObjects3()
'Delete/hide only all CommandButtons from the Control Toolbox
    Dim obj As OLEObject
    For Each obj In ActiveSheet.OLEObjects
        If TypeOf obj.Object Is MSForms.CommandButton Then
            obj.Delete
            ' or obj.Visible = False if you want to hide them
        End If
    Next
End Sub
```

Others are :

```
MSForms.CheckBox
MSForms.TextBox
MSForms.OptionButton
MSForms.ListBox
MSForms.ComboBox
MSForms.ToggleButton
MSForms.SpinButton
MSForms.ScrollBar
MSForms.Label
MSForms.Image
```

```
Sub OLEObjects4()
'Hide one ActiveX control(Control Toolbox)or a linked or embedded OLE object
    ActiveSheet.OLEObjects("CommandButton1").Visible = False
End Sub

Sub OLEObjects5()
```

```
'Delete one ActiveX control(Control Toolbox)or a linked or embedded OLE object
    ActiveSheet.OLEObjects("CommandButton1").Delete
End Sub
```
Because Control Toolbox controls are also a member of the Shapes collection you can also use this
```
Sub OLEObjects6()
'Hide one Control Toolbox button or Control
    ActiveSheet.Shapes("CommandButton1").Visible = False
End Sub

Sub OLEObjects7()
'Delete one Control Toolbox button or Control
    ActiveSheet.Shapes("CommandButton1").Delete
End Sub
```
Forms controls
```
Sub Forms1()
'Delete All Forms buttons
    ActiveSheet.Buttons.Delete
End Sub

Sub Forms2()
'Hide All Forms buttons
    ActiveSheet.Buttons.Visible = False
End Sub

Sub Forms3()
'Delete one Forms button
    ActiveSheet.Buttons("Button 1").Delete
End Sub

Sub Forms4()
'Hide one Forms button
    ActiveSheet.Buttons("Button 1").Visible = False
End Sub
```
Instead of Buttons you can also use

OptionButtons
CheckBoxes
DropDowns

Because Forms controls are also a member of the Shapes collection you can also use this
```
Sub Forms5()
'One Forms button or Control
    ActiveSheet.Shapes("Button 1").Delete
End Sub
```

```
Sub Forms6()
'One Forms button or Control
   ActiveSheet.Shapes("Button 1").Visible = False
End Sub
```

Entfernen von eingefügten Steuerelementen

Ich arbeite ja immer mit der Formular-Leiste - aber wenn mit der Steuerelement-Toolbox-Symbolleiste gearbeitet wurde, dann kann man die Command-Buttons nicht mit rechtem Mausclick ändern (löschen / neues Makro zuweisen / Text ändern …)

Um auf solche Commandboxen Zugriff zu erhalten, muss man die Steuerelement-Toolboxleiste einblenden und ein Element darin anklicken - anschließend erhält man wie gewöhnlich Zugriff auf den Button

FARBEN - Systemfarben einstellen der aktuellen Mappe

ActiveWorkbook.colors(17) = RGB(255, 255, 255)

Dies geht sogar in Excel 2010:

```
   ActiveCell.Interior.ColorIndex = 34 , zuerst Färben der Zelle mit der Systemfarbe 34
   ActiveWorkbook.Colors(34) = RGB(128, 128, 128) , dann ändern der Systemfarbe 34
```

ACHTUNG: Colorindex und SchemeColor sind 7 Nummern auseinander !

Colorindex 34=SchemeColor 41 !!

BSP markiere eine Linie im Excelblatt und starte folgendes Makro

```
   farbe = InputBox("Farbe 0-255")
   ActiveWorkbook.Colors(34) = RGB(255 - farbe, farbe, 255 - farbe)
   Range("A1").Interior.ColorIndex = 34
   Selection.ShapeRange.Line.ForeColor.SchemeColor = 41
```

LEIDER: wenn man nachträglich mit ActiveWorkbook.Colors(34) = RGB(… die Systemfarbe 34 ändert, würde zwar eine gefärbte Zelle die neue Farbe automatisch bekommen, aber die mit SchemeColor gefärbten Objekte werden nicht automatisch nachgezogen.

Schöne Farbrotationen wie sie z.B. bei AMOS (Amiga) möglich waren, gehen also nur für Zellfarben – aber nicht für Shapes.

FARBE RGB-Farbwerte anzeigen eines Objekts oder Zellfarbe

```
Dim Rot As Long, Gruen As Long, Blau As Long, Wert As Long
Wert = Selection.ShapeRange.Fill.BackColor ' oder  Wert = Range("B2").Interior.Color

On Error Resume Next
Rot = Wert Mod 256
Wert = (Wert - Rot) / 256
Gruen = Wert Mod 256
Wert = (Wert - Gruen) / 256
Blau = Wert Mod 256

MsgBox "Rot : " & Rot & " -  Grün : " & Gruen & " - Blau : " & Blau
```

Farben in Excel 2003 und in 2007/2010

Excel 2003 Color Palette has an index of 56 colors which can be modified using VBA. Each color in the palette is associated with a unique value in the index that can be changed programatically. At times it is useful to know the relative positioning of the various colors within this index as well as how various versions of Excel treat colors. This is the first in the series of articles that will expore this further.

How to generate Excel Color Index using VBA

Here's is a sample code snippet (with a few minor modifications) from mvps.org that helps one generate a color palette and place it an Excel worksheet :

```
Sub colors56()

'57 colors, 0 to 56

Dim i As Long
Dim str0 As String, str As String
Cells(1, 1) = "Interior"
Cells(1, 2) = "Font"
Cells(1, 3) = "HTML"
Cells(1, 4) = "RED"
Cells(1, 5) = "GREEN"
Cells(1, 6) = "BLUE"
Cells(1, 7) = "COLOR"


For i = 0 To 56
    Cells(i + 2, 1).Interior.ColorIndex = i
    Cells(i + 2, 2).Font.ColorIndex = i
    Cells(i + 2, 2).Value = "[Color " & i & "]"
    str0 = Right("000000" & Hex(Cells(i + 2, 1).Interior.Color), 6)
    'Excel shows nibbles in reverse order so make it as RGB
   str = Right(str0, 2) & Mid(str0, 3, 2) & Left(str0, 2)
    'generating 2 columns in the HTML table
   Cells(i + 2, 3) = "#" & str
    Cells(i + 2, 4).Formula = "=Hex2dec(""" & Right(str0, 2) & """)"
    Cells(i + 2, 5).Formula = "=Hex2dec(""" & Mid(str0, 3, 2) & """)"
    Cells(i + 2, 6).Formula = "=Hex2dec(""" & Left(str0, 2) & """)"
    Cells(i + 2, 7) = "[Color " & i & "]"
Next i
End Sub
```

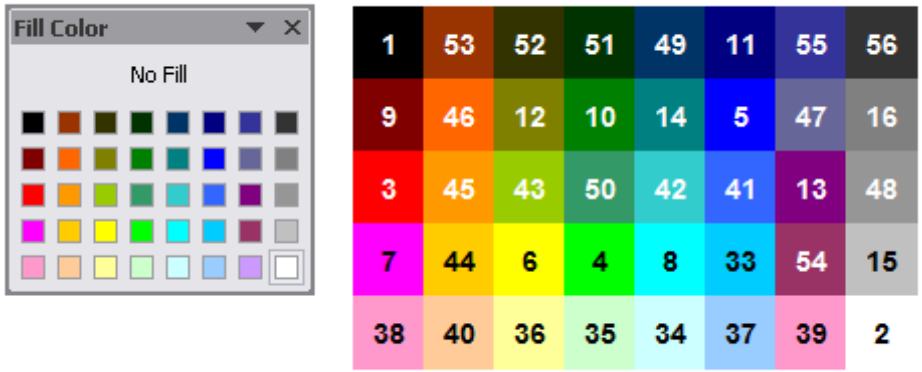
The output of the code will be something akin to what is shown below.

Interior	Font	HTML	RED	GREEN	BLUE	COLOR
	[Color 0]	#FFFFFF	255	255	255	[Color 0]
	[Color 1]	#000000	0	0	0	[Color 1]
		#FFFFFF	255	255	255	[Color 2]
	[Color 3]	#FF0000	255	0	0	[Color 3]
	[Color 4]	#00FF00	0	255	0	[Color 4]
	[Color 5]	#0000FF	0	0	255	[Color 5]
	[Color 6]	#FFFF00	255	255	0	[Color 6]
	[Color 7]	#FF00FF	255	0	255	[Color 7]
	[Color 8]	#00FFFF	0	255	255	[Color 8]
	[Color 9]	#800000	128	0	0	[Color 9]
	[Color 10]	#008000	0	128	0	[Color 10]
	[Color 11]	#000080	0	0	128	[Color 11]
	[Color 12]	#808000	128	128	0	[Color 12]
	[Color 13]	#800080	128	0	128	[Color 13]
	[Color 14]	#008080	0	128	128	[Color 14]
	[Color 15]	#C0C0C0	192	192	192	[Color 15]
	[Color 16]	#808080	128	128	128	[Color 16]
	[Color 17]	#9999FF	153	153	255	[Color 17]
	[Color 18]	#993366	153	51	102	[Color 18]
	[Color 19]	#FFFFCC	255	255	204	[Color 19]
	[Color 20]	#CCFFFF	204	255	255	[Color 20]
	[Color 21]	#660066	102	0	102	[Color 21]
	[Color 22]	#FF8080	255	128	128	[Color 22]
	[Color 23]	#0066CC	0	102	204	[Color 23]
	[Color 24]	#CCCCFF	204	204	255	[Color 24]
	[Color 25]	#000080	0	0	128	[Color 25]
	[Color 26]	#FF00FF	255	0	255	[Color 26]
	[Color 27]	#FFFF00	255	255	0	[Color 27]
	[Color 28]	#00FFFF	0	255	255	[Color 28]
	[Color 29]	#800080	128	0	128	[Color 29]
	[Color 30]	#800000	128	0	0	[Color 30]
	[Color 31]	#008080	0	128	128	[Color 31]
	[Color 32]	#0000FF	0	0	255	[Color 32]
	[Color 33]	#00CCFF	0	204	255	[Color 33]
	[Color 34]	#CCFFFF	204	255	255	[Color 34]
	[Color 35]	#CCFFCC	204	255	204	[Color 35]
	[Color 36]	#FFFF99	255	255	153	[Color 36]
	[Color 37]	#99CCFF	153	204	255	[Color 37]
	[Color 38]	#FF99CC	255	153	204	[Color 38]
	[Color 39]	#CC99FF	204	153	255	[Color 39]
	[Color 40]	#FFCC99	255	204	153	[Color 40]
	[Color 41]	#3366FF	51	102	255	[Color 41]
	[Color 42]	#33CCCC	51	204	204	[Color 42]
	[Color 43]	#99CC00	153	204	0	[Color 43]
	[Color 44]	#FFCC00	255	204	0	[Color 44]
	[Color 45]	#FF9900	255	153	0	[Color 45]
	[Color 46]	#FF6600	255	102	0	[Color 46]
	[Color 47]	#666699	102	102	153	[Color 47]
	[Color 48]	#969696	150	150	150	[Color 48]
	[Color 49]	#003366	0	51	102	[Color 49]

(Please note: Values -1 and 0 can be assigned to an object. However you cannot change those color values in the palette which would lead me to assume that Excel provides 58 (56 + 2) assignable color values while there being only 56 modifiable color values.)

Look's downright ugly isn't it. The interesting thing is that there seems to be no apparent logic to the allocation of color index values to various colors – it neither proceeds from light (#FFFFFF) to dark (#000000) or follow a sequential numbering pattern in the palette (left to right or up to down). One would think that the simplest way for anyone to create the palette would have been to number the colors in some logical fashion so that those colors could be modified using a program far more easily than by having to remember the position of each individual color in the palette.

Excel Color Index



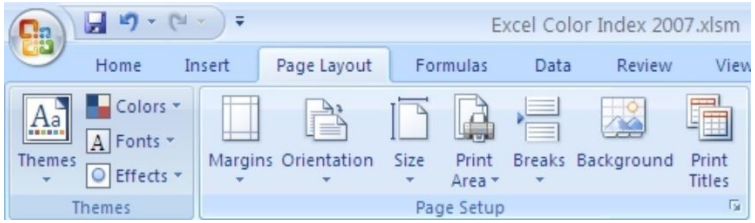
How to change Excel Color Palette using VBA

You can use the assign a new color to a the palette at a particular index position by simple specifying the index number and then using the .Colors function to assign a new RGB value to it. The R, G and B signify the red, green and blue hues that make up the color. (You can get the color index from the index chart shown above)

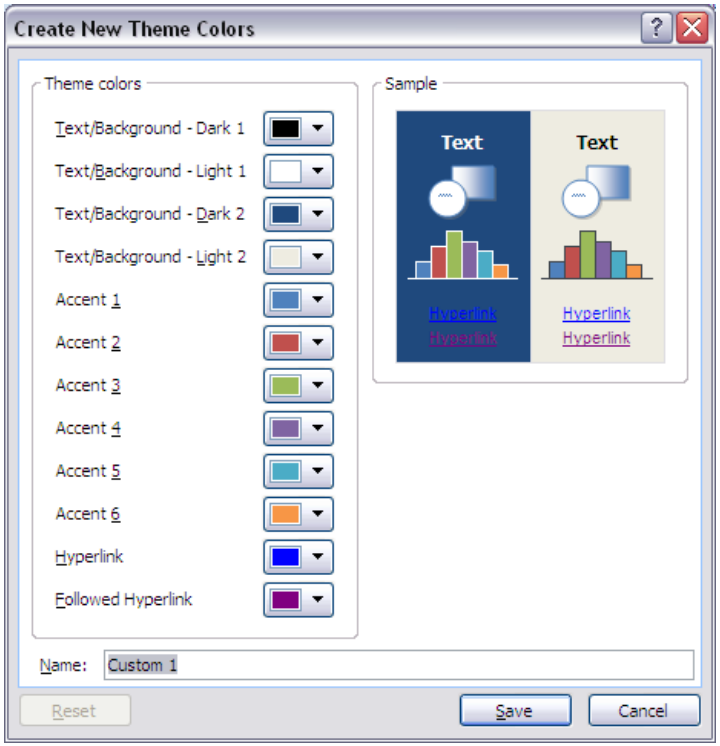
```
Sub change_palette_color
    dim color_index as long
    color_index = 10
    ActiveWorkbook.Colors(color_index) = RGB(128, 128, 128)
End sub
```

How to change Excel Color Palette manually in Excel 2007

Now Excel 2007 is a completely different ball of wax when it comes to color handling. The color palette theme can be change by accessing the "Colors" option under the "Page Layout" ribbon.



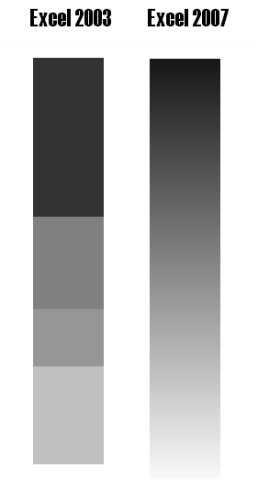
You can also define new color combinations (or select from a host of pre-existing ones) by using the options provided under this menu. The good part is that the color combinations are not just a mix of the existing 56 colors but new colors, which means that when you choose a new color theme, the RGB values of the new colors are different from the ones you had previously.



Accessing Color to object in Excel 2007 using VBA

The number of colors in the color index remains that same at 56. However additional properties such as **.ThemeColor** and **.TintAndShade** enable to work with nearly unlimited colors. This is a major improvement over Excel 2003 which used to “snap” custom colors to the ones which were already existing in the palette. Therefore, a piece of code would generate different results in Excel 2003 and 2007:

```
Sub test_color()
For i = 1 To 256
    Cells(i, 1).Interior.Color = RGB(i, i, i)
Next i
End Sub
```



As you can see, in Excel 2007 there is a smooth gradient from the darkest to the lightest hue while in Excel 2003, the colors change in bands.

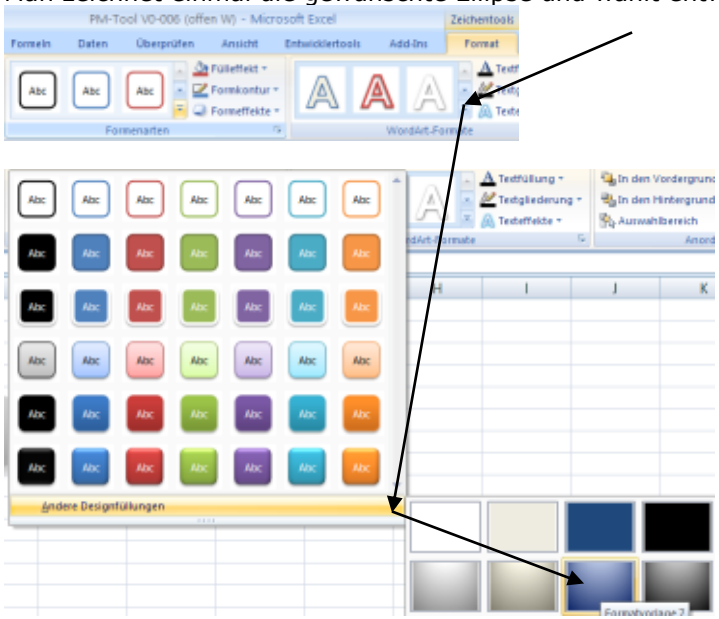
As I said before, you can use the **.ThemeColor** and **.TintAndShade** properties to change color tint and theme:

```
Sub test_color()
For i = 1 To 256
    Cells(i, 1).Interior.ThemeColor = xlThemeColorAccent1
    Cells(i, 1).Interior.TintAndShade = (i * 2 - 256) / 256
Next i
End Sub
```

(Just bear in mind that you can enter a number from -1 (darkest) to 1 (lightest) for the TintAndShade property with 0 (zero) being neutral. Attempting to set this property to a value less than -1 or more than 1 results in a run-time error: “The specified value is out of range.” This property works for both theme colors and nontheme colors.)

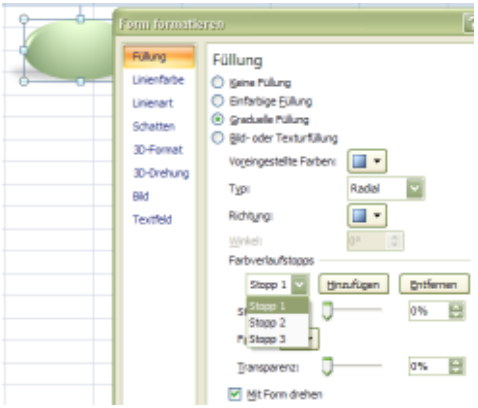
FARBEN eines Objekts einstellen

Man zeichnet einmal die gewünschte Ellipse und wählt entweder hier ein feines Preset aus:



Oder man spielt sich mit rechtem Mausklick und FORM FORMATIEREN vor allem in diesem Register herum:

und hier



Man muss eigentlich nur den gewünschten Verlauf von den Relfexionen und 3D-Effekten hinbekommen - die Farben selbst sind egal, weil man die ohnedies besser mit VBA fix färbt (sonst ist nur eine Systemfarbe hinterlegt, die auf jedem PC anders aussehen kann.

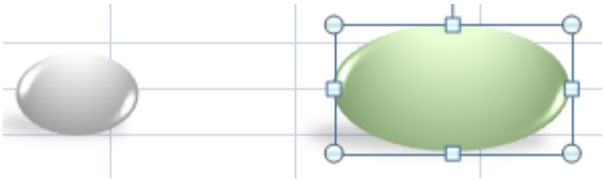
Mein Code, um meine speziellen Ellipsen gewünscht zu färben:

```
Sub FAERBE_OBJEKT()
' diese Prozedur kommt nie zum Einsatz
' sie diente nur ganz anfänglich dazu, die Ellipsen-Vorlagen einzufärben
' dazu musste man die betreffende Vorlagenellipse im Blatt SHAPES nur markieren und danach hier
' den gewünschten Code aktivieren
' Wichtig: ist das Element nicht markiert, kommt eine Fehlermeldung

' hellgrün dunkel
' Selection.ShapeRange.Fill.GradientStops(2).Color.RGB = RGB(235, 255, 215)
' Selection.ShapeRange.Fill.BackColor.RGB = RGB(104, 134, 84)

' hellgrün mittel
' Selection.ShapeRange.Fill.GradientStops(2).Color.RGB = RGB(240, 255, 225)
' Selection.ShapeRange.Fill.BackColor.RGB = RGB(114, 144, 94)

' hellgrün hell
 Selection.ShapeRange.Fill.GradientStops(2).Color.RGB = RGB(245, 255, 235)
 Selection.ShapeRange.Fill.BackColor.RGB = RGB(174, 194, 144)

' graugrün
' Selection.ShapeRange.Fill.GradientStops(2).Color.RGB = RGB(245, 250, 225)
' Selection.ShapeRange.Fill.BackColor.RGB = RGB(114, 124, 94)


End Sub
```

Theorie

Es nervt etwas, dass die Preset-Formate in Excel abhängig sind von der Hintergrundfarbe des Desktops:

Wenn man das Hintegrundweiß auf Grün einstellt - erhält man z.B. diese schönen grünen Ellipsen. Öffnet aber jemand diese Vorlage auf einem Rechner mit weiß, hat er nur eine graue Ellipse.



Mit nachfolgendem Code konnte ich die Ellipse wieder wie zuvor auf dunkelgrün (BackColor) und hellgrün (fill.gradientstops(2)) einstellen

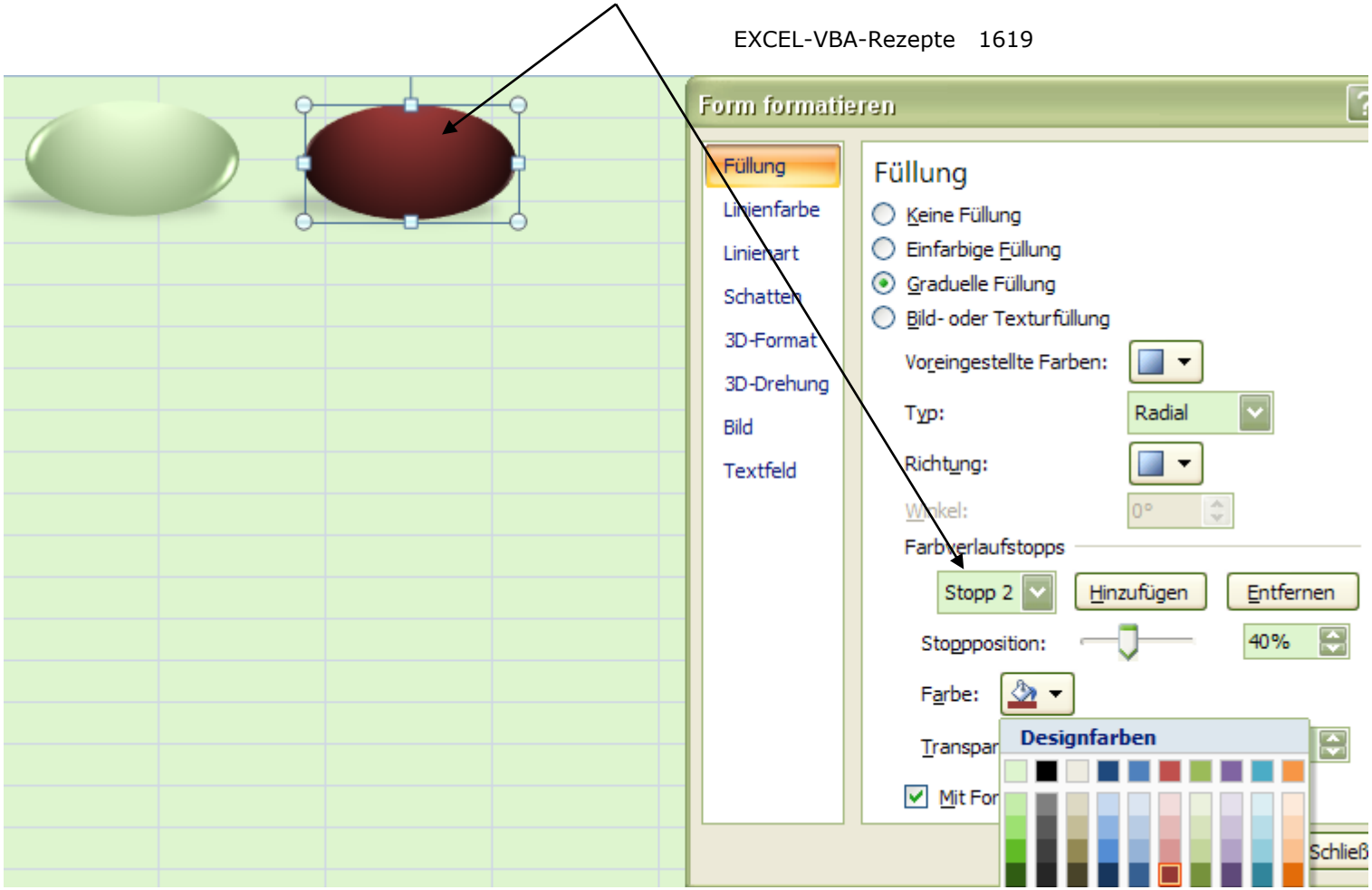
```
Selection.ShapeRange.Fill.GradientStops(2).Color.RGB = RGB(235, 255, 215)
Selection.ShapeRange.Fill.BackColor.RGB = RGB(104, 134, 84)
```

Der folgende Befehl bewirkt gar nichts an Veränderung

```
Selection.ShapeRange.Fill.ForeColor.RGB = RGB(200, 255, 200)
```

Daher: die hellere von den beiden Farben in der Ellipse im obigen Screenshot-BSp ist nicht die fill.forecolor

Die helle Farbe ist ein STOPP:



Setzen und Auslesen kann man sie mit

MYSHAPE.Fill.GradientStops(2).Color.RGB
bzw. mit
Selection.ShapeRange.Fill.GradientStops(2).Color.RGB

Im obigen Beispiel ausgelesen mit folgendem Code

```
Dim Rot As Long, Gruen As Long, Blau As Long, Wert As Long
Wert = Selection.ShapeRange.Fill.GradientStops(2).Color.RGB
On Error Resume Next
```

```
  Rot = Wert Mod 256
  Wert = (Wert - Rot) / 256
  Gruen = Wert Mod 256
  Wert = (Wert - Gruen) / 256
  Blau = Wert Mod 256
MsgBox "Rot : " & Rot & " -  Grün : " & Gruen & " - Blau : " & Blau
```

Als ich dann mal möglichst alle möglichen FILL-Parameter andruckte funktionierten folgende Zeilen (mit Semikolon sind am Schluss die Werte angezeigt)

```
MsgBox Selection.ShapeRange.Fill.BackColor.RGB ' 5539432 - das ist RGB(104, 134, 84)
MsgBox Selection.ShapeRange.Fill.GradientColorType ' 4
' MsgBox Selection.ShapeRange.Fill.GradientDegree ' -
' MsgBox Selection.ShapeRange.Fill.GradientStops ' -
MsgBox Selection.ShapeRange.Fill.GradientStops.Count ' 3
' MsgBox Selection.ShapeRange.Fill.GradientStops.Item.Color ' 3
' MsgBox Selection.ShapeRange.Fill.GradientStops.Item.Position ' -
MsgBox Selection.ShapeRange.Fill.GradientStyle ' -2
MsgBox Selection.ShapeRange.Fill.GradientVariant ' 0
' MsgBox Selection.ShapeRange.Fill.OneColorGradient ' -
MsgBox Selection.ShapeRange.Fill.Pattern ' -2
' MsgBox Selection.ShapeRange.Fill.Patterned ' -
' MsgBox Selection.ShapeRange.Fill.PresetGradient ' -
MsgBox Selection.ShapeRange.Fill.PresetGradientType ' -2
' MsgBox Selection.ShapeRange.Fill.TwoColorGradient ' -
MsgBox Selection.ShapeRange.Fill.Type ' 3
MsgBox Selection.ShapeRange.Fill.BackColor.ObjectThemeColor ' 2
MsgBox Selection.ShapeRange.Fill.BackColor.SchemeColor ' 57
MsgBox Selection.ShapeRange.Fill.BackColor.TintAndShade ' -0,8
MsgBox Selection.ShapeRange.Fill.BackColor.Type ' 2
MsgBox Selection.ShapeRange.Fill.ForeColor.ObjectThemeColor ' 0
MsgBox Selection.ShapeRange.Fill.ForeColor.SchemeColor ' 42
MsgBox Selection.ShapeRange.Fill.ForeColor.TintAndShade ' 0
```

```
MsgBox Selection.ShapeRange.Fill.ForeColor.Type ' 1
```

FARBEN - Powertools-VBA

' CODE DES MODULS

```
Option Explicit
Option Compare Text


Public Const C_RGB_RED As Long = &HFF&
Public Const C_RGB_GREEN As Long = &HFF00&
Public Const C_RGB_BLUE As Long = &HFF0000
Public Const C_RGB_WHITE As Long = &HFFFFFF
Public Const C_RGB_BLACK As Long = &H0&
Public Const C_MIN_COLOR_INDEX = 1
Public Const C_MAX_COLOR_INDEX = 56
Public Const C_MIN_RGB = C_RGB_BLACK
Public Const C_MAX_RGB = C_RGB_WHITE
Public Const C_SHIFT_ONE_BYTE = &H100&
Public Const C_SHIFT_TWO_BYTES = &H10000


Private Declare Function GetActiveWindow Lib "user32.dll" () As Long
Private Declare Function ChooseColorDlg Lib "comdlg32.dll" Alias "ChooseColorA" ( _
    pChoosecolor As CHOOSECOLOR) As Long


Private Const CC_RGBINIT = &H1&
Private Const CC_FULLOPEN = &H2&
Private Const CC_PREVENTFULLOPEN = &H4&
Private Const CC_SHOWHELP = &H8&
Private Const CC_ENABLEHOOK = &H10&
Private Const CC_ENABLETEMPLATE = &H20&
Private Const CC_ENABLETEMPLATEHANDLE = &H40&
Private Const CC_SOLIDCOLOR = &H80&
Private Const CC_ANYCOLOR = &H100&


Private Type CHOOSECOLOR
```

```
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    rgbResult As Long
    lpCustColors As Long
    flags As Long
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type


Private dwCustClrs(0 To 15) As Long
Private Init As Boolean

Public Function ChooseColorDialog(DefaultColor As Long) As Long
    Dim lpChoosecolor As CHOOSECOLOR
    With lpChoosecolor
        .lStructSize = Len(lpChoosecolor)
        .hwndOwner = GetActiveWindow
        .rgbResult = DefaultColor
        .lpCustColors = VarPtr(dwCustClrs(0))
        .flags = CC_ANYCOLOR Or CC_RGBINIT Or CC_FULLOPEN
    End With
    If ChooseColorDlg(lpChoosecolor) Then
        ChooseColorDialog = lpChoosecolor.rgbResult
    Else
        ChooseColorDialog = -1
    End If
End Function

Sub FORMAT_FARBEN_EINSTELLEN()
' Der Menüpunkt "EDITIERE GESPEICHERTE FARBEN"

Windows("Powertools.xls").Visible = True

End Sub

Sub FORMAT_FARBWAHL()

' Die Taste FARBE ÄNDERN in der Tabelle FARBE in den Powertools
    Dim RGBColor As Long
    Dim Default As Long
    Default = RGB(255, 0, 255) 'default to purple
```

```
    Default = ActiveCell.Offset(1, 0) 'default to purple
    RGBColor = ChooseColorDialog(DefaultColor:=Default)
    If RGBColor < 0 Then

    Else
        ActiveCell.Offset(1, 0) = RGBColor
        'MsgBox CDbl(ActiveCell.Offset(-1, 0))
        ActiveWorkbook.Colors(CDbl(ActiveCell.Offset(-1, 0))) = RGBColor
    End If



End Sub
Sub TESTER()

End Sub
```

' CODE FÜR FARBESETZEN

```
Sub FORMAT_FARBEN_NEU()
    ActiveWorkbook.Colors(3) = RGB(218, 0, 0)
    ActiveWorkbook.Colors(4) = RGB(173, 250, 134)
    ActiveWorkbook.Colors(5) = RGB(17, 57, 255)
    ActiveWorkbook.Colors(6) = RGB(255, 255, 155)
    ActiveWorkbook.Colors(7) = RGB(255, 37, 37)
    ActiveWorkbook.Colors(8) = RGB(109, 255, 199)
    ActiveWorkbook.Colors(9) = RGB(104, 0, 0)
    ActiveWorkbook.Colors(10) = RGB(32, 153, 23)
    ActiveWorkbook.Colors(11) = RGB(15, 0, 210)
    ActiveWorkbook.Colors(12) = RGB(255, 168, 51)
    ActiveWorkbook.Colors(13) = RGB(118, 0, 118)
    ActiveWorkbook.Colors(14) = RGB(0, 102, 138)
    ActiveWorkbook.Colors(15) = RGB(232, 232, 232)
    ActiveWorkbook.Colors(16) = RGB(176, 176, 176)
    ActiveWorkbook.Colors(33) = RGB(55, 150, 255)
    ActiveWorkbook.Colors(34) = RGB(200, 255, 255)
    ActiveWorkbook.Colors(35) = RGB(219, 255, 200)
    ActiveWorkbook.Colors(36) = RGB(255, 255, 200)
    ActiveWorkbook.Colors(37) = RGB(183, 210, 255)
    ActiveWorkbook.Colors(38) = RGB(255, 201, 201)
```

```
    ActiveWorkbook.Colors(39) = RGB(219, 183, 255)
    ActiveWorkbook.Colors(40) = RGB(255, 233, 200)
    ActiveWorkbook.Colors(41) = RGB(105, 141, 255)
    ActiveWorkbook.Colors(42) = RGB(87, 239, 255)
    ActiveWorkbook.Colors(43) = RGB(255, 226, 51)
    ActiveWorkbook.Colors(44) = RGB(254, 209, 144)
    ActiveWorkbook.Colors(45) = RGB(230, 181, 116)
    ActiveWorkbook.Colors(46) = RGB(218, 137, 70)
    ActiveWorkbook.Colors(47) = RGB(89, 62, 142)
    ActiveWorkbook.Colors(48) = RGB(211, 211, 211)
    ActiveWorkbook.Colors(49) = RGB(0, 12, 102)
    ActiveWorkbook.Colors(50) = RGB(78, 231, 57)
    ActiveWorkbook.Colors(51) = RGB(0, 90, 0)
    ActiveWorkbook.Colors(52) = RGB(164, 82, 0)
    ActiveWorkbook.Colors(53) = RGB(103, 58, 13)
    ActiveWorkbook.Colors(54) = RGB(155, 96, 234)
    ActiveWorkbook.Colors(55) = RGB(49, 43, 133)
    ActiveWorkbook.Colors(56) = RGB(77, 77, 77)

End Sub

Sub FORMAT_FARBEN_DEFAULT()
    ActiveWorkbook.Colors(1) = RGB(0, 0, 0)
    ActiveWorkbook.Colors(2) = RGB(255, 255, 255)
    ActiveWorkbook.Colors(3) = RGB(255, 0, 0)
    ActiveWorkbook.Colors(4) = RGB(0, 255, 0)
    ActiveWorkbook.Colors(5) = RGB(0, 0, 255)
    ActiveWorkbook.Colors(6) = RGB(255, 255, 0)
    ActiveWorkbook.Colors(7) = RGB(255, 0, 255)
    ActiveWorkbook.Colors(8) = RGB(0, 255, 255)
    ActiveWorkbook.Colors(9) = RGB(128, 0, 0)
    ActiveWorkbook.Colors(10) = RGB(0, 128, 0)
    ActiveWorkbook.Colors(11) = RGB(0, 0, 128)
    ActiveWorkbook.Colors(12) = RGB(128, 128, 0)
    ActiveWorkbook.Colors(13) = RGB(128, 0, 128)
    ActiveWorkbook.Colors(14) = RGB(0, 128, 128)
    ActiveWorkbook.Colors(15) = RGB(192, 192, 192)
    ActiveWorkbook.Colors(16) = RGB(128, 128, 128)
    ActiveWorkbook.Colors(33) = RGB(0, 204, 255)
    ActiveWorkbook.Colors(34) = RGB(204, 255, 255)
    ActiveWorkbook.Colors(35) = RGB(204, 255, 204)
    ActiveWorkbook.Colors(36) = RGB(255, 255, 153)
    ActiveWorkbook.Colors(37) = RGB(153, 204, 255)
    ActiveWorkbook.Colors(38) = RGB(255, 153, 204)
```

```
    ActiveWorkbook.Colors(39) = RGB(200, 153, 255)
    ActiveWorkbook.Colors(40) = RGB(255, 204, 153)
    ActiveWorkbook.Colors(41) = RGB(51, 102, 255)
    ActiveWorkbook.Colors(42) = RGB(51, 204, 204)
    ActiveWorkbook.Colors(43) = RGB(153, 204, 0)
    ActiveWorkbook.Colors(44) = RGB(255, 204, 0)
    ActiveWorkbook.Colors(45) = RGB(255, 153, 0)
    ActiveWorkbook.Colors(46) = RGB(255, 102, 0)
    ActiveWorkbook.Colors(47) = RGB(102, 102, 153)
    ActiveWorkbook.Colors(48) = RGB(150, 150, 150)
    ActiveWorkbook.Colors(49) = RGB(0, 51, 102)
    ActiveWorkbook.Colors(50) = RGB(51, 153, 102)
    ActiveWorkbook.Colors(51) = RGB(0, 51, 0)
    ActiveWorkbook.Colors(52) = RGB(51, 51, 0)
    ActiveWorkbook.Colors(53) = RGB(153, 51, 0)
    ActiveWorkbook.Colors(54) = RGB(153, 51, 102)
    ActiveWorkbook.Colors(55) = RGB(51, 51, 153)
    ActiveWorkbook.Colors(56) = RGB(51, 51, 51)
End Sub

Sub FORMAT_FARBEN_IN_USERFORM3_SETZEN()
    ' 4 Arten eine Farbe zu übergeben - hier jedesmal ROT
'    UserForm2.CommandButton1.BackColor = vbRed
'    UserForm2.CommandButton1.BackColor = &HFF& ' Langfassung = &H000000FF&
'    UserForm2.CommandButton1.BackColor = QBColor(12)
'    UserForm2.CommandButton1.BackColor = ActiveWorkbook.Colors(3)
    UserForm3.CommandButton1.BackColor = ActiveWorkbook.Colors(1)
    UserForm3.CommandButton2.BackColor = ActiveWorkbook.Colors(53)
    UserForm3.CommandButton3.BackColor = ActiveWorkbook.Colors(52)
    UserForm3.CommandButton4.BackColor = ActiveWorkbook.Colors(51)
    UserForm3.CommandButton5.BackColor = ActiveWorkbook.Colors(49)
    UserForm3.CommandButton6.BackColor = ActiveWorkbook.Colors(11)
    UserForm3.CommandButton7.BackColor = ActiveWorkbook.Colors(55)
    UserForm3.CommandButton8.BackColor = ActiveWorkbook.Colors(56)

    UserForm3.CommandButton9.BackColor = ActiveWorkbook.Colors(16)
    UserForm3.CommandButton10.BackColor = ActiveWorkbook.Colors(47)
    UserForm3.CommandButton11.BackColor = ActiveWorkbook.Colors(5)
    UserForm3.CommandButton12.BackColor = ActiveWorkbook.Colors(14)
    UserForm3.CommandButton13.BackColor = ActiveWorkbook.Colors(10)
    UserForm3.CommandButton14.BackColor = ActiveWorkbook.Colors(12)
    UserForm3.CommandButton15.BackColor = ActiveWorkbook.Colors(46)
    UserForm3.CommandButton16.BackColor = ActiveWorkbook.Colors(9)
```

```
    UserForm3.CommandButton17.BackColor = ActiveWorkbook.Colors(48)
    UserForm3.CommandButton18.BackColor = ActiveWorkbook.Colors(13)
    UserForm3.CommandButton19.BackColor = ActiveWorkbook.Colors(41)
    UserForm3.CommandButton20.BackColor = ActiveWorkbook.Colors(42)
    UserForm3.CommandButton21.BackColor = ActiveWorkbook.Colors(50)
    UserForm3.CommandButton22.BackColor = ActiveWorkbook.Colors(43)
    UserForm3.CommandButton23.BackColor = ActiveWorkbook.Colors(45)
    UserForm3.CommandButton24.BackColor = ActiveWorkbook.Colors(3)

    UserForm3.CommandButton25.BackColor = ActiveWorkbook.Colors(15)
    UserForm3.CommandButton26.BackColor = ActiveWorkbook.Colors(54)
    UserForm3.CommandButton27.BackColor = ActiveWorkbook.Colors(33)
    UserForm3.CommandButton28.BackColor = ActiveWorkbook.Colors(8)
    UserForm3.CommandButton29.BackColor = ActiveWorkbook.Colors(4)
    UserForm3.CommandButton30.BackColor = ActiveWorkbook.Colors(6)
    UserForm3.CommandButton31.BackColor = ActiveWorkbook.Colors(44)
    UserForm3.CommandButton32.BackColor = ActiveWorkbook.Colors(7)

    UserForm3.CommandButton33.BackColor = ActiveWorkbook.Colors(2)
    UserForm3.CommandButton34.BackColor = ActiveWorkbook.Colors(39)
    UserForm3.CommandButton35.BackColor = ActiveWorkbook.Colors(37)
    UserForm3.CommandButton36.BackColor = ActiveWorkbook.Colors(34)
    UserForm3.CommandButton37.BackColor = ActiveWorkbook.Colors(35)
    UserForm3.CommandButton38.BackColor = ActiveWorkbook.Colors(36)
    UserForm3.CommandButton39.BackColor = ActiveWorkbook.Colors(40)
    UserForm3.CommandButton40.BackColor = ActiveWorkbook.Colors(38)

' Untere Farben

    UserForm3.CommandButton48.BackColor = ActiveWorkbook.Colors(1)
    UserForm3.CommandButton47.BackColor = ActiveWorkbook.Colors(53)
    UserForm3.CommandButton46.BackColor = ActiveWorkbook.Colors(52)
    UserForm3.CommandButton45.BackColor = ActiveWorkbook.Colors(51)
    UserForm3.CommandButton44.BackColor = ActiveWorkbook.Colors(49)
    UserForm3.CommandButton43.BackColor = ActiveWorkbook.Colors(11)
    UserForm3.CommandButton42.BackColor = ActiveWorkbook.Colors(55)
    UserForm3.CommandButton41.BackColor = ActiveWorkbook.Colors(56)

    UserForm3.CommandButton49.BackColor = ActiveWorkbook.Colors(16)
    UserForm3.CommandButton50.BackColor = ActiveWorkbook.Colors(47)
    UserForm3.CommandButton51.BackColor = ActiveWorkbook.Colors(5)
    UserForm3.CommandButton52.BackColor = ActiveWorkbook.Colors(14)
    UserForm3.CommandButton53.BackColor = ActiveWorkbook.Colors(10)
    UserForm3.CommandButton54.BackColor = ActiveWorkbook.Colors(12)
```

```
    UserForm3.CommandButton55.BackColor = ActiveWorkbook.Colors(46)
    UserForm3.CommandButton56.BackColor = ActiveWorkbook.Colors(9)

    UserForm3.CommandButton57.BackColor = ActiveWorkbook.Colors(48)
    UserForm3.CommandButton58.BackColor = ActiveWorkbook.Colors(13)
    UserForm3.CommandButton59.BackColor = ActiveWorkbook.Colors(41)
    UserForm3.CommandButton60.BackColor = ActiveWorkbook.Colors(42)
    UserForm3.CommandButton61.BackColor = ActiveWorkbook.Colors(50)
    UserForm3.CommandButton62.BackColor = ActiveWorkbook.Colors(43)
    UserForm3.CommandButton63.BackColor = ActiveWorkbook.Colors(45)
    UserForm3.CommandButton64.BackColor = ActiveWorkbook.Colors(3)

    UserForm3.CommandButton65.BackColor = ActiveWorkbook.Colors(15)
    UserForm3.CommandButton66.BackColor = ActiveWorkbook.Colors(54)
    UserForm3.CommandButton67.BackColor = ActiveWorkbook.Colors(33)
    UserForm3.CommandButton68.BackColor = ActiveWorkbook.Colors(8)
    UserForm3.CommandButton69.BackColor = ActiveWorkbook.Colors(4)
    UserForm3.CommandButton70.BackColor = ActiveWorkbook.Colors(6)
    UserForm3.CommandButton71.BackColor = ActiveWorkbook.Colors(44)
    UserForm3.CommandButton72.BackColor = ActiveWorkbook.Colors(7)

    UserForm3.CommandButton73.BackColor = ActiveWorkbook.Colors(2)
    UserForm3.CommandButton74.BackColor = ActiveWorkbook.Colors(39)
    UserForm3.CommandButton75.BackColor = ActiveWorkbook.Colors(37)
    UserForm3.CommandButton76.BackColor = ActiveWorkbook.Colors(34)
    UserForm3.CommandButton77.BackColor = ActiveWorkbook.Colors(35)
    UserForm3.CommandButton78.BackColor = ActiveWorkbook.Colors(36)
    UserForm3.CommandButton79.BackColor = ActiveWorkbook.Colors(40)
    UserForm3.CommandButton80.BackColor = ActiveWorkbook.Colors(38)
End Sub
```

Fußzeilengröße und Fußzeilentext ändern

```
  With ActiveSheet
    .PageSetup.LeftFooter = "&24" & "text" ' 24 ist die Schriftgrösse
  End With
```

Grafik auswählen + einfügen

VERSION 1 - Poidls Journey

```
Sub LADE_BILD_KLEIN()

' Dies ist eine Hilfsprozedur, die im Hauptbildschirm das kleine Bild oben rechts lädt


' --- Variablen ---

    Dim BILD As String
    Dim MITTE_HORIZONTAL As Integer ' Mitte links/rechts des Tabellenblattes
    Dim MITTE_VERTIKAL As Integer ' gewünschter Mittelpunkt vertikal auf Tabellenblattes

' --- Code ---

' Festlegen der Allgemeinen Variablen

    DATEIPFAD = ThisWorkbook.Path
    BILDERPFAD = DATEIPFAD & "\BILDER\L" & Tabelle5.Range("C3") & "\"
    BILD = BILDERPFAD & "C5.jpg"
    MITTE_HORIZONTAL = 955
    MITTE_VERTIKAL = 215

' Testweise auslesen der Koordinaten des aktuellen Excelfensters

'    MsgBox ActiveWindow.VisibleRange.Top ' sollte 0 sein
'    MsgBox ActiveWindow.VisibleRange.Left ' sollte 0 sein
'    MsgBox ActiveWindow.VisibleRange.Width ' Breite
'    MsgBox ActiveWindow.VisibleRange.Height ' Höhe

' Bisheriges Bild löschen

    On Error Resume Next
    Tabelle2.Shapes("Bild").Delete
    On Error GoTo 0

' Bild einfügen


    ' Dieser Code macht nur eine Verknüpfung zum Bild
     ActiveSheet.Pictures.Insert(BILD).Select

    Selection.ShapeRange.LockAspectRatio = msoTrue
    'Selection.ShapeRange.IncrementLeft 7.5 ' verrücken links-rechts
    'Selection.ShapeRange.IncrementTop 5.25 ' verrücken oben-unten
    Selection.Width = 600
```

```
    Selection.ShapeRange.Left = MITTE_HORIZONTAL - Selection.Width / 2 ' Position links
    Selection.ShapeRange.Top = MITTE_VERTIKAL - Selection.Height / 2 ' Position oben

    ' Eingefügtes Bild mit Standardnamen benennen
    Selection.ShapeRange.Name = "Bild"
    Selection.Name = "Bild"

    ' Format übertragen
    Tabelle99.Shapes("Bild").PickUp
    Tabelle2.Shapes("Bild").Apply

    Range("A4").Select
```

VERSION 2 - RGC

Hier meine Prozedur, mit der wir im RGC-Tool Bilder einfügen - sie läuft auch unter Excel 2007/2010 und fügt dort das Originale Bild ein und nicht nur eine Verknüpfung

```
Sub Bild_Import()
' Auswahl der Datei
Dim DATEI
Dim DATEINAME As String            ' zB 1.jpg
Dim DATEINAMEOHNEENDUNG As String   ' zB 1
Dim DATEIPFAD As String            ' zB C:\Eigene Dateien\
Dim DATEIPFADUNDNAME As String      ' zB C:\Eigene Dateien\1.jpg

Dim MITTE_HORIZONTAL As Integer ' Mitte links/rechts des Tabellenblattes
Dim MITTE_VERTIKAL As Integer ' gewünschter Mittelpunkt vertikal auf Tabellenblattes


MITTE_HORIZONTAL = 230
MITTE_VERTIKAL = 300

' Sicherheitshalber EnableEvents
Application.EnableEvents = True

' Defaultverzeichnis einstellen
'    ChDrive ("C:\Programme\") ' also nur C
'    ChDir ("C:\Programme\") ' also gesamten Pfad als Default

' Auswahl der Datei
```

```
    DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)

' Auslesen Dateivariablen
    DATEIPFADUNDNAME = DATEI
    DATEIPFAD = Left(DATEIPFADUNDNAME, InStrRev(DATEIPFADUNDNAME, "\"))
    DATEINAME = Right(DATEIPFADUNDNAME, Len(DATEIPFADUNDNAME) - InStrRev(DATEIPFADUNDNAME, "\"))
    DATEINAMEOHNEENDUNG = Left(DATEINAME, InStrRev(DATEINAME, ".") - 1)
    'MsgBox DATEIPFADUNDNAME & vbCrLf & DATEIPFAD & vbCrLf & DATEINAME & vbCrLf & DATEINAMEOHNEENDUNG

' Einfügen des Bildes

    On Error GoTo KEINEDATEIAUSGEWÄHLT

    Range("A1").Select

    ' Alter Code, der nur Verknüpfungen zu Bildern macht:
    ' ActiveSheet.Pictures.Insert(DATEIPFADUNDNAME).Select

    ' Neuer Code (unter dem Befehl kommt die Erklärung)
    ActiveSheet.Shapes.AddPicture(DATEIPFADUNDNAME, False, True, 1, 2, 99, 74).Select
' 1 gibt die Position links, und 2 oben an.
' 99 und 74 sind die Bildgröße --> welche aber mit nachfolgendem Code noch bearbeitet wird
' False - nicht als Link speichern
' True - in Mappe speichern


    Selection.ShapeRange.LockAspectRatio = msoTrue
    'Selection.ShapeRange.IncrementLeft 7.5 ' verrücken links-rechts
    'Selection.ShapeRange.IncrementTop 5.25 ' verrücken oben-unten
    If Selection.Width > Selection.Height Then ' wenn Querformat
       Selection.Width = 400
      Else ' wenn Hchformat
       Selection.Height = 100
    End If
    If Selection.Width > 400 Then Selection.Width = 400
    If Selection.Height > 100 Then Selection.Height = 100
    Selection.ShapeRange.Left = MITTE_HORIZONTAL - Selection.Width / 2 ' Position links
    Selection.ShapeRange.Top = MITTE_VERTIKAL - Selection.Height / 2 ' Position oben

    Range("A1").Select

Application.EnableEvents = True
```

```
Exit Sub

KEINEDATEIAUSGEWÄHLT:
   MsgBox "You have selected either no or a non-supported file"

End Sub
```

CODE NEU AB EXCEL 2007

Der Befehl "ActiveSheet.Pictures.Insert(DATEIPFADUNDNAME).Select" liefert nicht mehr die gewünschte Funktion wie noch unter Excel 2003.

Ron De Bruin schreibt dazu:

This is a bug

You can use this in 2007 as a workaround

```
Sub Test()
   Dim myPict As Picture


   With Range("F32")
      Set myPict = .Parent.Pictures.Insert("T:\Sym-enh\CO\logo.bmp")
      myPict.Top = .Top
      myPict.Left = .Left
   End With
End Sub
```

Eine Alternative ist:

```
ActiveSheet.Shapes.AddPicture(("T:\diamod\bmp\" & Bild), False, True, poss, posz, 99, 74).Select
```

poss gibt die Position links, und posz oben an.
99,74 ist die Bildgröße --> welche sich dann aber noch bearbeiten lässt.
False -- nicht als Link speichern
True -- in Mappe speichern

CODE ALT VON MIR (bis 2003)

```
Sub Bild_Import()
' Auswahl der Datei
Dim DATEI
Dim DATEINAME As String            ' zB 1.jpg
Dim DATEINAMEOHNEENDUNG As String   ' zB 1
Dim DATEIPFAD As String            ' zB C:\Eigene Dateien\
Dim DATEIPFADUNDNAME As String      ' zB C:\Eigene Dateien\1.jpg

Dim MITTE_HORIZONTAL As Integer ' Mitte links/rechts des Tabellenblattes
Dim MITTE_VERTIKAL As Integer ' gewünschter Mittelpunkt vertikal auf Tabellenblattes


MITTE_HORIZONTAL = 200
MITTE_VERTIKAL = 300

' Defaultverzeichnis einstellen
'    ChDrive ("C:\Programme\") ' also nur C
'    ChDir ("C:\Programme\") ' also gesamten Pfad als Default

' Auswahl der Datei
   DATEI = Application.GetOpenFilename("Alle-Dateien (*.*),*.*,", MultiSelect:=xlNone)

' Auslesen Dateivariablen
   DATEIPFADUNDNAME = DATEI
   DATEIPFAD = Left(DATEIPFADUNDNAME, InStrRev(DATEIPFADUNDNAME, "\"))
   DATEINAME = Right(DATEIPFADUNDNAME, Len(DATEIPFADUNDNAME) - InStrRev(DATEIPFADUNDNAME, "\"))
   DATEINAMEOHNEENDUNG = Left(DATEINAME, InStrRev(DATEINAME, ".") - 1)
   'MsgBox DATEIPFADUNDNAME & vbCrLf & DATEIPFAD & vbCrLf & DATEINAME & vbCrLf & DATEINAMEOHNEENDUNG

' Einfügen des Bildes

   On Error GoTo KEINEDATEIAUSGEWÄHLT
```

```
    Range("A1").Select
    ActiveSheet.Pictures.Insert(DATEIPFADUNDNAME).Select
    Selection.ShapeRange.LockAspectRatio = msoTrue
    'Selection.ShapeRange.IncrementLeft 7.5 ' verrücken links-rechts
    'Selection.ShapeRange.IncrementTop 5.25 ' verrücken oben-unten
    If Selection.Width > Selection.Height Then ' wenn Querformat
       Selection.Width = 200
     Else ' wenn Hchformat
       Selection.Height = 100
    End If
    Selection.ShapeRange.Left = MITTE_HORIZONTAL - Selection.Width / 2 ' Position links
    Selection.ShapeRange.Top = MITTE_VERTIKAL - Selection.Height / 2 ' Position oben

    Range("A1").Select

Exit Sub

KEINEDATEIAUSGEWÄHLT:
    MsgBox "You have selected either no or a non-supported file"

End Sub
```

Grafix positionieren und Größe ändern

```
Tabelle1.Shapes("Bild1").Top ' Position oben

Tabelle1.Shapes("Bild1").Left ' Position links
Tabelle1.Shapes("Bild1").LockAspectRatio = msoTrue ' Seitenverhältnis sperren vor dem Größe ändern

Tabelle1.Shapes("Bild1").Width ' Breite
Tabelle1.Shapes("Bild1").Height ' Höhe
```

Feststellen der richtigen Position und Größe je nach Bildschirmauflösung

Variante 1

Sollte es mal schwierig sein ein Bild optimal zu Zoomen kann man entweder die Bildschirmauflösung auslesen bei voll maximiertem Excelfenster mit

```
'    MsgBox ActiveWindow.VisibleRange.Width ' Breite
'    MsgBox ActiveWindow.VisibleRange.Height ' Höhe
```

Danach abhängig von dieser Bildschirmgröße die Bildposition und Größe anpassen

Variante 2

Alternativ kann man auch auf einen bestimmten, sichtbarsein sollenden Zellbereich zoomen und die Zoomstufe auslesen - über die man auch erfährt wie groß die Bildschirmauflösung des Users ist

Variante 3

Auslesen was die aktuell oberste linke Zelle eines Bildes ist und was die aktuell unterste rechte Zelle ist und so lange kleiner machen, bis es passt

MsgBox ActiveSheet.Shapes("Bild").TopLeftCell.Address
MsgBox ActiveSheet.Shapes("Bild").BottomRightCell.Address

Gruppierungselemente betexten

Man muss eine Gruppierung nicht zwingend aufheben, um den Namen zu ändern-

ActiveSheet.Shapes("Group 23").GroupItems("shapeHeader1").Name = "NeuerName"

Gruppierungen aufheben (alle)

```
For each sh in activesheet.shapes
   if sh.Type = msoGroup then sh.Ungroup
Next
```

Kommentare formatieren, schreiben

1. Kommentare per Makro formatieren

```
Sub KommentarSchrift()
   Dim Cmt As Comment
   Set Cmt = ActiveCell.AddComment
   Cmt.Text "Mein Kommentar"
   With Cmt.Shape.TextFrame.Characters.Font
      .Name = "Arial"
      .Size = 14
```

```
    End With
End Sub
```

2. Zellbereich mit Kommentar versehen

```
Sub KommentarFestlegen()
    Dim C As Range
    For Each C In Selection
        If Not C.Comment Is Nothing Then
            C.NoteText "Kommentar!"
        End If
    Next C
End Sub
```

3. Grösse des Kommentarfensters automatisch festlegen

```
Sub Kommentargrösse()
     Dim Kommentarzelle As Range
    Application.DisplayCommentIndicator = xlCommentAndIndicator
    For Each Kommentarzelle In ActiveSheet.Cells.SpecialCells(1)
    Kommentarzelle.Comment.Shape.Select True
    Selection.AutoSize = True
    'Selection.ShapeRange.Width = 150
    'Selection.ShapeRange.Height = 100
    Next Application.DisplayCommentIndicator = xlCommentIndicatorOnly
End Sub
```

Kontrollkästchen anwählen

Wichtig: es muss immer das Tabellenblatt mit übergeben warden - es darf nicht mit Checkboxes… beginnen

```
ActiveSheet.CheckBoxes("Check Box 1").Value = True ' oder False
```

```
        ' Deaktiviere die beiden Checkboxen
        Worksheets("VERS1").CheckBoxes("Vers1_K100").Value = False
        Worksheets("VERS1").CheckBoxes("Vers1_V100").Value = False
```

Alle Kontrollkästchen abwählen:

```
On Error Resume Next
```

' Checkbox deaktivieren (hier: es gibt maximal 99 Checkboxen - bzw genau: die höchste Checkboxennummer hat die Nummer 99 - hat man 200 Checkboxen

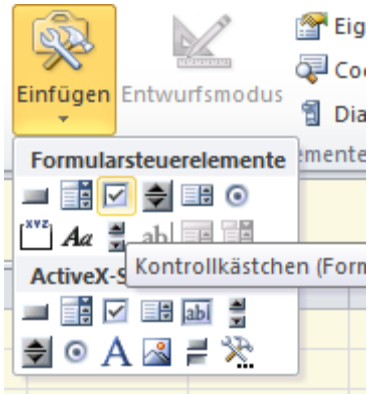
gemacht und die ersten 150 gelöscht, so hat man zwar nur 50 Checkboxen übrig, aber deren Nummern gehen von 150 bis 200)
```
For T = 1 To 99
ActiveSheet.CheckBoxes("Check Box " & T).Value = False
Next T
```

Kontrollkästchen auslesen

```
Sub Blende_DMP()
' dies ist der Code für 1.Checkbox zum Ein/Ausblenden der Dienstleistungs und Marktbezogenen Ziele

For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 1)) = "DMP" Then
      If Tabelle8.CheckBoxes("DMP").Value = 1 Then ' True
        Rows(Z).Hidden = False
       Else ' False
        Rows(Z).Hidden = True
      End If
   End If
Next Z
```

Kontrollkästchen / Optionsbutton als ActiveX oder Steuerelement

Es gibt sie ja zweimal – entweder als Formularsteuerelement oder als ActiveX-Element

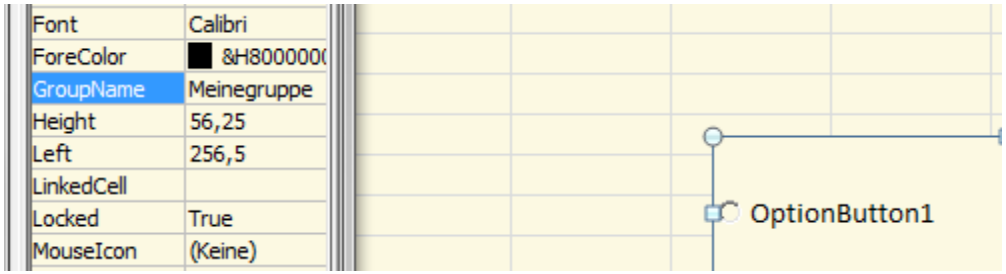


Ich arbeite ja lieber mit den Formularsteuerelementen, weil man da direkt in den Eigenschaften mehr machen kann – bei den ActiveX-Elementen muss man mehr in die VBA-Eigenschaften des Elementes gehen.

Auch das Auslösen von Makros ist verschieden: Formularelemente können direkt ein Makro zugewiesen bekommen, das irgendwo ist im Code. ActiveX-Elemente haben immer ihr eigenes Makro (im Tabellencode der Tabelle, wo sie sind und können höchstens von dort aus in ein anderes Makro verweisen).

Formularelemtente können einen Wert direkt in eine Zelle schreiben und können von dort auch quasi ausgelesen werden. ActiveX müssen immer selbst direkt ausgelesen werden.

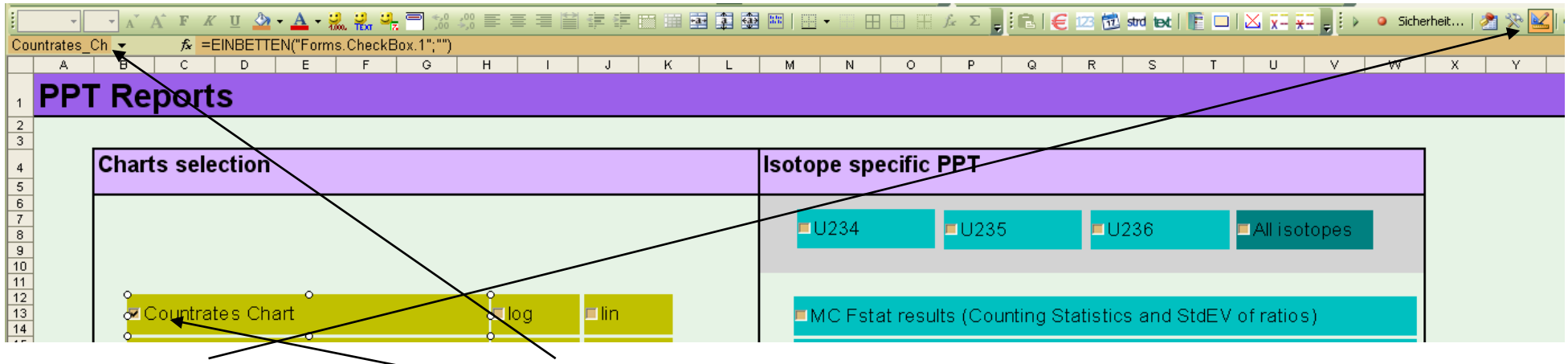
Formular-Optionsschaltflächen werden über ein Gruppenfeld zusammengefasst (das allerdings auch immer eine Rahmenlinie erzeugt) – ActiveX-Optionsschaltflächen werden über einen Wert im Feld Groups zusammengefasst:



Einen Vorteil haben die ActiveX-Elemente – man kann ihre Schriftgröße ändern hier im Feld FONT. Abwählbar sind sie aber nicht direkt in Excel sondern nur in der VBA-Umgebung. Alternative: man versieht sie mit einem Code, der mit Doppelclick das Kästchen wieder abwählt:

```
Private Sub OptionButton01_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
   Me.OptionButton01.Value = False
   Cancel = True
End Sub
```

Kontrollkästchen / Checkbox von Symbolleiste Steuerelement-Toolbox



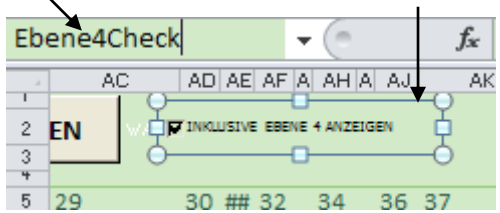
Ist man im Editiermodus kann man solche Checkboxes hier oben auf ihren Namen untersuchen.

Zum Abfragen des Status muss man immer den Tabellenblattnamen mit übergeben:

```
If Worksheets("PPT_Reports").Countrates_Charts = False Then
    MsgBox "false"
    Else
    MsgBox "True"
End If
```

Kontrollkästchen-Text ändern

Ich verwende ja immer die Kontrollkästchen der Formularsteuerelemente. Nach dem Einfügen ändere ich deren Namen (was am leichtesten in der Adresszelle hier geht, wenn das Kontrollkästchen angewählt ist):



Anschließend lautet der Code - bei aktivem Tabellenblatt

```
ActiveSheet.Shapes("Ebene4Check").TextFrame.Characters.Text = Sheets("Sprachen").Range("E103")
```

CODE 2

```
Public Sub test()
'Aus Symbolleiste Formular
Tabelle1.Shapes("Kontrollkästchen 1").TextFrame.Characters.Text = "Erste Auswahl"
'Aus SteuerelemteToolbox
Tabelle1.OLEObjects("Checkbox1").Object.Caption = "Erste Auswahl"
```

Koordinaten auf Tabellenblatt anzeigen

Zeichnet man eine Autoform auf ein Tabellenblatt unter Angabe der genauen Pixelposition (und verschiebt sie mit .IncrementLeft oder .IncrementTop um weitere Pixel), so sind das immer andere Koordinaten als der Mauszeiger hat, die man ja auch abrufen könnte. Zudem sind die Mauszeiger-Koordinaten von der Zoom-Stufe abhängig, aber nicht die Koordinaten beim Platzieren und Verschieben von Autoformen.

Um also die perfekten Koordinaten zu finden, arbeitet man am besten mit einer Kreuz-Autoform (zwei Linien), die man mit den Cursortasten verschiebt und deren jeweils aktuelle Position man sich anzeigen lässt in der Excel-Statuszeile. Hier mein Code

VERSION 1 mit Fadenkreuz

```
' nachfolgender Code dient dazu zuerst ein Kreuz zu zeichnen (Sub Zeichne_Kreuz),
' welches anschließend mit den Cursortasten verschoben werden kann
' wichtig: zuerst muss der Code Set_Cursor_Tasten ausgeführt werden
' dann der Code Zeichne_Kreuz
' die aktuellen Koordinaten des Kreuzes werden dann in der Statuszeile von Excel angezeigt


Public X_Line As Shape
Public Y_Line As Shape
Public X As Integer
Public Y As Integer

Sub SET_CURSOR_TASTEN()
    Application.OnKey "{LEFT}", "LINKS"
    Application.OnKey "{UP}", "UP"
    Application.OnKey "{DOWN}", "DOWN"
    Application.OnKey "{RIGHT}", "RECHTS"
End Sub

Sub UNSET_CURSOR_TASTEN()
```

```
    Application.OnKey "{LEFT}"
    Application.OnKey "{UP}"
    Application.OnKey "{DOWN}"
    Application.OnKey "{RIGHT}"
End Sub

Sub ZEICHNE_KREUZ()
    X = 10
    Y = 10

    Set X_Line = ActiveSheet.Shapes.AddLine(X - 5, Y, X + 5, Y)
    Set Y_Line = ActiveSheet.Shapes.AddLine(X, Y - 5, X, Y + 5)
End Sub

Sub LINKS()
    If X = 5 Then Exit Sub
    X = X - 1
    X_Line.IncrementLeft -1
    Y_Line.IncrementLeft -1
    Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub

Sub RECHTS()
    X = X + 1
    X_Line.IncrementLeft 1
    Y_Line.IncrementLeft 1
    Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub

Sub UP()
    If Y = 5 Then Exit Sub
    Y = Y - 1
    X_Line.IncrementTop -1
    Y_Line.IncrementTop -1
    Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub

Sub DOWN()
    Y = Y + 1
    X_Line.IncrementTop 1
    Y_Line.IncrementTop 1
    Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub
```

VERSION 2 mit AUTOFORM, die GANZ IN OBERSTE LINKE ECK VERSCHOBEN WURDE MIT DER MAUS UND DIE NOCH MARKIERT IST

```
Public X As Integer
Public Y As Integer

Sub SET_CURSOR_TASTEN()
   Application.OnKey "{LEFT}", "LINKS"
   Application.OnKey "{UP}", "UP"
   Application.OnKey "{DOWN}", "DOWN"
   Application.OnKey "{RIGHT}", "RECHTS"
End Sub

Sub UNSET_CURSOR_TASTEN()
   Application.OnKey "{LEFT}"
   Application.OnKey "{UP}"
   Application.OnKey "{DOWN}"
   Application.OnKey "{RIGHT}"
End Sub

Sub RESET_X_Y()
 X = 0
 Y = 0
End Sub

Sub LINKS()
   If X = 0 Then Exit Sub
   X = X - 1
   Selection.ShapeRange.IncrementLeft -1
   Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub

Sub RECHTS()
   X = X + 1
   Selection.ShapeRange.IncrementLeft 1
   Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub

Sub UP()
   If Y = 0 Then Exit Sub
   Y = Y - 1
   Selection.ShapeRange.IncrementTop -1
   Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub
```

```
Sub DOWN()
   Y = Y + 1
   Selection.ShapeRange.IncrementTop 1
   Application.StatusBar = "Meine Position: X " & X & ", Y " & Y
End Sub
```

Langsamer Shape-VBA-Code in Excel 2007

For reasons known only to Microsoft, macros that add and/or format shapes (rectangles, text boxes, etc.) on worksheets can easily take 10 or 20 times longer in Excel 2007 than in earlier versions.

Fortunately the slow code can be rewritten to achieve acceptable performance in Excel 2007.

What's slow code?

Much of the code that has been written to manipulate shapes works by selecting the shape and then operating on the "selection". Like this:

```
ActiveSheet.Shapes.AddTextBox(...).Select
Selection.Characters.Text = "ABC"
```

There was nothing wrong with this kind of code before Excel 2007. After all, the macro recorder produced code like this and how to write code that addressed shapes directly was not always obvious. And, in any case, there was little benefit in doing so.

Enter Excel 2007. It's not clear why but this kind of code runs like a turtle in quicksand under Excel 2007. And the more shapes you're dealing with the slower it goes.

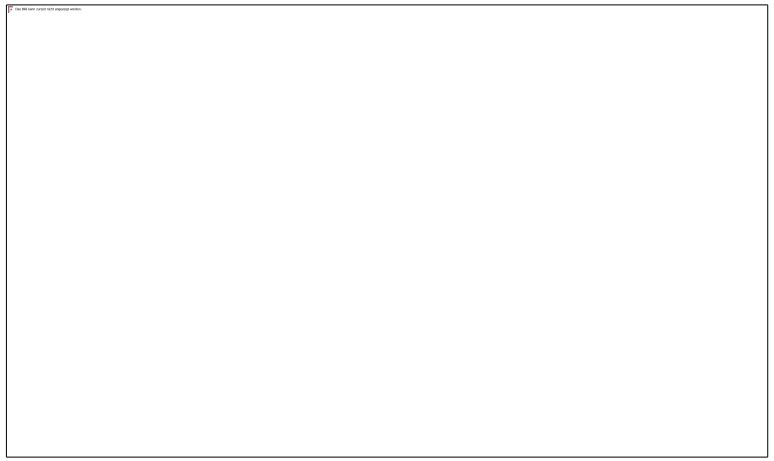
The remedy is to address shapes "directly". Like this:

```
Set sh = ActiveSheet.Shapes.AddTextBox(...)
sh.DrawingObject.Text = "ABC"
```

This code runs about 50% slower in Excel 2007 than Excel 2003, which is equal to or better than macro performance generally.

To see this performance penalty yourself and to get a more complete example of formatting shapes directly download SlowShapesXL2007.zip. The enclosed file, SlowShapesXL2007.xls created by Jim Rech,
is in Excel 97-2003 format so it can be run in Excel 2007 and in earlier versions.

Penalty Ratio from the test code in the workbook



Code run from a shape can crash Excel 2007

Note:This problem is fixed in Office 2007 SP2

If a macro that closes an Excel workbook is assigned to a shape, clicking it in Excel 2007 will cause a crash.
Note: If you have more workbooks open you not have this problem.

Note: It seems that If Office 2003 is also installed on the machine you not have this problem.

To see this, copy the code below into a standard module. Then add a shape, a rectangle for example, to worksheet, right click it, and assign this macro to it.

```
Sub CloseWithoutSaving()
   Dim Response

   Response = MsgBox("Are you sure you want to close the workbook without saving," _
               & vbNewLine & _
               "Click No and use the Save button to save your changes" _
               , vbYesNo, "My test Macro")

   If Response = vbYes Then
      ActiveWorkbook.Close SaveChanges:=False
   Else
      'Do nothing
   End If

End Sub
```

Note: This code runs correctly in Excel 97-2003.

The moral is that you should consider using a Forms or ActiveX button rather than a shape if there is any possibility Excel 2007 will be used. Those controls do not have this problem.

But there is a Workaround :

Copy this macro also in your module and right click on the shape and assign this macro to it.

```
Sub Assign_This_Macro_To_The_Shape()
   Application.OnTime Now + TimeSerial(0, 0, 1), _
                ThisWorkbook.Name & "!CloseWithoutSaving"
End Sub
```

When you click on the Shape it will wait for one second and then run the macro with the close code.
Thanks to Dave Peterson for this tip.

Listbox befüllen und Eintrag auswählen und übernehmen

Während die Combobox mehrspaltig ist, ist eine normal Listbox nur einspaltig.

Befüllen mit Listelementen:

```
Listbox1.AddItem "Eintrag 1"
Listbox1.AddItem "Eintrag 2"
Listbox1.AddItem "Eintrag 3"
```

Auswählen eines angelegten, vorhandenen Eintrages:

```
Listbox1.ListIndex=0 ' erstes Element
```

Unbedingt die Listbox mehrere Zeilen hoch machen.

Sobald man darin ein Element auswählt kann man mit

```
Sub Listbox1_Change
If Listbox1.ListIndex=1 then Cells (1,1)="Eintrag 2"
```

Listfelder siehe Dropdownfelder

Optionbutton: zu Gruppen zusammenfassen

Wenn Sie sehr viele Optionbuttons auf einer Tabelle gruppiert haben,.,... aber nicht zu einer Gruppe zusammengefasst haben :-), dann hilft Ihnen dieses Makro. Auf einfache Weise fasst es alle benötigten Optionbutton zusammen und bennennt diese zu einer Gruppe. Die Anzahl OptionButton pro Gruppe ist frei definierbar.

```
Sub OptionButton_Group()
'spricht alle OptionButton in der Tabelle 1
'und gruppiert diese in Gruppen von je x
'Es dürfen nur OptionButton Objecte in der Tabelle sein

On Error Resume Next
Dim i As Integer, n As Integer,
```

```
Dim myGroup As Integer, grpCounter As Integer, myCounter As Integer
grpCounter = 1
myCounter = 0
myGroup = Inputbox("Wieviele Optionbutton sollen in je eine Gruppe ?","Gruppierung",3)
If Group = "" Then Exit Sub
With Sheets("Tabelle1")
   For i = 1 To .OLEObjects.Count Step myGroup
      For n = 1 To myGroup
         .OLEObjects(i + myCounter).Object.GroupName = "Group" & grpCounter
         myCounter = myCounter + 1
      Next n
      myCounter = 0
      grpCounter = grpCounter + 1
   Next i
End With
End Sub
```

Optionschaltfläche anwählen

```
ActiveSheet.Shapes("Option Button 21").Select
With Selection
   .Value = xlOn
End With
```

Alternativ: wenn der Optionsschaltfläche einen Bezug zu einer Zelle hat, kann man auch deren Wert direkt ändern und die entsprechende Optionsschaltfläche wird angewählt - dies ist wichtig, wenn die betreffende Tabelle nicht sichtbar ist - dann kann man nur diesen Weg wählen !

Schleife durch alle Optionsschaltflächen eines Blattes

Gewünscht: Schleife durch alle Optionsschaltflächen eines Tabellenblattes und sie alle zurücksetzen

```
Sub RESET_OPTIONSSCHALTFLÄCHEN()

Application.ScreenUpdating = False
Application.EnableEvents = False

For i = 1 To ActiveSheet.OLEObjects.Count
  If TypeName(ActiveSheet.OLEObjects(i).Object) = "OptionButton" Then
      ActiveSheet.OLEObjects(i).Object.Value = False ' Optionsschaltfläche inaktiv stellen
  End If
```

```
Next i

Application.ScreenUpdating = True
Application.EnableEvents = True


End Sub
```

Schleife durch alle Shapes / Textfelder einer Arbeitsmappe

Version 1 - Schleife durch ALLE Shapes

Ich nehme gerne diesen Code und Textboxen haben den Wert 8 (Bezeichnungsfeld aus Formular-Symbolleiste), 17 (z.B. Textfeld aus Word) oder 12 (Steuerelement-Textbox) - je nach dem Typ von Textbox.

```
Sub GetShapeProperties()

    Dim sShapes As Shape ', lLoop As Long

    For Each sShapes In ActiveSheet.Shapes
        MsgBox sShapes.Type
        sShapes.Delete
    Next sShapes

End Sub
```

Version 2 - Schleife durch ALLE Shapes

Diese Schleife erzeugt ein neues Tabellenblatt und listet dort alle Shapes aller Blätter auf inkl. Typ

```
Sub GetShapeProperties()

    Dim sShapes As Shape, lLoop As Long
    Dim wsStart As Worksheet, WsNew As Worksheet


    Set wsStart = ActiveSheet
    Set WsNew = Sheets.Add

    'Add headings for our lists. Expand as needed
```

```
    WsNew.Range("A1:F1") = Array("Shape Name", "Shape Type", "Height", "Width", "Left", "Top")

    'Loop through all shapes on active sheet
    For Each sShapes In wsStart.Shapes

        'Increment Variable lLoop for row numbers
        lLoop = lLoop + 1

        With sShapes

            'Add shape properties
            WsNew.Cells(lLoop + 1, 1) = .Name
            WsNew.Cells(lLoop + 1, 2) = .OLEFormat.Object.Name
            WsNew.Cells(lLoop + 1, 3) = .Height
            WsNew.Cells(lLoop + 1, 4) = .Width
            WsNew.Cells(lLoop + 1, 5) = .Left
            WsNew.Cells(lLoop + 1, 6) = .Top
            'Follow the same pattern for more

        End With

    Next sShapes

    'AutoFit Columns.
    WsNew.Columns.AutoFit

End Sub
```

Version 3

```
For Each ws In ActiveWorkbook.Worksheets
    For Each shp In ws.Shapes
        If shp.Type = msoTextBox Then
            With shp.OLEFormat.Object
                .HorizontalAlignment = xlJustify
                .VerticalAlignment = xlCenter
                .ShapeRange.TextFrame.MarginLeft = 7.09
                .ShapeRange.TextFrame.MarginRight = 7.09
                .ShapeRange.TextFrame.MarginTop = 3.69
                .ShapeRange.TextFrame.MarginBottom = 3.69
                .ShapeRange.Line.Weight = 0.25
                .ShapeRange.Fill.Visible = msoTrue
                .ShapeRange.Fill.Solid
```

```
            .ShapeRange.Fill.ForeColor.SchemeColor = 65
            .ShapeRange.Fill.Transparency = 0#
        End With
    End If
  Next shp
Next ws
```

Version 4

```
Sub TextBoxClear2()
  For Shts = 1 To ActiveWorkbook.Sheets.Count
    Sheets(Shts).Select
    For TextCnt = 1 To ActiveSheet.Shapes.Count
      ActiveSheet.Shapes(TextCnt).Select
      … hier fehlt noch der Code um den Text des Shapes zu löschen
    Next
  Next
End Sub
```

Version 5 durch alle Shapes

```
Sub LoopThruShapes()
    Dim sh As Shape
    Dim I As Integer
    I = 1
    For Each sh In ActiveSheet.Shapes
        If sh.Type = msoLine Then
             Cells(I, 1).value = sh.name
             I = I + 1
        End If
    Next
End Sub
```

Statusleiste – Schleifen-Durchlauf-Prozent von 1-100

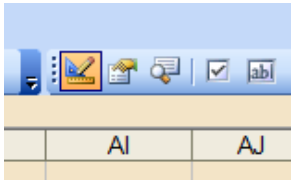
Wenn eine Schleife lange dauert, weil sie z.B. durch viele Zeilen durchwandert, kann man die Prozent der erledigten Schleifendurchläufe in der Statuszeile anzeigen. (Dabei ist Application.Screenupdating = True NICHT notwendig :o)

```
' ----------------------------------------
```

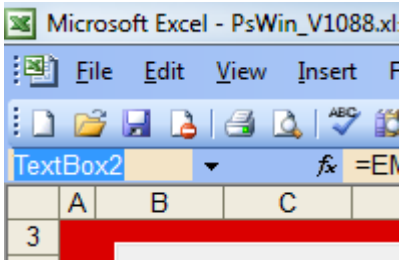
```
' ---  Such- Schleife durch alle Zeilen ---
' ----------------------------------------

Application.ScreenUpdating = False

LETZTEZEILE = LETZTEZELLE(ActiveSheet.Name).Row

For Z = 11 To LETZTEZEILE
' …. der Code

    ' Schleifenprozent
    SCHLEIFENPROZENT = (Z - 10) / (LETZTEZEILE - 10) * 100  ' Da unsere Schleife erst in Zeile 11 beginnt, müssen wir die ersten 10 Zeilen immer abziehen)
    If Round(SCHLEIFENPROZENT, 0) > PROZENT Then
        PROZENT = Round(SCHLEIFENPROZENT, 0)
        Application.StatusBar = PROZENT
    End If

Next Z

Application.ScreenUpdating = True
Application.StatusBar = ""
```

Textbox/=Textfeld (von Symbolleiste Steuerelement-Toolbox) Text ändern

Wichtig: man darf nicht im DESIGN-Modus sein in Excel, um Steuerelemente per VBA zu ändern !



Man sieht oben in der Adresszelle den Namen des Steuerelements – zB Textbox2

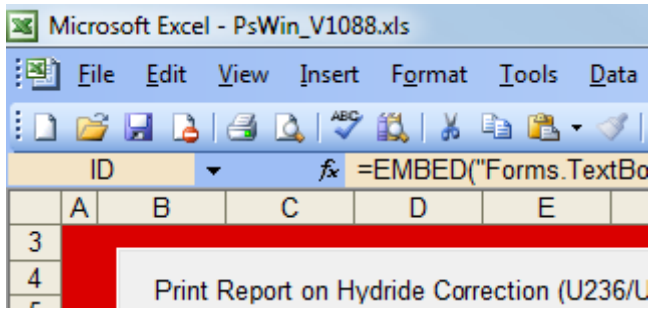


Gib dem Steuerelement hier in der Adress-Zelle den Namen HALLO

Seinen Inhalt ändert man ganz einfach so:

Worksheets("Toolbox").HALLO = "test"

Achtung: im nächsten Beispiel befindet sich das zu ändernde Textfeld ID in einer anderen Arbeitsmappe



Es reicht nicht zuerst die andere Mappe zu aktivieren und dann das Tabellenblatt zu aktivieren und dann die ID zu ändern

Man hat vielmehr folgende zwei Möglichkeiten

1.) Direkt komplett adressieren

```
Sub testen()
Workbooks("PsWin_V1088.xls").Worksheets("ToolBox").ID = "hallo3"
End Sub
```

2.) Aktivieren und dann mit Active… arbeiten

```
Workbooks("PsWin_V1088.xls").Activate
Worksheets("ToolBox").Activate
ActiveSheet.ID = "hallo5"
```

Textbox/=Textfeld (von Symbolleiste Zeichnen) erzeugen und ändern

1. Simples Beispiel für unsichtbares Textfeld (nicht ausgefüllt / keine Rahmenlinie)

```
Set TBOX = WS.Shapes.AddTextbox(msoTextOrientationHorizontal, X, Y, Breite, Höhe)

TBOX.TextFrame2.TextRange.Text = "mein Text"

With TBOX.TextFrame2.TextRange.Font
   .Size = 10
   .Fill.ForeColor.RGB = RGB(255, 200, 0)
End With
With TBOX.Fill
   .Visible = msoFalse
End With
With TBOX.Line
   .Visible = msoFalse
End With
```

2. DETAILS

Wie die Shapes-Beschriftungen, gibt es auch bei der Textbox zwei Arten der Beschriftung:

1.) Textframe (geht auch in Excel vor 2007)
2.) Textframe2 (geht erst ab Excel 2007, kann aber mehr)

Wichtig: bei beiden kann man den Text mit VBCRLF manuell umbrechen an der gewünschten Stelle !

Hier in Variante A sind beide Arten beschrieben - in Variante B ist noch mal die Art 1 mit Textframe - in Variante A ist die Bündigkeit mit zwei Befehlen einzustellen (siehe die beiden roten Zeilen)

Varianten A

```
Sub Orga_Textfeld(X As Integer, Y As Integer, Breite As Integer, Hoehe As Integer, Bezeichnung, Typ As Integer)

' Typ: Haupttitel der zentralen Projektellipse

   Dim ws As Worksheet
```

```
    Dim tbox As Shape

    Set ws = ActiveSheet

    ' Variante 1 mit Textframe (auch schon in Excel vor 2007)

'    Set tbox = ws.Shapes.AddTextbox(msoTextOrientationHorizontal, X, Y, Breite, Hoehe)
'
'    tbox.TextFrame.Characters.Text = Bezeichnung
'    With tbox.TextFrame.Characters.Font
'        .Name = "Calibri"
'        .FontStyle = "Fett" ' oder "Standard"
'        .Size = 10
'        .ColorIndex = xlAutomatic
'    End With
'    With tbox.TextFrame
'        .HorizontalAlignment = xlCenter
'        .VerticalAlignment = xlCenter
'        .Orientation = msoTextOrientationHorizontal
'    End With

   ' Variante 2 mit Textframe2 (ab 2007 und kann mehr)

    Set tbox = ws.Shapes.AddTextbox(msoTextOrientationHorizontal, X, Y, Breite, Hoehe)

    tbox.TextFrame2.TextRange.Text = Bezeichnung

    With tbox.TextFrame2.TextRange.Characters
       '.Font.Fill.ForeColor.RGB = RGB(45, 91, 76) ' kann man auch direkt setzen bei Textframe2.Textrange.Font (siehe unten)
       '.Font.Name = "Calibri"  ' kann man auch direkt setzen bei Textframe2.Textrange.Font (siehe unten)
    End With
    With tbox.TextFrame2.TextRange.Font
       .Size = 10 ' default-Typ 0
       If Typ = 1 Then .Size = 14 ' Zentrale Projekt-Team-Beschriftung
       If Typ = 2 Then .Size = 11 ' Projektmanager + Auftraggeber
       If Typ = 3 Then .Size = 11 ' Team-Ellipsenbeschriftung
       .Bold = msoFalse
       If Typ = 1 Or Typ = 3 Then .Bold = msoTrue
       .Name = "Calibri"
       .Fill.ForeColor.RGB = RGB(255, 255, 255)
       If Typ = 1 Or Typ = 3 Then .Fill.ForeColor.RGB = RGB(45, 91, 76)
    End With
    With tbox.TextFrame2
       .MarginBottom = 10
```

```
        .MarginLeft = 10
        .MarginRight = 10
        .MarginTop = 10
        .VerticalAnchor = msoAnchorMiddle
        .HorizontalAnchor = msoAnchorCenter
        If Typ = 3 Then .HorizontalAnchor = msoAnchorNone
    End With
    With tbox.TextFrame2.TextRange.ParagraphFormat
        .Alignment = msoAlignCenter
        If Typ = 3 Then .Alignment = msoAlignLeft
    End With
    With tbox.Fill
        .Visible = msoFalse
    End With
    With tbox.Line
        .Visible = msoFalse
    End With

End Sub
```

Variante B

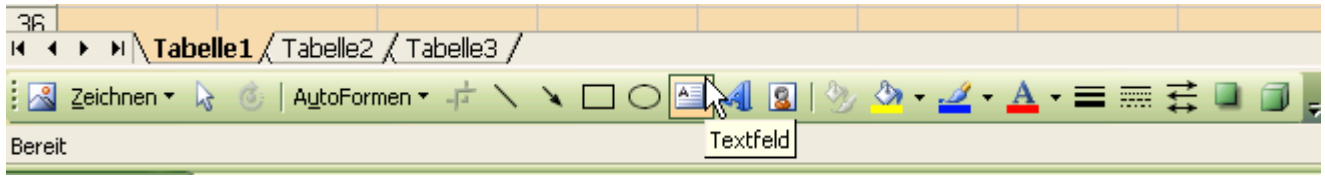
```
    ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, 259.5, 121.5, _
        345#, 75.75).Select
    Selection.Characters.Text = "dies ist ein Test"
    With Selection.Characters(Start:=1, Length:=17).Font
        .Name = "Arial"
        .FontStyle = "Standard"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Fett"
        .Size = 11
        .Strikethrough = False
```

```
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .ReadingOrder = xlContext
    .Orientation = xlHorizontal
    .AutoSize = False
End With
Selection.ShapeRange.Fill.Visible = msoTrue
Selection.ShapeRange.Fill.Solid
Selection.ShapeRange.Fill.ForeColor.SchemeColor = 9
Selection.ShapeRange.Fill.Transparency = 0#
Selection.ShapeRange.Line.Weight = 0.75
Selection.ShapeRange.Line.DashStyle = msoLineSolid
Selection.ShapeRange.Line.Style = msoLineSingle
Selection.ShapeRange.Line.Transparency = 0#
Selection.ShapeRange.Line.Visible = msoFalse
```

Textbox: Text in Textboxen (von Symbolleiste Zeichnen) ändern

Textboxen, die man von der Steuerelementtoolbox einfügt, kann man nicht per VBA verändern.

Den gleichen Effekt wie diese, aber volle Bearbeitbarkeit für User und VBA haben die Textboxen aus der Zeichnenleiste




```
Sub Textbox_Neuen_Text_zuweisen()
"'Text Box 1" ist die englische Bezeichnung
'Im Namensfeld der Tabelle steht "Textfeld1"
With ActiveSheet.Shapes("Text Box 1")
   .TextFrame.Characters.Text = "Neuer Text"
```

```
End With
End Sub
```

Es gibt auch Textboxen, wo der Code ohne dem .Textframe-Teil angesprochen werden

Textbox: Text in Textbox automatisch aus einer Zelle übernehmen

Man macht dies, indem man der Textbox eine Formel zuweist (was nur mittels VBA und nicht direkt geht)

```
Sub Textbox_Neuen_Text_zuweisen()
"'Text Box 1" ist die englische Bezeichnung
'Im Namensfeld der Tabelle steht "Textfeld1"
With ActiveSheet.Shapes("Text Box 1")
    .OLEFormat.Object.Formula = "A1"
End With
End Sub
```

Textbox: ein und ausblenden

```
Sub Show_Textbox()
ActiveSheet.Shapes("Text Box 1").visible = true 'false
End Sub
```

Textbox: Zeilenumbruch

das Textfeld hat zwei Eigenschaften, die du ändern musst, um es mehrzeilig zu machen:

EnterKeyBehavior = True
MultiLine = True

Wenn die Eigenschaften eingestellt sind, kannst du beim Tippen mit der Eingabetaste einen Umbruch erzeugen. In VBA geht das am besten mit der Konstante für Wagenrücklauf und Zeilenvorschub vbCrLf:

```
Me.Textbox = "Zeile 1" & vbCrLf & "Zeile 2"
```

Textbox mit mehr als 255 Zeichen befüllen

Die Textboxen schaffen mehr als 255 und Stringvariablen schaffen auch mehr - aber sobald man eine Stringvariable mit mehr als 255 Zeichen an eine Textbox übergibt, kommt drüben nichts mehr an. Man muss also den Text in max 255-Zeichen große Blöcke reinschaufeln

Lösung 1 verwendet eine Schleifenstruktur um eine lange Zeichenfolge in ein Textfeld einzufügen.

```
Sub Looper()
   Dim i as Integer
   Dim mytxt As String

   ' Create a string 1000 characters in length.
   mytxt = WorksheetFunction.Rept("test", 250)

   ActiveSheet.Shapes("Text Box 1").Select

   With Selection

      ' Initialize text in text box.
      .Text = ""
      For i = 0 To Int(Len(mytxt) / 255)
      .Characters(.Characters.Count + 1).Text = Mid(mytxt, (i * 255) + _
         1, 255)
      Next

   End With

End Sub
```

In Lösung 2 wird die Länge der einzelnen Variablen auf 100 gesetzt und jede **Insert** -Methode fügt eine andere Zeichenfolge an der Position am Ende die vorhergehende Zeichenfolge.

```
Sub NoLoop()
   Dim var1 As String, var2 As String, var3 As String, var4 As String
   Dim first As Integer, second As Integer, third As Integer

   ' Create four text string, each 100 characters in length.
   var1 = String(100,"a")
   var2 = String(100,"b")
   var3 = String(100,"c")
   var4 = String(100,"d")
```

```
   ' The character length of each variable string is 100 characters.
   ' Define the variable equal to length of the first string.
   first = Len(var1) + 1
   ' Define variables equal to length of the original string plus
   ' the length of each additional string.
   second = first + Len(var2)
   third = second + Len(var3)

   ' Select the text box on the worksheet.
   ActiveSheet.Shapes("Text Box 1").Select

   With Selection

      ' Place the first string into the text box.
      .Characters.Text = var1

      ' Place the second, third, and fourth text strings into the
      ' text box.
      .Characters(Start:=first).Text = var2
      .Characters(Start:=second).Text = var3
      .Characters(Start:=third).Text = var4
   End With
End Sub
```

Textfeld mit Mausklick neu schreiben

```
' Wenn man in der Tabelle CHECKLISTE auf die KLIENTENNUMMER (Textfeld) klickt
' Leg das Makro direkt auf das Textfeld

Dim INHALT As String

' Auslesen des alten Textes - bei Textfeldern kann man den Text nicht direkt ändern,
' sondern muss das Feld zuerst wählen
'(Alternative: kein Excel-Textfeld, sondern ein Bezeichnungsfeld nehmen)
ActiveSheet.Shapes("Text Box 636").Select
```

```
INHALT = Selection.Characters.Text

'Abfrage neuer Text
INHALT = InputBox("NEUE KLIENTENNNUMMER EINGEBEN", "KLIENTENNUMMER", INHALT)

' Aussteigen, falls ABBRUCH
If INHALT = "" Then Exit Sub

Application.ScreenUpdating = False

' Zuweisen neuer Text
ActiveSheet.Shapes("Text Box 636").Select ' Auswahl des Textfeldes
Selection.Characters.Text = INHALT ' Zuweisen des Textfeldinhaltes
Range("I20").Select ' Markierung der Gruppe aufheben durch Markierung einer anderen Zelle

Sheets("MWR").Select
Sheets("MWR").Shapes("Text Box 8").Select
Selection.Characters.Text = INHALT
Range("K14").Select

Sheets("Fragen").Select
Sheets("FRAGEN").Shapes("Text Box 17").Select
Selection.Characters.Text = INHALT
Range("N8").Select

Sheets("Checkliste").Select

Application.ScreenUpdating = True
```

Textbox/Textfeld - bei allen die Rahmenlinie ausblenden

```
Sub TextfelderOhneRand()

    Dim i As Integer

    For i = 1 To ActiveDocument.Shapes.Count
        If ActiveDocument.Shapes(i).Type = msoTextBox Then
            ActiveDocument.Shapes(i).Line.Visible = msoFalse
        End If
    Next i

End Sub
```

Textbox/Textfeld löschen

Alle Textfelder löschen

```
For Each objShape In ActiveSheet.Shapes
    If objShape.AutoShapeType = 1 Then ' AutoShapeType 1 ist Textfeld
        objShape.Delete
    End If
Next objShape
```

Wahlweise ein Logo einblenden

Die WP teilte sich eine Vorlage mit der IB - es gab zwei Logos (in verschiedenen Tabellenbereichen) und abwechselnd sollte das eine und dann das andere angezeigt werden, je nach gewählter Optionsschaltfläche.

Ich habe an die Optionsschaltfläche - deren Werte in Zelle L2 geschrieben wird - folgenden VBA-Code angehängt, der zwei Rechtecke, die ohne rahmenlinie sind und quasi unsichtbar die Logos wechselweise abdeckten:

```
If Range("L2") = 2 Then
    ' IB
    ActiveSheet.Shapes("Rectangle 27").Select
    Selection.ShapeRange.ZOrder msoSendToBack
    ActiveSheet.Shapes("Picture 29").Select
    Selection.ShapeRange.ZOrder msoSendToBack
  Else
    ' HH
    ActiveSheet.Shapes("Rectangle 26").Select
    Selection.ShapeRange.ZOrder msoSendToBack
    ActiveSheet.Shapes("Picture 23").Select
    Selection.ShapeRange.ZOrder msoSendToBack
End If
Range("E9").Select ' Damit Focus nicht mehr auf Grafik ist
```

Wordart Text ändern

Eigentlich sollten alle 4 gehen - ich verwende den ersten Code

```
MsgBox KA.Shapes("Titel").TextFrame.Characters.Text
MsgBox KA.Shapes("Titel").TextFrame.Characters.Caption

MsgBox KA.Shapes("Titel").TextFrame2.TextRange.Text
```

```
MsgBox KA.Shapes("Titel").TextFrame2.TextRange.Characters
```

Wordart mit Mausklick neu schreiben

```
Sub FORMAT_KLIENTENNAME()

' Prozedur direkt als Makro auf das Wordart legen

Dim TEXT As String
Dim WORDART As Shape

TEXT = ActiveSheet.Shapes("WordArt 633").TextEffect.TEXT
TEXT = InputBox("NEUEN KLIENTENNAMEN EINGEBEN", "NAME", TEXT)

    ActiveSheet.Shapes("WordArt 633").TextEffect.TEXT = TEXT
    'Selection.ShapeRange.ScaleWidth Len(TEXT) / 5, msoFalse, msoScaleFromTopLeft

If Len(TEXT) < 18 Then
    ActiveSheet.Shapes("WordArt 633").Height = 41#
    ActiveSheet.Shapes("WordArt 633").Width = Len(TEXT) * 30
 Else
    ActiveSheet.Shapes("WordArt 633").Height = 31#
    ActiveSheet.Shapes("WordArt 633").Width = Len(TEXT) * 20
End If
    Range("B8").Select
End Sub
```

Zellbereich als Bild speichern (JPG)

```
Sub Zellen_als_Bild_erportieren()
'erstellt von den markierten Zellen eine Bilddatei (GIF)
Dim Zellbereich As Range
Dim Anz_Markierungen As Integer
Dim Bild As Picture
Dim Diagramm As ChartObject

On Error GoTo ABBRUCH      'falls "Abbrechen" gedrückt wird
'Zellen markieren (Bildbereich)
Set Zellbereich = Application.InputBox _
```

```
  (prompt:="Markieren Sie die Zellen für das Bild" & vbNewLine & _
          "mit Strg + Maus Mehrfachmarkierungen möglich", _
   Title:="Bildauswahl", Type:=8)
On Error GoTo 0

Application.ScreenUpdating = False

'Schleife falls mehrere Bereiche markiert wurden
'für jeden Bereich ein GIF erstellen
For Anz_Markierungen = 1 To Zellbereich.Areas.Count

  Zellbereich.Areas(Anz_Markierungen).Copy
  Worksheets.Add
  Set Bild = ActiveSheet.Pictures.Paste(Link:=True)
  Bild.CopyPicture Appearance:=xlScreen, Format:=xlPicture

  Set Diagramm = ActiveSheet.ChartObjects.Add(0, 0, Bild.Width, Bild.Height)

  With Diagramm
    .Chart.Paste
    .Chart.Export Filename:=ActiveWorkbook.Path & "\test" & Anz_Markierungen & ".gif", FilterName:="gif"
  End With

  Application.DisplayAlerts = False
  ActiveSheet.Delete
  Application.DisplayAlerts = True
  Application.ScreenUpdating = True
Next Anz_Markierungen

Application.ScreenUpdating = True

Set Diagramm = Nothing
Set Bild = Nothing
Set Zellbereich = Nothing
Exit Sub

ABBRUCH:
  MsgBox "", , "Abbruch"
End Sub
```

TASTEN + MAUS

Diverse Maus- und Tastaurfunktionen

```
' ------------------------------------------------------------------------
'
'  Title: Miscellaneous mouse and keyboard routines
'


' ------------------------------------------------------------------------

Option Explicit

' MouseKeyboard version 1.1 by Lord Orwell

' ----------------------------
' Constants & API Declarations
' ----------------------------

' this sub contains all of the mouse and keyboard subs.
Public Const KEYEVENTF_KEYUP = &H2
Private Const SPI_SCREENSAVERRUNNING = 97&
Declare Function CharToOem& Lib "user32" Alias "CharToOemA" (ByVal lpszSrc As String, ByVal lpszDst As String)
Declare Function GetAsyncKeyState% Lib "user32" (ByVal vKey As Long)
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer
'Declare Function GetKeyboardState& Lib "user32" (pbKeyState As String)
Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As
Long)
Declare Function MapVirtualKey Lib "user32" Alias "MapVirtualKeyA" (ByVal wCode As Long, ByVal wMapType As Long) As Long
Declare Function OemKeyScan& Lib "user32" (ByVal wOemChar As Integer)
```

```
Declare Function ShowCursor Lib "user32" (ByVal bShow As Long) As Long
Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As
Long, lpvParam As Any, ByVal fuWinIni As Long) As Long
Declare Function VkKeyScan% Lib "user32" Alias "VkKeyScanA" (ByVal cChar As Byte)


' -----------
' Functions
' -----------

Sub ShowMouseCursor()
    Dim rtn As Long
    rtn = ShowCursor(True)
End Sub

Sub HideMouseCursor()
    Dim rtn As Long
    rtn = ShowCursor(False)
End Sub

Sub DisableTaskKeys()
    'disables ctrl-alt-del, alt-tab, ctrl-f4, etc., keeping you in charge...
    Dim rtn As Long
    rtn = SystemParametersInfo(SPI_SCREENSAVERRUNNING, 1&, 0&, 0)
End Sub

Sub EnableTaskKeys()
    Dim rtn As Long
    rtn = SystemParametersInfo(SPI_SCREENSAVERRUNNING, 0&, 0&, 0)
End Sub

Function ScanCodeToAscii(ScanCode As Long) As Long
    ScanCodeToAscii = MapVirtualKey(ScanCode, 2)
End Function

Function WasKeyPressed(VBKey As Long)
    Dim ScanCode As Integer
    ScanCode = GetKeyState(VBKey)
    If ScanCode And &HFFF0 > 0 Then
        WasKeyPressed = True
    Else
        WasKeyPressed = False
```

```
    End If
End Function

Function IsKeyPressed(VBKey As Long) As Boolean
    Dim KeyState As Integer
    KeyState = GetAsyncKeyState(VBKey)

    If KeyState And &H8000 = &H8000 Then
        IsKeyPressed = True     ' : Debug.Print VBKey
    Else
        IsKeyPressed = False
    End If
End Function

Function IsAsciiKeyPressed(Ascii As Integer) As Boolean
    '
End Function

Public Sub TypeAsciiKey(ByVal Char As Integer, AllStates As Integer)
    Dim c As String
    Dim vk%
    Dim Scan%
    Dim OemChar$
'    Dim dl&
    Dim ShiftState As Integer
'    Dim ss As Long
    Dim CtrlState As Integer
    Dim AltState As Integer
    Dim ShiftScanCode As Integer
    Dim CtrlScanCode As Integer
    Dim AltScanCode As Integer
    ShiftScanCode = MapVirtualKey(vbKeyShift, 0)
    CtrlScanCode = MapVirtualKey(vbKeyControl, 0)
    AltScanCode = MapVirtualKey(vbKeyMenu, 0)
    c = ChrW$(Char)
    'ss = (VkKeyScan(AscW(c$)) And &H100) / &HFF
    ' MsgBox ss
    ' Get the virtual key code for this character
    vk% = VkKeyScan(Char) And &HFF
    If AllStates = 0 Then AllStates = (VkKeyScan(AscW(c$)) And &H100) / &HFF
    ShiftState = AllStates And 1
    CtrlState = (AllStates And 2) / 2
```

```
    AltState = (AllStates And 4) / 4
    OemChar$ = "  "                                          ' 2 character buffer
    ' Get the OEM character - preinitialize the buffer
    CharToOem left$(c$, 1), OemChar$
    ' Get the scan code for this key
    Scan% = OemKeyScan(AscW(OemChar$)) And &HFF
    If ShiftState = 1 Then
        keybd_event vbKeyShift, ShiftScanCode, 0, 0
        DoEvents
    End If
    If CtrlState = 1 Then
        keybd_event vbKeyControl, CtrlScanCode, 0, 0
        DoEvents
    End If
    If AltState = 1 Then
        keybd_event vbKeyMenu, AltScanCode, 0, 0
        DoEvents
    End If
    ' Send the key down
    keybd_event vk%, Scan%, 0, 0
    DoEvents
    ' Send the key up
    keybd_event vk%, Scan%, KEYEVENTF_KEYUP, 0
    DoEvents
    If ShiftState = 1 Then
        keybd_event vbKeyShift, ShiftScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
    If CtrlState = 1 Then
        keybd_event vbKeyControl, CtrlScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
    If AltState = 1 Then
        keybd_event vbKeyMenu, AltScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
End Sub

Sub TypeKey(KeyToType As String, ToggleKeys As Integer)
    'togglekeys: bit1 = shift, 2 = ctrl, 3 = alt.
    Call TypeAsciiKey(AscW(KeyToType), ToggleKeys)
End Sub
```

```
Sub MySendKeys(StringToType As String)
'    MsgBox StringToType
    'doesn't work exactly like the vb version.
    'to send a shifted character, simply type it.
    'it will be converted automatically.  There are
    'three exceptions: ~^%{.  Each of these keys has to
    'be typed 2 times in a row.  They are used to toggle
    'on shift, ctrl, and/or alt states of the keyboard
    'for the next character typed.
    'special codes: + = shift, ^ = ctrl, % = alt
    'a * after the ' means it is implemented, otherwise
    ' it is slated for future implementation
    '*BACKSPACE   {BACKSPACE}, {BS}, or {BKSP}
    '*BREAK   {BREAK}
    '*CAPS LOCK   {CAPSLOCK}
    '*DEL or DELETE   {DELETE} or {DEL}
    '*DOWN ARROW  {DOWN}
    '*END {END}
    '*ENTER   {ENTER}or ~
    '*ESC {ESC}
    ' HELP    {HELP}
    ' HOME    {HOME}
    '*INS or INSERT   {INSERT} or {INS}
    '*LEFT ARROW  {LEFT}
    ' NUM LOCK    {NUMLOCK}
    ' PAGE DOWN   {PGDN}
    ' PAGE UP {PGUP}
    '*PRINT SCREEN    {PRTSC}
    '*RIGHT ARROW {RIGHT}
    ' SCROLL LOCK {SCROLLLOCK}
    '*TAB {TAB}
    '*UP ARROW    {UP}
    ' F1  {F1}
    ' F2  {F2}
    ' F3  {F3}
    ' F4  {F4}
    ' F5  {F5}
    ' F6  {F6}
    ' F7  {F7}
    ' F8  {F8}
    ' F9  {F9}
```

```
    ' F10 {F10}
    ' F11 {F11}
    ' F12 {F12}
    ' F13 {F13}
    ' F14 {F14}
    ' F15 {F15}
    ' F16 {F16}
'    Dim LcaseStringToType As String
    Dim ToggleKeys As Integer
    Dim Char As String
    ToggleKeys = 0
    Dim cl As Long
    cl = 1
    Do While cl <= Len(StringToType)
        ToggleKeys = 0
'        LcaseStringToType = LCase$(StringToType)
        Do While Char = "%" Or Char = "^" Or Char = "+" Or Char = "{" Or Char = "~"
            Char = Mid$(StringToType, cl, 1)
            If Char = "{" Then
                cl = cl + 1
                If Mid$(StringToType, cl - 1, 2) = "{{" Then
                    cl = cl + 1
                    Call TypeKey(Char, 0)
                Else
                    If StrComp(Mid$(StringToType, cl, 10), "backspace}", vbTextCompare) = 0 Then
                        cl = cl + 10
                        VirtualKeyPress vbKeyBack
                    ElseIf StrComp(Mid$(StringToType, cl, 3), "bs}", vbTextCompare) = 0 Then
                        cl = cl + 3
                        VirtualKeyPress vbKeyBack
                    ElseIf StrComp(Mid$(StringToType, cl, 5), "bksp}", vbTextCompare) = 0 Then
                        cl = cl + 5
                        VirtualKeyPress vbKeyBack
                    ElseIf StrComp(Mid$(StringToType, cl, 7), "delete}", vbTextCompare) = 0 Then
                        cl = cl + 7
                        VirtualKeyPress vbKeyDelete
                    ElseIf StrComp(Mid$(StringToType, cl, 4), "del}", vbTextCompare) = 0 Then
                        cl = cl + 4
                        VirtualKeyPress vbKeyDelete
                    ElseIf StrComp(Mid$(StringToType, cl, 5), "home}", vbTextCompare) = 0 Then
                        cl = cl + 5
                        VirtualKeyPress vbKeyHome
```

```
        ElseIf StrComp(Mid$(StringToType, cl, 6), "enter}", vbTextCompare) = 0 Then
            cl = cl + 6
            VirtualKeyPress vbKeyReturn
        ElseIf StrComp(Mid$(StringToType, cl, 4), "tab}", vbTextCompare) = 0 Then
            cl = cl + 4
            VirtualKeyPress vbKeyTab
        ElseIf StrComp(Mid$(StringToType, cl, 6), "prtsc}", vbTextCompare) = 0 Then
            cl = cl + 6
            VirtualKeyPress vbKeySnapshot
        ElseIf StrComp(Mid$(StringToType, cl, 4), "end}", vbTextCompare) = 0 Then
            cl = cl + 4
            VirtualKeyPress vbKeyEnd
        ElseIf StrComp(Mid$(StringToType, cl, 4), "ins}", vbTextCompare) = 0 Then
            cl = cl + 4
            VirtualKeyPress vbKeyInsert
        ElseIf StrComp(Mid$(StringToType, cl, 7), "insert}", vbTextCompare) = 0 Then
            cl = cl + 7
            VirtualKeyPress vbKeyInsert
        ElseIf StrComp(Mid$(StringToType, cl, 5), "left}", vbTextCompare) = 0 Then
            cl = cl + 5
            VirtualKeyPress vbKeyLeft
        ElseIf StrComp(Mid$(StringToType, cl, 6), "right}", vbTextCompare) = 0 Then
            cl = cl + 6
            VirtualKeyPress vbKeyRight
        ElseIf StrComp(Mid$(StringToType, cl, 6), "break}", vbTextCompare) = 0 Then
            cl = cl + 6
            VirtualKeyPress vbKeyPause
        ElseIf StrComp(Mid$(StringToType, cl, 9), "capslock}", vbTextCompare) = 0 Then
            cl = cl + 9
            VirtualKeyPress vbKeyCapital
        ElseIf StrComp(Mid$(StringToType, cl, 10), "downarrow}", vbTextCompare) = 0 Then
            cl = cl + 10
            VirtualKeyPress vbKeyDown
        ElseIf StrComp(Mid$(StringToType, cl, 8), "uparrow}", vbTextCompare) = 0 Then
            cl = cl + 8
            VirtualKeyPress vbKeyUp
        ElseIf StrComp(Mid$(StringToType, cl, 4), "esc}", vbTextCompare) = 0 Then
            cl = cl + 4
            VirtualKeyPress vbKeyEscape
        End If
    End If
ElseIf Char = "~" Then
```

```
                cl = cl + 1
                If Mid$(StringToType, cl, 1) = "~" Then
                    cl = cl + 1
                    Call TypeKey(Char, 0)
                Else
                    VirtualKeyPress vbKeyReturn
                End If
            ElseIf Char = "+" Then
                cl = cl + 1
                If Mid$(StringToType, cl, 1) = "+" Then
                    cl = cl + 1
                    Call TypeKey(Char, 0)
                Else
                    ToggleKeys = ToggleKeys + 1
                End If
            ElseIf Char = "^" Then
                cl = cl + 1
                If Mid$(StringToType, cl - 1, 2) = "^^" Then
                    cl = cl + 1
                    Call TypeKey(Char, 0)
                Else
                    ToggleKeys = ToggleKeys + 2
                End If
            ElseIf Char = "%" Then
                cl = cl + 1
                If Mid$(StringToType, cl - 1, 2) = "%%" Then
                    cl = cl + 1
                    Call TypeKey(Char, 0)
                Else
                    ToggleKeys = ToggleKeys + 4
                End If
            End If
        Loop
'        Debug.Print Char, ToggleKeys
        If Len(Char) <> 0 Then Call TypeKey(Char, ToggleKeys)
    Loop
End Sub

Sub VirtualKeyPress(VirtualKeyCode As Integer)
    Dim ScanCode As Integer
    ScanCode = MapVirtualKey(VirtualKeyCode, 0)
    keybd_event VirtualKeyCode%, ScanCode, 0, 0
```

```
    DoEvents
    keybd_event VirtualKeyCode, ScanCode, KEYEVENTF_KEYUP, 0
    DoEvents
End Sub

Public Sub VirtualKeyPressEx(VirtualKeyCode As Integer, Shift As Integer, Ctrl As Integer, Alt As Integer)
    Dim ShiftScanCode As Integer
    Dim CtrlScanCode As Integer
    Dim AltScanCode As Integer

    ShiftScanCode = MapVirtualKey(vbKeyShift, 0)
    CtrlScanCode = MapVirtualKey(vbKeyControl, 0)
    AltScanCode = MapVirtualKey(vbKeyMenu, 0)

    ' Get the virtual key code for this character
    Dim ScanCode As Integer
    ScanCode = MapVirtualKey(VirtualKeyCode, 0)

'    Shift = AllStates And 1
'    Ctrl = (AllStates And 2) / 2
'    Alt = (AllStates And 4) / 4

    If Ctrl = 1 Then
        keybd_event vbKeyControl, CtrlScanCode, 0, 0
        DoEvents
    End If
    If Shift = 1 Then
        keybd_event vbKeyShift, ShiftScanCode, 0, 0
        DoEvents
    End If
    If Alt = 1 Then
        keybd_event vbKeyMenu, AltScanCode, 0, 0
        DoEvents
    End If

    ' Send the key down
    keybd_event VirtualKeyCode%, ScanCode, 0, 0
    DoEvents
    ' Send the key up
    keybd_event VirtualKeyCode, ScanCode, KEYEVENTF_KEYUP, 0
    DoEvents
```

```
    If Shift = 1 Then
        keybd_event vbKeyShift, ShiftScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
    If Ctrl = 1 Then
        keybd_event vbKeyControl, CtrlScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
    If Alt = 1 Then
        keybd_event vbKeyMenu, AltScanCode, KEYEVENTF_KEYUP, 0
        DoEvents
    End If
End Sub
```

F9-Taste Daten zwischen Mappen austauschen

Siehe ARBEITSMAPPEN ganz oben unter

! F9-Taste Daten zwischen Mappen austauschen

F9-Taste Werte aktualisieren

Application.Calculate

Maustasten abfragen

BSP 0

In Seibersdorf wollten wir beim Selectieren eines Diagramm-Datenpunktes abfragen, ob man diesen mit der linken oder mit der rechten Maustaste angewählt hat. Wir brauchen eine Function und zwei Konstanten und zuletzt die Funktion (alles direkt im Chart-Code)

Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

Const VK_LBUTTON = &H1 ' Linker Mausbutton
Const VK_RBUTTON = &H2 ' Rechter Mausbutton

Private Sub Chart_Select(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long)

```
    MsgBox GetAsyncKeyState(VK_LBUTTON) & " und " & GetAsyncKeyState(VK_RBUTTON)

End Sub
```

Angezeigtes Ergebnis
bei linkem Mausklick "-32767 und 0" oder "1 und 0"
bei rechtem Mausclick „0 und – 32767“ oder "1 und 1"
(warum es bei linkem Mausklick zwei verschiedene Anzeigen gab, weiß ich nicht – Eigenheit des Getasynckeystate-Befehls ist ja, dass er immer anzeigt, was seit dem letzten Abrufen geschehen ist.

Als ich die Prozedur umänderte auf

```
Private Sub Chart_Select(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long)
    Dim IDNum As Long
    Dim a As Long
    Dim b As Long
    Dim ANSWER

    ANSWER = GetAsyncKeyState(VK_LBUTTON)

    MsgBox GetAsyncKeyState(VK_LBUTTON) & " und " & GetAsyncKeyState(VK_RBUTTON)

End Sub
```

gab es
bei linkem Mausklick: „-32767 und 0“ oder „0 und 0“
vei rechtem Mauscklick: „0 und -32767“ oder „0 und 1“

```
Public Sub test()
    If GetAsyncKeyState(&H10) Then MsgBox "Shift key is pressed"
    ' &H11 = Ctrl
    ' &H12)= Alt
End Sub
```

BSP 1

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y _
  As Single)

  Select Case Button
```

```
    Case vbLeftButton
      ' Linke Maustaste
      ' ...

    Case vbRightButton
      ' Rechte Maustaste
      ' ...

    Case vbMiddleButton
      ' Mittlere Maustaste (wenn vorhanden)
      ' ...

  End Select

End Sub
```

BSP 2

Geht ganz einfach. An einem Beispiel "Rechtsklick auf eine Form":

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

    If Button = vbRightButton Then
      'Deine Befehle hier...
    End If

End Sub
```

BSP 3

Hallo Experten!

Ich hab mal wieder ne Frage:

Wie kann ich abfragen, ob die linke Maustaste gedrückt wurde?

Beispielcode, wenn Du das mit der API probieren möchtest.
Beginne ein neues Projekt und lege nur einen Timer auf die Form. Dann füge den Code ein, starten ...

```
Option Explicit
```

```
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer
Private Sub Form Load()
Timer1.Interval = 100
Timer1.Enabled = True
End Sub
Private Sub Timer1_Timer()
If GetAsyncKeyState(1) <> 0 Then
Me.Caption = "down"
Else
Me.Caption = "up"
End If
End Sub
```

BSP 4

GetAsyncKeyState-Funktion

Diese Funktion ermittelt, ob seit ihrem letzten Aufruf eine bestimmte Tasteneingabe erfolgt ist.

Betriebssystem: Win95, Win98, WinNT 3.1, Win2000, WinME

Deklaration:

```
Declare Function GetAsyncKeyState Lib "user32.dll" ( ByVal vKey As Long ) As Long
```

Beschreibung:
Diese Funktion ermittelt, ob seit ihrem letzten Aufruf eine bestimmte Tasteneingabe erfolgt ist.

Parameter:

vKey Erwartet eine der Tastaturkonstanten die auf einen schon betätigten Tastendruck hin überprüft werden soll.

vKey Konstanten:

```
Const VK_LBUTTON = &H1
' Linker Mausbutton

Const VK_RBUTTON = &H2
' Rechter Mausbutton

Const VK_MBUTTON = &H4
' Mittlerer Mausbutton

Const VK_BACK = &H8
' Backspace Taste

Const VK_TAB = &H9
```

```
' Tab Taste

Const VK_CLEAR = &HC
' Numpad 5 ohne Numlock

Const VK_RETURN = &HD
' Enter Taste

Const VK_SHIFT = &H10
' Shift Taste

Const VK_CONTROL = &H11
' STRG Taste

Const VK_MENU = &H12
' Alt Taste

Const VK_PAUSE = &H13
' Pause/Untbr

Const VK_CAPITAL = &H14
' Caps Lock/Feststelltaste

Const VK_ESCAPE = &H1B
' Escape

Const VK_SPACE = &H20
' Space/Leertaste

Const VK_PRIOR = &H21
' PageUp/Bild hoch

Const VK_NEXT = &H22
' PageDown/Bild runter

Const VK_END = &H23
' Ende

Const VK_HOME = &H24
' Home/Pos1

Const VK_LEFT = &H25
' Linke Pfeiltaste

Const VK_UP = &H26
' Obere Pfeiltaste

Const VK_RIGHT = &H27
' Rechte Pfeiltaste

Const VK_DOWN = &H28
' Untere Pfeiltaste
```

```
Const VK_PRINT = &H2A
' Drucken (Nokia Tastaturen)

Const VK_SNAPSHOT = &H2C
' Drucken/S-Abf

Const VK_INSERT = &H2D
' Einfügen

Const VK_DELETE = &H2E
' Entfernen

Const VK_HELP = &H2F
' Hilfe

Const VK_0 = &H30
' Taste 0

Const VK_1 = &H31
' Taste 1

Const VK_2 = &H32
' Taste 2

Const VK_3 = &H33
' Taste 3

Const VK_4 = &H34
' Taste 4

Const VK_5 = &H35
' Taste 5

Const VK_6 = &H36
' Taste 6

Const VK_7 = &H37
' Taste 7

Const VK_8 = &H38
' Taste 8

Const VK_9 = &H39
' Taste 9

Const VK_A = &H41
' Taste A

Const VK_B = &H42
' Taste B

Const VK_C = &H43
' Taste C
```

```
Const VK_D = &H44
' Taste D

Const VK_E = &H45
' Taste E

Const VK_F = &H46
' Taste F

Const VK_G = &H47
' Taste G

Const VK_H = &H48
' Taste H

Const VK_I = &H49
' Taste I

Const VK_J = &H4A
' Taste J

Const VK_K = &H4B
' Taste K

Const VK_L = &H4C
' Taste L

Const VK_M = &H4D
' Taste M

Const VK_N = &H4E
' Taste N

Const VK_O = &H4F
' Taste O

Const VK_P = &H50
' Taste P

Const VK_Q = &H51
' Taste Q

Const VK_R = &H52
' Taste R

Const VK_S = &H53
' Taste S

Const VK_T = &H54
' Taste T

Const VK_U = &H55
```

```
' Taste U

Const VK_V = &H56
' Taste V

Const VK_W = &H57
' Taste W

Const VK_X = &H58
' Taste X

Const VK_Y = &H59
' Taste Y

Const VK_Z = &H5A
' Taste Z

Const VK_STARTKEY = &H5B
' Startmenütaste

Const VK_CONTEXTKEY = &H5D
' Kontextmenü

Const VK_NUMPAD0 = &H60
' Numpad Taste 0

Const VK_NUMPAD1 = &H61
' Numpad Taste 1

Const VK_NUMPAD2 = &H62
' Numpad Taste 2

Const VK_NUMPAD3 = &H63
' Numpad Taste 3

Const VK_NUMPAD4 = &H64
' Numpad Taste 4

Const VK_NUMPAD5 = &H65
' Numpad Taste 5

Const VK_NUMPAD6 = &H66
' Numpad Taste 6

Const VK_NUMPAD7 = &H67
' Numpad Taste 7

Const VK_NUMPAD8 = &H68
' Numpad Taste 8

Const VK_NUMPAD9 = &H69
' Numpad Taste 9
```

```
Const VK_MULTIPLY = &H6A
' Numpad Multiplikations Taste (*)

Const VK_ADD = &H6B
' Numpad Additions Taste (+)

Const VK_SUBTRACT = &H6D
' Numpad Subtrations Taste (-)

Const VK_DECIMAL = &H6E
' Numpad Komma Taste (,)

Const VK_DIVIDE = &H6F
' Numpad Dividierungs Taste (/)

Const VK_F1 = &H70
' F1 Taste

Const VK_F2 = &H71
' F2 Taste

Const VK_F3 = &H72
' F3 Taste

Const VK_F4 = &H73
' F4 Taste

Const VK_F5 = &H74
' F5 Taste

Const VK_F6 = &H75
' F6 Taste

Const VK_F7 = &H76
' F7 Taste

Const VK_F8 = &H77
' F8 Taste

Const VK_F9 = &H78
' F9 Taste

Const VK_F10 = &H79
' F10 Taste

Const VK_F11 = &H7A
' F11 Taste

Const VK_F12 = &H7B
' F12 Taste

Const VK_F13 = &H7C
' F13 Taste
```

```
Const VK_F14 = &H7D
' F14 Taste

Const VK_F15 = &H7E
' F15 Taste

Const VK_F16 = &H7F
' F16 Taste

Const VK_F17 = &H80
' F17 Taste

Const VK_F18 = &H81
' F18 Taste

Const VK_F19 = &H82
' F19 Taste

Const VK_F20 = &H83
' F20 Taste

Const VK_F21 = &H84
' F21 Taste

Const VK_F22 = &H85
' F22 Taste

Const VK_F23 = &H86
' F23 Taste

Const VK_F24 = &H87
' F24 Taste

Const VK_NUMLOCK = &H90
' Numlock Taste

Const VK_OEM_SCROLL = &H91
' Scroll Lock

Const VK_LSHIFT = &HA0
' Linke Shift-Taste

Const VK_RSHIFT = &HA1
' Rechte Shift-Taste

Const VK_LCONTROL = &HA2
' Linke STRG-Taste

Const VK_RCONTROL = &HA3
' Rechte STRG-Taste

Const VK_LMENU = &HA4
```

```
' Linke ALT-Taste

Const VK_RMENU = &HA5
' Rechte ALT-Taste

Const VK_OEM_1 = &HBA
' ";"-Taste

Const VK_OEM_PLUS = &HBB
' "

Const VK_OEM_COMMA = &HBC
' ","-Taste

Const VK_OEM_MINUS = &HBD
' "-"-Taste

Const VK_OEM_PERIOD = &HBE
' "."-taste

Const VK_OEM_2 = &HBF
' "/"-Taste

Const VK_OEM_3 = &HC0
' "`"-Taste

Const VK_OEM_4 = &HDB
' "["-Taste

Const VK_OEM_5 = &HDC
' "\"-Taste

Const VK_OEM_6 = &HDD
' "]"-Taste

Const VK_OEM_7 = &HDE
' "

Const VK_ICO_F17 = &HE0
' F17 einer Olivetti Tastatur (Intern)

Const VK_ICO_F18 = &HE1
' F18 einer Olivetti Tastatur (Intern)

Const VK_OEM102 = &HE2
' "<"-Taste oder "|"-Taste einer IBM-Kompatiblen 102 Tastatur (Nicht US)

Const VK_ICO_HELP = &HE3
' Hilfetaste einer Olivetti Tastatur (Intern)

Const VK_ICO_00 = &HE4
' 00-Taste einer Olivetti Tastatur (Intern)
```

```
Const VK_ICO_CLEAR = &HE6
' Löschen Taste einer Olivetti Tastatur (Intern)

Const VK_OEM_RESET = &HE9
' Reset Taste (Nokia)

Const VK_OEM_JUMP = &HEA
' Springen Taste (Nokia)

Const VK_OEM_PA1 = &HEB
' PA1 Taste (Nokia)

Const VK_OEM_PA2 = &HEC
' PA2 Taste (Nokia)

Const VK_OEM_PA3 = &HED
' PA3 Taste (Nokia)

Const VK_OEM_WSCTRL = &HEE
' WSCTRL Taste (Nokia)

Const VK_OEM_CUSEL = &HEF
' WSCTRL Taste (Nokia)

Const VK_OEM_ATTN = &HF0
' ATTN Taste (Nokia)

Const VK_OEM_FINNISH = &HF1
' Fertig Taste (Nokia)

Const VK_OEM_COPY = &HF2
' Kopieren Taste (Nokia)

Const VK_OEM_AUTO = &HF3
' Auto Taste (Nokia)

Const VK_OEM_ENLW = &HF4
' ENLW Taste (Nokia)

Const VK_OEM_BACKTAB = &HF5
' BackTab Taste (Nokia)

Const VK_ATTN = &HF6
' ATTN-Taste

Const VK_CRSEL = &HF7
' CRSEL-Taste

Const VK_EXSEL = &HF8
' EXSEL-Taste

Const VK_EREOF = &HF9
' EREOF-Taste
```

```
Const VK_PLAY = &HFA
' PLAY-Taste

Const VK_ZOOM = &HFB
' ZOOM-Taste

Const VK_NONAME = &HFC
' NONAME-Taste

Const VK_PA1 = &HFD
' PA1-Taste

Const VK_OEM_CLEAR = &HFE
' OEM_CLEAR-Taste
```

Rückgabewert:
Ist die Funktion erfolgreich und die angegeben Taste gedrückt, so wird das Bit **"&H8000"** in der Rückgabe gesetzt. Ist die Taste momentan nicht gedrückt, wurde aber seit dem letzten Funktionsaufruf betätigt, so ist in der Rückgabe das **"&H1"** Bit gesetzt. Wurde und ist die Taste nicht gedrückt worden oder trat ein Fehler bei dem Funktionsaufruf auf so ist derWert **"0"** die Rückgabe.

Beispiel:

```
Private Declare Function GetAsyncKeyState Lib "user32.dll" ( _
  ByVal vKey As Long) As Long

' GetAsyncKeyState vKey-Konstanten
Private Const VK_LBUTTON = &H1  ' Linker Mausbutton
Private Const VK_RBUTTON = &H2  ' Rechter Mausbutton
Private Const VK_MBUTTON = &H4  ' Mittlerer Mausbutton
Private Const VK_BACK = &H8  ' Backspace Taste
Private Const VK_TAB = &H9  ' Tab Taste
Private Const VK_CLEAR = &HC  ' Numpad 5 ohne Numlock
Private Const VK_RETURN = &HD  ' Enter Taste
Private Const VK_SHIFT = &H10  ' Shift Taste
Private Const VK_CONTROL = &H11  ' STRG Taste
Private Const VK_MENU = &H12  ' Alt Taste
Private Const VK_PAUSE = &H13  ' Pause/Untbr
Private Const VK_CAPITAL = &H14  ' Caps Lock/Feststelltaste
Private Const VK_ESCAPE = &H1B  ' Escape
Private Const VK_SPACE = &H20  ' Space/Leertaste
Private Const VK_PRIOR = &H21  ' PageUp/Bild hoch
Private Const VK_NEXT = &H22  ' PageDown/Bild runter
Private Const VK_END = &H23  ' Ende
Private Const VK_HOME = &H24  ' Home/Pos1
Private Const VK_LEFT = &H25  ' Linke Pfeiltaste
Private Const VK_UP = &H26  ' Obere Pfeiltaste
Private Const VK_RIGHT = &H27  ' Rechte Pfeiltaste
Private Const VK_DOWN = &H28 ' Untere Pfeiltaste
Private Const VK_PRINT = &H2A  ' Drucken (Nokia Tastaturen)
Private Const VK_SNAPSHOT = &H2C  ' Drucken/S-Abf
```

```
Private Const VK_INSERT = &H2D  ' Einfügen
Private Const VK_DELETE = &H2E  ' Entfernen
Private Const VK_HELP = &H2F  ' Hilfe
Private Const VK_0 = &H30  ' Taste 0
Private Const VK_1 = &H31  ' Taste 1
Private Const VK_2 = &H32  ' Taste 2
Private Const VK_3 = &H33   ' Taste 3
Private Const VK_4 = &H34  ' Taste 4
Private Const VK_5 = &H35  ' Taste 5
Private Const VK_6 = &H36  ' Taste 6
Private Const VK_7 = &H37  ' Taste 7
Private Const VK_8 = &H38   ' Taste 8
Private Const VK_9 = &H39   ' Taste 9
Private Const VK_A = &H41   ' Taste A
Private Const VK_B = &H42   ' Taste B
Private Const VK_C = &H43   ' Taste C
Private Const VK_D = &H44   ' Taste D
Private Const VK_E = &H45   ' Taste E
Private Const VK_F = &H46   ' Taste F
Private Const VK_G = &H47   ' Taste G
Private Const VK_H = &H48  ' Taste H
Private Const VK_I = &H49  ' Taste I
Private Const VK_J = &H4A  ' Taste J
Private Const VK_K = &H4B  ' Taste K
Private Const VK_L = &H4C   ' Taste L
Private Const VK_M = &H4D   ' Taste M
Private Const VK_N = &H4E  ' Taste N
Private Const VK_O = &H4F  ' Taste O
Private Const VK_P = &H50   ' Taste P
Private Const VK_Q = &H51   ' Taste Q
Private Const VK_R = &H52 ' Taste R
Private Const VK_S = &H53  ' Taste S
Private Const VK_T = &H54   ' Taste T
Private Const VK_U = &H55  ' Taste U
Private Const VK_V = &H56  ' Taste V
Private Const VK_W = &H57   ' Taste W
Private Const VK_X = &H58  ' Taste X
Private Const VK_Y = &H59  ' Taste Y
Private Const VK_Z = &H5A  ' Taste Z
Private Const VK_STARTKEY = &H5B   ' Startmenütaste
Private Const VK_CONTEXTKEY = &H5D  ' Kentextmenü
Private Const VK_NUMPAD0 = &H60  ' Numpad Taste 0
Private Const VK_NUMPAD1 = &H61  ' Numpad Taste 1
Private Const VK_NUMPAD2 = &H62  ' Numpad Taste 2
Private Const VK_NUMPAD3 = &H63  ' Numpad Taste 3
Private Const VK_NUMPAD4 = &H64  ' Numpad Taste 4
Private Const VK_NUMPAD5 = &H65 ' Numpad Taste 5
Private Const VK_NUMPAD6 = &H66  ' Numpad Taste 6
Private Const VK_NUMPAD7 = &H67  ' Numpad Taste 7
Private Const VK_NUMPAD8 = &H68   ' Numpad Taste 8
Private Const VK_NUMPAD9 = &H69   ' Numpad Taste 9
Private Const VK_MULTIPLY = &H6A ' Numpad Multiplikations Taste (*)
Private Const VK_ADD = &H6B  ' Numpad Additions Taste (+)
```

```
Private Const VK_SUBTRACT = &H6D   ' Numpad Subtrations Taste (-)
Private Const VK_DECIMAL = &H6E   ' Numpad Komma Taste (,)
Private Const VK_DIVIDE = &H6F ' Numpad Devidierungs Taste (/)
Private Const VK_F1 = &H70  ' F1 Taste
Private Const VK_F2 = &H71  ' F2 Taste
Private Const VK_F3 = &H72   ' F3 Taste
Private Const VK_F4 = &H73   ' F4 Taste
Private Const VK_F5 = &H74   ' F5 Taste
Private Const VK_F6 = &H75   ' F6 Taste
Private Const VK_F7 = &H76  ' F7 Taste
Private Const VK_F8 = &H77  ' F8 Taste
Private Const VK_F9 = &H78   ' F9 Taste
Private Const VK_F10 = &H79  ' F10 Taste
Private Const VK_F11 = &H7A   ' F11 Taste
Private Const VK_F12 = &H7B   ' F12 Taste
Private Const VK_F13 = &H7C  ' F13 Taste
Private Const VK_F14 = &H7D  ' F14 Taste
Private Const VK_F15 = &H7E   ' F15 Taste
Private Const VK_F16 = &H7F   ' F16 Taste
Private Const VK_F17 = &H80   ' F17 Taste
Private Const VK_F18 = &H81   ' F18 Taste
Private Const VK_F19 = &H82   ' F19 Taste
Private Const VK_F20 = &H83   ' F20 Taste
Private Const VK_F21 = &H84   ' F21 Taste
Private Const VK_F22 = &H85   ' F22 Taste
Private Const VK_F23 = &H86   ' F23 Taste
Private Const VK_F24 = &H87   ' F24 Taste
Private Const VK_NUMLOCK = &H90  ' Numlock Taste
Private Const VK_OEM_SCROLL = &H91  ' Scroll Lock
Private Const VK_LSHIFT = &HA0 ' Linke Shift-Taste
Private Const VK_RSHIFT = &HA1  ' Rechte Shift-Taste
Private Const VK_LCONTROL = &HA2  ' Linke STRG-Taste
Private Const VK_RCONTROL = &HA3 ' Rechte STRG-Taste
Private Const VK_LMENU = &HA4  ' Linke ALT-Taste
Private Const VK_RMENU = &HA5  ' Rechte ALT-Taste
Private Const VK_OEM_1 = &HBA  ' ";"-Taste
Private Const VK_OEM_PLUS = &HBB   ' "
Private Const VK_OEM_COMMA = &HBC   ' ","-Taste
Private Const VK_OEM_MINUS = &HBD   ' "-"-Taste
Private Const VK_OEM_PERIOD = &HBE   ' "."-taste
Private Const VK_OEM_2 = &HBF   ' "/"-Taste
Private Const VK_OEM_3 = &HC0   ' "`"-Taste
Private Const VK_OEM_4 = &HDB   ' "["-Taste
Private Const VK_OEM_5 = &HDC   ' "\"-Taste
Private Const VK_OEM_6 = &HDD   ' "]"-Taste
Private Const VK_OEM_7 = &HDE   ' "
Private Const VK_ICO_F17 = &HE0   ' F17 einer Olivette Tastatur (Intern)
Private Const VK_ICO_F18 = &HE1   ' F18 einer Olivette Tastatur (Intern)
Private Const VK_OEM102 = &HE2   ' "<"-Taste oder "|"-Taste einer
 ' IBM-Kompatiblen 102 Tastatur (Nicht US)
Private Const VK_ICO_HELP = &HE3   ' Hilfetaste einer Olivetti Tastatur (Intern)
Private Const VK_ICO_00 = &HE4  ' 00-Taste einer Olivetti Tastatur (Intern)
Private Const VK_ICO_CLEAR = &HE6   ' Löschen Taste einer Olivetti Tastatur (Intern)
```

```
Private Const VK_OEM_RESET = &HE9  ' Reset Taste (Nokia)
Private Const VK_OEM_JUMP = &HEA   ' Springen Taste (Nokia)
Private Const VK_OEM_PA1 = &HEB   ' PA1 Taste (Nokia)
Private Const VK_OEM_PA2 = &HEC  ' PA2 Taste (Nokia)
Private Const VK_OEM_PA3 = &HED  ' PA3 Taste (Nokia)
Private Const VK_OEM_WSCTRL = &HEE   ' WSCTRL Taste (Nokia)
Private Const VK_OEM_CUSEL = &HEF   ' WSCTRL Taste (Nokia)
Private Const VK_OEM_ATTN = &HF0   ' ATTN Taste (Nokia)
Private Const VK_OEM_FINNISH = &HF1  ' Fertig Taste (Nokia)
Private Const VK_OEM_COPY = &HF2   ' Kopieren Taste (Nokia)
Private Const VK_OEM_AUTO = &HF3   ' Auto Taste (Nokia)
Private Const VK_OEM_ENLW = &HF4   ' ENLW Taste (Nokia)
Private Const VK_OEM_BACKTAB = &HF5   ' BackTab Taste (Nokia)
Private Const VK_ATTN = &HF6   ' ATTN-Taste
Private Const VK_CRSEL = &HF7   ' CRSEL-Taste
Private Const VK_EXSEL = &HF8   ' EXSEL-Taste
Private Const VK_EREOF = &HF9   ' EREOF-Taste
Private Const VK_PLAY = &HFA  ' PLAY-Taste
Private Const VK_ZOOM = &HFB   ' ZOOM-Taste
Private Const VK_NONAME = &HFC   ' NONAME-Taste
Private Const VK_PA1 = &HFD   ' PA1-Taste
Private Const VK_OEM_CLEAR = &HFE  ' OEM_CLEAR-Taste
' Überprüfen ob eine Taste gedrückt wurde
Private Sub Command1_Click()
  Dim Retval As Long

  ' Überprüfen ob seit dem letzten Funktionsaufruf die Taste X gedrückt wurde
  Retval = GetAsyncKeyState(VK_X)

  ' Auswerten des Ergebnisses
  If CBool(Retval And &H8000) Then
    MsgBox "Die Taste ""X"" ist momentan gedrückt."

  ElseIf CBool(Retval And &H1) Then
    MsgBox "Die Taste ""X"" wurde seit dem letztem Check gedrückt."

  Else
    MsgBox "Die Taste ""X"" wurde seit der letzten Überprüfung nicht gedrückt."
  End If
End Sub
```

BSP 5

Mausklick systemweit abfragen

Wie man systemweit erkennen kann, wann und welche Maustaste gedrückt wurde, erfahren Sie in diesem Tipp.

Wie man einen Mausklick systemweit abfragen kann, d.h. wann welche Maustaste gedrückt wurde, das erfahren Sie heute. Erstellen Sie hierzu ein neues Projekt, platzieren auf die Form ein Timer-Control, sowie ein Label.

Fügen Sie nachfolgenden Code in das Codefenster der Form ein:

```
Option Explicit

' zunächst die benötigten API-Deklarationen
Private Declare Function GetAsyncKeyState Lib "user32" ( _
  ByVal vKey As Long) As Integer

Private Const VK_LBUTTON = &H1
Private Const VK_RBUTTON = &H2
Private Const VK_MBUTTON = &H4
Private Sub Form_Load()
  ' Timer initialisieren
  Timer1.Interval = 150
  Timer1.Enabled = True

  ' Wichtig! Zunächst zurücksetzen
  Call GetAsyncKeyState(VK_LBUTTON)
  Label1.Caption = ""
End Sub
Private Sub Timer1_Timer()
  ' linke Maustaste gedrückt?
  If GetAsyncKeyState(VK_LBUTTON) Then
    Label1.Caption = "linke Maustaste"
  ElseIf GetAsyncKeyState(VK_RBUTTON) Then
    Label1.Caption = "rechte Maustaste"
  ElseIf GetAsyncKeyState(VK_MBUTTON) Then
    Label1.Caption = "mittlere Maustaste"
  Else
    Label1.Caption = ""
  End If
End Sub
```

Da die API-Funktion **GetAsyncKeyState** den Status der (Maus-)Taste seit der letzten Abfrage zurückgibt, ist es wichtig, die Funktion einmalig vor der dauerhaften Abfrage aufzurufen.

MAUS - X+Y-Koordinaten

Die Mauskoordinaten (Bildschirmkoordinaten) weichen ab von den Koordinaten der eigentlichen Excelarbeitsmappe - nicht nur durch den Rand des Excelapplicationfensters, sondern auch durch Zoomstufe .... - hier in diesem Abschnitt werden die absoluten Bildschirmkoordinaten des Mauszeigers angezeigt

Möchte man die exakten Koordinaten auf dem Tabellenblatt anzeigen, um z.B. Autoformen oder Textfelder ganz genau setzen zu können, muss man die Koordinationen ohne der Maus anzeigen lassen. Es gibt von mir dazu eigenen Code im Abschnitt "T-ELEMENTE / Koordinaten auf Tabellenblatt anzeigen"

VERSION 0

Zuerst die F12-Taste mit dem Makro für die Anzeige festlegen

```
Private Sub Workbook_Open()
   Application.OnKey "{F12}", "MAUS_KOORDINATEN"
End Sub
```

Und hier der Code für die Anzeige

```
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Type POINTAPI
   x As Long
   y As Long
End Type

Sub MAUS_KOORDINATEN()
   Dim pTargetPoint As POINTAPI
   Dim lRetVal As Long

   lRetVal = GetCursorPos(pTargetPoint)
   Application.StatusBar = "Meine Position: X " & pTargetPoint.x & ", Y " & pTargetPoint.y
End Sub
```

VERSION 1

```
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Type POINTAPI
    x As Long
    y As Long
End Type

Sub WhereAmI()
    Dim pTargetPoint As POINTAPI
    Dim lRetVal As Long
    lRetVal = GetCursorPos(pTargetPoint)
    MsgBox "Meine Position:" & vbLf & _
      pTargetPoint.x & "," & pTargetPoint.y
End Sub
```

```
VERSION 2


Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Type POINTAPI
    x As Long
    y As Long
End Type

Sub CurosrXY_Pixels()

    Dim lngStatus As Long
    Dim typWhere As POINTAPI

    lngStatus = GetCursorPos(typWhere)
    MsgBox "x: " & typWhere.x & Chr(13) & "y: " & typWhere.y, vbInformation, "Pixels"

End Sub
```

MAUSCLICKS emulieren

so macht die Maus einen Linksklick:

Code:

```
Private Declare Sub mouse event Lib "user32.dll" (
    ByVal dwFlags As Long,
    ByVal dx As Long,
    ByVal dy As Long,
    ByVal dwdata As Long,
    ByVal dwExtraInfo As Long)

Private Const MOUSEEVENT LEFTDOWN = &H2
Private Const MOUSEEVENT LEFTUP = &H4

Public Sub test()
    mouse event MOUSEEVENT LEFTDOWN, 0&, 0&, 0&, 0&
    mouse event MOUSEEVENT LEFTUP, 0&, 0&, 0&, 0&
End Sub
```

MAUS soll Pixel in Userform zeichnen

Hi!
Ich lese mit folgendem Quellcode die Mauskoordinaten aus und zeichne Pixel auf die UserForm. Trotzdem werden die Punkte versetzt gezeichnet, wo liegt mein Fehler?

Vielen Dank für Eure Hilfe! MfG, Andi

```
Private Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Private Declare Function SetPixel Lib "gdi32" (ByVal hDC As Long, ByVal X As Long, ByVal Y As Long, ByVal crColor As Long) As Long
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Type POINTAPI
X As Long
Y As Long
End Type
Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
Dim hWnd As Long
Dim hDC As Long
Dim PT As POINTAPI
hWnd = FindWindow(vbNullString, Me.Caption)
hDC = GetDC(hWnd)
GetCursorPos PT
SetPixel hDC, PT.X + KorrekturX, PT.Y + KorrekturY, vbBlack
End Sub
```

ANTWORT

Hi Andi,

die API gibt Dir die Koordinaten der Maus auf dem Bildschirm zurück. Wenn Du einen Punkt setzt, gibst Du aber die Koordinaten auf dem Formular an. Dann fehlt Dir natürlich der Abstand des Formulars von Rand, der Pixel wird um diesen Betrag versetzt gesetzt.

Dann hast Du auf der Form noch das Image. Wenn Du zum Zeichnen Koordinaten im Image angibst, mußt Du auch noch die Lage des Images auf der form berücksichtigen.

Dazu kommt noch, daß die Koordinaten tatsächlich 'verschoben' sind, eine Konstante kommt tatsächlich noch dazu. Ein Beispiel in VB:

```
Option Explicit
Private Declare Function GetCursorPos Lib "user32" (lpPoint As _
POINTAPI) As Long
Private Type POINTAPI
x As Long
y As Long
End Type
Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
Dim Result&, P As POINTAPI
Dim TF As Integer
TF = 15     'Twip-Faktor :-) Pixel in Twips wandeln
Result = GetCursorPos(P)
If Button = 1 Then
Picture1.PSet ((((P.x - 132) * TF) - Me.Left + Picture1.Left), ((P.y - 87) * TF) - Me.Top + Picture1.Top)
End If
End Sub
```

Mit F2-Enter-Tastensimulation die Zellen neu übernehmen (wichtig nach Import von Daten in Excel)

```
Sub ZellenAufbereiten()

Dim Cell As Range
Application.ScreenUpdating = False
For Each Cell In Selection
Cell.Select
Application.SendKeys "{F2}+{ENTER}", True
Next
Application.ScreenUpdating = True
End Sub
```

Variante 2 für Current Region

```
Sub DatenUmwandeln()
Dim MyRange As Range
Dim Cell As Range
Application.ScreenUpdating = False
Set MyRange = ActiveCell.CurrentRegion.Columns(7)
For Each Cell In MyRange
Cell.Select
```

```
Application.SendKeys "{F2}+{ENTER}", True
Next Cell
End Sub
```

Rechte Maustaste deaktivieren

GANZ WICHTIG - dieses Abdrehen greift leider auch nach einem Excel Neustart - daher unbedingt auch den Code für das WIEDER-AKTIVIEREN BEIM SCHLIESSEN DER MAPPE EINBAUEN (UNTEN ROT)

Für ganze Arbeitsmappe

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
  If ADMIN=1 Then Exit Sub ' wenn wir im Adminmodus sind, soll Mausklick gehen
  Cancel = True ' Man kann nirgendwo einen rechten Mausklick machen
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.CommandBars("Cell").Reset ' Rechten Mausklick wieder aktivieren
End Sub
```

Für einzelnes Tabellenblatt

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
  If ADMIN=1 Then Exit Sub ' wenn wir im Adminmodus sind, soll Mausklick gehen
  Cancel = True
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.CommandBars("Cell").Reset ' Rechten Mausklick wieder aktivieren
End Sub
```

VERSION 1

wie kann man die rechte maustaste in einer Arbeitsmappe komplett deaktivieren?

Ich habe was gefunden, was zwar in der Mappe selbst, aber nicht für meine selbsterstellte Menüleiste funktioniert:

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Excel.Range, Cancel As Boolean)
Cancel = True
End Sub
```

Ich möchte aber, daß die rechte Maustaste komplett "lahm" gelegt wird...

Wer hat ne Lösung?

Hallo TF,

wie waer's mit

Application.CommandBars("Cell").Enabled = False bzw. True

Gruss
Volker Croll

VERSION 2

Hallo Ihr Koenner,

kann mir einer von euch sagen wo und welchen Befehl ich benutzen muss um in einem Formular die rechte Maustaste und die F-Tasten zu unterbinden.

Ich habe mir eine Datenbank gebastelt in der ich Daten eingeben, abrufen und loeschen kann. Nur moechte ich nicht das die rechte Maustaste und die F-Tasten benutzt werden.

Besten Dank im voraus.

Hallo!

Formulareigenschaften - **Tastenvorschau** (KeyPreview) auf **Ja** (True)

Formular-Ereignis **Bei Taste Ab** (KeyUp):

Code:

```
    Select Case KeyCode
      Case vbKeyF1:      KeyCode = 0
```

```
      Case vbKeyF2:       KeyCode = 0
      Case vbKeyF3:       KeyCode = 0
      Case vbKeyF4:       KeyCode = 0
      Case vbKeyF5:       KeyCode = 0
      Case vbKeyF6:       KeyCode = 0
      Case vbKeyF7:       KeyCode = 0
      Case vbKeyF8:       KeyCode = 0
      Case vbKeyF9:       KeyCode = 0
      Case vbKeyF10:      KeyCode = 0
      Case vbKeyF11:      KeyCode = 0
      Case vbKeyF12:      KeyCode = 0
      Case vbKeyRButton: KeyCode = 0    'rechte Maustaste
    End Select
```

Hi Ihr Zwei,

oder auch etwas einfacher

Code:

```
    Select Case KeyCode
      Case vbKeyF1, vbKeyF2, vbKeyF3, vbKeyF4, vbKeyF5, vbKeyF6,
           vbKeyF7, vbKeyF8, vbKeyF9, vbKeyF10, vbKeyF11, vbKeyF12
            KeyCode = 0
' oder auch einfach
'      Case vbKeyF1 To vbKeyF12
'           KeyCode = 0
    End Select
```

Das mit **vbKeyRButton** funktioniert soweit mir bekannt nicht.

Aber wenn es Dir um das Kontextmenue geht,
dann kannst Du die Formular-Eigenschaft **Kontextmenue** (ShortcutMenu) auf **Nein** (False) setzen.

Rechte Maustaste wieder aktivieren

Hat man den rechten Mausklick deaktiviert mit verschiedenen VBA-Befehlen (auf Zelle zum Kopieren, auf Tabellenblattname zum Kopieren etc), dann setzt folgender Code wieder den rechten Mausklick komplett zurück. Leider bleibt dieses Deaktivieren auch nach Neustart erhalten von Excel. Daher einmal in eine Prozedur einfügen und ausführen und FÖDDI.

Application.CommandBars("Cell").Reset

Rechte Maustaste auf Blattregister deaktivieren

```
Application.commandbars("Ply").enabled=false ' kein rechter Mausklick auf Registerkarte
Application.CommandBars("Ply").Enabled = True ' Mausklick ist wieder aktiviert
```

SENDKEYS - Simulation von Tastenanschlägen

Ein a senden

SendKeys ("a")

Ein Shift-a senden

SendKeys ("+a")

STRG ist ^ und ALT ist %

Möchte man + oder ^ oder % als Zeichen übergeben muss man sie in "{}" setzen:

SendKeys ("{+}") sendet also ein echtes Plus

Um z.B. eine Auswahl eines Buttons / einer Autoform aufzuheben, reicht es eine andere Zelle zu aktivieren oder man sendet die ESC nach Excel:

Application.SendKeys ("{ESC}")

Strg-V: SendKeys ("^v")

ESC-Tastendruck nach Word senden

Als ich bei der IAEA ein Worddokument – mit dem Dateinamen Coverletter XZY.doc – von Excel aus öffnete und mit Bildern und Textfeldern versah, gelang es mir nicht die Textfelder-Markierung (die ich brauchte um Text reinzuschreiben und diesen Mittig zu machen) wieder aufzuheben. Der vom Macrorecorder aufgezeichnete Selection.Collapse-Befehl funktionierte nicht.

Da auch zweimaliges ESC-Tastendrücken reicht, sende ich einfach von Excel-VBA nach Word die ESC-Taste.

Da ich das Wordfenster gar nicht aktiviert hatte während ich zuvor dort reinschrieb, muss es zuerst aktiviert werden mit

```
AppActivate ("Cover")
```

Dass der Dateiname wesentlich länger ist und das gesamte Word-Fenster, dass ich mit AppActivate aktiviere, neben dem Dateinamen auch noch „(schreibgeschützt)" und „ – Microsoft Word" dort stehen hatte, ist egal – der VBA-Befehl durchsucht alle offenen Fensternamen und nimmt das erste, das mit unserem Wort Cover beginnt und aktiviert es.

Nun kommt die eigentliche Übergabe der zweimaligen ESC-Tastendrucke:

```
SendKeys "{ESC}"
SendKeys "{ESC}"
```

Mit diesen drei blauen Zeilen haben wir erfolgreich Word aktiviert und darin zweimal auf ESC gedruckt.

SendKeys-Anweisung (Beispiel)

In diesem Beispiel wird die **Shell**-Funktion verwendet, um die Rechner-Anwendung von Microsoft Windows auszuführen. Anschließend werden mit der **SendKeys**-Anweisung Tastenanschläge an den Rechner gesendet, um Zahlen zu addieren und den Rechner wieder zu beenden. (Sie probieren das Beispiel aus, indem Sie es in eine Prozedur einfügen und diese anschließend ausführen. Da **AppActivate** den Fokus auf die Rechner-Anwendung setzt, können Sie den Code nicht mit Einzelschritten ausführen.) Verwenden Sie auf dem Macintosh anstelle des Windows-Rechners eine Macintosh-Anwendung, die Tastatureingabe akzeptiert.

```
Dim Ergebnis, I
Ergebnis = Shell("CALC.EXE", 1)    ' Rechner starten.
AppActivate Ergebnis     ' Rechner aktivieren.
For I = 1 To 100    ' Zählschleife beginnen.
   SendKeys I & "{+}", True    ' Tastenanschläge senden, um die
Next I    ' Werte von I zu addieren.
SendKeys "=", True    ' Gesamtsumme abrufen.
SendKeys "%{F4}", True    ' Rechner mit ALT+F4 beenden.
```

SendKeys-Anweisung

Sendet eine Tastenfolge (die aus einem oder mehreren Tastenanschlägen bestehen kann) an das aktive Fenster, als ob sie über die Tastatur eingegeben worden wäre.

Syntax
SendKeys ***string***[, ***wait***]
Die Syntax der **SendKeys**-Anweisung verwendet die folgenden benannten Argumente:

Teil	Beschreibung
string	Erforderlich. Ein Zeichenfolgenausdruck, der die zu sendende Tastenfolge angibt.
wait	Optional. Ein Wert vom Typ Boolean, der den Wartemodus angibt. Wenn der Wert **False** ist (Voreinstellung), setzt die Prozedur die Ausführung fort, unmittelbar nachdem die Tastenfolge gesendet wurde. Wenn der Wert **True** ist, muß die Tastenfolge verarbeitet werden, bevor die Prozedur die Ausführung fortsetzen kann.

Bemerkungen
Jede Taste wird durch mindestens ein Zeichen repräsentiert. Ein einzelnes Zeichen auf der Tastatur kann mit dem Zeichen selbst angegeben werden. "A" für das Argument ***string*** repräsentiert beispielsweise den Buchstaben A. Sie geben mehrere Zeichen an, indem Sie die Zeichen aneinanderhängen. "ABC" für ***string*** repräsentiert zum Beispiel die Buchstaben A, B und C.

Das Pluszeichen (**+**), Caret-Zeichen (**^**), Prozentzeichen (**%**), die Tilde (**~**) und die Klammern **( )** haben bei der **SendKeys**-Anweisung eine spezielle Bedeutung. Sie müssen jedes dieser Zeichen in geschweifte Klammern einschließen ({}), um es verwenden zu können. Für das Pluszeichen geben Sie beispielsweise {+}an. Eckige Klammern ([ ]) haben bei der **SendKeys**-Anweisung zwar keine spezielle Bedeutung, müssen aber auch in geschweifte Klammern eingeschlossen werden, da sie in anderen Anwendungen eine spezielle Bedeutung haben, insbesondere im Zusammenhang mit dynamischem Datenaustausch (DDE). Die Zeichen für die geschweiften Klammern legen Sie unter Verwendung von {{} und {}} fest.

Für Zeichen, die beim Drücken einer Taste nicht angezeigt werden (z.B. die EINGABETASTE oder TAB-TASTE) und für bestimmte Aktionstasten können Sie die folgenden Codes verwenden:

Taste	Code
RÜCKTASTE	{BACKSPACE}, {BS} oder {BKSP}
PAUSE	{BREAK}
FESTSTELLTASTE	{CAPSLOCK}
ENTF	{DELETE} oder {DEL}
NACH-UNTEN	{DOWN}
ENDE	{END}
EINGABETASTE	{ENTER}oder ~
ESC	{ESC}
HILFE	{HELP}
POS 1	{HOME}

EINFG	{INSERT} oder {INS}
NACH-LINKS	{LEFT}
NUM-FESTSTELL	{NUMLOCK}
BILD-AB	{PGDN}
BILD-AUF	{PGUP}
DRUCK	{PRTSC}
NACH-RECHTS	{RIGHT}
ROLLEN-FESTSTELL	{SCROLLLOCK}
TAB	{TAB}
NACH-OBEN	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}

F15	{F15}
F16	{F16}

Sie können Tastenkombinationen mit der UMSCHALTTASTE, STRG-TASTE oder ALT-TASTE angeben, indem Sie vor dem normalen Tasten-Code einen oder mehrere der folgenden Codes angeben:

Taste	**Code**
UMSCHALT	+
STRG	^
ALT	%

Wenn UMSCHALT, STRG und ALT gleichzeitig mit anderen Tasten gedrückt werden müssen, schließen Sie die Codes für die Tasten in Klammern ein. Wenn zum Beispiel die UMSCHALTTASTE gleichzeitig mit den Tasten E und C gedrückt werden soll, geben Sie "+(EC)" an. Wenn die UMSCHALTTASTE zusammen mit E gedrückt werden soll und im Anschluß daran C ohne UMSCHALTTASTE, geben Sie "+EC" an.

Tastenwiederholungen können Sie in der Form {Taste Zahl} angeben. Das Leerzeichen zwischen Taste und Zahl ist dabei zwingend erforderlich. {LEFT 42} wird zum Beispiel als 42-maliges Drücken der NACH-LINKS-TASTE interpretiert, {h 10} als 10-maliges Drücken der Taste H.

Anmerkung **SendKeys** kann keine Tastenanschläge an Anwendungen senden, die nicht unter Microsoft Windows oder auf dem Macintosh ausgeführt werden können. **SendKeys** kann auch die DRUCK-TASTE {PRTSC} an keine Anwendung senden.

WICHTIG: man darf bei der SENDKEYS-Anweisung die Bildschirmaktualisierung (Application.ScreenUpdating) nicht deaktivieren, sonst kommt es zu Fehlern oder Crashes

BSP: VBA-Code macht etwas in einer anderen Anwendung, worauf dort eine Sicherheitsabfrage kommt (innerhalb von Excel könnte man die ja deaktivieren), die mit der RETURN-Taste beantwortet werden muss:

```
Application.SendKeys ("{ENTER}") verwenden um die Bestätigung durch ENTER-Tastendruck zu simulieren...
```

SENDKEYS andere Anwendung steuern

Steuern von Drittprogrammen mit Hilfe der SendKeys-Methode

Nicht alle Programme auf einem Windows-System können von Visual Basic for Application aus erreicht werden. Um Programme die ihre Ressourcen nicht z.B. als COM-Objekte anbieten trotzdem in einem VB/VBA-Projekt einzubinden muss deshalb ein anderer Weg beschritten werden. Ein Möglichkeit dazu bietet die "SendKeys-Methode".

Einleitung

Die meisten Programme sind über Tastaturkürzel steuerbar. So gelangt man z.B. bei MS-Word mit der Taste [Alt] in die Menüleiste und kann dann alle angebotenen Funktionen nutzen indem man die unterstrichenen Buchstaben auf der Tastatur drückt. Die Kombination [Alt] und dann [D] öffnet z.B. das Menü "Datei". Auch einige Steuertasten wie die Pfeiltasten oder die Taste [RETURN] sind nutzbar. Diese Eigenschaft, ein Programm über die Tastatur zu steuern wird bei der "SendKeys-Methode" ausgenutzt. Die Tastenkombinationen werden dann nicht vom Anwender eingegeben, sondern von VB/VBA aus an das geöffnete Programm bzw. an die geöffnete Applikation gesendet. Bei der Angabe der Tastaturcodes ist die Groß-/Kleinschreibung relevant und muss beachtet werden. Vor der eigentlichen Programmierung sollten die einzelnen Schritte unbedingt manuell nachvollzogen bzw. ausprobiert und dokumentiert werden.

Bei dem Verfahren ist es allerdings sehr wichtig sicherzustellen, dass die zu steuernde Applikation geladen und aktiv ist. Sie muss "den Fokus" haben. "Fokus" bedeutet hier, die nächste Tastatureingabe gilt dem betreffenden Programm. Wird dieser Punkt nicht beachtet, landen die abgesandten Tastaturkürzel in dem Programm, dass zufällig den Fokus hat (das kann auch Windows selbst sein) und es werden unvorhersehbare Aktionen ausgelöst.
Die allgemeine Vorgehensweise ist also:

- Sicherstellen dass das zu steuernde Programm den Fokus hat.
- Tastaturkürzel senden.
- Ergebnis(se) sichern.

Der Fokus des Programms

Um sicherzustellen, dass das gewünschte Programm "den Fokus" hat, gibt es verschiedene Möglichkeiten. Besonders sicher und einfach anzuwenden ist die Benutzung von AppActivate. Die allgemeine Syntax lautet:

```
AppActivate (Titel, [Wait])
```

Listing 1

"Titel" ist vom Typ String und enthält den Namen oder die Application-ID der Anwendung. Dieser erscheint im Normalfall in der Titelleiste der Anwendung. Es reicht, wenn man ein Wort oder die ersten paar Buchstaben aus dem Titel angibt. Dies kann aber zu unerwünschten Ergebnissen führen, wenn die Angabe nicht eindeutig ist. Zum Beispiel beginnen die Programme Microsoft Excel und Microsoft Access beide mit "Microsoft" - sofern keine Dokumente geladen sind. Am sichersten ist es deshalb, anstelle des Namens die Application-ID (AppID) zu verwenden.

Beispiel für den "Titel" bei MS-Word mit geladener Datei "SendKeysMethode.doc":

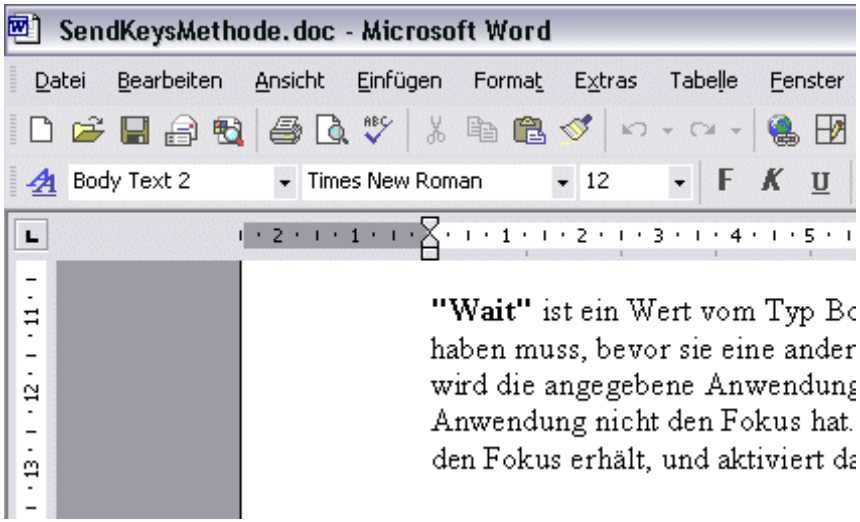


Abbildung 1

"Wait" ist ein Wert vom Typ Boolean, der angibt, ob die aufrufende Anwendung den Fokus haben muss, bevor sie eine andere Anwendung aktiviert. Beim Wert False (Voreinstellung) wird die angegebene Anwendung mit sofortiger Wirkung aktiviert, auch wenn die aufrufende Anwendung nicht den Fokus hat. Beim Wert True wartet die aufrufende Anwendung, bis sie den Fokus erhält, und aktiviert dann die angegebene Anwendung.

Die Anweisung: AppActivate "MeinDokument1.doc - Microsoft Word", Wait:=True würde das bereits gestartete Textverarbeitungsprogramm mit der geladenen Datei MeinDokument1.doc in den Vordergrund bringen.

Die sicherste Methode überhaupt ist es, das Programm von VB/VBA selbst aus zu starten. Die gewünschten Funktionen mit Hilfe der Sendkeys abzurufen und danach das Programm wieder zu schließen. Dies gelingt z.B. mit Hilfe eines Shell-Aufrufes. Dabei kann dann auch die Application-ID aufgefangen werden, mit der später eine eindeutige Zuordnung zum Programm möglich ist. Ein Beispiel dazu:

```
Dim appID As Integer
appID = Shell("notepad.exe", vbNormalFocus)
AppActivate appID
```

Listing 2

Die erste Zeile enthält eine gewöhnliche Deklaration. Die zweite Zeile startet den Windows-Editor Notepad und füllt die Integervariable appID mit der zugehörigen Application-ID. Die Zahl 1 (oder vbNormalFocus) hinter dem String "notepad.exe" ist übrigens optional. Dieser Parameter bestimmt wie das Programm gestartet wird. Dabei bedeuten:

- 1 (vbNormalFocus) Das Programm wird in Normalgröße gestartet und erhält den Fokus

- 2 (vbMinimizedFocus) Das Programm wird minimiert gestartet und erhält den Fokus
- 3 (vbMaximizedFocus) Das Programm wird maximiert gestartet und erhält den Fokus

Die dritte Zeile garantiert, dass die soeben gestartete Applikation (Editor Notepad im Beispiel) den Fokus erhält, den der Wert von appID ist unique. Diese Zeile kann je nach Erfordernis irgendwo - auch mehrfach - im Programmtext stehen. Die Anordnung unmittelbar nach dem Shell-Aufruf wie im Beispiel oben ist eigentlich überflüssig, da das Programm damit bereits den Fokus erhält. Typisch ist es diese Zeile unmittelbar vor dem Absenden der Tastaturcodes anzuordnen.

Im obigen kleinen Beispiel wurde in der zweiten Zeile stillschweigend vorausgesetzt, dass der Pfad zur Datei notepad.exe in Windows bekannt ist. Falls dies nicht garantiert werden kann, ist natürlich die komplette Pfadangabe zu übergeben:

```
appID = Shell("C:\WINNT\notepad.exe", vbNormalFocus)
```

Listing 3

Tastaturkürzel senden

Nachdem das zu steuernde Programm auf diese Weise gestartet wurde und den Fokus erhalten hat, können die Tastaturcodes abgesandt werden. Die allgemeine Syntax dazu lautet wie folgt:

```
SendKeys String [,Wait]
```

"String" beinhaltet den zu sendenden Tastaturcode, der als String übergeben wird.

"Wait" ist optional und vom Typ Boolean. Hat "Wait" den Wert True, dann wird gewartet bis die Anweisung komplett ausgeführt wurde. Erst danach erfolgt die Fortsetzung der Programmausführung in der nächsten Zeile. Falls "Wait" den Wert False hat, wird der Tastaturcode abgesandt und unmittelbar danach in der nächsten Zeile weitergemacht. Im Prinzip kann dies ausgenutzt werden um parallele Prozesse anzuwerfen (was aber einige Gefahren in sich birgt).

Beispiel: das Programm, das gerade zufällig den Fokus hat soll geschlossen werden.

```
SendKeys String:="%{F4}", Wait:=True
```

Listing 4

oder kürzer:

```
SendKeys "%{F4}", True
```

Listing 5

"%{F4}" bedeutet hier [Alt]+[F4].

Einige Tasten haben spezielle Aufgaben. Eine Liste der bei Spezialtasten zu verwendenden Strings ist nachfolgend abgebildet.

Backspace	{BACKSPACE} oder {BS} oder {BKSP}
Caps Lock	{CAPSLOCK}
Druck	{PRTSC}
Einfügen	{INSERT} oder {INS}
Entfernen	{DELETE} oder {DEL}
Ende	{END}
Help	{HELP}
NUM Lock	{NUMLOCK}
Pause	{BREAK}
Pos1	{HOME}
Return/Enter	{ENTER} oder ~
Scroll Lock	{SCROLLLOCK}
Tabulator	{TAB}
←	{LEFT}
↑	{UP}
→	{RIGHT}
↓	{DOWN}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}
Shift/Umschalt	+
Ctrl bzw. Strg	^
Alt	%

Abbildung 2

Es gibt ein paar Grundregeln bei der Übergabe der Tastaturcodes, deren Kenntnis den Umgang mit der Methode sehr vereinfacht:

- die Tastaturcodes werden in Anführungsstrichen übergeben (wegen des Variablentyps String)

- Tasten mit speziellen Funktionen werden in geschwungenen Klammern gesetzt.
- die Zeichen ~%()+{}[] müssen in runden Klammern übergeben werden
- Wenn UMSCHALT, STRG und ALT gleichzeitig mit anderen Tasten gedrückt werden müssen, wird der Code für die Tasten in Klammern eingeschlossen. Wenn zum Beispiel die UMSCHALTTASTE (Shift-Taste) gleichzeitig mit den Tasten [E] und [C] gedrückt werden soll, muss "+(EC)" übergeben werden. Wenn die UMSCHALTTASTE zusammen mit [E] gedrückt werden soll und im Anschluß daran [C] ohne UMSCHALTTASTE, lautet der Ausdruck richtig "+EC".
- Soll eine Taste mehrfach betätigt werden, kann dies hinter dem Kürzel in der geschweiften Klammer angegeben werden. Wenn zum Beispiel die Tabulator-Taste zweimal betätigt werden soll kann geschrieben werden: SendKeys "{TAB 2}", True

Anwendungsbeispiel: Mit Hilfe des Windows-Calculators (calc.exe) soll die Summe aller Zahlen von 1 bis 100 berechnet werden. Anschließend wird der Calculator geschlossen. Das Ergebnis wird nicht weiterverwertet.

```
Dim Ergebnis, I

'Starten des Windows-Calculators und Auffangen der Applikation-ID
appID = Shell("CALC.EXE", 1)

'Sicherstellen dass der Calculator den Fokus hat
AppActivate appID

'berechnen der Summe der Zahlen von 1 bis 100
For I = 1 To 99
    'send keys to add the values of I
    SendKeys I & "{+}", True
Next I
SendKeys ("100"), True
SendKeys "=", True

'Schließen des Calculators mit [Alt]+[F4]
SendKeys "%{F4}", True
```

Listing 6

Ergebnis sichern

Im vorangegangenen Beispiel wurde etwas mit dem Calculator berechnet. Bei der Programmausführung kann man dies am Bildschirm mitverfolgen. Das Ergebnis wurde aber wieder vergessen bzw. nicht verwendet. Normalerweise möchte man aber das Resultat aus der fremden Applikation nutzen. Es stellt sich also die berechtigte Frage, wie kann das Ergebnis einer SendKeys-Aktion in einem VB/VBA-Programm (oder in einem anderen, dritten Programm) benutzt werden. Dazu bieten sich drei Wege mit unterschiedlichen Vor- und Nachteilen an:

- a) über das Dateisystem (Festplatte)
- b) über die Windows-Zwischenablage

- c) über direkten Zugriff auf den Arbeitsspeicher

Punkt c) ist eher umständlich und bietet nur bei wirklich zeitkritischen Routinen einen echten Vorteil. Er wird deshalb im Rahmen dieses Aufsatzes nicht weiter behandelt. Interessanter und einfacher zu handhaben ist der Fall a) oder b).

Fall a): Ergebnisse über das Dateisystem (Festplatte) sichern

Die Möglichkeit a) ergibt sich häufig aus der Aufgabenstellung. Dazu ein Beispiel: Es soll mit Hilfe des Windows-Programms "Paint" (mspaint.exe) eine Grafikdatei vom bmp-Format in das jpg-Format umgewandelt werden. In diesem Fall ist das Ergebnis bereits eine Datei.

Im Beispiel wird der Dateiname der zu konvertierenden bmp-Datei samt Pfad in eine TextBox eingetragen. Die erzeugte jpg-Datei wird im selben Ordner abgelegt in dem sich auch die Originaldatei befindet. Dazu muss das Programm "Paint", dass sich im Windows-Systemordner befindet gestartet werden. Anschließend muss der Fokus auf dieses Programm sichergestellt werden. Dann wird die Originaldatei eingeladen und als jpg-Datei wieder abgespeichert. Zum Schluss wird das Programm Paint beendet.

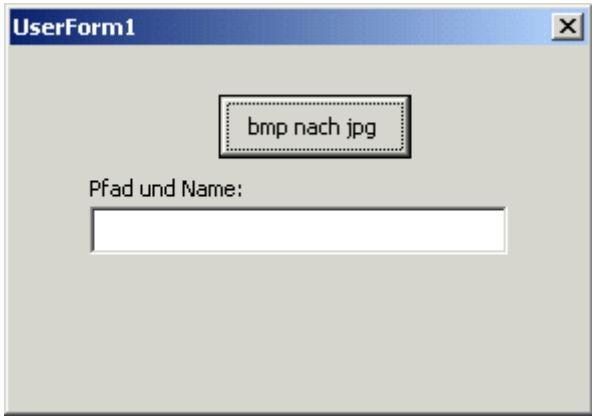


Abbildung 3: Oberfläche des Grafik-Konvertierungsprogramms

```
Private Sub grafikkonverter_Click()
    Dim appident As Integer
    Dim Eingang As String

    'Pfad zu den zu konvertierenden bmp-Dateien
    Eingang = TextBox1.Value

    'Starten des Programms MS-Paint und den Fokus darauf setzen
    appident = Shell("C:\WINNT\system32\mspaint.exe", 1)
    AppActivate appident

    'Einladen der Datei in MS-Paint
    SendKeys "%Df", True
    SendKeys Eingang, True
```

```
    SendKeys "{ENTER}", True

    'Abspeichern der eingeladenen Datei als jpg-Typ im selben Verzeichnis
    SendKeys "%Du", True
    SendKeys "{TAB}j", True
    SendKeys "{ENTER}", True

    'MS-Paint beenden
    SendKeys "%({F4})"
End Sub
```

Listing 7

Nachfolgend die Bedeutung der im obigen Beispiel verwendeten Tastenkürzel:

Das Einladen der Datei erfolgt mit

- % Alt, um in die Menüleiste zu gelangen
- D zum Öffnen des Pulldown-Menüs "Datei"
- f den Dialog zum Einladen einer Datei öffnen

Beim Abspeichern sieht es wie folgt aus:

- % Alt, um in die Menüleiste zu gelangen
- D zum Öffnen des Pulldown-Menüs "Datei"
- u zum Öffnen des Dialogs "Speichern unter"

In diesem Zustand ist der Fokus auf die Eingabe des Dateinamens gesetzt. Mit {TAB} gelangt man in das nächste Feld, mit dem der Dateityp vorgegeben wird. Man kann nun durch Eintippen des Anfangsbuchstabens eine Auswahl treffen. Mit "j" wird das jpg-Format ausgewählt.

Dieses kleine Programm ist natürlich noch nicht alltagstauglich, denn wenn z.B. die Zieldatei bereits im Ordner vorhanden ist gelingt das Abspeichern (und in diesem Falle das Überschreiben) und Beenden des Programmes nicht. MS-Paint würde anhalten und fragen ob die vorhandene Datei überschrieben werden darf. Da es bei der SendKeys-Methode keine Rückmeldungen gibt, müsste also vorher mit herkömmlichen VB/VBA-Methoden überprüft werden, ob der Zielordner leer ist bzw. ob sich vielleicht eine gleichnamige Datei darin befindet.

Außerdem wäre es sinnvoll wenn alle bmp-Dateien die sich in einem Ordner befinden automatisch in das jpg-Format konvertiert würden, ohne dass man einen Dateinamen auswählen muss. Aber auch dafür gibt es Lösungen.

Ein etwas umfangreicheres Beispiel: Alle Bilder (jpg-Dateien) in einem Verzeichnis sollen verkleinert werden, wobei das Höhe-Breiten-Verhältnis und auch der Inhalt nicht verändert werden darf. Die Resultate sollen eine einheitliche Höhe von 400 Pixel aufweisen und im gleichen Verzeichnis unter einem neuen Namen abgelegt werden. Auch hier wird VB nur für die Steuerung eines Fremdprogrammes, das dann die eigentliche Arbeit erledigt, eingesetzt. Das Windows-Programm mspaint.exe ist mit dieser Aufgabe überfordert. Man kann sich ein über die Tastatur steuerbares Free- oder Sharewareprogramm beschaffen und die Aufgabe mit dessen Hilfe lösen. Im hier gezeigten Beispiel wird das Programm PaintShopPro (Shareware) in der Version 7.06 (engl.) verwendet. Dieses Programm ist im Internet verfügbar. Es kann zu Testzwecken kostenlos heruntergeladen und installiert werden. Um das Beispiel nachzuvollziehen muss PSP 7.06 auf dem Windows-System installiert sein. Eine andere Version dieses Programms würde eventuell eine Anpassung der Tastaturcodes erfordern.

Betrachtet man den Programmcode, so zeigt sich, dass bei der SendKeys-Methode noch eine weitere Schwierigkeit auftaucht: Es gibt keine Rückmeldung, ob eine Anweisung fertig ausgeführt wurde. Bei den meisten Vorgängen hängt die Ausführungsdauer von der verwendeten Hardware ab. Sendet man aber einen Folgecode zu früh, also wenn z.B. ein Speichervorgang noch nicht abgeschlossen ist, so kann dies zu unvorhersehbaren Aktionen führen. Aus diesem Grund wurden im Programmcode an kritischen Stellen Pausen eingearbeitet. Eine schnelle Programmausführung läßt die Methode also nicht zu. Außerdem ist die Ausführungsdauer von der Anzahl der umzuarbeitenden Dateien abhängig. Das heißt, dass Programm PaintShopPro ist eine geraume Zeit maximiert auf dem Bildschirm zu sehen und hat den Fokus. Wegen der eingefügten, notwendigen Pausen steht das Programm zeitweise still. Klickt ein Anwender in dieser Phase auf einen Menüpunkt oder gewährt einem anderen Programm den Fokus, so werden die Voraussetzung für das Weiterarbeiten verändert und die nächsten gesendeten Steuercodes laufen ins Leere oder führen in einem Programm das zufällig den Fokus hat, unvorhersehbare Aktionen aus. Es ist also sowohl bei der Programmierung, als auch bei der Anwendung Vorsicht geboten!

```
'Dieses Programm verwendet PSP 7.06 (engl.)
'um eine Anzahl von max. 100 Bildern
'auf eine einheitliche Höhe von 400 Pixel zu bringen.
'Das Seitenverhältnis Breite/Höhe bleibt erhalten.

Option Explicit
Option Base 1

'die Bildhöhe fest vorgeben
Const einheitlichehoehe As Integer = 400

'den Pfad mit den zu bearbeitenden Dateien fest vorgeben
Const quellpfad As String = "E:\Fotos\"

'Array mit den Namen der zu ändernden Dateien (maximal 100)
Dim dateinamen(100) As String


Sub verkleinern()
    Dim l_anzahlreturn As Integer
    Dim l_zaehler As Integer
    Dim l_appid As Integer
    Dim l_pfadundname As String
    Dim l_returnpause As String
```

```
'ermitteln wieviele Bilddateien verkleinert
'werden sollen (Funktionsaufruf)
l_anzahlreturn = wievieledateien(quellpfad)

'nur arbeiten, wenn wenigstens eine Bilddatei im Verzeichnis liegt
If l_anzahlreturn > 0 Then
'PaintShopPro öffnen und die Application-ID auffangen
l_appid = Shell(C:\Programme\Jasc Software Inc\ _
Paint Shop Pro 7\psp.exe, 3)
'3 bedeutet: Maximiert, mit Fokus

'PaintShopPro erscheint im vorliegenden Fall mit einem Start-
'Fenster und einem Dialogfenster; beides muss quittiert werden.
'Des Start-Fenster verschwindet nach ein paar Sekunden von alleine.
'Das Dialogfeld muss quittiert werden.
'Pauschal 10 s warten bis dass das Start-Fenster verschwunden ist.
l_returnpause = pause(10)

'Das Dialogfenster schließen mit [RETURN]
AppActivate l_appid, False
SendKeys "{ENTER}", True

    For l_zaehler = 1 To l_anzahlreturn

    'den Öffnen-Dialog starten
    SendKeys "^o", True 'entspricht File Open

    'ein Bild einladen
    l_pfadundname = quellpfad & dateinamen(l_zaehler)
    SendKeys l_pfadundname, True
    SendKeys "{ENTER}", True

    'Pauschal 5s warten bis die Datei eingelesen ist
    l_returnpause = pause(5)

    'Bildgröße anpassen (die eigentliche Umrechnung)
    SendKeys "+S", True    'entspricht Image Resize
    SendKeys "400", True    'entspricht Eingabe der Breite 400
    SendKeys "{ENTER}", True  'Übernahme der Einstellung

    'Zielpfad- und name festlegen
    l_pfadundname = quellpfad & "klein_" & dateinamen(l_zaehler)

    'Pauschal 1s warten bis die Umrechnung abgeschlossen ist
    l_returnpause = pause(1)

    'Speicherdialog aufrufen, neuen Dateinamen eingeben und
    'in das selbe Verzeichnis abspeichern
    SendKeys "{F12}", True 'File SaveAs-Dialog öffnen
    SendKeys l_pfadundname, True
    SendKeys "{ENTER}", True

    'falls die Datei schon da ist: Dialog quittieren und Datei
```

```
        'überschreiben; falls die Datei noch nicht existiert stört das
        '"J" auch nicht
        SendKeys "J", True

        'Pauschal 3s warten bis abgespeichert ist (da die Bilder jetzt
        'sehr klein sind nur 3 Sekunden pro Bild)
        l_returnpause = pause(3)

        'Bild schließen
        SendKeys "%", True   'entspricht [Alt]
        SendKeys "FC", True  'entspricht File Close
        SendKeys "N", True   'entspricht Nein, das Original nicht
        'überschreiben

        Next l_zaehler

    'PaintShopPro beenden
    SendKeys "%{F4}", True

    Else
        MsgBox "Keine Dateien im Verzeichnis vorhanden."
    End If

End Sub


Function wievieledateien(pfadangabe As String) As Integer
'hier wird festgestellt wieviele Dateien im Quellverzeichnis vorhanden
'sind und welche Namen diese haben

    Dim l_zaehler As Integer
    Dim l_dname As String

    l_zaehler = 0

    'erste Datei in pfadangabe lesen und dir initialisieren
    l_dname = Dir(pfadangabe)

    'falls mindestens eine Datei vorhanden ist versuchen alle weiteren
    'festzustellen; die Dateiendung wird nicht geprüft. Es werden jpg-
    'Dateien vorausgesetzt.
    If l_dname <> "" Then
        l_zaehler = 1
        dateinamen(l_zaehler) = l_dname
        Do
            l_dname = Dir()
            If l_dname <> "" Then
            l_zaehler = l_zaehler + 1
            dateinamen(l_zaehler) = l_dname

            'Sicherstellen das nur maximal 100 Dateien bearbeitet
            'werden
            If l_zaehler = 100 Then
```

```
                Exit Do
            End If
            Else
            'falls weniger als 100 Dateien vorhanden sind, weitermachen
            Exit Do
            End If
        Loop

        'Anzahl der Dateien zurückgeben
        wievieledateien = l_zaehler
    Else
        wievieledateien = 0
    End If
End Function


Function pause(wartesekunden As Integer) As String
'Dieser Funktion wird eine Wartezeit in Sekunden (Ganzzahl)
'angegeben.
    Dim l_zeit As Double
    Dim l_wartezeit As Double
    Dim l_deltazeit As Double
    l_zeit = CDbl(Time)
    l_wartezeit = (1 / 86400) * wartesekunden '86400 = 24*3600
    Do
        l_deltazeit = Time - l_zeit
        If l_deltazeit > l_wartezeit Then
            Exit Do
        End If
    Loop
    pause = "Pause beendet!"
End Function
```

Listing 8

Wenn nicht Dateien, sondern einzelne Daten (z.B. Zahlenwerte) übergeben werden müssen, können diese in eine Datei auf die Festplatte abgelegt und dann mit VB/VBA-Mitteln geöffnet und übernommen werden. Das Ablegen der Werte geschied z.B. mit der SendKeys-Methode indem man üblicherweise den Dialog "Datei/Speichern" unter bzw. "File/Save as" ansteuert. Die angelegte Datei wird dann unter zuhilfenahme üblicher VB/VBA-Mittel (Open....) eingelesen und die Daten können ausgewertet werden. Wie dies gemacht wird ist nicht Thema dieses Aufsatzen und vermutlich bekannt. Ein Beispiel erübrigt sich deshalb hier.

Fall b) Ergebnisse über die Windows-Zwischenablage sichern

Diese Methode ist besonders einfach und schnell anzuwenden. Nehmen wir das gezeigte Beispiel mit dem Windows-Calculator. Das Ergebnis der Berechnung wurde dort nicht verwendet - es wurde einfach wieder vergessen. Soll es in die Zwischenablage übernommen werden, ist vor dem Schließen des Calculators die folgende Zeile einzufügen:

```
SendKeys "^c", True
```

Listing 9

Beim String "^c" handelt es sich um die übliche Bedienung der Zwischenablage über die Tastatur:

- Strg + C : das markierte Elemente bzw. die markierten Elemente als Kopie in die Zwischenablage legen
- Strg + X : das markierte Element ausschneiden und in die Zwischenablage legen
- Strg + V : den Inhalt der Zwischenablage an die bezeichnete Stelle ablegen (kopieren)

Das "markierte Element" ist im vorliegenden Fall der im Calculator errechnete Wert. Möglich wäre auch die Steuerung über die spezifischen Tastaturkürzel des jeweiligen Programms. Dies sähe beim Windows-Calculator wie folgt aus:

```
SendKeys "%BK", True
```

Listing 10

Dies entspricht der Tastaturbedienung [Alt] [Bearbeiten] [Kopieren]. Das Ergebnis befindet sich dann in der Zwischenablage. Um es von dort ohne Umweg abzuholen und z.B. in einer Textbox auszugeben kann die "GetFromClipboard"-Methode benutzt werden.

Beispiel: Mit Hilfe des Windows-Calculators soll eine Dezimalzahl in eine Binärzahl umgewandelt werden. Der Wert wird in einem Eingabe-Textfeld entgegengenommen, dann über die Zwischenablage dem Windows-Calculator übergeben und dort binär dargestellt. Anschließend wird der im Calculator dargestellte Wert - also das Ergebnis - in die Zwischenablage gebracht und in einem Ausgabe-Textfeld aus der Zwischenablage übernommen.

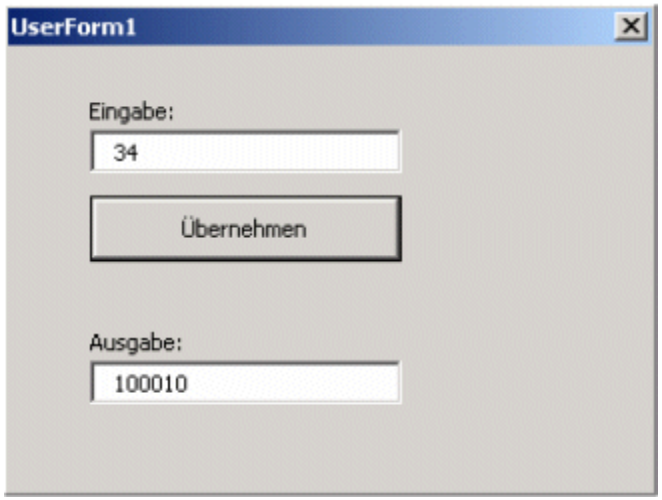


Abbildung 4

```
Option Explicit

'Umrechnung einer Dezimalzahl in eine Binärzahl.
Private Sub uebernehmen_Click()

    Dim l_appid As Integer
    Dim l_inputvalue As String

    'Eingabewert in der TextBox markieren und in die Zwischenablage bringen
    TextBox1.SelStart = 0
    TextBox1.SelLength = TextBox1.TextLength
    TextBox1.Copy

    'Windows-Calculator maximiert starten und Fokus geben
    l_appid = Shell("C:\WINNT\system32\calc.exe", 3)

    'auf wissenschaftliche Darstellung schalten
    SendKeys String:="%AW", Wait:=True

    'Inhalt der Zwischenablage einfügen
    SendKeys String:="^v", Wait:=True

    'Umschalten auf Binärdarstellung
    SendKeys String:="%ab", Wait:=True

    'Binärzahl in die Zwischenablage bringen
    'und den Calculator schließen
    SendKeys String:="^c", Wait:=True
    AppActivate l_appid 'Sicherstellen dass der Calculator den Fokus hat
    SendKeys String:="%{F4}", Wait:=True
```

```
    'Ergebnis anzeigen
    TextBox2.Paste

End Sub
```

Listing 11

Schlusswort

Die Beispiele haben sicherlich die Einfachheit der Anwendung der SendKeys-Methode aufgezeigt. Allerdings birgt sie auch Gefahren: Wenn der Fokus nicht sorgfältig auf das zu steuernde Programm gesetzt wird, können unvorhersehbare Aktionen ausgelöst werden. Dies trifft auch zu, wenn die zu sendenden Tastaturcodes einen Fehler enthalten. Es ist deshalb dringend anzuraten, die auszuführenden Aktionen vor dem eigentlichen Programmieren zunächst einmal per Handarbeit auszuführen und die verwendeten Tastaturkürzel sorgfältig zu notieren.

In jedem Fall sollte, wie in den Beispielen zu sehen, immer der Parameter Wait auf True gesetzt sein. Unterläßt man dies und ein Befehl wird nicht ausgeführt, weil sich vielleicht die Umstände geändert haben (z.B. eine abzuspeichernde Datei ist bereits auf der Festplatte, etc.) können die übrigen Anweisungen unter Umständen unvorhersehbare Aktionen auslösen. Ist der Parameter Wait jedoch auf True gesetzt, werden die nach einem nicht ausführbaren Kommando folgenden Anweisungen ignoriert.

Zusätzlich sind ausreichend Pausen einzubauen die sicherstellen, dass ein Tastaturcode erst dann abgesandt wird, wenn das zu steuernde Programm bereit dafür ist - d.h. nicht mit irgendwelchen Aktionen oder Berechnungen beschäftigt ist.

SENDKEYS ALLE KÜRZEL

Syntax

Send Keys string, [wait]

Die Syntax der SendKeys-Anweisung weist folgende benannte Argumente auf:

Zeichenfolge Erforderlich. Zeichenfolgenausdruck, der die zu
sendende Tastenfolge angibt.

wait Optional. Boolescher Wert, der den Wartemodus
angibt Wenn False (Standardwert), wird das Steuerelement unmittelbar an die Prozedur zurückgegeben, nachdem die Tasten gesendet werden. Wenn True, muss die Tastenfolge verarbeitet werden, bevor das Steuerelement an die Prozedur zurückgegeben wird.

Hinweise

Jede Taste wird durch ein oder mehrere Zeichen dargestellt. Um ein einzelnes Tastaturzeichen anzugeben, verwenden Sie das Zeichen selbst. Um den Buchstaben „A" darzustellen, verwenden Sie z. B. "A" für Zeichenfolge. Um mehrere Zeichen anzugeben, hängen Sie jedes zusätzliche Zeichen an das vorherige an. Um die Buchstaben A, B und C darzustellen, verwenden Sie z. B. "ABC" für Zeichenfolge.
Pluszeichen (+), Caretzeichen (^), Prozentzeichen (%), Tilde (** ~ **) und Klammern 0 haben für SendKeys eine spezielle Bedeutung.

Wenn Sie eines der folgenden Zeichen angeben möchten, setzen Sie es in Klammern ({}). Wenn Sie zum Beispiel das Pluszeichen angeben möchten, verwenden Sie {+} .

Klammern ([]) haben keine besondere Bedeutung für SendKeys, Sie müssen diese jedoch in Klammern gesetzt werden. In anderen Anwendungen haben Klammern eine besondere Bedeutung, die möglicherweise relevant ist, wenn dynamischer Datenaustauscherfolgt. Um Klammern anzugeben, verwenden Sie {{} und {}} .
Für Zeichen, die beim Drücken einer Taste nicht angezeigt werden (z. B. die EINGABETASTE oder TAB-TASTE) und für bestimmte Aktionstasten können Sie die Codes in der folgenden Tabelle verwenden:

Taste	Code
RÜCKTASTE	{BACKSPACE} , {BS} oder {BKSP}
PAUSE	{BREAK}
FESTSTELLTASTE	{CAPSLOCK}
ENTF	{DELETE} Oder {DEL}
NACH-UNTEN	{DOWN}
ENDE	{END}
EINGABETASTE	{ENTER} oder ~
ESC	{ESC}

HILFE	{HELP}
START	{HOME}
EINFG	{INSERT} oder {INS}
NACH-LINKS	{LEFT}
NUM	{NUMLOCK}
BILD-AB	{PGDN}
BILD-AUF	{PGUP}
DRUCK	{PRTSC}
NACH-RECHTS	{RIGHT}

Taste	Code
ROLLEN	{SCROLLLOCK}
TAB	{TAB}
NACH-OBEN	{UP}

Fl	{Fl}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
Fll	{Fll}
F12	{F12}
F13	{F13}

F14 {F14}

F15 {F15}

F16 {F16}

Sie können Tastenkombinationen mit der UMSCHALTTASTE, STRG-TASTE oder ALT-TASTE angeben, indem Sie vor dem normalen Tasten-Code einen oder mehrere der folgenden Codes angeben:

UMSCHALT	+
STRG	
ALT	%

Wenn UMSCHALT, STRG und ALT gleichzeitig mit anderen Tasten gedrückt werden müssen, schließen Sie die Codes für die Tasten in Klammern ein. Wenn zum Beispiel die UMSCHALTTASTE gleichzeitig mit den Tasten E und C gedrückt werden soll, geben Sie +(EC) an.

Um wiederkehrende Tasten anzugeben, verwenden Sie {key number} . Sie müssen ein Leerzeichen zwischen key und number einfügen. {LEFT 42} bedeutet zum Beispiel, dass die NACH-LINKS- TASTE 42 Mal gedrückt wird; {h 10} bedeutet, dass „H" 10 Mal gedrückt wird.

SendKeys kann keine Tastenanschläge an Anwendungen senden, die nicht unter Microsoft Windows oder auf dem Macintosh ausgeführt werden können. Sendkeys kann auch die DRUCK-TASTE {PRTSC} an keine Anwendung senden.

Beispiel

In diesem Beispiel wird die Shell-Funktion verwendet, um die unter Microsoft Windows integrierte Anwendung „Rechner" auszuführen.Es verwendet die SendKeys-Anweisung zum Senden von Tastaturanschlägen, um einige Zahlen hinzuzufügen und den Rechner zu beenden. (Um das Beispiel anzuzeigen, fügen Sie dies in einer Prozedur hinzu, und führen Sie dann die Prozedur aus. Da AppActivate den Fokus auf die Anwendung „Rechner" ändert, können Sie nicht schrittweise den Code durchlaufen.)

```
Dim ReturnValue, I
ReturnValue = Shell("CALC.EXE", 1)                    ' Run Calculator.
AppActivate ReturnValue ' Activate the Calculator.
For I = 1 To 100                ' Set up counting loop.
    SendKeys I & "{+}", True ' Send keystrokes to Calculator Next I ' to add each value of I.
SendKeys              True ' Get grand total.
1.1.2019
SendKeys "%{F4}", True
SendKeys-Anweisung (VBA) | Microsoft Docs
' Send ALT+F4 to close Calculator.
```

SENDKEYS - Theorie mit vielen BSP

» Begriffserklärungen
» Tasten-Namen

Einleitung

Inhalt dieser Seite

Auf dieser Seite finden Sie umfassende und ausführliche Informationen zum Thema "Senden von Tastenfolgen an eine Anwendung". Lesen Sie hier allerlei Interessantes zu diesen Dingen:
- warum und in welchen Situationen man Tastenfolgen senden sollte
- was man grundsätzlich beachten muss
- welche Risiken und Gefahren existieren
- welche Probleme und Fehler bekannt sind
- was man bereits vor dem Programmieren beachten sollte
- was aus technischer Sicht beim Senden von Tastenfolgen vorgeht
- wie die in VBA zur Verfügung stehenden Anweisungen und Funktionen heissen
- wie man mit VBA Tastenfolgen sendet
- was man bei Auftreten eines Fehlers tun kann
- welche Tasten nicht an eine Anwendung gesendet werden können
- wie man Spezialtasten wie u.a. CapsLock aufrufen kann
- wie eine andere Anwendung gestartet wird
- welche Alternativen es zum Senden von Tastenbefehlen gibt
- wie man überprüfen kann, ob Tastenfolgen korrekt verarbeitet wurden
- welche geheime und undokumentierte Features existieren
- wie man SendKeys-Programmcode stabil und fehlerresistent macht
- wie die offiziellen Tastenbezeichnungen lauten
- wo man weitere themenbezogene Informationen erhält

▲

Zweck von Tastenbefehlen senden

Das Senden von Tastenfolgen stellt eine einfache Lösung dar, eine beliebige Anwendung zu steuern, Funktionen dieser Anwendung auszuführen, Menübefehle aufzurufen, Daten zu übergeben bzw. zu übernehmen, und so weiter. Die Möglichkeiten sind praktisch unbegrenzt, wobei es ein paar wenige Einschränkungen gibt (siehe Kapitel Einschränkungen und Grenzen von SendKeys).
Üblicherweise werden Tastenfolgen an eine andere Anwendung gesendet. Man kann jedoch problemlos auch Tastenbefehle an die eigene Anwendung senden, sprich an diejenige Anwendung, die den Programmcode enthält und ausführt, und somit die aktive Anwendung darstellt.

▲

Einsatzgebiete

In den meisten Fällen gibt es bessere Lösungen als das Senden von Tastenbefehlen, nämlich unter anderem die Steuerung der anderen Anwendung mittels Automation (auch Office Automation genannt, früher OLE Automation genannt), oder, wenn die eigene Anwendung gesteuert werden soll, die Verwendung der entsprechenden, standardmässig vorhandenen VBA-Anweisungen und Objekt-Methoden.
Alternative 1: Objekt-Methode
Anstelle das Senden der Tastenkombination Strg+X verwendet man innerhalb Microsoft Excel zum Ausschneiden einer Zelle besser die Cut-Methode. Mit der Codezeile "ActiveCell.Cut" wird beispielsweise die gerade aktive Zelle ausgeschnitten.
Alternative 2: VBA-Anweisung
Zum Kopieren einer Datei muss man nicht den Windows Explorer mittels Shell-Funktion starten, die gewünschte Datei selektieren und dann den Kopieren-Befehl des Bearbeiten-Menüs anhand von gesendeten Tastenanschlägen aufrufen. Einfacher geht es mit der FileCopy-Anweisung von VBA.
Alternative 3: Automation
Ein neues Word-Dokument kann aus einer anderen Anwendung heraus sehr einfach mittels Office Automation angelegt werden. Man muss also nicht Microsoft Word mit VBA (z.B. mit Shell oder ActivateMicrosoftApp) starten und dann anhand von Tastenfolgen den Menübefehl Datei/Neu von Word aufrufen.
Der VBA-Programmcode dieser Automation-Lösung, welche Word starten, ein neues Dokument erstellt, das Dokument speichert und anschliessend Word beendet sieht folgendermassen aus:

```
Sub CreateNewDocument()
  Dim appWord As Object
  Set appWord = CreateObject("Word.Application")
  appWord.Documents.Add.SaveAs "D:\NewDocument.doc"
  appWord.Quit
  Set appWord = Nothing
End Sub
```

Wenn es keine Alternative gibt
Wie gesagt funktioniert in den meisten Fällen mindestens eine der oben aufgeführten Alternativen. Es gibt jedoch ein paar Fälle, bei denen weder eine VBA-Anweisung, noch eine Objekt-Methode, noch eine Automation-Lösung in Frage kommt, und zwar ganz einfach aus dem Grund, weil es keine entsprechende Lösungsmöglichkeit gibt.
Verschiedene Aufgabenstellungen lassen sich ausschliesslich durch das Senden von Tastenfolgen lösen. In Bezug auf Microsoft Excel kann man diese in zwei Gruppen einteilen:
1. Dialogmasken und Fenster, die in Excel zwar aufgerufen werden können, aber nicht mit VBA angesprochen werden können
2. Funktionen, Befehle und Optionen, die im Excel-Objektmodell nicht als Methoden oder Eigenschaften existieren
Hier ein paar solcher Aufgaben:

Aufgabe	Beschreibung	Beispiel
Drucker-Eigenschaften ändern	Die Drucker-Eigenschaften können nicht mit VBA eingestellt werden, da entsprechende Eigenschaften im Objektmodell fehlen.	-
Bearbeiten-Modus aktivieren	Der Zellbearbeitungsmodus wird in Excel mittels Doppelklick auf eine Zelle oder mit der Funktionstaste F2 aktiviert. Excel-VBA stellt keine entsprechende Methode zur Verfügung.	-

Bestimmter Datensatz in Datenmaske anzeigen	-	-
Optionen-Dialogfenster öffnen	-	-
Optionen des VBA-Editors ändern	-	-

Voraussetzungen für das Senden von Tasten

Damit eine Taste an eine Anwendung gesendet und von dieser verarbeitet werden kann, müssen insbesondere drei Bedingungen erfüllt sein:
1. Die Empfänger-Anwendung muss aktiv sein
2. Die Empfänger-Anwendung muss die an sie gesendete Taste interpretieren können
3. Die Empfänger-Anwendung muss die an sie gesendete Taste verarbeiten können

In den folgenden drei Abschnitten werden diese drei Bedingungen vorgestellt.

1. Anwendung muss aktiv sein

Gesendete Tastenfolgen werden immer von derjenigen Anwendung empfangen, die gerade aktiv ist. Die aktive Anwendung erkennt man daran, dass das Fenster dieser Anwendung aktiv ist, sprich den Fokus besitzt. Eine Anwendung, die kein Fenster besitzt oder nicht aktiviert werden kann, kann folglich auch keine Tastenbefehle empfangen (z.B. ein Systemprozess oder eine Anwendung mit einem ausgeblendeten Fenster).

2. Anwendung muss Taste verstehen können

Obwohl diese Voraussetzung an sich 'logisch' ist, darf man sie nicht vergessen und schon gar nicht als selbstverständlich betrachten. Beispielsweise die an Microsoft Excel gesendete Tastenfolge "ALT+D, I" bewirkt, dass das Dialogfenster "Eigenschaften" geöffnet wird, weil mit ALT+D das Menü "Datei" geöffnet und mit der I-Taste der Menübefehl "Eigenschaften" ausgeführt wird. Es ist soweit eigentlich klar, dass Excel die Tastenfolge nur dann versteht, wenn es ein Menü "Datei" mit unterstrichenem Buchstabe D gibt und beim Menübefehl "Eigenschaften" der Buchstabe I unterstrichen ist. Gäbe es dies nicht, würde Excel die Tasten zwar ausführen, was jedoch zu einem unerwünschten oder gar keinem Resultat führen würde.
In diesem Zusammenhang wird oft vergessen, dass das Menü "Datei", um beim obigen Beispiel zu bleiben, nicht zwingend die Zugriffstaste ALT+D besitzt. Einerseits ist es für einen Excel-Anwender kein Problem, diesem Menü eine andere oder gar keine Zugriffstaste zu vergeben. Andererseits lautet das Menü nur in der deutschsprachigen Excelversion "Datei". In beispielsweise der englischen Version heisst dieses Menü "File" und wird mit der Tastenkombination ALT+F aufgerufen. Die Tastenfolge "ALT+D, I" wird somit bei der englischen Excelversion auf jeden Fall nicht das Dialogfenster "Eigenschaften" öffnen, sondern eine andere (oder gar keine) Aktion ausführen.

3. Anwendung muss Taste verarbeiten können

Es gibt Tasten, die von einer Anwendung nicht verarbeitet werden können. Es ist beispielsweise möglich, die Feststell-Taste (Caps Lock) mittels SendKeys zu senden (bzw. zu drücken):

```
SendKeys "{CAPSLOCK}"
```

Allerdings wird nichts passieren, wenn Sie diese Anweisung ausführen. Es liegt nicht etwa daran, weil der Tastencode "CAPSLOCK" falsch wäre (er existiert sehr wohl) oder weil die Feststell-Taste zwei Zustände Ein/Aus haben kann (auch die Druck-Taste funktioniert nicht). Der Grund ist ganz einfach, dass die aktive Anwendung mit der Feststell-Taste nichts anfangen und daher nicht verarbeiten kann. Nur das Betriebssystem - im Falle von VBA ist dies gewöhnlich Windows - kann diese Taste verarbeiten. Weil aber mit SendKeys Tasten immer an das aktive Anwendungsfenster gesendet werden und nie an Windows, funktioniert die obige VBA-Anweisung nicht.

Die verschiedenen Möglichkeiten zum Senden von Tastenfolgen

Sinn und Zweck von SendKeys

SendKeys sendet Tastenfolgen an die aktive Anwendung bzw. das aktive Fenster. Die Methode SendKeys des Application-Objektes von Excel (Application.SendKeys) besitzt grundsätzlich den gleichen Zweck, besitzt allerdings ein paar kleine Unterschiede. Mit DDEExecute kann man ebenfalls Tastenanschläge an eine andere Anwendung senden.

SendKeys (VBA-Anweisung)

Syntax

SendKeys *string*[, *wait*]

Die Syntax der SendKeys-Anweisung verwendet die folgenden benannten Argumente:

Argument	Beschreibung
string	Erforderlich. Ein Zeichenfolgenausdruck, der die zu sendende Tastenfolge angibt.
wait	Optional. Ein Wert vom Typ Boolean, der den Wartemodus angibt. Wenn der Wert False ist (Voreinstellung), setzt die Prozedur die Ausführung fort, unmittelbar nachdem die Tastenfolge gesendet wurde. Wenn der Wert True ist, muss die Tastenfolge verarbeitet werden, bevor die Prozedur die Ausführung fortsetzen kann.

Bemerkungen

Jede Taste wird durch mindestens ein Zeichen repräsentiert. Ein einzelnes Zeichen auf der Tastatur kann mit dem Zeichen selbst angegeben werden. "A" für das Argument *string* repräsentiert beispielsweise den Buchstaben A. Sie geben mehrere Zeichen an, indem Sie die Zeichen aneinanderhängen. "ABC" für *string* repräsentiert zum Beispiel die Buchstaben A, B und C.

Das Pluszeichen (+), Caret-Zeichen (^), Prozentzeichen (%), die Tilde (**~**) und die Klammern ( ) haben bei der SendKeys-Anweisung eine spezielle Bedeutung. Sie müssen jedes dieser Zeichen in geschweifte Klammern einschliessen ({}), um es verwenden zu können. Für das Pluszeichen geben Sie beispielsweise {+} an. Eckige Klammern ([ ]) haben bei der SendKeys-Anweisung zwar keine spezielle Bedeutung, müssen aber auch in geschweifte Klammern eingeschlossen werden, da sie in anderen Anwendungen eine spezielle Bedeutung haben, insbesondere im Zusammenhang mit dynamischem Datenaustausch (DDE). Die Zeichen für die geschweiften Klammern legen Sie unter Verwendung von {{} und {}} fest.

Für Zeichen, die beim Drücken einer Taste nicht angezeigt werden (z.B. die EINGABETASTE oder TAB-TASTE) und für bestimmte Aktionstasten können Sie die folgenden Codes verwenden:

Taste	Code
RÜCKTASTE	{BACKSPACE}, {BS} oder {BKSP}
PAUSE	{BREAK}

FESTSTELLTASTE	{CAPSLOCK}
ENTF	{DELETE} oder {DEL}
NACH-UNTEN	{DOWN}
ENDE	{END}
EINGABETASTE	{ENTER} oder ~
ESC	{ESC}
HILFE	{HELP}
POS 1	{HOME}
EINFG	{INSERT} oder {INS}
NACH-LINKS	{LEFT}
NUM-FESTSTELL	{NUMLOCK}
BILD-AB	{PGDN}
BILD-AUF	{PGUP}
DRUCK	{PRTSC}
NACH-RECHTS	{RIGHT}
ROLLEN-FESTSTELL	{SCROLLLOCK}
TAB-TASTE	{TAB}
NACH-OBEN	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

Sie können Tastenkombinationen mit der UMSCHALTTASTE, STRG-TASTE oder ALT-TASTE angeben, indem Sie vor dem normalen Tasten-Code einen oder mehrere der folgenden Codes angeben:

Taste	Code
UMSCHALT	+
STRG	^
ALT	%

Wenn UMSCHALT, STRG und ALT gleichzeitig mit anderen Tasten gedrückt werden müssen, schliessen Sie die Codes für die Tasten in Klammern ein. Wenn zum Beispiel die UMSCHALTTASTE gleichzeitig mit den Tasten E und C gedrückt werden soll, geben Sie "+(EC)" an. Wenn die UMSCHALTTASTE zusammen mit E gedrückt werden soll und im Anschluss daran C ohne UMSCHALTTASTE, geben Sie "+EC" an.
Tastenwiederholungen können Sie in der Form {Taste Zahl} angeben. Das Leerzeichen zwischen Taste und Zahl ist dabei zwingend erforderlich. {LEFT 42} wird zum Beispiel als 42-maliges Drücken der NACH-LINKS-TASTE interpretiert, {h 10} als 10-maliges Drücken der Taste H.

Anmerkung

SendKeys kann keine Tastenanschläge an Anwendungen senden, die nicht unter Microsoft Windows oder auf dem Macintosh ausgeführt werden können. SendKeys kann auch die DRUCK-TASTE {PRTSC} an keine Anwendung senden.

Application.SendKeys (Objekt-Methode)

Syntax

Ausdruck.SendKeys*(Keys*, *Wait*)

Ausdruck: Ein optionaler Ausdruck, der ein Application-Objekt zurückgibt.

Keys: Variant erforderlich. Die Taste oder Tastenkombination, die Sie als Zeichenfolge an die Anwendung senden möchten.

Wait: Variant optional. Falls True, wartet Microsoft Excel das Ende der Verarbeitung ab, bevor die Steuerung an das Makro zurückgegeben wird. False oder keine Angabe führt das Makro weiter aus, ohne auf einen Tastenkombination zu warten.

Anmerkungen

Diese Methode schreibt die Tastenkombination in einen Tastenpuffer. In einigen Fällen müssen Sie diese Methode vor der Methode aufrufen, die die Tastenkombination verarbeitet. Wenn Sie z.B. ein Kennwort an ein Dialogfeld senden möchten, müssen Sie zuerst die SendKeys-Methode aufrufen, bevor Sie das Dialogfeld anzeigen.
Das Argument *Keys* kann eine einzelne Taste oder eine Taste kombiniert mit einer oder mehreren der Tasten ALT, STRG oder UMSCHALT sein. Jede Taste wird durch ein oder mehrere Zeichen dargestellt, wie z.B. "a" für das Zeichen a oder "{EINGABE}" für die EINGABETASTE.
Wenn Sie Zeichen angeben möchten, die beim Drücken der entsprechenden Taste nicht angezeigt werden (z. B. EINGABE oder TAB), verwenden Sie die Codes aus der folgenden Tabelle. Jeder Code in der Tabelle steht für eine Taste auf der Tastatur.

Taste	Code
RÜCKTASTE	{RÜCKTASTE} oder {RÜCK}
UNTBR oder BREAK	{UNTBR}

FESTSTELLTASTE	{FESTSTELLTASTE}
CLEAR oder LÖSCHTASTE	{SÄUBERN}
ENTF oder LÖSCH	{ENTF} oder {LÖSCH}
NACH-UNTEN	{UNTEN}
ENDE	{ENDE}
EINGABETASTE (Ziffernblock)	{EINGABE}
EINGABETASTE	~ (Tilde)
RETURNTASTE	{EINGABE}
ESC	{ESCAPE} oder {ESC}
HILFE	{HILFE}
POS1	{POS1}
EINFG	{EINFG}
NACH-LINKS	{LINKS}
NUM	{NUMFT}
BILD-AB	{BILDU}
BILD-AUF	{BILDO}
NACH-RECHTS	{RECHTS}
ROLLEN oder SCROLL	{UNTERBR}
TAB	{TAB}
NACH-OBEN	{OBEN}
F1 bis F15	{F1} bis {F15}

Sie können jede beliebige Tastenkombination mit UMSCHALT, STRG und ALT angeben. Kombinieren Sie eine Taste mit einer oder mehreren anderen Tasten entsprechend der folgenden Tabelle.

Kombinieren mit	**Vorangestelltes Zeichen**
UMSCHALT	+ (Pluszeichen)
STRG	^ (Caret-Zeichen)
ALT	% (Prozentzeichen)

Application.DEExecute (Objekt-Methode)

Syntax

Ausdruck.DDEExecute(*Channel*, *String*)

Ausdruck Optional. Ein Ausdruck, der ein Application-Objekt zurückgibt.

Channel Long erforderlich. Die Kanalnummer, die von der DDEInitiate-Methode zurückgegeben wird.

String String erforderlich. Die Nachricht, die in der empfangenden Anwendung definiert wurde.

Anmerkungen

Die DDEExecute-Methode dient dazu, Befehle an eine andere Anwendung zu senden. Mit Hilfe dieser Methode lassen sich auch Tastenanschläge an die andere Anwendung senden, obwohl für diesen Zweck die SendKeys-Methode geeigneter ist. Das *String*-Argument kann jede einzelne Taste angeben, die mit ALT, STRG oder UMSCHALT sowie jeder Kombination dieser Tasten kombiniert wird. Jede Taste wird durch ein oder mehrere Zeichen repräsentiert, etwa "a" für das Zeichen a, oder "{ENTER}" für die EINGABETASTE.

Zur Angabe von Zeichen, die beim Drücken der entsprechenden Taste nicht angezeigt werden (z. B. EINGABE oder TAB), verwenden Sie die in der Tabelle unten angeführten Codes. Jeder Code in der Tabelle repräsentiert eine Taste auf Ihrer Tastatur.

Taste	Code
RÜCKTASTE	{RÜCKTASTE} oder {RÜCK}
UNTBR	{UNTBR}
FESTSTELLTASTE	{FEST}
CLEAR oder LÖSCHTASTE	{SÄUBERN}
LÖSCH oder ENTF	{ENTF} oder {LÖSCH}
NACH-UNTEN	{UNTEN}
ENDE	{ENDE}
EINGABETASTE (Zehnertastatur)	{EINGABE}
EINGABETASTE	~ (Tilde)
ESC	{ESCAPE} oder {ESC}
HILFE	{HILFE}
POS1	{POS1}
EINFG	{EINFG}
NACH LINKS	{LINKS}
NUM	{NUMFT}
BILD-AB	{BILDU}
BILD-AUF	{BILDO}
RETURNTASTE (Macintosh)	{EINGABE}
NACH RECHTS	{RECHTS}
ROLLEN	{ROLLEN}
TAB	{TAB}
NACH OBEN	{OBEN}
F1 bis F15	{F1} bis {F15}

Sie können zudem Tasten angeben, die mit UMSCHALT und/oder STRG und/oder ALT kombiniert sind. Um eine Taste anzugeben, die mit einer oder mehreren der oben erwähnten Tasten kombiniert wird, verwenden Sie die folgende Tabelle.

Kombination mit	Vorangestelltes Zeichen
UMSCHALT	+ (Pluszeichen)

STRG oder CTRL	^ (Caret-Zeichen)
ALT	% (Prozentzeichen)

Technische Informationen zu SendKeys, Application.SendKeys und DDEExecute

SendKeys (VBA-Anweisung)

SendKeys ist eine Anweisung die zum Sprachumfang von VBA gehört. Die SendKeys-Anweisung ist allerdings keine Core-Anweisung von VBA (d.h. keine Anweisung, die zum Kernumfang der Sprache gehört) wie beispielsweise die Anweisungen "For...Next", "Exit" oder "End". SendKeys ist in der Objektbibliothek von VBA (genau wie Excel besitzt auch VBA eine Objektbibliothek) definiert, und zwar im Modul "Interaction" (Achtung: Interaction ist ein Modul und keine Klasse). Das Interaction-Modul enthält VBA-Anweisungen wie unter anderem "AppActivate", "CreateObject", "MsgBox" und "Shell", also alles Anweisungen, die eine Interaktion mit dem Benutzer oder mit einer Anwendung bezwecken.

Library
VBA
Path
D:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA332.dll
Description
Visual Basic For Applications
Module
Interaction
Syntax
SendKeys(String As String, [Wait])
Samples
SendKeys
Interaction.SendKeys
VBA.Interaction.SendKeys
VBA.SendKeys

Application.SendKeys (Methode des Application-Objektes)

Application.SendKeys gehört zum Sprachumfang von Excel-VBA. Die SendKeys-Methode ist in der Objektbibliothek von Microsoft Excel definiert, und zwar in der Klasse Application.

Library
Excel
Path
D:\Programme\Microsoft Office\Office\EXCEL8.OLB
Description
Microsoft Excel 8.0 Object Library

Class
Application (Global)
Syntax
SendKeys(Keys, [Wait])
Samples
Application.SendKeys
Excel.Application.SendKeys
Excel.SendKeys
Excel.Global.SendKeys
Excel.Global.Application.SendKeys

SendKeys oder Application.SendKeys oder DDEExecute verwenden?

Unterschiede zwischen SendKeys und Application.SendKeys

Die SendKeys-Anweisung von VBA und die SendKeys-Methode des Application-Objektes von Excel sind an sich identisch. Allerdings können Sie SendKeys und Application.SendKeys in einem VBA-Programm nicht ohne weiteres austauschen. Ich habe unzählige Tests mit beiden Anweisungen durchgeführt und insbesondere drei markante Unterschiede festgestellt:

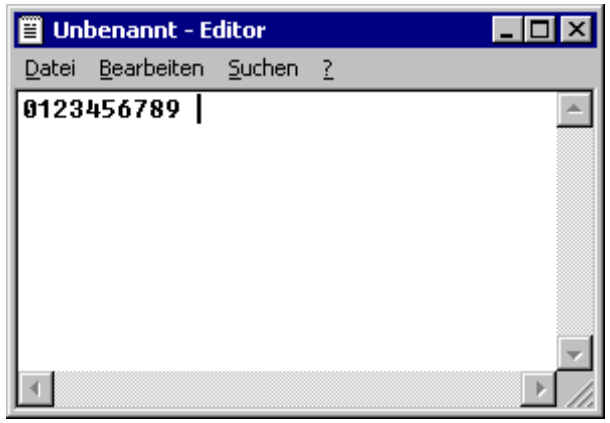
1. Die VBA-Anweisung SendKeys verwendet englische Bezeichnungen für die Tastencodes. Die SendKeys-Methode von Excel dagegen verwendet deutsche Abkürzungen.
2. Enter-Taste ist nicht gleich Enter-Taste. Wie in Punkt 1 erwähnt, werden deutsche oder englische Tastencodes verwendet. Der Code {ENTER} darf nicht mit {EINGABE} gleichgesetzt werden. Der Tastencode {EINGABE} steht für die Eingabetaste der Zehnertastatur. Auch wenn diese Eingabetaste in der Empfänger-Anwendung den genau gleichen Zweck wie die Eingabetaste der Alpha-Tastatur besitzt, verhält sie sich bei Application.SendKeys anders.
3. Ein auf True gestellter Wait-Parameter funktioniert bei Application.SendKeys je nach Windows-Version nur eingeschränkt oder sogar überhaupt nicht. Das Argument "Wait:=True" von SendKeys funktioniert dagegen in jeder Situation korrekt.

Damit man sich diese Unterschiede besser vorstellen kann, nachfolgend ein kleiner Praxis-Test mit ein paar VBA-Beispielen zu den obigen Punkten 1 und 2.

Ein kleiner Praxis-Test

Testvorbereitung

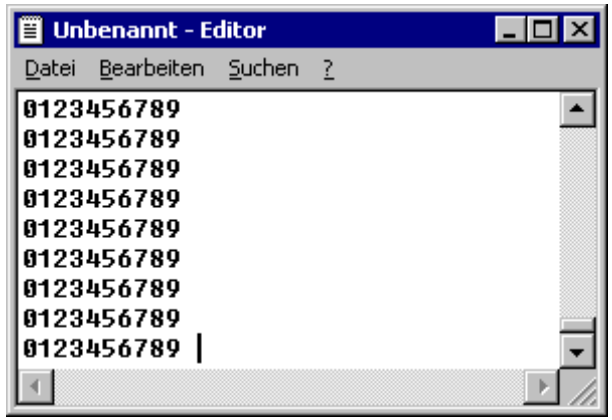
1. Starten Sie zuerst den Windows Editor (Notepad.exe).
2. Geben Sie die Ziffern 0 bis 9 sowie einen Leerschlag ein, d.h. "0123456789 ". Lassen Sie nach der Eingabe den Textcursor einfach stehen, sodass er nach dem Leerzeichen blinkt. Drücken Sie nicht die Enter-Taste. Speichern Sie auch nicht (es ist wichtig, dass im Fenstertitel "Unbenannt" steht).



Schritt 1: Testen von SendKeys
Führen Sie das Makro NotepadSendKeysTest1 aus. Das Makro sendet diese Tastenfolgen insgesamt 10 mal: Menübefehl Bearbeiten/Alles markieren, Menübefehl Bearbeiten/Kopieren, Tasten Pfeil nach unten und Eingabe, Menübefehl Bearbeiten/Einfügen.

```
Sub NotepadSendKeysTest1()
  Dim intCounter As Integer
  AppActivate "Unbenannt", True
  For intCounter = 1 To 10
    SendKeys "%(BM)", True
    SendKeys "%(BK)", True
    SendKeys "{DOWN}{ENTER}", True
    SendKeys "%(BE)", True
  Next intCounter
End Sub
```

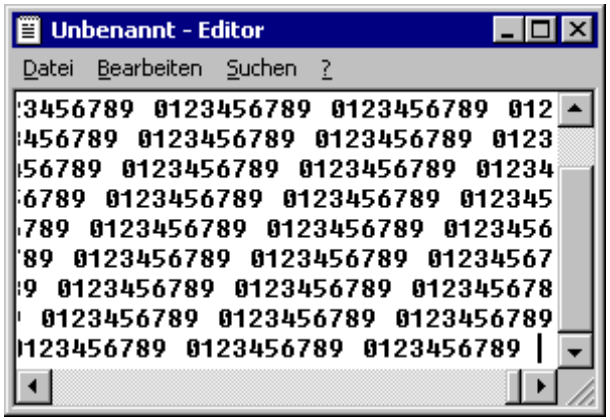
Sobald die Makroausführung beendet ist, sehen Sie im Editor den eingegebenen Text "0123456789 ", der mehrmals untereinander aufgelistet ist:



Schritt 2: Testen von Application.SendKeys
Löschen Sie den Inhalt des Notizblocks (Menübefehl Datei/Neu) und geben Sie den Text "0123456789 " erneut ein. Führen Sie jetzt das Makro NotepadSendKeysTest2 aus. Dieses Makro macht genau das gleiche wie das erste Makro, mit dem einzigen Unterschied, dass Application.SendKeys anstatt SendKeys benutzt wird.

```
Sub NotepadSendKeysTest2()
  Dim intCounter As Integer
  AppActivate "Unbenannt", True
  For intCounter = 1 To 10
    Application.SendKeys "%(BM)", True
    Application.SendKeys "%(BK)", True
    Application.SendKeys "{DOWN}{ENTER}", True
    Application.SendKeys "%(BE)", True
  Next intCounter
End Sub
```

Sobald die Makroausführung fertig ist, sehen Sie im Editor den eingegebenen Text "0123456789 ", der mehrmals nebeneinander steht:

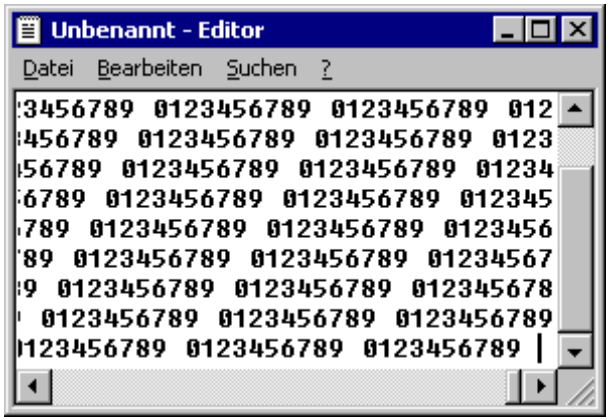


Das zweite Makro führt zu einem anderen Ergebnis als das erste Makro. Und dies, obwohl lediglich Application.SendKeys anstelle von SendKeys benutzt wird.
Der Grund für das abweichende Ergebnis liegt an den verwendeten Tastencodes. Die SendKeys-Methode verwendet im Gegensatz zur SendKeys-Anweisung deutsche Abkürzungen. Mit den beiden Codes {DOWN} und {ENTER} kann sie nichts anfangen. Unschön finde ich, dass diese Tastencodes einfach übergangen werden, das heisst ohne eine Fehler- oder Hinweismeldung nicht verarbeitet werden. Zumindest ist jetzt klar, warum der Text nebeneinander und nicht untereinander erscheint: Das {ENTER} wurde nicht verarbeitet und folglich keine Zeilenschaltung eingefügt.

Schritt 3: Testen von Application.SendKeys (zweiter Versuch)
Löschen Sie wieder den Inhalt des Editors und erfassen den Text "0123456789 ". Starten Sie nun das Makro NotepadSendKeysTest3. Dieses Makro entspricht dem zweiten Makro. Lediglich die Tastencodes {DOWN} und {ENTER} wurden mit {UNTEN} bzw. {EINGABE} übersetzt.

```
Sub NotepadSendKeysTest3()
  Dim intCounter As Integer
  AppActivate "Unbenannt", True
  For intCounter = 1 To 10
    Application.SendKeys "%(BM)", True
    Application.SendKeys "%(BK)", True
    Application.SendKeys "{UNTEN}{EINGABE}", True
    Application.SendKeys "%(BE)", True
  Next intCounter
End Sub
```

Nach Ausführung des Makros sehen Sie dieses Bild im Editor:



Im Notizblock steht der Text wieder nebeneinander anstatt untereinander - obwohl deutschsprachige Tastencodes verwendet wurden. Wo liegt der Fehler? Nun, der Fehler liegt an der Übersetzung des englischen Tastencodes {ENTER}. Der Code {ENTER} steht sowohl für die Eingabetaste der alphanummerischen Tastatur als auch für die Eingabetaste der Zehnertastatur. Der deutsche Code {EINGABE} dagegen bezeichnet nur die Eingabetaste der Zehnertastatur. Für die Eingabetaste der Alpha-Tastatur muss das "~"-Zeichen (Tilde) benutzt werden, und zwar ohne das geschweifte Klammernpaar "{}".

Schritt 4: Testen von Application.SendKeys (dritter Versuch)
Das korrekte Makro mit Application.SendKeys muss folgedessen so aussehen:

```
Sub NotepadSendKeysTest4()
  Dim intCounter As Integer
  AppActivate "Unbenannt", True
  For intCounter = 1 To 10
    Application.SendKeys "%(BM)", True
    Application.SendKeys "%(BK)", True
    Application.SendKeys "{UNTEN}~", True
    Application.SendKeys "%(BE)", True
  Next intCounter
End Sub
```

Wenn Sie dieses Makro ausführen, erhalten Sie das gleiche Ergebnis wie beim ersten Makro "NotepadSendKeysTest1", nämlich den untereinander aufgelisteten Text "0123456789 ".

Zusammenfassung
Wie man erkennen kann, ist das Austauschen von SendKeys durch Application.SendKeys nicht ganz so einfach wie es zuerst aussieht.

Unterschiedliches Verhalten je nach Windows-Version

Das Verhalten der beiden Befehle SendKeys und Application.SendKeys ist nicht immer gleich, da der - bei beiden Anweisungen existierende - Wait-Parameter je nach Betriebssystem (Windows-Version) unterschiedlich reagiert.

Windows-Version	Empfohlene Anweisung	Wait-Parameter	Beispiel (Neuberechnung auslösen)
Windows 95	SendKeys	True	SendKeys "%^{F9}", True
Windows 98	SendKeys	True	SendKeys "%^{F9}", True
Windows ME	SendKeys	True	SendKeys "%^{F9}", True
Windows NT	Application.SendKeys	False	Application.SendKeys "%^{F9}":DoEvents
Windows 2000	Application.SendKeys	False	Application.SendKeys "%^{F9}":DoEvents
Windows XP	Application.SendKeys	False	Application.SendKeys "%^{F9}":DoEvents

▲

Verwendung von DoEvents

DoEvents übergibt wie allgemein bekannt ist die Ablaufsteuerung an das Betriebssystem. Das Programm erhält erst dann die Steuerung wieder zurück, wenn das Betriebssystem alle wartenden Ereignisse verarbeitet hat und alle wartenden Tastenanschläge in der SendKeys-Warteschlange abgearbeitet sind.
In der VBA-Hilfe steht, dass der Wait-Parameter der SendKeys-Methode - wenn auf True gesetzt - die vollständige Verarbeitung der Tastenanschläge abwartet (d.h. Application.SendKeys "...", True). Das kann man vergessen. Ich hab's in Excel 97 unter Windows NT 4 getestet: Funktioniert nicht. Daher die obige Lösung mit dem DoEvents. Bei "Wait:=True" wird die Steuerung erst wieder an das Makro übergeben, wenn Excel mit der Verarbeitung fertig ist. Das klappt jedoch bei präemptiven Multitasking-System wie Windows NT 4, Windows 2000 und Windows XP nicht, da nicht der ausgeführte Task den Zeitpunkt der Rückgabe bestimmen kann. Das entscheided bei diesen Betriebssystemen ein spezieller Systemprozess (der sogenannte "Scheduler").

▲

Was bewirkt der Wait-Parameter von SendKeys?

Mit dem Wait-Parameter wird gesteuert, wann der Programmcode weiter ausgeführt wird:

- Bei "Wait = False" oder Weglassen des Parameters wartet das VBA-Programm nicht ab, bis sämtliche gesendeten Tastenfolgen verarbeitet sind und fährt mit der Programmausführung fort.
- Bei "Wait = True" wird das VBA-Programm erst weiter ausgeführt, wenn alle gesendeten Tastenfolgen verarbeitet sind.

Wie im vorangegangenen Abschnitt über die Unterschiede bereits kurz erwähnt wurde, verhält sich der auf True eingestellte Wait-Parameter der beiden Anweisungen je nach Windows-Version nicht immer gleich. Während mit "SendKeys "<Taste>", True" die Verarbeitung der gesendeten Tasten korrekt abgewartet wird, wird bei "Application.SendKeys "<Taste>", True" das VBA-Programm zu früh fortgeführt bzw. keine Pause eingelegt.

Ein praktisches Beispiel

Anhand eines kleines VBA-Beispiels lässt sich Zweck des Wait-Parameters gut erkennen. Starten Sie zuerst den Windows Editor (Notepad) und führen dann dieses Makro im VBA-Editor aus:

```
Sub NotepadTest()
  AppActivate "Unbenannt"
  SendKeys "Hallo", True
End Sub
```

Als Resultat sollten Sie das geöffnete Notepad mit dem Wort "Hallo" auf dem Bildschirm sehen. Im Programm wurde der Wait-Parameter auf True gestellt. Das Makro wurde somit erst weiter ausgeführt, nachdem alle Tastenanschläge (d.h. alle fünf Buchstaben des Wortes "Hallo") verarbeitet wurden. Da nach der SendKeys-Anweisung das Makro zu Ende war, also keine weiteren Befehle ausser dem "End Sub" (der das Makroende kennzeichnet), blieb der Fokus beim Notepad-Anwendungsfenster.
Nun wird Wait auf False gesetzt und das Makro nochmals ausgeführt. Löschen Sie aber zuerst das "Hallo" aus dem Notepad, sodass Sie wieder eine leere Seite sehen.

```
Sub NotepadTest()
  AppActivate "Unbenannt"
  SendKeys "Hallo", False
End Sub
```

Nach diesem Makro sehen Sie anstelle des Notepads den VBA-Editor vor sich. Wenn Sie zum Notepad wechseln, werden Sie lediglich den Buchstaben "H" des Wortes "Hallo" im Textdokument vorfinden. Wegen dem auf False gestellten Wait-Parameter hat das Makro nicht gewartet, bis alle Buchstaben an das Notepad übermittelt wurden. Doch wo sind die restlichen vier Buchstaben "allo" geblieben? Nun, da nach Senden des ersten Buchstabens das Makro bereits wieder die Kontrolle übernahm, sprich den Fensterfokus zurückerhielt, hat das Makro die restlichen Buchstaben quasi sich selbst geschickt! Das gesuchte "allo" wurde in ein Fenster des VBA-Editors eingefüllt, und zwar genau an der Stelle, wo sich der Textcursor befand, als das Makro gestartet wurde.

▲

Wann muss der Wait-Parameter auf True gesetzt werden?

Ob die Verarbeitung der gesendeten Tastenfolgen abgewartet werden muss oder nicht, was mit dem Wait-Parameter von SendKeys gesteuert wird, lässt sich anhand von Situationen aus der Praxis gut erkennen. Diese Situationen werden im nächsten Kapitel So sendet man Tastenanschläge anhand verschiedener Beispiele erläutert.

▲

So sendet man Tastenanschläge

Tastenanschläge zur Steuerung einer Dialogmaske an Excel senden

Aktion	VBA-Anweisung	SendKeys	Wait	Beispiel
Dialog wird mittels Dialogs-Auflistung geöffnet	Dialogs(...).Show	Vor Dialog-Aufruf	False	`SendKeys "%z200"` `Application.Dialogs(xlDialogCalculation).Show`
Dialog wird mit der speziellen, dafür zuständigen Methode geöffnet	Application.GetOpenFilename Application.GetSaveAsFilename Application.FindFile Worksheet.ShowDataForm Worksheet.CheckSpelling Range.FunctionWizard	Vor Dialog-Aufruf	False	`SendKeys "%s" & strBegriff & "%t"` `Worksheet.ShowDataForm`

Dialog wird durch Simulation eines Klicks auf eine Symbolleisten-Schaltfläche geöffnet	CommandBars.FindControl().Execute	Vor Dialog-Aufruf	False	`SendKeys "%n" & strID & "{TAB}"`
				`Application.CommandBars.FindControl _`
				`  (Id:=860).Execute`

Dialog wird durch eine sonstige Methode geöffnet, die ihrerseits den Dialog einblendet	Workbooks.Open mit einer kennwortgeschützten Mappe	Vor Dialog-Aufruf	False	`SendKeys "Heute"` `Workbooks.Open "D:\Kennwort.xls"`
Dialog wird mit einer Funktionstaste oder Tastaturabkürzung geöffnet	SendKeys	-	True	`SendKeys "{F5}", True`
Dialog wird über einen Menübefehl geöffnet	SendKeys	-	True	`SendKeys "%di", True`

Sonstige Tastenanschläge an Excel senden

Aktion	Wait-Parameter	Bemerkung	Beispiel

Makro wird in der VBE gestartet	Immer True, egal ob weitere Codezeilen nach SendKeys folgen oder nicht	Excel muss immer zuerst mit AppActivate aktiviert werden
		Wait muss immer True sein, egal ob nach SendKeys weitere Befehle folgen oder nicht
			`Sub OpenNamesDialog()`
			`  AppActivate "Microsoft Excel"`
			`  SendKeys "^{F3}", True`
			`  'Evtl. weiterer Code...`
			`End Sub`

Makro wird in Excel gestartet	False oder True, wenn Makro nach SendKeys beendet ist	-	`Sub OpenNamesDialog()`
			`  SendKeys "^{F3}"`
			`End Sub`

Makro wird in Excel gestartet	False, wenn die Codezeilen nach SendKeys ausgeführt werden müssen	-	`Sub OpenNamesDialog()`
			`  SendKeys "^{F3}"`
			`  Application.StatusBar = "Name wählen"`
			`End Sub`

Makro wird in Excel gestartet	False, wenn die Codezeilen nach SendKeys ausgeführt werden dürfen	-	`Sub OpenNamesDialog()`
			`  SendKeys "^{F3}"`
			`  intZahl = intZahl + 1`
			`End Sub`

Makro wird in Excel gestartet	True, wenn die Codezeilen nach SendKeys nicht ausgeführt werden dürfen	-	`Sub OpenNamesDialog()`
			`  SendKeys "^{F3}", True`
			`  Application.StatusBar = "Name ok"`
			`End Sub`

Tastenfolgen an eine andere Anwendung senden

Aktion	Wait	Bemerkung	Beispiel

Makro wird in der VBE mit F5 gestartet	True	Die Anwendung muss immer zuerst mit AppActivate aktiviert werden	`Sub OpenNamesDialog()`
			`  AppActivate "Explorer"`
			`  SendKeys "a", True`
			`End Sub`

Einschränkungen und Grenzen von SendKeys

Nicht unterstützte Tasten

SendKeys besitzt ein paar Einschränkungen. Nicht alle auf einer PC- oder Mac-Tastatur angeordneten Tasten können mit SendKeys aufgerufen werden, sprich lassen sich an eine Anwendung senden.
Hier eine Liste der nicht unterstützten Tasten mit den Tastencodes:

Taste	Code
FESTSTELLTASTE	{CAPSLOCK}
HILFE	{HELP}
NUM-FESTSTELL	{NUMLOCK}
DRUCK	{PRTSC}
ROLLEN-FESTSTELL	{SCROLLLOCK}

Zudem können die Spezialtasten einer Tastatur, wie unter anderem
- SysRq (System Request),
- Pause und
- die beiden Windows-Tasten (links und rechts der Leertaste),

nicht mit SendKeys angesprochen werden.

Umgehungslösung für nicht unterstützte Tasten

Mit dem nachfolgenden VBA-Programmcode kann beispielsweise die CapsLock-Taste umgeschaltet werden:

```
Private Const VER_PLATFORM_WIN32_NT = 2
Private Const VER_PLATFORM_WIN32_WINDOWS = 1
Private Const VK_CAPITAL = &H14
Private Const KEYEVENTF_EXTENDEDKEY = &H1
Private Const KEYEVENTF_KEYUP = &H2
Private Type OSVERSIONINFO
  dwOSVersionInfoSize As Long
  dwMajorVersion As Long
  dwMinorVersion As Long
  dwBuildNumber As Long
  dwPlatformId As Long
  szCSDVersion As String * 128
End Type
```

```
Private Declare Function GetVersionEx Lib "kernel32" Alias "GetVersionExA" _
  (lpVersionInformation As OSVERSIONINFO) As Long
Private Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal bScan _
  As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
Private Declare Function GetKeyboardState Lib "user32" (pbKeyState As Byte) As Long
Private Declare Function SetKeyboardState Lib "user32" (lppbKeyState As Byte) As Long
Public Sub ToggleCapsLock(TurnOn As Boolean)
  'To turn Caps Lock on, set TurnOn to True
  'To turn Caps Lock off, set TurnOn to False

  Dim bytKeys(255) As Byte
  Dim bCapsLockOn As Boolean
  Dim typOS As OSVERSIONINFO

  'Get state of the 256 virtual keys
  GetKeyboardState bytKeys(0)

  bCapsLockOn = bytKeys(VK_CAPITAL)

  If bCapsLockOn <> TurnOn Then   'If current state <> requested state
    If typOS.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS Then    '=== Win95/98
      bytKeys(VK_CAPITAL) = 1
      SetKeyboardState bytKeys(0)
    Else   '=== WinNT/2000
      'Simulate Key Press
      keybd_event VK_CAPITAL, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
      'Simulate Key Release
      keybd_event VK_CAPITAL, &H45, KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0
    End If
  End If
End Sub
```

```
Public Sub PressAstericsKey()
  Dim bytKeys(255) As Byte
  Dim typOS As OSVERSIONINFO

  If typOS.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS Then    '=== Win95/98
    bytKeys(VK_ASTERICS) = 1
    SetKeyboardState bytKeys(0)
  Else    '=== WinNT/2000
    keybd_event VK_ASTERICS, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
    keybd_event VK_ASTERICS, &H45, KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0
  End If
End Sub
```

Bestimmen der Ausführungsumgebung (Excel oder VBE)

Wie in einem Kapitel weiter oben bereits erwähnt wurde, kann ein Makro, welches mittels SendKeys Tastenfolgen an Microsoft Excel sendet, nicht so ohne weiteres im VBA-Editor bzw. aus dem VBA-Editor heraus gestartet werden. Das funktioniert aus zwei Gründen nicht:

1. Das Anwendungsfenster von Excel muss zuerst aktiviert werden, da ansonsten die Tastenfolgen von der VBE empfangen werden und nicht von Excel.
2. Nach Beendigung der Makroausführung wird automatisch zur VBE zurückgekehrt. Das darf erst erfolgen, nachdem alle Tastenfolgen von Excel verarbeitet wurden. Anderenfalls kann es passieren, dass zu früh zum VBA-Editor gewechselt wird und somit einzelne noch anstehende Tastenfolgen in der VBE verarbeitet werden.

Die beiden obigen Punkte bedeuten nichts anderes, als dass die Anweisung AppActivate und die Parameter-Einstellung "Wait:=True" verwendet werden müssen, wenn das Makro in der VBE gestartet wird. Damit ein und dasselbe Makro sowohl aus der VBE als auch aus Excel heraus gleichermassen funktioniert, bedient man sich am besten einer globalen Variablen, mit welcher man den Makroablauf komfortabel steuern kann.

VBA-Programmcode

Hier der entsprechende VBA-Code zum Aktivieren des Office-Programmes:

```
Public RunInVBE as Boolean
Sub OpenNamesDialog()
  If RunInVBE = True Then
    AppActivate "Microsoft Excel"
  End If
  SendKeys "^{F3}", RunInVBE
End Sub
```

Die globale Variable RunInVBE steuert den einzuschlagenden Makroablauf. Wenn das Makro in der VBE ausgeführt wird, muss RunInVBE auf True gesetzt werden. Dadurch weiss das Makro, dass vor dem SendKeys die Anwendung Excel aktiviert (Codezeile mit der AppActivate-Anweisung) und auf die vollständige Abarbeitung der Tastenfolgen gewartet werden muss (der Wait-Parameter von SendKeys steht auf RunInVBE, wobei RunInVBE den Wert True besitzt).

Empfänger-Anwendung starten und aktivieren

Anwendung starten mit "Shell"

Shell startet eine Anwendung asynchron und gibt die Task-ID der ausgeführten Anwendung zurück. Asynchron bedeutet, dass das VBA-Programm weiterläuft, während Windows die Anwendung startet. Es kommt häufig vor, dass in einer VBA-Codezeile eine Anwendung mittels Shell gestartet wird und unmittelbar danach, in der nächsten Codezeile, auf eben diese Anwendung zugegriffen werden soll (z.B. mit AppActivate, SendKeys oder dergleichen). Wenn die zu startende Anwendung sehr klein ist, oder die Anwendungsdatei (exe) auf einer lokalen Festplatte liegt, oder wenn sich die Anwendung noch im Arbeitsspeicher befindet, so startet sie sofort und ist folgedessen augenblicklich bereit, wenn die Codezeile nach der Shell-Funktion ausgeführt wird. Benötigt jedoch die Anwendung zum vollständigen Aufstarten etwas mehr Zeit, so ist sie noch nicht bereit, wenn die Codezeile nach der Shell-Funktion ausgeführt wird.

Anwendung aktivieren mit "AppActivate"

Die von Shell gelieferte Task-ID ist nützlich, wenn die Anwendung mit AppActivate aktiviert werden soll. Der AppActivate-Anweisung kann anstelle des Fenstertitels der Anwendung nämlich die Task-ID mitgegeben werden. Leider funktioniert die Aktivierung mittels Task-ID in der Praxis je nach Empfänger-Anwendung nicht in jeder Situation und nicht bei jeder Anwendung.
Die folgende Tabelle zeigt das Verhalten verschiedener Windows-Programme:

Anwendung	Aktivierung mittels Task-ID	Codebeispiel
Calculator (Rechner)	Funktioniert korrekt	`TaskID = Shell("calc.exe", vbNormalFocus)` `AppActivate TaskID`
Paint/Paint Brush	Funktioniert korrekt	`TaskID = Shell("mspaint.exe", vbNormalFocus)` `AppActivate TaskID`
Registry Editor	Funktioniert korrekt bei der ersten Aktivierung Funktioniert nicht korrekt bei allen weiteren Aktivierungen	`TaskID = Shell("regedit.exe", vbNormalFocus)` `AppActivate TaskID`
Explorer	Funktioniert nicht immer korrekt [* siehe Hinweis]	`TaskID = Shell("explorer.exe", vbNormalFocus)` `AppActivate TaskID`
Notepad (Editor)	Funktioniert korrekt	`TaskID = Shell("notepad.exe", vbNormalFocus)` `AppActivate TaskID`
WordPad	Funktioniert nicht korrekt	`TaskID = Shell("write.exe", vbNormalFocus)` `AppActivate TaskID`
CD Player	Funktioniert korrekt bei der ersten Aktivierung Funktioniert nicht korrekt bei allen weiteren Aktivierungen	`TaskID = Shell("cdplayer.exe", vbNormalFocus)` `AppActivate TaskID`

Clock (Uhr)	Funktioniert korrekt	`TaskID = Shell("clock.exe", vbNormalFocus)` `AppActivate TaskID`
CharMap (Zeichentabelle)	Funktioniert korrekt	`TaskID = Shell("charmap.exe", vbNormalFocus)` `AppActivate TaskID`
Event Viewer (Ereignisanzeige)	Funktioniert korrekt	`TaskID = Shell("eventvwr.exe", vbNormalFocus)` `AppActivate TaskID`
User Manager (Benutzerverwaltung)	Funktioniert korrekt	`TaskID = Shell("musrmgr.exe", vbNormalFocus)` `AppActivate TaskID`
Task Manager	Funktioniert korrekt bei der ersten Aktivierung Funktioniert nicht korrekt bei allen weiteren Aktivierungen	`TaskID = Shell("taskmgr.exe", vbNormalFocus)` `AppActivate TaskID`

* Hinweis zum Starten des Windows Explorers
Beschreibung folgt...

Anwendung starten und aktivieren mit "ActivateMicrosoftApp"

Muss mit Excel-VBA eine andere Office-Anwendung gestartet werden, bedient man sich häufig der Shell-Funktion von VBA. Damit die Anwendung jedoch gestartet werden kann, muss der Shell-Funktion der genaue Pfad der ausführbaren Datei (.exe) mitgegeben werden. Solange die zu startende Anwendung aus der gleichen Office-Version wie Excel stammt, könnte man behelfsmässig den Pfad von Excel verwenden (d.h. Application.Path) und voraussetzen, dass die exe-Datei der anderen Anwendung im gleichen Verzeichnis wie Excel.exe steht. Sobald es sich um eine Anwendung einer anderen Office-Version handelt, weicht der Programmpfad sehr wahrscheinlich ab. Damit nicht ein (eher schwierig zu programmierender) Zugriff auf die Windows Registry vorgenommen werden muss, um den Pfad der gesuchten Anwendung herauszufinden, stellt Excel eine spezielle Methode zum Starten von Office-Anwendungen bereit: Die ActivateMicrosoftApp-Methode.
Der wohl grösste Vorteil von ActivateMicrosoftApp ist - verglichen mit der Shell-Funktion - das synchrone Starten der angegebenen Anwendung. Das VBA-Programm läuft somit erst dann weiter, wenn die andere Anwendung vollständig aufgestartet und bereit ist.
VBA-Syntax

```
Application.ActivateMicrosoftApp(Index As XlMSApplication)
```

Konstanten für **XlMSApplication**

- xlMicrosoftAccess (Wert 4)
- xlMicrosoftFoxPro (5)
- xlMicrosoftMail (3)
- xlMicrosoftPowerPoint (2)
- xlMicrosoftProject (6)
- xlMicrosoftSchedulePlus (7)
- xlMicrosoftWord (1)

Wenn die als Argument angegebene Office-Anwendung nicht auf der Arbeitsstation installiert ist, erscheint der Laufzeitfehler 1004 mit einem entsprechenden Fehlertext (Beispiel FoxPro for Windows):

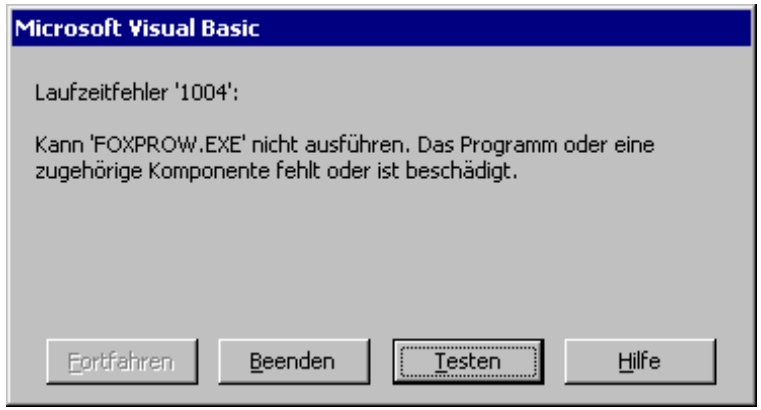


Abbildung: Laufzeitfehler bei nicht auffindbarer Programmdatei foxprow.exe

Das Geheimnis von ActivateMicrosoftApp

Übrigens besitzt ActivateMicrosoftApp ein gut verstecktes und höchst interessantes Geheimnis: Die beiden boolschen Werte True (Wahr bzw. -1) und False (Falsch bzw. 0) können nebst den XlMSApplication-Konstanten ebenfalls als Argument eingesetzt werden.
Mit der Programmzeile

```
Application.ActivateMicrosoftApp False
```

kann der Taschenrechner von Windows (calc.exe) gestartet werden.
Mit der Codezeile

```
Application.ActivateMicrosoftApp True
```

lässt sich das Kartenspiel Solitär von Windows (sol.exe) ausführen.

Handelt es sich um einen Zufall, dass für True bzw. False genau die beiden Programme Solitär bzw. Rechner gewählt wurden? Oder steht vielleicht etwa der Rechner für das "falsche" Programm (symbolisiert die Arbeit), während das Kartenspiel Solitär die "wahre" oder "richtige" Anwendung darstellt (symbolisiert das Spielen/den Spass)?
Der Parameterwert True bzw. -1 ist VBA-intern fix mit der ausführbaren Datei sol.exe des Kartenspiels Solitär verknüpft. Wenn sich weder im Windows- oder Windows System-Verzeichnis noch in den mit der Systemumgebungsvariable "Path" definierten Verzeichnissen eine Datei namens sol.exe befindet, tritt der Laufzeitfehler 1004 auf:

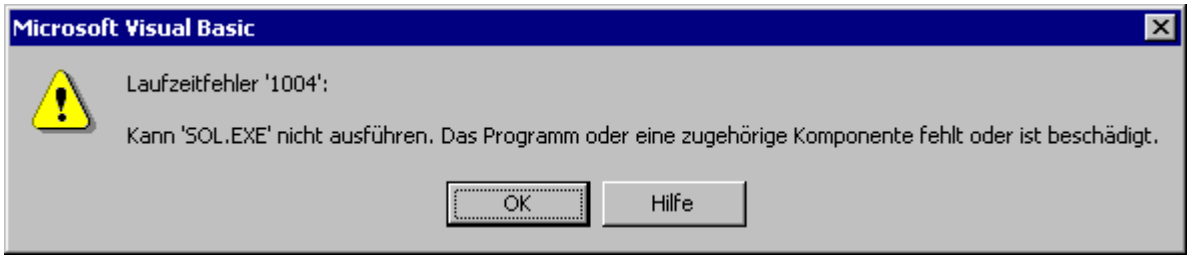


Abbildung: Laufzeitfehler bei nicht auffindbarer Programmdatei sol.exe
Das gleiche gilt für den False bzw. 0; dieser Wert ist fest mit der Datei 'calc.exe' verbunden.

XL: Custom Button Images Start Solitaire or Calculator
http://support.microsoft.com/default.aspx?scid=kb;en-us;112374

Eine alltägliche Aufgabe: Anwendung starten, Fenster aktivieren, Tasten senden

Diese versteckte Möglichkeit zum Starten von Solitär hat mich auf eine Idee gebracht, wie man ein in der Praxis häufig anzutreffende schwierige Aufgabe sehr einfach lösen könnte. Die Aufgabe, die schwierig zu lösen ist, definiert sich wie folgt:
- Eine Nicht-Office-Anwendung soll von einem Server-Laufwerk gestartet werden
- Nach dem Start ist die Anwendung zu aktivieren, damit Tastenfolgen gesendet werden können

Sie fragen sich jetzt vielleicht, wo denn das Problem liegt. Schliesslich muss man lediglich mit Shell die Anwendung starten und dann mit AppActivate den Fokus setzen. Der VBA-Code würde dann lediglich zwei Zeilen umfassen und ungefähr so aussehen:

```
Sub AnwendungAktivieren()
  x = Shell("P:\Programme\FeuerwehrPlaner.exe", vbNormalFocus)
  AppActivate "Feuerwehr-Planer 1.0 - Administrator"
End Sub
```

Grundsätzlich haben Sie recht, denn genau so würden viele Programmierer diese Aufgabe lösen. Aber seien Sie auf der Hut! In der Praxis sieht das Ganze nämlich gar nicht mehr so einfach aus. Bis der Programmcode - für diese eigentlich simple Aufgabe - fehlerfrei und stabil läuft, müssen Sie noch ein paar ganz interessante Probleme lösen:
- Woher kennen Sie den bei Shell erforderlichen Pfad zur Anwendung, sprich zur Datei FeuerwehrPlaner.exe?
- Was machen Sie, wenn der Programmstart misslingt (fehlende Berechtigung, Pfad falsch usw.)?
- Nach Shell wird das Makro weiter ausgeführt, ohne den vollständigen Programmstart abzuwarten. AppActivate wird daher höchstwahrscheinlich fehlschlagen. Was können Sie dagegen tun?
- Bei AppActivate ist im Anwendungstitel die Version der Anwendung angegeben. Was tun Sie, wenn die Versionsnummer ändert?
- Bei AppActivate ist ebenfalls im Titel der angemeldete Benutzer mit "Administrator" angegeben. Was machen Sie, damit die Aktivierung bei allen Benutzern funktioniert?

ActivateMicrosoftApp für eigene Zwecke missbrauchen

Wie unter "Das Geheimnis von ActivateMicrosoftApp" zu lesen ist, können anstelle eines offiziellen Konstantenwertes auch die Werte True oder False als Argument verwendet werden.

- Nach Shell wird das Makro weiter ausgeführt, ohne den vollständigen Programmstart abzuwarten. AppActivate wird daher höchstwahrscheinlich fehlschlagen. Was

```
Application.ActivateMicrosoftApp True
AppActivate "Feuerwehr"
```

Aktivieren einer bereits laufenden Empfänger-Anwendung

Wenn die Empfänger-Anwendung bereits läuft, verfügt man über keine Task-ID (da die Shell-Funktion nicht verwendet wurde, welche die Task-ID liefern würde). In diesem Fall muss man die Anwendung aktiviert werden, indem man AppActivate einsetzt und als Argument den Titel des zu aktivierenden Anwendungsfensters übergibt.

VBA-Syntax

```
AppActivate Title [, Wait]
```

Die Syntax der AppActivate-Anweisung verwendet die folgenden benannten Argumente:

Argument	Beschreibung
Title	Erforderlich. Ein Zeichenfolgenausdruck, der den Titel in der Titelleiste des zu aktivierenden Anwendungsfensters angibt. Die von der Shell-Funktion zurückgegebene Task-ID kann anstelle von *title* verwendet werden, um eine Anwendung zu aktivieren.
Wait	Optional. Ein Wert vom Typ Boolean, der angibt, ob die aufrufende Anwendung den Fokus hat, bevor sie eine andere Anwendung aktiviert. Beim Wert False (Voreinstellung) wird die angegebene Anwendung mit sofortiger Wirkung aktiviert, auch wenn die aufrufende Anwendung nicht den Fokus hat. Beim Wert True wartet die aufrufende Anwendung, bis sie den Fokus erhält, und aktiviert dann die angegebene Anwendung.

Bemerkungen

Die AppActivate-Anweisung setzt den Fokus auf die angegebene Anwendung oder das angegebene Fenster, hat aber keinen Einfluss darauf, ob diese maximiert oder minimiert dargestellt werden. Das aktivierte Anwendungsfenster verliert den Fokus, wenn der Benutzer das Fenster schliesst oder den Fokus auf ein anderes Fenster setzt. Verwenden Sie die Shell-Funktion, wenn Sie eine Anwendung starten und den Fensterstil festlegen möchten.

Welche Anwendung aktiviert werden soll, wird bestimmt, indem *Title* mit der Zeichenfolge des Titels jeder einzelnen momentan ausgeführten Anwendung verglichen wird. Wenn es keine genaue Übereinstimmung gibt, wird eine beliebige Anwendung, deren Zeichenfolge mit *Title* beginnt, aktiviert. Wenn mehr als eine Instanz dieser Anwendung mit *Title* bezeichnet ist, wird willkürlich eine dieser Instanzen aktiviert.

Der Wait-Parameter von AppActivate

Die AppActivate-Anweisung wird oft zusammen oder genauer gesagt nach der Shell-Funktion verwendet.

```
TaskID = Shell("Calc.exe", 1)
AppActivate TaskID, True
```

Programm-Stabilität und -Fehlerfreiheit

Makro-Unterbrechung verhindern

Wenn Sie mit SendKeys oder Application.SendKeys arbeiten, ist es wichtig, dass die Ausführung des Programmcodes möglichst nicht unterbrochen wird bzw. nicht unterbrochen werden kann. Insbesondere beim Senden von Tastenfolgen an eine andere Anwendung kann es - untertrieben ausgedrückt - sehr unangenehm sein, wenn das VBA-Programm aufgrund eines Fehlers oder einer Intervention durch den Benutzer irgendwo abbricht. Der Programmabbruch infolge eines Laufzeitfehlers kann man recht einfach abfangen, indem im Code eine Fehlerbehandlungsroutine hinzugefügt wird. Innerhalb des Error Handlers kann fehlerverursachende Situation bereinigt werden, sodass das Programm fortgesetzt werden kann.

Weil die zu steuernde Anwendung zum Empfangen von Tastenfolgen aktiviert sein muss, lässt sich die Ausführung eines unterbrochenen VBA-Makros nicht so einfach fortsetzen. Dies, weil man sich bei der weiteren Ausführung des Programmcodes im VBA-Editor befindet, und folgedessen nicht das Fenster der Empfänger-Anwendung aktiv ist sondern das Hauptfenster des VBA-Editors. Die für die korrekte Code-Fortführung benötigte Anweisung AppActivate muss somit zuerst ausgeführt werden.

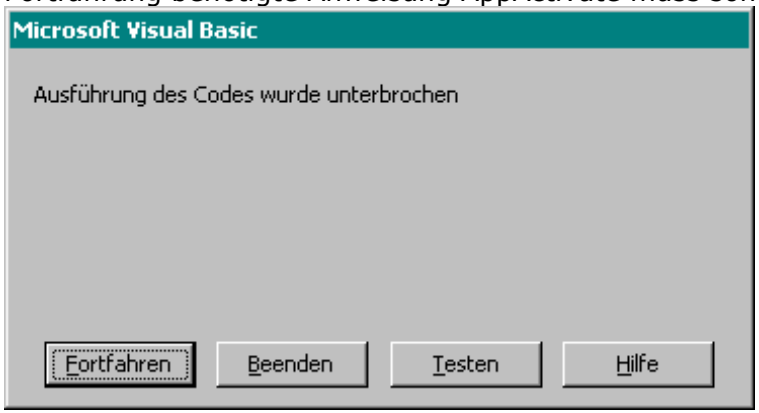


Abbildung: Hinweismeldung "Ausführung des Codes wurde unterbrochen"

▲

Makro-Abbruch verhindern

Beschreibung folgt...

Anwendung kann nicht mit AppActivate aktiviert werden

-> Laufzeitfehler Nr. 5

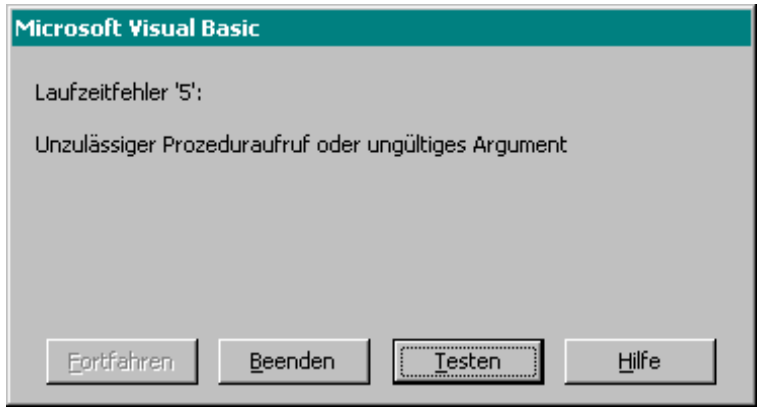


Abbildung: Laufzeitfehler 5

▲

Unterbrochenes Makro fortsetzen

Beschreibung folgt...

▲

Checkliste für die Programmierung

Diese Checkliste hilft Ihnen bei der Programmierung von Anwendungen, mit denen Sie Tastenfolgen senden. Überprüfen Sie anhand der Fragen, ob allenfalls wichtige Punkte bei der Realisierung ungenügend beachtet oder sogar vergessen wurden.

Nr.	Themenbereich	Frage	Wenn Antwort 'Nein' ist...
1	Programmabbruch verhindern	Besitzt die Prozedur einen Error Handler, damit bei Auftreten eines Laufzeitfehlers das Programm nicht abgebrochen wird bzw. abstürzt?	Fügen Sie in die Prozedur die Anweisung "On Error Goto <ErrorHandler>" ein und fangen Sie alle Laufzeitfehler ab. Muster siehe VBA Codebeispiele
2	Programmunterbrechung verhindern	Wurde die Tastenkombination Strg+Unterbrechen (Ctrl+Break) sowie die Escape-Taste (Esc) abgefangen, damit die Codeausführung nicht durch die Benutzer unterbrochen werden kann?	Leiten Sie mit der Codezeile "Application.EnableCancelKey = xlErrorHandler" die Codeunterbrechung an den Error Handler weiter. Muster siehe VBA Codebeispiele
3	VBE-Fehlerbehandlungsoptionen prüfen	Ist sichergestellt, dass die Fehlerbehandlungsoption "Bei jedem Fehler unterbrechen" nicht eingestellt ist?	Wenn Sie nicht davon ausgehen können, dass die Optionen immer wie erwartet eingestellt sind, sollte dies im Programm berücksichtigt werden. Anderenfalls bricht die Ausführung des VBA-Codes bei einem Laufzeitfehler trotz korrekter Fehlerbehandlungsroutine ab. Muster siehe VBA Codebeispiele

4	Timing-Probleme berücksichtigen	Wurde im Code berücksichtigt, dass in bestimmten Situationen Timing-Probleme auftreten könnten, z.B. wenn die Empfänger-Anwendung sehr lange benötigt, bis sie vollständig gestartet und bereit zum Empfangen von Tastenfolgen ist?	

Programmverhalten testen

Die folgenden Testmakros rufen den Eigenschaften-Dialog über das Menü Datei auf und setzen die Texteinfügemarke in das Eingabefeld "Hyperlink-Basis". Wenn das Dialogfenster korrekt geöffnet wird, ohne dass die Meldung "Das war ..." erscheint, funktioniert die im Makro benutzte Lösungsvariante zum Senden von Tastenfolgen.

```
Sub SendKeysTest1()
  SendKeys "%di{tab 8}", True
  MsgBox "Das war SendKeys mit Wait=True."
End Sub
Sub SendKeysTest2()
  SendKeys "%di{tab 8}"
  DoEvents
  MsgBox "Das war SendKeys mit Wait=False und DoEvents."
End Sub
Sub SendKeysTest3()
  Application.SendKeys "%di{tab 8}", True
  MsgBox "Das war Application.SendKeys mit Wait=True"
End Sub
Sub SendKeysTest4()
  Application.SendKeys "%di{tab 8}"
  DoEvents
  MsgBox "Das war Application.SendKeys mit Wait=False und DoEvents."
End Sub
```

VBA Codebeispiele

Beispiel aus der VBA-Hilfe

```
Dim Ergebnis, I
Ergebnis = Shell("CALC.EXE", 1)      ' Rechner starten.
AppActivate Ergebnis                 ' Rechner aktivieren.
For I = 1 To 100                     ' Zählschleife beginnen.
  SendKeys I & "{+}", True           ' Tastenanschläge senden, um die
Next I                               ' Werte von I zu addieren.
SendKeys "=", True                   ' Gesamtsumme abrufen.
SendKeys "%{F4}", True               ' Rechner mit ALT+F4 beenden.
```

Begriffserklärungen

Synchron
Asynchron
Task-ID
VBE
Präemptives Multitasking
Kooperatives Multitasking

Tasten-Namen

In general, spell key names as they appear in the following list, whether the name appears in text or in a procedure. Use all caps unless otherwise noted.
Note This list applies to Microsoft and IBM-type keyboards unless otherwise noted. Differences with the Macintosh keyboard are noted.

Key	Comment
ALT	-
ALT GR	-
Application key	Microsoft Natural Keyboard only
arrow keys	Not *direction keys, directional keys,* or *movement keys*.
BACKSPACE	-
BREAK	-
CAPS LOCK	-
CLEAR	-
COMMAND	Macintosh keyboard only. Use the bitmap to show this key whenever possible, because the key is not named on the keyboard.
CONTROL	Macintosh keyboard only. Does not always map to the CTRL key on the PC keyboard. Use correctly.

CTRL	-
DEL	Macintosh keyboard only. Use to refer to the forward delete key.
DELETE	Use to refer to the back delete key on the Macintosh keyboard.
DOWN ARROW	Use the and key with the arrow keys except in key combinations or key sequences. Always spell out. Do not use graphical arrows.
END	-
ENTER	On the Macintosh, use only when functionality requires it.
ESC	Always use ESC, not ESCAPE or Escape, especially on the Macintosh.
F1–F12	-
HELP	Macintosh keyboard only. Always use "the HELP key" to avoid confusion with the Help button.
HOME	-
INSERT	-
LEFT ARROW	Use *the* and *key* with the arrow keys except in key combinations or key sequences.
NUM LOCK	-
OPTION	Macintosh keyboard only.
PAGE DOWN	-
PAGE UP	-
PAUSE	-
PRINT SCREEN	-
RESET	-
RETURN	Macintosh keyboard only.
RIGHT ARROW	Use *the* and *key* with the arrow keys except in key combinations or key sequences.
SCROLL LOCK	-
SELECT	-
SHIFT	-
SPACEBAR	Precede with *the* except in procedures, key combinations, or key sequences.
SYS RQ	-
TAB	Use *the* and *key* except in key combinations or key sequences.
UP ARROW	Use *the* and *key* with the arrow keys except in key combinations or key sequences.
Windows logo key	Microsoft Natural Keyboard only.

STRG-UNTBR unterbinden - Abbruch von VBA deaktivieren

Es soll das Ausführen von Makrocode nicht mit STRG-UNTBR nicht unterbrochen werden können, damit nicht Makros zur Registrierung unterbrochen werden können:

Für USERFORMS:

Private Sub UserForm_Activate()
Application.EnableCancelKey = xlDisabled
End Sub

Private Sub UserForm_Terminate()
Application.EnableCancelKey = xlInterrupt
End Sub

ALLGEMEIN:

```
Sub Abbruch()
    Application.EnableCancelKey = xlDisabled
End Sub
Sub Abbruch()
    Application.EnableCancelKey = xlErrorHandler
End Sub
```

Tasten ein Makro zuweisen

Einfach in die Workbook_open Prozedur folgenden Befehle einbauen, um der ENDE-Taste das Makro "EXTRA_SPRING_ZUR_LETZTEN_ZEILE" zuzuordnen

Private Sub Workbook_Open()
Application.OnKey "{END}", "EXTRA_SPRING_ZUR_LETZTEN_ZEILE"
End sub

Cursortasten heißen DOWN, UP, LEFT, RIGHT

Mit folgender Zeile würde die Makrozuordnung wieder aufgehoben:
Application.OnKey "{END}"

Wichtig: bei Excel 2007 müssen zwingend die englischen Tastenbezeichnungen verwendet werden - das "POS1", das bei Excel 2003 noch ging, muss nun zwingend "HOME" heißen.

Shift key = "+" (plus sign)
Ctrl key = "^" (caret)

Alt key = "%" (percent sign)

BSP der Tastenkombination Shift-STRG-Pfeilnachrechts wird die Prozedur "TESTLAUF" zugeordnet

Application.OnKey "+^{RIGHT}", " TESTLAUF "

Hier die Tastennamen:

BILD-AB {PGDN}
BILD-AUF {PGUP}
EINFG {INSERT}
EINGABETASTE ~ (Tilde)
EINGABETASTE {RETURN}
EINGABETASTE (Zehnertastatur) {ENTER}
ENDE {END}
ENTF {CLEAR}
ENTFERNEN oder ENTF {DELETE} oder {DEL}
ESC {ESCAPE} oder {ESC}
F1 bis F15 {F1} bis {F15}
FESTSTELLTASTE {CAPSLOCK}
HILFE {HELP}
NACH-LINKS-TASTE {LEFT}
NACH-OBEN-TASTE {UP}
NACH-RECHTS-TASTE {RIGHT}
NACH-UNTEN-TASTE {DOWN}
NUM {NUMLOCK}
POS1 {HOME}
ROLLEN {SCROLLLOCK}
RÜCKTASTE {BACKSPACE} oder {BS}
TAB {TAB}
UNTBR {BREAK}

Tasten-Umleitungen auf Makro/Sperren aufheben

Hier der Code, um ALLE Tastaturkürzel zurückzusetzen

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    ' Alle Buchstaben-Tasten mit STRG/ALT/SHIFT zurücksetzen
    With Application
        For i = 97 To 122
            .OnKey "^" & Chr(i)
            .OnKey "%" & Chr(i)
            .OnKey "+" & Chr(i)
        Next i
    End With

    ' Alle Funktionstasten zurücksetzen
   For i = 1 To 12
      Application.OnKey "{F" & i & "}"
   Next i

End Sub
```

Noch umfangreicher hier:

```
Sub AlleEin()
    Dim i As Integer
    On Error Resume Next
    For i = 1 To 256
        Application.OnKey "^{" & i & "}" 'Strg
        Application.OnKey "%{" & i & "}" 'Alt
        Application.OnKey "^%{" & i & "}" 'Strg-Alt
    Next i
End Sub
```

Tasten : Kopieren über STRG-C und Menü unterbinden

(Für die rechte Maustaste zu sperren, siehe "Rechte maustaste deaktivieren")

```
Sub Kopieren_Aktivieren()
'Tastenkombinationen einschalten
Application.OnKey "^x"
Application.OnKey "^c"
Application.OnKey "^v"
Application.OnKey "+{DEL}"
Application.OnKey "+{INSERT}"

'Drag & Drop wieder erlauben
Application.CellDragAndDrop = True

'Schaltflaechen in Menüleiste => Bearbeiten aktivieren
procControlEnableDisable 21, True ' Ausschneiden
procControlEnableDisable 19, True 'Kopieren
procControlEnableDisable 22, True 'Einfuegen
procControlEnableDisable 755, True 'Inhalte einfuegen
procControlEnableDisable 809, True 'Office-&Zwischenablage

End Sub

Sub Kopieren_Deaktivieren()
'Tastenkombinationen deaktivieren
Application.OnKey "^x", ""
Application.OnKey "^c", ""
Application.OnKey "^v", ""
Application.OnKey "+{DEL}", ""
Application.OnKey "+{INSERT}", ""

'Drag & Drop ausschalten
Application.CellDragAndDrop = False

'Schaltflaechen in Menüleiste => Bearbeiten deaktivieren
procControlEnableDisable 21, False ' Ausschneiden
procControlEnableDisable 19, False 'Kopieren
procControlEnableDisable 22, False 'Einfuegen
procControlEnableDisable 755, False 'Inhalte einfuegen
procControlEnableDisable 809, False 'Office-&Zwischenablage
```

```
End Sub

Sub procControlEnableDisable(intId As Integer, _
bolStatus As Boolean)
Dim cmbSuche As CommandBar
Dim cmbcSteuerelement As CommandBarControl
On Error Resume Next
For Each cmbSuche In Application.CommandBars
Set cmbcSteuerelement = _
cmbSuche.FindControl(ID:=intId, recursive:=True)
If Not cmbcSteuerelement Is Nothing Then
cmbcSteuerelement.Enabled = bolStatus
End If
Next
End Sub
```

Tastenkombinationen verbieten oder zuweisen

Mein Code

```
Private Sub Workbook_Open()

    ' Sperren von Tastenkombinationen
    Application.OnKey "^{+}", ""    ' unterbinden STRG +
    Application.OnKey "^{-}", ""    ' unterbinden von STRG -
    Application.OnKey "^{109}", ""  ' minus
    Application.OnKey "^{107}", ""  ' plus
    Application.OnKey "^{c}", ""    ' STRG-C
    Application.OnKey "^{v}", ""    ' STRG-V
    Application.OnKey "^{x}", ""    ' unterbinden von STRG-X
    Application.OnKey "+{DEL}", ""
    Application.OnKey "+{INSERT}", ""

    ' Drag & Drop ausschalten
    Application.CellDragAndDrop = False

End Sub
```

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
  Cancel = True ' Man kann nirgendwo einen rechten Mausklick machen
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)

    Application.CommandBars("Cell").Reset ' Rechten Mausklick wieder aktivieren

    ' Erlauben von Tastenkombinationen
    Application.OnKey "^{+}"    ' STRG +
    Application.OnKey "^{-}"    ' STRG -
    Application.OnKey "^{109}"  ' minus
    Application.OnKey "^{107}"  ' plus
    Application.OnKey "^{c}"    ' STRG-C
    Application.OnKey "^{v}"    ' STRG-V
    Application.OnKey "^{x}"    ' STRG-X
    Application.OnKey "+{DEL}"
    Application.OnKey "+{INSERT}"

    ' Drag & Drop einschalten
    Application.CellDragAndDrop = True

End Sub
```

ALLGEMEIN

Strg-A oder Alt-F11 und andere Tastenkombinationen können in Excel sehr leicht deaktiviert / aktiviert werden:

Die OnKey-Anweisung, die zum Festelegen einer auszuführenden Prozedur verwendet wird, führt durch Zuweisen der Prozedur "" dazu, dass die betreffende Tastenkombination deaktiviert wird:

```
Sub DeaktivierenTastenkombination_STRG_F11()
  Application.OnKey "%{F11}", "" ' %= STRG
End Sub
```

Wieder aktivieren der default-Funktion von Excel: man übergibt einfach keinen Wert für die Prozedur

```
Sub AktivierenTastenkombination()
  Application.OnKey "%{F11}"
End Sub
```

Zuweisen einer eigenen Prozedur:

```
Sub DeaktivierenTastenkombination()
  Application.OnKey "%{F11}", "TESTEN"
End Sub

Sub TESTEN()
    MsgBox "hallo"
End Sub
```

Zusätzlich gedrückte Tasten haben folgende Codes:
Shift ist +
STRG ist ^
Alt ist %

Die Sondertasten der Tastatur können natürlich ebenso belegt werden:

Taste	Code
BILD-AB	{PGDN}
BILD-AUF	{PGUP}
EINFG	{INSERT}
EINGABETASTE	~ (Tilde)
EINGABETASTE	{RETURN}
EINGABETASTE (Zehnertastatur)	{ENTER}
ENDE	{END}
ENTF	{CLEAR}
ENTFERNEN oder ENTF	{DELETE} oder {DEL}
ESC	{ESCAPE} oder {ESC}
F1 bis F15	{F1} bis {F15}
FESTSTELLTASTE	{CAPSLOCK}
HILFE	{HELP}
NACH-LINKS-TASTE	{LEFT}
NACH-OBEN-TASTE	{UP}
NACH-RECHTS-TASTE	{RIGHT}

NACH-UNTEN-TASTE	{DOWN}
NUM	{NUMLOCK}
POS1	{HOME}
ROLLEN	{SCROLLLOCK}
RÜCKTASTE	{BACKSPACE} oder {BS}
TAB	{TAB}
UNTBR	{BREAK}

BSP der Tastenkombination Shift-STRG-Pfeilnachrechts wird die Prozedur "TESTLAUF" zugeordnet

Application.OnKey "+^{RIGHT}", " TESTLAUF "

Wichtig ist die Rücksetzung auf Standard, wenn deine Mappe deaktiviert wird, denn andere Tabellen kommen mit Deinem Makro wahrscheinlich nicht klar...

Tasten / -kombinationen unterbinden o. Makro ausführen

Wichtig: Übergibt man den Onkey-Befehle mit , "" - dann wird in ein leeres Makro umgeleitet und damit die Funktion gesperrt
Übergibt man ihn ohne , "" dann wird wieder aktiviert

Man kann die Taste in {} setzen, muss aber nicht sein - außer wenn es sich um Tasten handelt, die zugleich Kürzel sind
^ ... STRG + ... Shift % ... Alt diese müssen unter {} sein

Achtung: Application.OnKey "^{+}", "" sperrt nur die PLUS-Taste auf der Tastatur, nicht auf dem Zehnerblock
Lösung: siehe nächster Abschnitt STRG-PLUS/MINUS auf Zehnerblock sperren !

```
Sub Test()

   Application.OnKey "^p", ""     ' unterbinden von STRG-P
   Application.OnKey "^p"      ' wieder aktivieren von STRG-P

   Application.OnKey "^x", "" ' unterbinden von STRG-X
   Application.OnKey "^{+}", ""  ' unterbinden STRG +
   Application.OnKey "^{-}", "" ' unterbinden von STRG -
Application.OnKey "^{109}", ""   'minus-Taste auf Zehnerblock
Application.OnKey "^{107}", ""   'plus-Taste auf Zehnerblock


End Sub
```

```
'in "Diese Arbeitsmappe
'deaktivieren
Private Sub Workbook_Open()
Application.OnKey "%{F2}", ""
Application.OnKey "%{F8}", ""
Application.OnKey "%{F11}", ""
Application.OnKey "%+{F2}", ""
Application.OnKey "+{F12}", ""
Application.OnKey "^{F12}", ""
Application.OnKey "{F12}", ""
Application.OnKey "^{s}", ""
Application.OnKey "^{o}", ""
Application.OnKey "^{c}", ""
Application.OnKey "^{v}", ""
End Sub

'Standardfunktion wieder aktivieren
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Application.OnKey "%{F2}"
Application.OnKey "%{F8}"
Application.OnKey "%{F11}"
Application.OnKey "%+{F2}"
Application.OnKey "+{F12}"
Application.OnKey "^{F12}"
Application.OnKey "{F12}"
Application.OnKey "^{s}"
Application.OnKey "^{o}"
Application.OnKey "^{c}"
Application.OnKey "^{v}"
End Sub
```

```
RÜCKTASTE {BACKSPACE} oder {BS}
PAUSE {BREAK}
FESTSTELLTASTE {CAPSLOCK}
ENTF {CLEAR}
ENTF {DELETE} oder {DEL}
NACH-UNTEN-TASTE {DOWN}
ENDE {END}
EINGABETASTE (Zehnertastatur) {ENTER}
EINGABETASTE ~ (Tilde)
ESC {ESCAPE} oder {ESC}
HILFE {HELP}
POS1 {HOME}
EINFG {INSERT}
NACH-LINKS-TASTE {LEFT}
NUM {NUMLOCK}
```

BILD-AB {PGDN}
BILD-AUF {PGUP}
EINGABETASTE {RETURN}
NACH-RECHTS-TASTE {RIGHT}
ROLLEN {SCROLLOCK}
TAB {TAB}
NACH-OBEN-TASTE {UP}
F1 bis F15 {F1} bis {F15}

How do I use Application.Onkey

With Application.Onkey you can disable a particular key or key combination or run
a macro when you use a particular key or key combination.
Below you can read information from Excel's VBA help about Onkey.

The Key argument can specify any single key combined with ALT, CTRL, or SHIFT,
or any combination of these keys. Each key is represented by one or more characters,
such as "a" for the character a, or "{ENTER}" for the ENTER key.

Shift key = "+" (plus sign)
Ctrl key = "^" (caret)
Alt key = "%" (percent sign)

This example assigns "YourMacroName" to the key sequence SHIFT+CTRL+RIGHT ARROW
Application.OnKey "+^{RIGHT}", "YourMacroName"

This example returns SHIFT+CTRL+RIGHT ARROW to its normal meaning.
Application.OnKey "+^{RIGHT}"

This example disables the SHIFT+CTRL+RIGHT ARROW key sequence.
Application.OnKey "+^{RIGHT}", ""

You can also use it to disable a built-in shortcut like Ctrl p that you can use to Print.
Application.OnKey "^p", ""

Das Bild kann zurzeit nicht angezeigt werden.

Note: For some reason you can't disable every key combination with Onkey.
I am not able to disable Ctrl- and Ctrl+ for example but you can protect the worksheet to disable these two shortcuts that popup the Insert and Delete dialog.

Tip: If you want to use Onkey only for one workbook you can place the code in the Activate and Deactivate event in the ThisWorkbook module of that file.

See how here
http://www.rondebruin.nl/code.htm

```
Private Sub Workbook_Activate()
    Application.OnKey "+^{RIGHT}", "YourMacroName"
End Sub

Private Sub Workbook_Deactivate()
    Application.OnKey "+^{RIGHT}"
End Sub
```

Disable almost every key or key combination

If you want to disable every key or key combination you can use the code below.
If you have suggestions to make the code better let me know.

```
Sub Disable_Keys()
    Dim StartKeyCombination As Variant
    Dim KeysArray As Variant
    Dim Key As Variant
    Dim I As Long

    On Error Resume Next

    'Shift key = "+"  (plus sign)
    'Ctrl key = "^"   (caret)
    'Alt key = "%"    (percent sign
    'We fill the array with this keys and the key combinations
```

```
    'Shift-Ctrl, Shift- Alt, Ctrl-Alt, Shift-Ctrl-Alt

    For Each StartKeyCombination In Array("+", "^", "%", "+^", "+%", "^%", "+^%")

        KeysArray = Array("{BS}", "{BREAK}", "{CAPSLOCK}", "{CLEAR}", "{DEL}", _
                "{DOWN}", "{END}", "{ENTER}", "~", "{ESC}", "{HELP}", "{HOME}", _
                "{INSERT}", "{LEFT}", "{NUMLOCK}", "{PGDN}", "{PGUP}", _
                "{RETURN}", "{RIGHT}", "{SCROLLLOCK}", "{TAB}", "{UP}")

        'Disable the StartKeyCombination key(s) with every key in the KeysArray
        For Each Key In KeysArray
            Application.OnKey StartKeyCombination & Key, ""
        Next Key

        'Disable the StartKeyCombination key(s) with every other key
        For I = 0 To 255
            Application.OnKey StartKeyCombination & Chr$(I), ""
        Next I

        'Disable the F1 - F15 keys in combination with the Shift, Ctrl or Alt key
        For I = 1 To 15
            Application.OnKey StartKeyCombination & "{F" & I & "}", ""
        Next I

    Next StartKeyCombination


    'Disable the F1 - F15 keys
    For I = 1 To 15
        Application.OnKey "{F" & I & "}", ""
    Next I

    'Disable the PGDN and PGUP keys
    Application.OnKey "{PGDN}", ""
    Application.OnKey "{PGUP}", ""
End Sub
```

Enable key or key combinations :

```
Sub Enable_Keys()
    Dim StartKeyCombination As Variant
    Dim KeysArray As Variant
    Dim Key As Variant
```

```
    Dim I As Long

    On Error Resume Next

    'Shift key = "+"  (plus sign)
    'Ctrl key = "^"   (caret)
    'Alt key = "%"    (percent sign
    'We fill the array with this keys and the key combinations
    'Shift-Ctrl, Shift- Alt, Ctrl-Alt, Shift-Ctrl-Alt

    For Each StartKeyCombination In Array("+", "^", "%", "+^", "+%", "^%", "+^%")

        KeysArray = Array("{BS}", "{BREAK}", "{CAPSLOCK}", "{CLEAR}", "{DEL}", _
                  "{DOWN}", "{END}", "{ENTER}", "~", "{ESC}", "{HELP}", "{HOME}", _
                  "{INSERT}", "{LEFT}", "{NUMLOCK}", "{PGDN}", "{PGUP}", _
                  "{RETURN}", "{RIGHT}", "{SCROLLLOCK}", "{TAB}", "{UP}")

        'Enable the StartKeyCombination key(s) with every key in the KeysArray
        For Each Key In KeysArray
            Application.OnKey StartKeyCombination & Key
        Next Key

        'Enable the StartKeyCombination key(s) with every other key
        For I = 0 To 255
            Application.OnKey StartKeyCombination & Chr$(I)
        Next I

        'Enable the F1 - F15 keys in combination with the Shift, Ctrl or Alt key
        For I = 1 To 15
            Application.OnKey StartKeyCombination & "{F" & I & "}"
        Next I

    Next StartKeyCombination


    'Enable the F1 - F15 keys
    For I = 1 To 15
        Application.OnKey "{F" & I & "}"
    Next I

    'Enable the PGDN and PGUP keys
    Application.OnKey "{PGDN}"
    Application.OnKey "{PGUP}"
End Sub
```

Taste STRG-PLUS und -MINUS auf Zehnerblock sperren

Hallo alle,

ich habe die Lösung gefunden!

```
'schaltet die Tastenkombinationen Strg+... für Ziffernblock aus
Sub NumAus()
Application.OnKey "^{111}", ""  'geteilt
Application.OnKey "^{106}", ""  'mal
Application.OnKey "^{109}", ""  'minus
Application.OnKey "^{107}", ""  'plus
End Sub
```

Hilfe brachte mir letztlich diese Seite:
http://www.mrexcel.com/board2/viewtopic.php?t=87676&start=0

Man kann zur Codeermittlung eine Textbox in einer Userform verwenden:

```
Private Sub TextBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
MsgBox KeyCode
End Sub
```

Freudige Grüße,
Matthias

Tastatur – weitere Funktionen

```
'  Title: Various Keyboard routines
'


' ----------------------------------------------------------------------------

'--------- Class Name: clsKeys

Option Explicit

Private Declare Function MapVirtualKey Lib "user32" Alias _
```

```
   "MapVirtualKeyA" (ByVal wCode As Long, _
   ByVal wMapType As Long) As Long

Private Declare Function VkKeyScan Lib "user32" Alias "VkKeyScanA" (ByVal _
   cChar As Byte) As Integer

Private Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal _
   bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long)

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As _
   Long) As Integer

Private Const KEYEVENTF_EXTENDEDKEY = &H1
Private Const KEYEVENTF_KEYUP = &H2

Public Enum EnumKeys
    keyBackspace = &H8
    keyTab = &H9
    keyReturn = &HD
    keyShift = &H10
    keyControl = &H11
    keyAlt = &H12
    keyPause = &H13
    keyEscape = &H1B
    keySpace = &H20
    keyEnd = &H23
    keyHome = &H24
    keyLeft = &H25
    KeyUp = &H26
    keyRight = &H27
    KeyDown = &H28
    keyInsert = &H2D
    keyDelete = &H2E
    keyF1 = &H70
    keyF2 = &H71
    keyF3 = &H72
    keyF4 = &H73
    keyF5 = &H74
    keyF6 = &H75
    keyF7 = &H76
    keyF8 = &H77
    keyF9 = &H78
```

```
    keyF10 = &H79
    keyF11 = &H7A
    keyF12 = &H7B
    keyNumLock = &H90
    keyScrollLock = &H91
    keyCapsLock = &H14
End Enum

'Presses the single key represented by sKey
Public Sub PressKey(sKey As String, Optional bHold As Boolean, Optional _
   bRelease As Boolean)

    Dim nVK As Long
    nVK = VkKeyScan(Asc(sKey))

    If nVK = 0 Then
        Exit Sub
    End If

    Dim nScan As Long
    Dim nExtended As Long

    nScan = MapVirtualKey(nVK, 2)
    nExtended = 0
    If nScan = 0 Then
        nExtended = KEYEVENTF_EXTENDEDKEY
    End If
    nScan = MapVirtualKey(nVK, 0)

    Dim bShift As Boolean
    Dim bCtrl As Boolean
    Dim bAlt As Boolean

    bShift = (nVK And &H100)
    bCtrl = (nVK And &H200)
    bAlt = (nVK And &H400)

    nVK = (nVK And &HFF)

    If Not bRelease Then
        If bShift Then
            keybd_event EnumKeys.keyShift, 0, 0, 0
        End If
        If bCtrl Then
```

```
            keybd_event EnumKeys.keyControl, 0, 0, 0
        End If
        If bAlt Then
            keybd_event EnumKeys.keyAlt, 0, 0, 0
        End If

        keybd_event nVK, nScan, nExtended, 0
    End If

    If Not bHold Then
        keybd_event nVK, nScan, KEYEVENTF_KEYUP Or nExtended, 0

        If bShift Then
            keybd_event EnumKeys.keyShift, 0, KEYEVENTF_KEYUP, 0
        End If
        If bCtrl Then
            keybd_event EnumKeys.keyControl, 0, KEYEVENTF_KEYUP, 0
        End If
        If bAlt Then
            keybd_event EnumKeys.keyAlt, 0, KEYEVENTF_KEYUP, 0
        End If
    End If

End Sub

'Loop through a string and calls PressKey for each character (Does not
' parse strings like SendKeys)
Public Sub PressString(ByVal sString As String, Optional bDoEvents As Boolean = True)
    Do While sString <> ""
        PressKey Mid(sString, 1, 1)

        Sleep 20
        If bDoEvents Then
            DoEvents
        End If

        sString = Mid(sString, 2)
    Loop
End Sub

'Presses a specific key (this is used for keys that don't have a
' ascii equilivant)
Public Sub PressKeyVK(keyPress As EnumKeys, Optional bHold As Boolean, _
   Optional bRelease As Boolean, Optional bCompatible As Boolean)
```

```
    Dim nScan As Long
    Dim nExtended As Long

    nScan = MapVirtualKey(keyPress, 2)
    nExtended = 0
    If nScan = 0 Then
       nExtended = KEYEVENTF_EXTENDEDKEY
    End If
    nScan = MapVirtualKey(keyPress, 0)

    If bCompatible Then
       nExtended = 0
    End If

    If Not bRelease Then
       keybd_event keyPress, nScan, nExtended, 0
    End If

    If Not bHold Then
       keybd_event keyPress, nScan, KEYEVENTF_KEYUP Or nExtended, 0
    End If

End Sub

'Returns (in the boolean variables) the status of the various Lock keys
Public Sub GetLockStatus(bCapsLock As Boolean, bNumLock As Boolean, _
   bScrollLock As Boolean)

    bCapsLock = GetKeyState(EnumKeys.keyCapsLock)
    bNumLock = GetKeyState(EnumKeys.keyNumLock)
    bScrollLock = GetKeyState(EnumKeys.keyScrollLock)
End Sub
```

VBA-Code mit ESC-Taste Abbrechen

Bisweilen möchte man dem User die Möglichkeit einräumen, dass er einen zu lange dauernden VBA-Code mittels ESC-Taste jederzeit abbrechen kann

VARIANTE A mit Abzweigen in Errorhandler (geht nur in Excel)

```
Sub TEST()
```

```
 On Error GoTo ABBRUCH
' Definieren der ESC-Taste

 Application.EnableCancelKey = xlErrorHandler ' xlInterrupt = es kommt VBA-STRG-PAUSE-Fenster mit Ende/Debuggen/Abbruch -   xlDisabled=nix geschieht

' nachfolgende Schleife dauert ca 15 sek
For T = 1 To 2000
 For S = 1 To 999999
 Next S
Next T

MsgBox "Schleife ganz durchlaufen ohne Abbruch"

Exit Sub

ABBRUCH:

MsgBox "Es wurde mit ESC abgebrochen"


End Sub
```

VARIANTE B1 mit Verzweigen direkt in der Prozedur (auch in Outlook, Word etc)

Bisweilen wird man nicht wollen, dass der Errorhandler für Fehlerbehandlung nicht mehr zur Verfügung steht, weil er mit der ESC-Taste belegt ist.
Dann hilft dieser Code, der zudem auch z.B. in Outlook wunderbar läuft

```
Private Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As Long

Private Const VK_ESCAPE = &H1B ' ESC Taste

Public Sub ENDLOSSCHLEIFE()

   Dim TASTENDRUCK As Long

 While 1 = 1
   TASTENDRUCK = GetAsyncKeyState(VK_ESCAPE) ' schauen ob die Esc-Taste gedrückt wurde
   If CBool(TASTENDRUCK) Then ' falls ja (wahr)
      MsgBox "Sie haben ESC gedrückt."
      Exit Sub
```

```
   End If
 Wend


End Sub
```

VARIANTE B2 mit Abzweigen in eigene Prozedur (auch in Outlook, Word etc)

Bisweilen wird man nicht wollen, dass der Errorhandler für Fehlerbehandlung nicht mehr zur Verfügung steht, weil er mit der ESC-Taste belegt ist.
Dann hilft dieser Code, der zudem auch z.B. in Outlook wunderbar läuft

```
Private Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As Long

Private Const VK_ESCAPE = &H1B ' ESC Taste

Public Sub ENDLOSSCHLEIFE()

   Dim TASTENDRUCK As Long

 While 1 = 1
   TASTENDRUCK = GetAsyncKeyState(VK_ESCAPE) ' schauen ob die Esc-Taste gedrückt wurde
   If CBool(TASTENDRUCK) Then ' falls ja (wahr)
      Call ABBRUCH
      Exit Sub
   End If
 Wend


End Sub


Sub ABBRUCH()

  MsgBox "Sie haben ESC gedrückt."

End Sub
```

Variante Code, der nach dem Ablauf ausgibt, wenn ESC- NICHT gedrückt wurde.

```
Private Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As Long

Private Const VK_ESCAPE = &H1B ' ESC Taste

Public Sub StartZeitGeber()
  Dim RetVal As Long

 Application.Wait Now + TimeValue("0:0:2")
 RetVal = GetAsyncKeyState(VK_ESCAPE)

 If Not CBool(RetVal) Then
 Call Makro_Automatik
 End If

End Sub


Sub Makro_Automatik()

   MsgBox "Nicht ESC gedrückt =("

End Sub
```

USER-ABFRAGEN

Benutzername auslesen

0. Datum, Usernamen und letzte Belegnummer ausgeben

```
' Speichern des letzten Datums, des letzten Users und der letzten Belegnummer

E.Range("U1") = "am " & Left(Now, 10) & " um " & Right(Now, 8) & vbCrLf & _
"von " & Application.UserName & vbCrLf & _
"mit letzter Belegnummer: " & Cells(Range("B65536").End(xlUp).Row, 2)
```

1.Möglichkeit Application.Username

aus unerfindlichen Gründen klappte dieser Befehl bei den meisten Thin Clients und Stand-PCs, aber nicht bei einen Mitarbeiter, das klappte nur die 2 Variante durch Auslesen mittels API (siehe unten)

Msgbox Application.UserName

Möchte man Vornamen und Nachnamen auftrennen und großschreiben:

```
Sub MITARBEITER_NAME()

Dim BENUTZERNAME
Dim VORNAME
Dim NACHNAME

MITARBEITERNAME = Application.UserName
VORNAME = UCase(Left(MITARBEITERNAME, 1)) & Mid(MITARBEITERNAME, 2, InStr(1, MITARBEITERNAME, ".") - 2)
NACHNAME = UCase(Mid(MITARBEITERNAME, InStr(1, MITARBEITERNAME, ".") + 1, 1)) & Mid(MITARBEITERNAME, InStr(1, MITARBEITERNAME, ".") + 2)
Range("I7") = VORNAME & " " & NACHNAME

End Sub
```

Zweite Möglichkeit zum Auslesen mittels API:

```
Option Explicit

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

Function fOSUserName() As String
    Dim lngLen As Long, lngX As Long
    Dim strUserName As String

    strUserName = String$(254, 0)
    lngLen = 255
    lngX = apiGetUserName(strUserName, lngLen)
    If lngX <> 0 Then
       fOSUserName = Left$(strUserName, lngLen - 1)
      Else
       fOSUserName = ""
    End If
End Function

Sub DEMO
   MsgBox "Zur Zeit angemeldet ist: " & fOSUserName
End Sub
```

3. Möglichkeit

Wer bin ich ?

Sollte man eigentlich wissen,,.. in einem Netzwerk aber nicht immer selbstverständlich ;-)

```
'Eigenes Kürzel definieren
Private Declare Function GCN Lib "kernel32" Alias "GetComputerNameA" (ByVal myPara As String, myLen As Long) As Long
Private Declare Function GUN Lib "advapi32.dll" Alias "GetUserNameA" (ByVal myPara As String, myLen As Long) As Long
'Standardeklarationen lauten sonst im allgemeinen
'Private Declare Function GetComputerNameA Lib "kernel32" (ByVal lpBuffer As String, nSize As Long) As Long
'Private Declare Function GetUserNameA Lib "advapi32.dll" (ByVal lpBuffer As String, nSize As Long) As Long

'Prozeduren:
Public Function ActiveUserName() As String
'Benutzernamen auslesen
Dim AUN As String * 100
Dim AunLen As Byte
'100 Zeichen reichen in den meisten Fällen aus
AunLen = 100
If GUN(AUN, Len(AUN)) Then
   'Siehe Hinweis *
   ActiveUserName = Left(AUN, AunLen)
Else
   ActiveUserName = "User can not be Identified"
End If
End Function


Public Function ActiveComputerName() As String
'Benutzernamen auslesen
Dim ACN As String * 100
Dim AcnLen As Byte
AcnLen = 100
If GCN(ACN, Len(ACN)) Then
   'Siehe Hinweis*
   ActiveComputerName = Left(ACN, AcnLen)
Else
   ActiveComputerName = "User can not be Identified"
End If
End Function


Sub wer_und_was_bin_ich()
Dim Qe As Byte
MsgBox ("Mein Rechner heisst" & ActiveComputerName)
MsgBox ("Aktuell angemeldeter User ist: " & ActiveUserName)
```

```
End Sub
```

Hinweis

Das Problem das nun auftritt, ist, dass eine Überprüfung des Benutzernamens mit grosser Wahrscheinlichkeit/Sicherheit fehlschlägt. Ihr Benutzer heisst "Uwe" und eine Überprüfung in der Art

If ActiveUserName = "Uwe" Then

schlägt fehl, obwohl im Debug.Fenster der richtige Name angezeigt wird. Das Problem liegt in der Dimensionierung der Variablen "AUN As String * 100" Nach der Rückgabe aus der API-Funktion *"GetUserNameA"* wird die Variable mit sogenannten 0-Zeichen (ASCII 0) aufgefüllt bis der String eben die definierten 100 Zeichen enthält.

Testen können Sie dies, indem Sie am Punkt "If GUN...:" einen Haltepunkt setzen und mit dem Mauszeiger über die Variable AUN fahren. EXCEL zeigt Ihnen nun den Inhalt der Variablen "AUN" in einem Kommentarfeld an. Dieses wird in der Regel dann in etwa so aussehen:

"USERNAME]]]]]]]]]]]].....

Um diese *0*-Zeichen zu eliminieren können Sie diese Funktion verwenden

Left(AUN, InStr(AUN, vbNullChar) - 1)

oder

Left(ACN, InStr(ACN, vbNullChar) - 1)

Mit "InStr" suchen Sie innerhalb des Strings AUN nach der Position des ersten Zeichen das diesem ASCII 0 Zeichen entspricht. Dieses Zeichen können sie mit "vbNullChar" identifizieren.

Von dieser Positionsnummer subtrahieren Sie 1 und haben den "reinen" Benutzernamen, bzw. den "reinen" Computernamen.

Alternativ geht's auch mit einer, etwas umständlichen ;-), eigenen Funktion der Sie die Variable "AUN" übergeben

```
Function PureName(tmpName As String)
Dim i
For i = 1 To 100
   Select Case UCase(Mid(tmpName, i, 1))
      Case vbNullChar
         'Hier wird die Position des
         'Zeichens ebenfalls gefunden
         Debug.Print "Pos: " & i
         PureName = Left(tmpName, i - 1)
         Exit For
   End Select
Next i
End Function
Sub wer_und_was_bin_ich()
MsgBox ("Mein Rechner ist" & PureName(ActiveComputerName) & PureName(ActiveUserName))
End Sub
```

Dann klappt's auch mit dem Vergleich ;-)

4. Version

```
Declare Function GetUserName Lib "advapi32.dll" Alias
```

```
"GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Sub ShowUserName()
Dim Buffer As String * 100
Dim BuffLen As Long
BuffLen = 100
GetUserName Buffer, BuffLen
MsgBox Left(Buffer, BuffLen - 1)
End Sub
Function NetUserName()
Dim Buffer As String * 100
Dim BuffLen As Long
BuffLen = 100
GetUserName Buffer, BuffLen
NetUserName = Left(Buffer, BuffLen - 1)
End Function
```

Dialogbox positionieren

```
Declare Function SetWindowPos Lib "User" (ByVal hwnd%, ByVal _
   hwndAfter%, ByVal x%, ByVal y%, ByVal cx%, ByVal cy%, ByVal _
   Flags%) As Integer
Declare Function FindWindow Lib "User" (ByVal szClass$, ByVal _
   szTitle$) As Integer
Const SWP_NOSIZE = 1
Const SWP_NOMOVE = 2
Const SWP_NOZORDER = 4
Const SWP_NOREDRAW = 8
Const SWP_NOACTIVATE = &h10
Sub ShowDialogboxByPos(ByVal x%, ByVal y%)
   Dim hwndDlg As Integer
   hwndDlg = FindWindow("bosa_sdm_XL", ActiveDialog.DialogFrame.Text)
   If hwndDlg <> 0 Then
      SetWindowPos hwndDlg, 0, x%, y%, 0, 0, SWP_NOSIZE + _
        SWP_NOACTIVATE + SWP_NOZORDER
   End If
End Sub 'ShowDialogboxByPos


Sub CentreDialog32()
On Error Resume Next
'*** DIMENSION VARIABLES ***
Dim V_rect As Rect32
'Variables to retrieve the screen dimensions with GetSystemMetrics
API.
```

```
Dim V_scrn_w As Long
Dim V_scrn_h As Long
'Variable to store the window handle with FindWindow API.
Dim V_hwnd As Long
'Variables to calculate the new dimensions for the window.
Dim V_width As Long
Dim V_height As Long
Dim V_left As Long
Dim V_top As Long
'Get the handle of the dialog box window - 'bosa_sdm_XL' is the class
name
'for an Excel dialog box.
V_hwnd = FindWindow32("bosa_sdm_XL", ActiveDialog.DialogFrame.Text)
'Only continue if a valid handle is returned
If V_hwnd <> 0 Then
'Get the width and height of the screen in pixels
   V_scrn_w = GetSystemMetrics32(0)
   V_scrn_h = GetSystemMetrics32(1)
'Get the dimensions of the dialog box window in pixels
   GetWindowRect32 V_hwnd, V_rect
'Calculate the width and height of the dialog box
   V_width = Abs(V_rect.Right - V_rect.Left)
   V_height = Abs(V_rect.Top - V_rect.Bottom)
'Calculate the new position of the dialog box in pixels
   V_left = (V_scrn_w - V_width) / 2
   V_top = (V_scrn_h - V_height) / 2
'Move the dialog box to the centre of the screen
   Movewindow32 V_hwnd, V_left, V_top, V_width, V_height, True
End If
End Sub
'TRY IT HERE!
Sub ShowDialog()
ThisWorkbook.DialogSheets("Dialog1").Show
End Sub
```

Dialogbox : entfernen der Steuerungselemente im Rahmen

```
Declare Function FindWindowA Lib "user32" _
(ByVal lpClassName As Any, _
ByVal lpWindowName As String) As Long
Declare Function GetWindowLongA Lib "user32" _
(ByVal hwnd As Long, _
ByVal nIndex As Integer) As Long
Declare Function SetWindowLongA Lib "user32" _
```

```
(ByVal hwnd As Long, ByVal nIndex As Integer, _
ByVal dwNewLong As Long) As Long
Global Const GWL_STYLE = (-16)
Global Const WS_SYSMENU = &H80000
' Assign to dialogframe's OnAction event
Sub RemoveControlMenuExcel32()
    Dim WindowStyle As Long
    Dim hwnd As Long
    Dim Result
    'bosa_sdm_xl is the class name for an
    'Excel 5/7 dialog box.
    'In Excel 97 it is bosa_sdm_xl8
    '(i.e. XL 5/7 style dialogs in XL97, notuserforms)
    hwnd = FindWindowA("bosa_sdm_xl", ActiveDialog.DialogFrame.Text)
    'Get the current windowstyle
    WindowStyle = GetWindowLongA(hwnd, GWL_STYLE)
    'Turn off the System menu
    WindowStyle = WindowStyle And (Not WS_SYSMENU)
    'Set the style
    Result = SetWindowLongA(hwnd, GWL_STYLE, WindowStyle)
End Sub
```

Messagebox

damit kann man sich schnell Variablen anzeigen lassen

Msgbox variable1
oder
Msgbox "Variable1 hat den Wert" & variable1

& vbTab & "Text" ' vbTab zählt als Tabulatorsprung
& vbCrLf ' Wagenrücklauf und Zeilenvorschub

MEHRZEILIGE MESSAGBOXEN:

Dim BUTTONTEXT As String

Buttontext=Buttontext+"DIES IST DER TEXT 1" & vbCrLf
Buttontext=Buttontext+"DIES IST DER TEXT 2" & vbCrLf

Messagebox mit Titelzeilentext

MsgBox "Hier kommt meine eigentliche Nachricht", vbInformation, "Bitte lesen"

Messagebox frei positionierbar

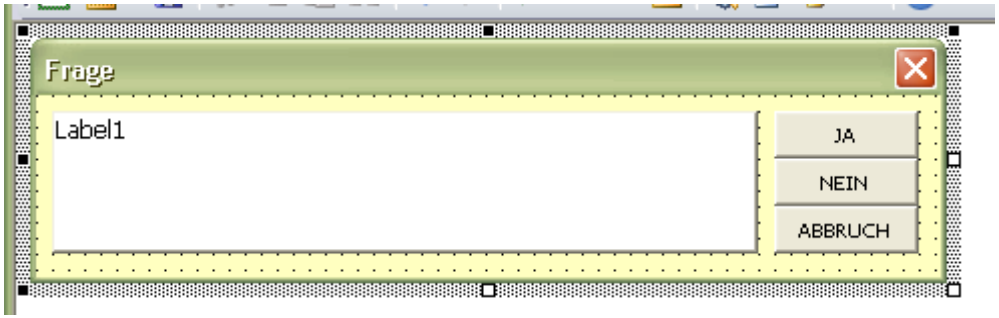
Dies geht nur über eine Userform. Ich habe eine solche in der Vorlage im Pfad: I:\' VBA\EXCEL\' EXCEL-VORLAGEN
Frei positionierbare Userform als MSGBOX-Ersatz.xlsm

Alternative: Inputboxen sind frei positionierbar - Schreib den MSGBOX-Text in die Inputbox und gib als Eingabefeld einfach nur ein "OK" ein und mit OK-Button kann man sie schließen.

Hier ist der Code für das Modul1

```
Public Sub MBox(Text As String, Titel As String, Starttop As Long, Startleft As Long)

    With UserForm1
    .StartUpPosition = 0
    .Top = 10 ' Starttop
    .Left = 10 ' Startleft
    .Label1 = Text
    .Caption = Titel
    End With
    UserForm1.Show
End Sub
Sub Test()

   MBox "Hier kommt mein Text" & vbCrLf & vbCrLf & "Und hier die Zeile 2", "Nachricht", 10, 10

End Sub
```

Hier der Code für die Userform




```
Private Sub CommandButton1_Click()
    ' JA-Button
    Cells(z, 14) = DIFFERENZ
    Cells(z, 19) = DIFFERENZ
    Unload UserForm1
End Sub


Private Sub CommandButton2_Click()
    ' NEIN-Button
    Unload UserForm1
End Sub

Private Sub CommandButton3_Click()
    ' ABBRUCH-Button
    Unload UserForm1
    End
End Sub

Private Sub UserForm_Activate()
    ' UserForm1.Top = 10
    ' UserForm1.Left = 10
End Sub
```

INPUTBOX mit Sternchen für Passworteingabe

```
Private Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, _
ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long
Private Declare Function GetModuleHandle Lib "kernel32" Alias "GetModuleHandleA" (ByVal lpModuleName As String) As Long
Private Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" _
(ByVal idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, _
ByVal dwThreadId As Long) As Long
Private Declare Function UnhookWindowsHookEx Lib "user32" (ByVal hHook As Long) As Long
Private Declare Function SendDlgItemMessage Lib "user32" Alias "SendDlgItemMessageA" _
(ByVal hDlg As Long, ByVal nIDDlgItem As Long, ByVal wMsg As Long, _
ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function GetClassName Lib "user32" Alias "GetClassNameA" (ByVal hWnd As Long, _
ByVal lpClassName As String, _
ByVal nMaxCount As Long) As Long
Private Declare Function GetCurrentThreadId Lib "kernel32" () As Long
Private Const EM_SETPASSWORDCHAR = &HCC
Private Const WH_CBT = 5
Private Const HCBT_ACTIVATE = 5
Private Const HC_ACTION = 0
Private hHook As Long

Public Function NewProc(ByVal lngCode As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
        Dim RetVal
        Dim strClassName As String, lngBuffer As Long
        If lngCode < HC_ACTION Then
           NewProc = CallNextHookEx(hHook, lngCode, wParam, lParam)
           Exit Function
        End If
        strClassName = String$(256, " ")
        lngBuffer = 255
         If lngCode = HCBT_ACTIVATE Then
           RetVal = GetClassName(wParam, strClassName, lngBuffer)
              If Left$(strClassName, RetVal) = "#32770" Then
              SendDlgItemMessage wParam, &H1324, EM_SETPASSWORDCHAR, Asc("*"), &H0
              End If
         End If
        CallNextHookEx hHook, lngCode, wParam, lParam
End Function

Public Function InputBoxDK(Prompt, Optional Title, Optional Default, Optional XPos, _
        Optional YPos, Optional HelpFile, Optional Context) As String
        Dim lngModHwnd As Long, lngThreadID As Long
        lngThreadID = GetCurrentThreadId
        lngModHwnd = GetModuleHandle(vbNullString)
        hHook = SetWindowsHookEx(WH_CBT, AddressOf NewProc, lngModHwnd, lngThreadID)
```

```
        InputBoxDK = InputBox(Prompt, Title, Default, XPos, YPos, HelpFile, Context)
        UnhookWindowsHookEx hHook
End Function

Sub TESTFRAGE ()

Passwort=InputBoxDK("Bitte Passwort eingeben", "Passwort")
Msgbox Passwort

End Sub
```

INPUTBOX

```
MAILADDY ="default@default.at"
BUTTONTEXT = "NEUE E-MAIL-ADRESSE BEI ROYAL EINGEBEN"
MAILADDY = InputBox(BUTTONTEXT, "ÄNDERUNG E-MAIL-ADRESSE", MAILADDY)
```

Bei den Parametern der Inputbox ist der erste die Meldung die ausgegeben werden soll.
Der zweite Parameter gibt die Überschrift in der Titelleiste an.
Der dritten Parameter beinhaltet den Wert, der im Textfeld ausgegeben werden soll.
Die letzten beiden Parameter geben schließlich die x-Postition und die y-Postition der Inputbox vom linken und oberen Rand des Bildschirmes in Tips an.

INPUTBOX auf bestimmte Typen beschränken (Zahl, Text)

Natürlich kann man den Inhalt einer Inputbox-Abfrage auch nachträglich analysieren und dem User Feedback geben, wenn er statt einer Zahl einen Buchstaben eingegeben hat.

Man kann dies aber auch automatisch von der Inputbox machen lassen, die auch als Funktion für das Application-Obejkt zur Verfügung steht und (nur) dort die Typenprüfung erlaubt – konkret im 8. Parameter TYPE. Wichtig: dieser kann auch kombinierte Werte erhalten, um z.B. Zahlen UND Text zu erlauben

Anmerkungen

In der folgenden Tabelle sind die Werte aufgeführt, die im Argument *Type* übergeben werden können. Dies kann ein Wert oder eine Summe der Werte sein. Legen Sie beispielsweise *Type* auf **1 + 2** fest, damit in einem Eingabefeld sowohl Text als auch Zahlen eingegeben werden können.

Wert	Bedeutung
0	Formel
1	Zahl
2	Text (Zeichenfolge)
4	Wahrheitswert (**True** oder **False**)

8	Zellbezug, z. B. ein **Range**-Objekt
16	Fehlerwert, z. B. **#NV**
64	Wertearray

```
Sub Test()

    Dim Zahl As Double


    Zahl = Application.InputBox("Welche Zahl", "Zahl", , , , , , 1)

    MsgBox "Sie haben gewählt " & Zahl

End Sub
```

Ja/Nein-Frage mit mehrzeiliger Messagebox

```
    If MsgBox("Sind Sie sicher?", vbQuestion + vbYesNo + vbDefaultButton2, "FRAGE") = vbYes Then
…
    End If
```

An erster Stelle kann das angezeigte Symbol eingestellt werden:

vbCritical	Meldung für kritischen Fehler
vbQuestion	Warnung mit Abfrage
vbExclamation	Warnmeldung
vbInformation	Informationsmeldung

An zweiter Stelle kann eingestellt werden, welche Schaltflächen zur Verfügung stehen:

vbOKOnly	Nur Schaltfläche OK (Voreinstellung)
vbOKCancel	Schaltflächen OK und Abbrechen
vbAbortRetryIgnore	Schaltflächen Abbruch, Wiederholen und Ignorieren
vbYesNoCancel	Schaltflächen Ja, Nein und Abbrechen
vbYesNo	Schaltflächen Ja und Nein
vbRetryCancel	Schaltflächen Wiederholen und Abbrechen

An dritter Stelle kann noch der Default-Button gewählt werden:

vbDefaultButton1	Erste Schaltfläche ist Voreinstellung (Voreinstellung)

vbDefaultButton2	Zweite Schaltfläche ist Voreinstellung
vbDefaultButton3	Dritte Schaltfläche ist Voreinstellung
vbDefaultButton4	Vierte Schaltfläche ist Voreinstellung

Der Rückgabewert (im obigen Beispiel direkt abgefragt If Msgbox …= vbyes / im unteren Beispiel über die Variable Wahl ausgewertet) kann folgende Werte haben:

vbOK	Schaltfläche OK gedrückt
vbCancel	Schaltfläche Abbrechen gedrückt
vbAbort	Schaltfläche Abbruch gedrückt
vbRetry	Schaltfläche Wiederholen gedrückt
vbIgnore	Schaltfläche Ignorieren gedrückt
vbYes	Schaltfläche Ja gedrückt
vbNo	Schaltfläche Nein gedrüc

```
Dim BUTTONTEXT As String
Dim Wahl As Integer

BUTTONTEXT = "TEXT ZEILE 1" & vbCrLf
BUTTONTEXT = BUTTONTEXT & vbCrLf & "TEXT ZEILE 3"
Wahl = MsgBox(BUTTONTEXT, vbYesNo, "TITEL DER MESSAGEBOX")

If Wahl = vbNo Then
        end
End if
```

Schließen eines Dialogfensters verhindern

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
'Prevent user from closing with the Close box in the title bar.
If CloseMode <> 1 Then Cancel = 1
End Sub
```

SUCHEN-ERSETZEN-FENSTER ÖFFNEN UND NACH SCHLIESSEN CODE AUSFÜHREN

Es ist nicht ganz einfach Code zu erzeugen, der das Suchen/Ersetzen-Fenster von Excel öffnet und nach dessen Schließen dann mit Code weiter zu machen. hier die Lösung

```
'=====================================================================
```

```
' Fensterhandle ermitteln
'========================================================================
Public Declare Function FindWindow Lib "user32" Alias _
      "FindWindowA" (ByVal lpClassName As String, ByVal _
      lpWindowName As String) As Long

'========================================================================
' Windowstimer setzen
'========================================================================
Declare Function SetTimer Lib "user32" _
    (ByVal hwnd As Long, _
    ByVal nIDEvent As Long, _
    ByVal uElapse As Long, _
    ByVal lpTimerFunc As Long) _
As Long

'========================================================================
' Windowstimer löschen
'========================================================================
Declare Function KillTimer Lib "user32" _
    (ByVal hwnd As Long, _
    ByVal nIDEvent As Long) As Long

Public Const lngAPITIMER    As Long = &H10000
Public lngTIMERID           As Long

Sub FindAll()

    '---------------------------------------------------------------------
    ' Fehlerbehandlung ausschalten
    '---------------------------------------------------------------------
    On Error Resume Next

    '---------------------------------------------------------------------
    ' Ruft den Suchen- und Ersetzendialog auf
    '---------------------------------------------------------------------
    Application.CommandBars.FindControl(ID:=1849).Execute

    '---------------------------------------------------------------------
    ' Startet den Windowstimer und ruft alle 100 Millisekunden die Prozedur
    ' "Timer_Procedure" auf
    '---------------------------------------------------------------------
    lngTIMERID = SetTimer(0, lngAPITIMER, 100, AddressOf Timer_Procedure)
```

```
End Sub

Sub Timer_Procedure()

    '---------------------------------------------------------------------
    ' Fehlerbehandlung ausschalten
    '---------------------------------------------------------------------
    On Error Resume Next

    '---------------------------------------------------------------------
    ' Wird das Fenster "Suchen und Ersetzen" nicht mehr gefunden, wird der
    ' gesetzte Windowstimer wieder gelöscht und die Programmausführung in der
    ' If-Abfrage fortgesetzt.
    '---------------------------------------------------------------------
    If FindWindow("bosa_sdm_XL9", vbNullString) = 0 Then
        KillTimer 0, lngTIMERID
        MsgBox "Ende"
    End If

End Sub
```

Usernamen im Excelfenster-Titel anzeigen

```
Sub FensterName()
ActiveWindow.Caption = ActiveWindow _
.Caption & " " & Application.UserName
End Sub
```

USER-FORMS - LISTENFELDER

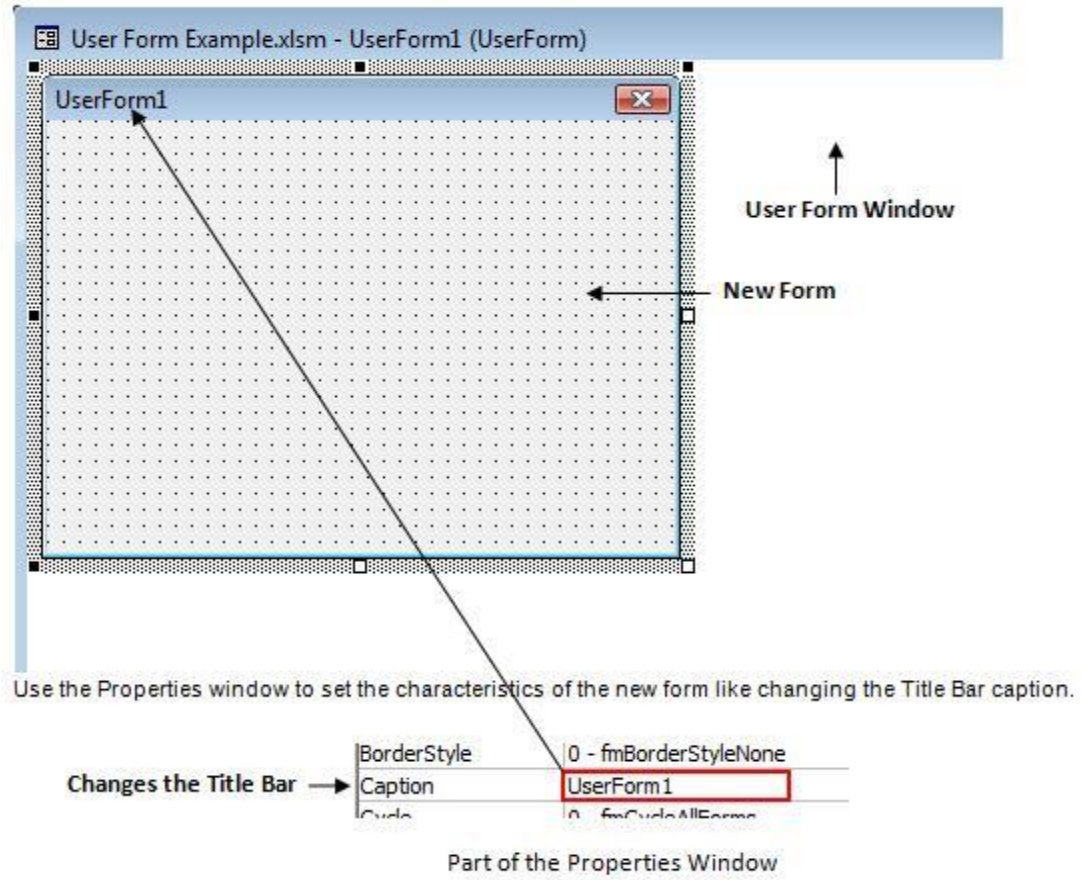
-→ Grundlagen USERFORMS

Pre-defined **Dialog Boxes** like InputBox and MsgBox functions are useful and quick to use. However, designing your own **Dialog Boxes** (or **User Form**), allows you to add other controls and personalise your application.

Creating a new User Form

Make sure you are in the Visual Basic Editor and not Excel.

Select **Insert**, **UserForm**:



Note: To load the Properties Window, press the **F4** function key.

In addition to the **Properties** window, when active on the new form, the **Toolbox** Toolbar automatically appears. (*If this is missing, use the View, Toolbox command to show it.*)

Also, you may want to display the 'UserForm' toolbar to align and rearrange the controls on the form. Select **View**, **Toolbars** and choose **UserForm**.

Userform Toolbar

This toolbar is only available for designing and arranging objects when creating or modifying forms (*user forms*).

1 **Bring to Front** moves the selected object to the front of all other objects.

2 **Send to Back** moves the selected object to the back of all other objects.

3 **Group** two or more selected objects together as one.

4 **Ungroup** where a single object was made up of two or more objects.

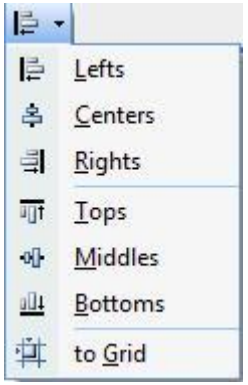
5 **Alignments** of selected objects to various alignments - *see below*.

6 **Vert/Horiz Alignments** of selected objects - *see below*.

7 **Sizes** a number of selected objects to the same dimensions - *see below*.

8 **Zoom** the User Form by magnifying/diminishing by percentage.

Alignments (Button 5)

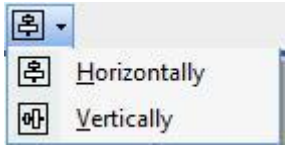


Choose from one of the alignments as to how a number of selected objects will be placed together.

This keeps controls on a form or Dialog Box symmetrically aligned and therefore professional looking.

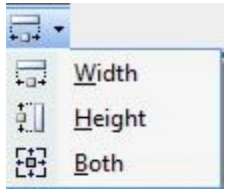
Objects can also be numerically set using the Properties Window.

Vertical/Horizontal Alignments (Button 6)



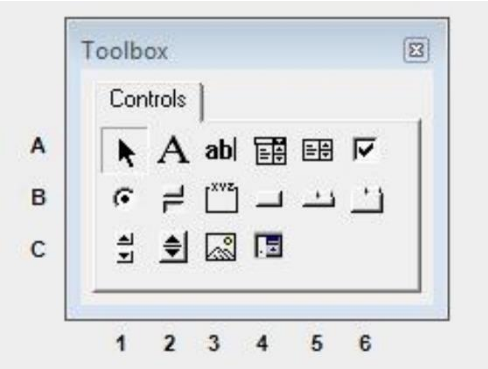
These are two repeated types of alignments as mentioned above allowing objects to be centred.

Sizes (Button 7)



Changes the size of selected objects to the same dimension as each other. This can also be set from the Properties Window.

Toolbox Toolbar



A1 **Select Objects**: When there is no control to draw, this mode allows you to select other controls.

A2 **Label**: Allows you to create text (*caption*) that a user does not change.

A3 **TextBox**: Allows you to create an edit box which a user types into.

A4 **ComboBox**: Allows the user to select from a drop down box items predefined.

A5 **ListBox**: As above but shows many item in one view with a vertical scroll bar.

A6 **CheckBox**: Allows the user to create a CheckBox where an item can only have a yes or no (*true or false*) answer.

B1 **OptionButton**: Allows you to display multiple options with a frame where only one can be selected at a time.

B2 **ToggleButton**: Like a CheckBox, but a button version.

B3 **Frame**: Allows you to create a frame to store controls in one group (*usually option buttons*).

B4 **CommandButton**: Creates a button like the OK, Cancel and Other... Buttons.

B5 **TabStrip**: Allows you to create multiple pages of the same Dialog Box controls.

B6 **MultiPage**: Allows you to create multiple pages of different controls (*multi- tab Dialog Boxes*).

C1 **ScrollBar**: Provides a graphical scroll bar to allow scrolling through a list of values.

C2 **SpinButton**: This button allows you to set values by scrolling up or down through ranges.

C3 **Image**: Allows the user to store graphics in a Dialog Box.

C4 **RefEdit**: Allows a range to be plotted into this control from a spreadsheet. (*Not available in Excel 97*).

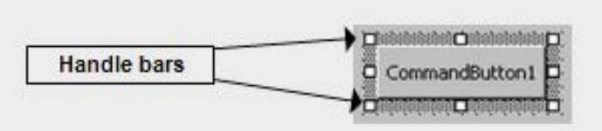
Adding Controls to a Custom Dialog Box

By using the **Toolbox**, standard controls (*command buttons, text boxes and others*) can be added to a user form.

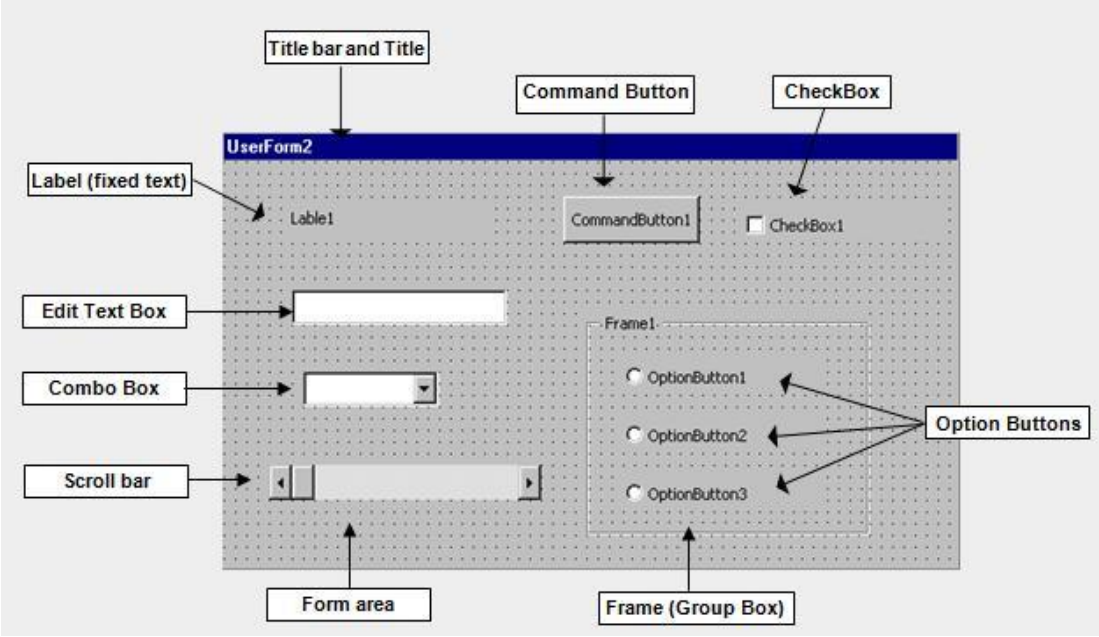
Make sure the form is the active window.

Click on the required control and then click on the user form roughly where the control is to be positioned or drag and drop the control from the **Toolbox** to the area on the form.

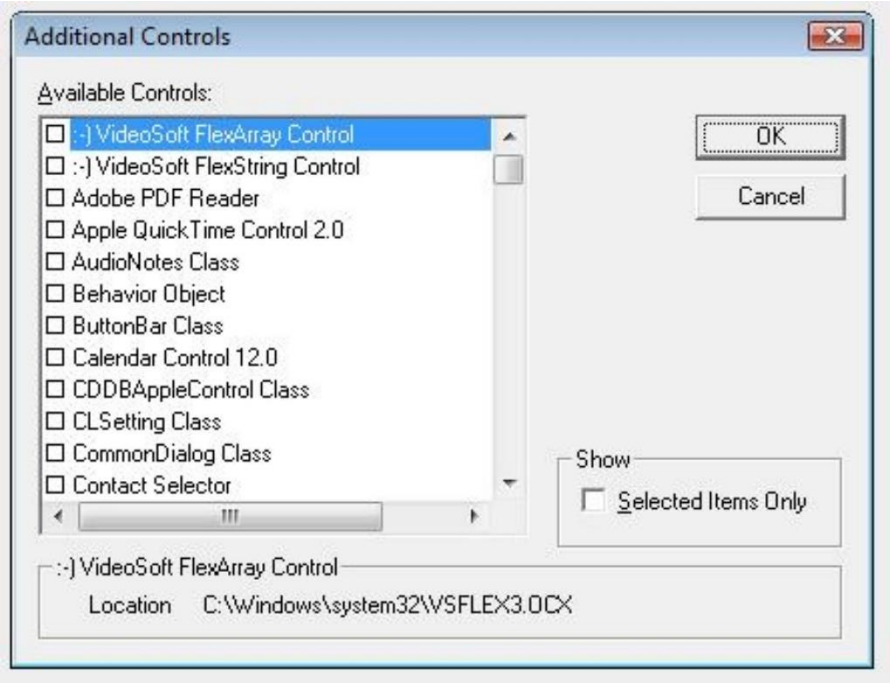
Handlebars appear around the selected control. This allows control(s) to be resized and positioned.



Common Controls



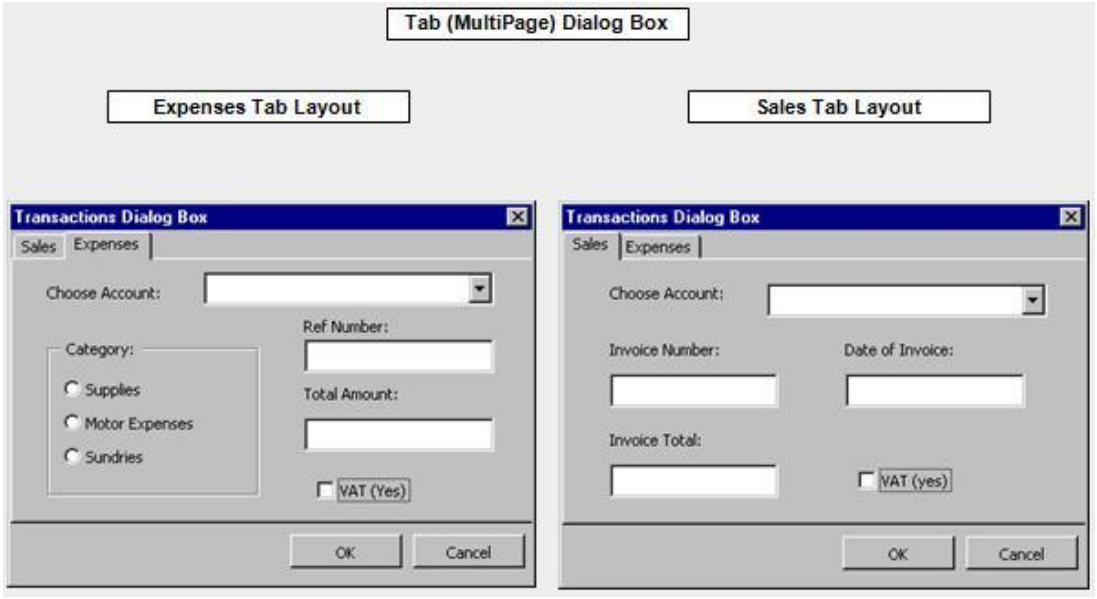
There are many more controls than the standard set which is installed with Microsoft Excel and can be added. Select **Tools**, **Additional Controls....**



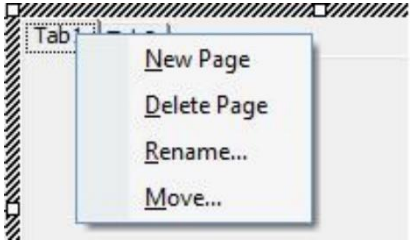
Tick the required item and choose the **OK** Button. This updates the **Toolbox** toolbar.

Creating Tabs for Userforms

If a single user form needs to handle a large number of controls that can be sorted into categories, use a **Tab** Control.



Tab pages can be added and deleted using the properties of the tab (itself) of the control itself by right-mouse clicking the item.



Setting Controls with Properties (*Design Time*)

Some controls are better set during the design side of a user form by using the **Properties** window.

Typical examples of setting these controls:

- Sorting out the Tab Order of controls.
- Setting the default values to edit boxes, checkboxes and many more.

- Creating control tool tips and captions.
- Setting the Accelerator key (*underscored letter*) of a control.
- Other colours and graphics that really do not need code handling.

Initialising Controls with Code (*Run Time*)

Some controls (*some included as above*) can be set as the userform is running or before the form is displayed by setting properties with code.

Typical examples of setting controls with code:

- Setting initial values of controls like an Edit Box or a Combo Box.
- Setting the focus of a control.
- Validating values in a Dialog Box.
- Changing values in a Dialog Box while it is running.
- Showing and hiding other controls.
- Enabling and disabling controls.

These controls are set to different events of a control. These include:

- Click (*and double click*) of a control - command buttons, checkboxes.
- Change of a control - edit boxes.
- Initialising of a control - as the form starts up.
- Exiting a control.

Example:

```
Sub MyDialogBox_Initialize()
```

```
                                TextBox1.Text = "Sales"
                                Checkbox1.Enabled = True
                                                Me.Command1.SetFocus
                                OptionCommand1.Visible = False
                                Checkbox2.Value = False
End Sub
```

Five controls are set when the form (*MyDialogBox*) is shown (*Initialised*).

1. **TextBox1** displays “Sales” in it.
2. **Chexkbox1** control is active.
3. **Command1** button has the focus.
4. **OptionCommand1** is not visible (*hidden*).
5. **Checkbox2** is not ticked (*False value*).

Displaying a Userform

Once the userform has been created, the next stage is to test to see how the user form will look. You can use the **Run** (**F5** function key in design time mode) command when the active form is displayed. But, writing code is ultimately how a user form will be used.

Decide where the code is to be stored (*in a Module, Worksheet or Workbook*).

Use the name of the form with the **Show** method command.

Example:

```
Sub DisplayMyUserform()
    MyUserForm.Show

End Sub
```

The user form is known, as ‘*MyUserForm’* and the **Show** method will display the user form.

There is one optional argument called **Modal** which can be explicitly defined and has a value of **0** or **1** .

1 = a modal state which means that users have to complete the form and can not click anywhere else (in the background).

0 = a modeless state which allows users to click outside the form area.

The default is **1** if omitted.

MyUserForm.Show 0

MyUserForm.Show 1 *or* **MyUserForm.Show**

Adding Code to respond with User Forms

Each control will have its own set of events. These events store the code and are executed when that event is triggered.

For example, a **Button** recognises the *Click_Event*, a **Form** recognises an *Intitialize_Event* and a **Combo Box** recognises a *Change_Event*.

To assign code to a control, display the form and double click on that control. This opens the module and the main event allowing code to be written:

Example:

When the **OK** Button is clicked...

```
Private Sub cmdOK_Click()
    Range("NameResult").Select
    EnterText.Hide
    If txtName.Text = "" Then
        MsgBox "Must enter a name. Try again."
        EnterText.Show
    End If
    ActiveCell.Value = txtName
    Unload Me
End Sub
```

When the userform is displayed, if the **OK** Button is clicked, the above code is executed and checks to see if this Textbox (*txtName*) is empty or not using the **If** statement. If false, it displays a message prompt and shows the user form again. If true, it enters the data into a spreadsheet (*range - NameResult*).

Unload Me is the way to close a form (itself)

The Me Property

The **Me** property returns a reference of the form itself that the code is currently running. This is used as shorthand for the full reference of a form.

Example:

Suppose you have the following procedure in a module:

```
Sub ChangeFormColour(FormName As Form)
    FormName.BackColor = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub
```

You can call this procedure and pass the current instance of the Form as an argument using the following statement:

```
Sub cmdColour_Click()
    ChangeFormColour Me
End Sub
```

The **ChangeFormColour** procedure is passed to the **Me** property in the current form running which therefore changes the colour of a specified control(s) to the colour defined.

`RGB(Rnd * 256, Rnd * 256, Rnd * 256)` is a Red, Green and Blue colour function.

Userforms allgemein – Fokus auslesen, setzen ...

Aktvierreihenfolge

Die Reihenfolge, wie die verschiedenen Formularelemente angesprungen werden, legt man fest im Menü Ansicht / Aktivierreihenfolge - diese kann aber auch mittels VBA festgelegt werden (siehe Tabindex-Befehl)

Aktuelles Userformelement auslesen

MsgBox Userform1.ActiveControl.Name

Mit MsgBox Userform1.ActiveControl wird der Wert ausgelesen, wenn es sich dabei z.B. um ein Textfeld / Label etc handelt.

Focus auf bestimmtes Element der Userform setzen

TextBox3.SetFocus ' setzt den Schreibcursor auf bestimmtes Element - hier Textbox

Inhalt eines Elements gleich markieren vor Ausfüllen

Möchte man den Text in der Textbox3 nach dem Setfocus-Vorgang gleich auch noch markieren, damit er beim Ausfüllen ersetzt wird, so geht das so:

```
With TextBox3
   .SelStart = 0
   .SelLength = Len(.TEXT)
End With
```

Die EnterFieldBehavior-Eigenschaft eines Feldes legt fest, ob beim Anspringen mit TAB automatisch der Text markiert werden soll. Dies gilt aber nicht für Anspringen mittels SETFOCUS und auch nicht, wenn man mit der Maus draufclickt.

Will man mit einem Mausclick auf ein Feld den Text markieren:

```
Private Sub TextBox1_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   With TextBox1
   .SelStart = 0
   .SelLength = Len(.Text)
   End With
End Sub
```

Eine andere Sache ist das Auffangen eines Ereignisses, wenn ein ActiveX-Element (Kombinationsfeld / Textfeld / Listfeld ...) direkt in einer Tabelle ist und man dort abfangen will, wenn es ausgewählt wird:

```
Private Sub Combobox24_GotFocus()
    [code]
End Sub
```

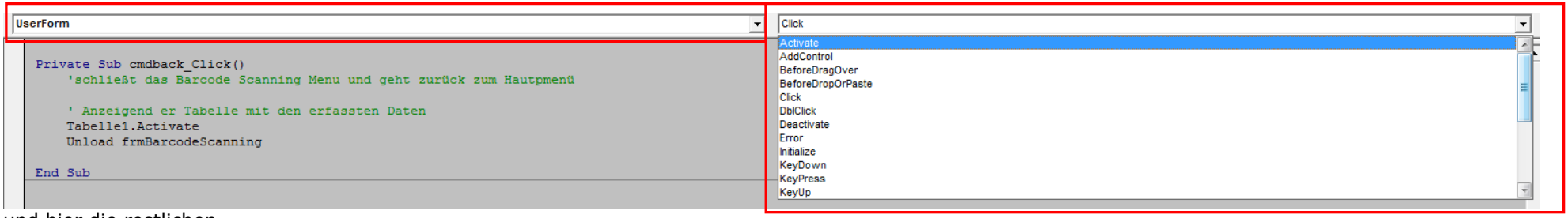
Eine Veriante, die meiner Erfahrung nach besser klappt als Gotfocus ist folgende. Das Problem ist ja immer, wie kam der User in das Feld: durch Mausclick, durch ENTER, durch Tab ...

```
Private Sub Combobox24_Enter()
    [code]
End Sub
```

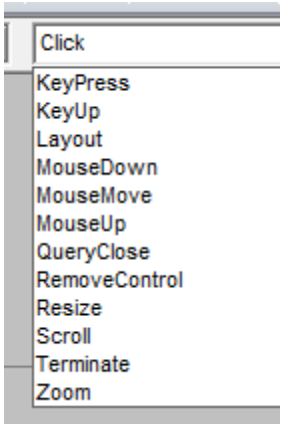
Activecontrol.Name enthält den Namen des Formularelementes, das den Fokus enthält

ComboBox1.ListIndex = 1 ' setzt im Kombinationsfeld das vorausgewählte Element

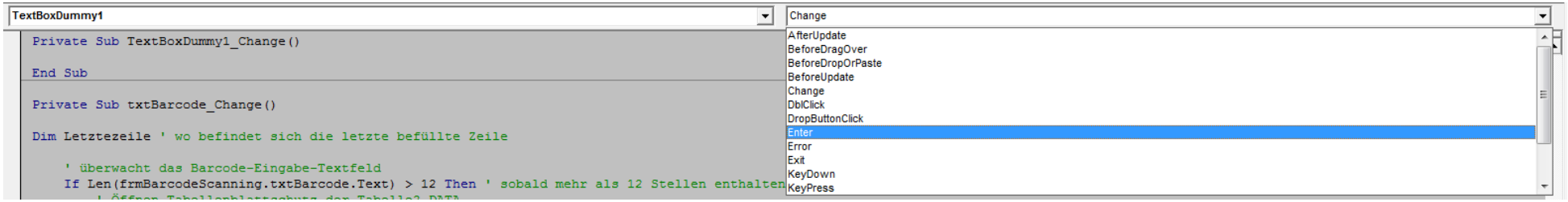
Wie beim Worksheet-Element, gibt es auch ein allgemeines USERFORM-Element, wo man oben zur Auswahl der möglichen Prozeduren kommt:



und hier die restlichen

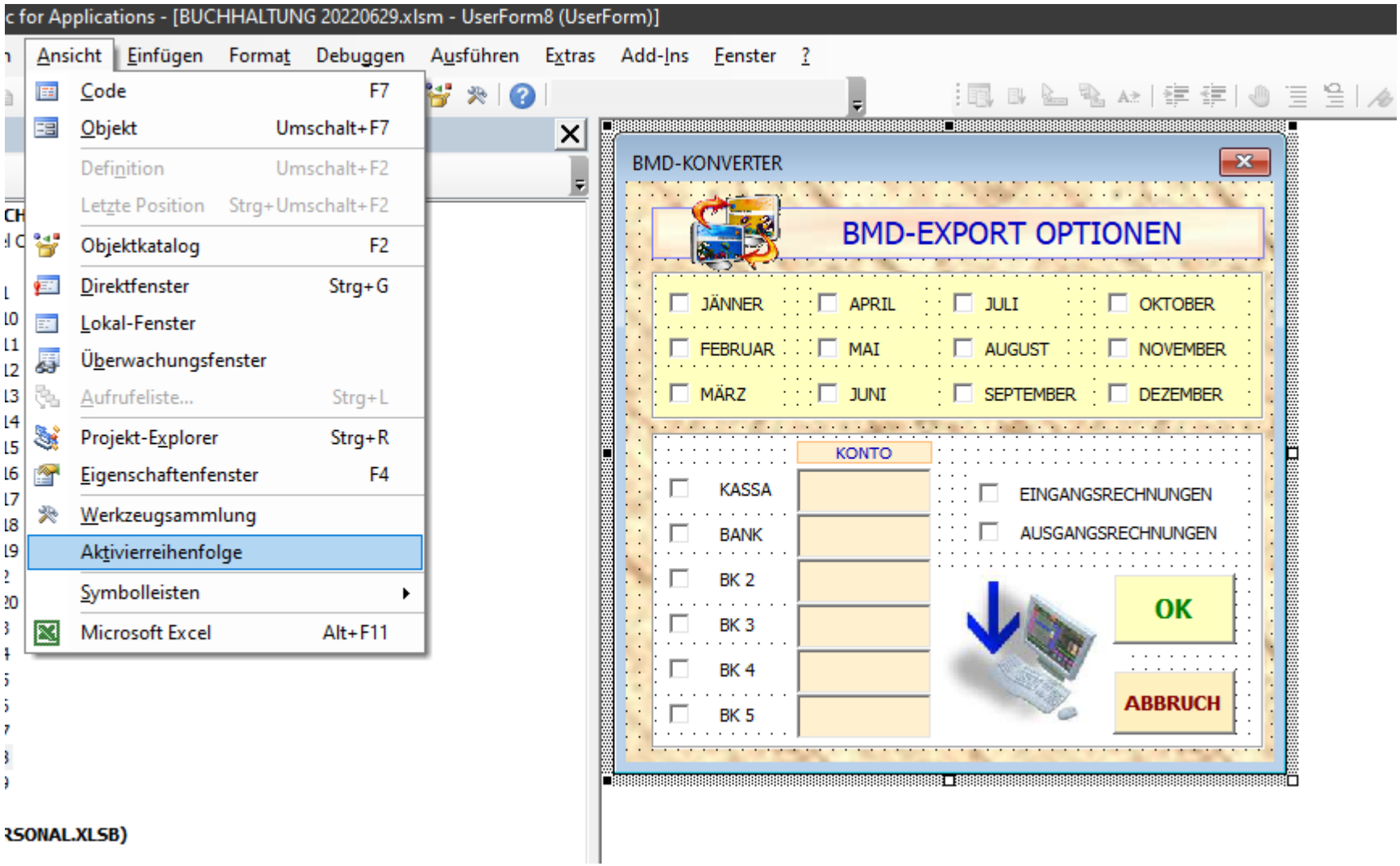


Wählt man IRGENDEINEN Code von einem Userformelement ein, kann man oben einblenden, welche andere Code-Möglichkeiten es noch gibt – hier z.B. von einem Textfeld:



Aktivierreihenfolge der Elemente in Userform einstellen

Es darf nur die Userform angewählt sein und kein Element



Nun kann man in der Userform eventuelle Frames in die richtige Reihenfolge bringen.

Danach markiert man den gewünschten Frame und geht wieder in die Aktivierreihenfolge und kann nun das gewünschte Element des Frames in die gewünschte

Reihenfolge bringen.

Nachträglich den Fokus auf ein Element setzen geht mit

TextBox3.SetFocus ' setzt den Schreibcursor auf bestimmtes Element - hier Textbox

Aktuelles Userformelement auslesen

MsgBox Userform1.ActiveControl.Name

Allgemeines BSP von mir

```
Sub USTRechner()

' ZURÜCKSETZEN DES FORMULARS

'    UserForm1.TextBox1.Value = ""
'    UserForm1.TextBox2.Value = ""
'    UserForm1.TextBox3.Value = "0"
'    UserForm1.Label7.Caption = "KONTROLLE"

 ' ANZEIGEN DES FORMULARS

    Userform7.Show

End Sub

Private Sub CommandButton2_Click()
    Unload Userform7
End Sub
```

Ausrechnen von Usereingaben mit + - x und /

Public Sub ZELLKONVERTER()

```
    ' Dient für die Userforms um Beträge mit Rechenoperationen auszuwerten (zB 2,14+3,14) und wird von dort aus aufgerufen
    Dim i As Integer ' Laufvariable
    Dim RECHNEN As Integer ' merkt sich, ob die Eingabe eine Rechenoperation enhält
    Dim ZEICHEN As String ' zum Zerschneiden des Strings in einzelne Zeichen
    Dim ERGEBNIS As String ' zum Umwandeln von Variable ZELLEKONVERTIERT
    ZELLEKONVERTIERT = ""
    ERGEBNIS = ""
    RECHNEN = 0
    ' Abschneiden eventueller Rechenzeichen (+,-,=) am Ende der Eingabe
    If Right(ZELLEINGABE, 1) = "=" Or Right(ZELLEKONVERTIERT, 1) = "+" Or Right(ZELLEKONVERTIERT, 1) = "-" Then
       ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)
    End If

    ' Leider hat "evaluate"-Befehl einen Fehler - ein "evaulate ("24")" ergibt 90 und nicht 24 =>
    ' evaluate-Befehl nur ausführen, wenn eine Rechenoperation im String enthalten ist
    ' weiters verdaut "evaluate" keine "," - er nimmt nur "."
    ' und zudem löscht Excel beim Übergeben einer Zahl, die ein "." enthält, an eine Variable
    ' diesen Punkt (aus "2.12" wird 212)
    ' netterweise gibt der Evaluate-Befehl, der zwar "." braucht, das Ergebnis korrekt wieder mit "," aus
    ' => Im zu berechnenden String nur dann die "," in "." umwandeln, wenn anschließend ein Evaluate erfolgt


' Wenn Zelleingabe mit einem + o. - endet : hinzufügen von 0, damit gerechnet werden kann - aber das Ergebnis sich nicht ändert

    If Right(ZELLEINGABE, 1) = "+" Or Right(ZELLEINGABE, 1) = "-" Then ZELLEINGABE = ZELLEINGABE + "0"

' Wenn Zelleingabe mit * oder / endet : hinzufügen von 1, damit gerechnet werden kann, aber Ergebnis gleich bleibt

    If Right(ZELLEINGABE, 1) = "*" Or Right(ZELLEINGABE, 1) = "/" Then ZELLEINGABE = ZELLEINGABE + "1"

' Wenn Zelleingabe mit "/0" endet : abschneiden, damit gerechnet werden kann

    If Right(ZELLEINGABE, 2) = "/0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 2)

' Wenn Zelleingabe mit "/0," endet : abschneiden, damit gerechnet werden kann

    If Right(ZELLEINGABE, 3) = "/0," Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 3)

' Wenn Zelleingabe mit "/0,0" endet : abschneiden, damit gerechnet werden kann
```

```
    If Right(ZELLEINGABE, 4) = "/0,0" Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 4)

' Wenn Zelleingabe mit " " endet - abschneiden

    If Right(ZELLEINGABE, 1) = " " Then ZELLEINGABE = Left(ZELLEINGABE, Len(ZELLEINGABE) - 1)

' KONTROLLE OB EINE RECHENOPERATION ENTHALTEN IST (nur dann wird Evaluate ausgeführt)
    For i = 1 To Len(ZELLEINGABE)
       ZEICHEN = Mid(ZELLEINGABE, i, 1)
       If ZEICHEN = "+" Or ZEICHEN = "-" Or ZEICHEN = "*" Or ZEICHEN = "/" Then RECHNEN = 1
    Next i

' WENN RECHENOPERATION ENTHALTEN DANN VOR EVALUATE DAS "," in "." umwandeln
    If RECHNEN = 1 Then

       ' Zeichenweise "umschaufeln" in Zielvariable und aus "," einen "." machen
       For i = 1 To Len(ZELLEINGABE)
          If Mid(ZELLEINGABE, i, 1) = "," Then
             ERGEBNIS = ERGEBNIS + "."
          Else
             ERGEBNIS = ERGEBNIS + Mid(ZELLEINGABE, i, 1)
          End If
       Next
       ERGEBNIS = Evaluate(ERGEBNIS)
      Else
       ERGEBNIS = ZELLEINGABE
    End If
   If ERGEBNIS = "" Then ERGEBNIS = "0"
   ERGEBNIS = Str(Round(ERGEBNIS, 2))


 ' Anschließend das Ergebnis, falls mit "." wieder umwandeln auf ","

        For i = 1 To Len(ERGEBNIS)
          If Mid(ERGEBNIS, i, 1) = "." Then
             ZELLEKONVERTIERT = ZELLEKONVERTIERT + ","
          Else
             ZELLEKONVERTIERT = ZELLEKONVERTIERT + Mid(ERGEBNIS, i, 1)
          End If
       Next

End Sub
```

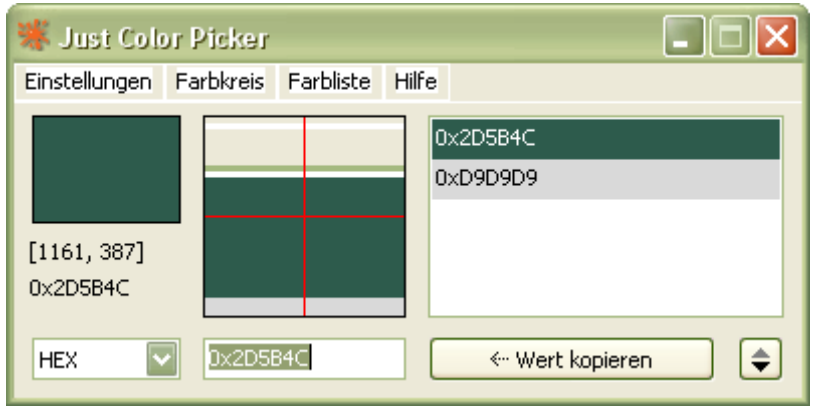
Befehlsschaltfläche - Fokus auf sie in Userform setzen

Möchte man den Fokus per VBA in einer Userform auf einen Commandbutton setzen, damit dieser z.B. vom User direkt mit ENTER gedrückt werden kann, geht dies ganz einfach:

UserForm1.CommandButton194.SetFocus

Farbcodes von Userform-Elementen

Am besten die Farben auslesen mit dem JUST COLOR PICKER



Und dort die Anzeige auf HEX einstellen

In VBA muss die Reihenfolge der 3 2-er Gruppen umgedreht werden: aus oben 2D5B4C wird 4CB52D

Die wesentlichen "Schrauben" für manuelles Drehen sind also die Stellen 5-10 und die Werte gehen von 00 bis FF. Die erste Position ist das Blau, dann das Grün und dann das Rot in VBA.

Labeltext vertikal zentrieren

Userform-Labels können nur horizontal zentriert werden per Code. Entweder man macht die Schriftgröße so groß, dass der Text wieder mittig wird oder man verschiebt mit Leerzeichen den Text in die zweite Zeile

ODER aber die schönste Lösung

mache zwei Userforms – eine, die mit dem eigentlichen Text, ist auf transparent gestellt im Feld BACKSTYLE – den VBA-Code gibt man beiden gleich.

Listenfelder (in Tabellenblatt oder Userform)

1.) **Listboxen, die man direkt im Tabellenblatt hat:**

```
Dim LetztezeileSK as Integer ' Nummer der letzten Zelle im Tabellenblatt "SK"
 Worksheets("SK").Select
 Range("G65536").End(xlUp).Select  ' anwählen der letzten benutzen Zelle in Spalte G
 LetztezeileSK = ActiveCell.Row  ' Auslesen der Zeilennummer dieser letzten Zelle
 Worksheets("JAN").Activate
 ActiveSheet.Shapes("Drop Down 24").ControlFormat.ListFillRange = "SK!$G$10:$G$" & LetztezeileSK
```

Hier wird zB für das Listenfeld 24 im Tabellenblatt JAN der Zellbezug für die Auflistdaten eingestellt auf das Tabellenblatt "SK" und dort die Einträge in der Spalte G von 10 bis zum Wert der Variable "LetztezeileSK", sprich der Zeile mit der letzten Zelle in Spalte G

Listboxen direkt in einem Programm kann man mit rechtem Mausclick mit einem Makro verknüpfen, das bei Auswahl der Listbox ausgeführt wird. Ich schreibe meist den Wert des Listbox-Outputwertes direkt in die Zelle unter die Listbox. Auf diesen Wert greife ich mittels des Makros zu.

2.) **Listboxen (Combobox), die man in einem Formular hat:**
Wieder könnte das Listing gleich wie obiges sein - nur der Bezug wird anders geändert

```
UserForm2.ComboBox1.RowSource = "PK!$G$10:$G$" & LetztezeileSK
```

UserForm1.ComboBox1.Value enthält den Wert des ausgewählten Kombinationsfeldeintrages
UserForm1.ComboBox1.ListIndex enthält die Zahl des wievielten Eintrages (minus 1, da es mit 0 beginnt)

Das Problem mit dem gleichbleibenden Listfeldwert, der das Makro nicht aufruft.

ActiveSheet.Shapes("Drop Down 24").ControlFormat.ListIndex = "1"
bzw
ComboBox1.ListIndex = 0

setzt den ersten Listfeldeintrag als voreingestelltes Preset.
Mit "0" wird kein Listfeldeintrag angezeigt. Das Problem ist leider, dass das Makro hinter dem Listfeld nur ausgeführt wird, wenn man ein anderes als den angezeigten Listfeldeintrag anwählt. Wählt man etwa den Eintrag Nummer 5 an = wird dieser angezeigt und würde man beim nächsten Mal wieder den Eintrag 5 anwählen wollen, würde das Makro nicht ausgeführt werden. (Das macht Sinn, wenn man mit einem Listfeld ohnedies nur eine einzige Zelle ändern will - aber nicht, wenn man damit verschiedene Zellen ändern will) => das Makro hinter dem Listfeld muss nach dessen Ausführung den Presetwert des Listfeldes auf 0 (=nichts) oder auf den ersten Eintrag einstellen, so dieser eine Überschrift und gar kein wirklicher Eintrag ist.

LISTFELD ODER ANDERES ELEMENT SOLL IMMER EIN MAKRO AUFRUFEN (onaction-Befehl)

Normalerweise kann man mit rechtem Mauschlick einstellen, welches Makro aufgerufen werden soll, wenn man auf ein Element in einem Tabellenblatt clickt - egal ob Listenfeld etc.

Man kann dies aber auch via VBA einstellen - dazu gibt man im Worksheet-Code des betreffenden Tabellenblattes folgenden Code ein

```
Private Sub Worksheet_Activate()
  Shapes("Drop Down 24").OnAction = "LISTFELDCLICK"
End Sub
```

Sobald das betreffende Tabellenblatt aktiviert wird (o. anfangs geöffnet wird), wird dem Element "Drop Down 24" (also ein Listfeld in diesem Beispiel) das Makro "LISTFELDCLICK" zugewiesen. Sobald man etwas im Listfeld auswählt, wird nun immer dieses Makro (das man normal im Modul1 programmiert und nicht im Worksheetcode) ausgeführt.

LISTBOXEN IN FORMULAR - NACH ANWÄHLEN MIT MAUS, SOLL EREIGNIS FOLGEN

unbedingt _Mouseup wählen und nicht _MouseDown, da nur bei ersterem schon der Auswahlcursor wirklich auf die Wahl des Listfeldes gesetzt wird und abgerufen werden kann (Mouseup= wenn Maus losgelassen wird)

```
Private Sub ListBox1_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
        [code]
End Sub
```

ABFRAGE VON TASTEN

Diese Prozedur - hinterlegt bei einem Listenfeld - fragt ab, wenn eine Taste gedrückt und WIEDER LOSGELASSEN wurde und führt den Code aus

```
Private Sub ListBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
        MsgBox KeyCode ' 13 für Enter , 40 für Cursor down etc
End Sub
```

VBA-Beispiele

Hier zwei kleine VBA-Beispiele, die zeigen sollen, wie man beim Öffnen einer Exceldatei eine gedrückte Taste abfragen kann - die Abfrage erfolgt dann nur einmalig beim Öffnen der Vorlage - also ob während dem Öffnen der Vorlage gerade zusätzlich eine Taste gedrückt wurde.. Fügen Sie den Programmcode in das Modul "DieseArbeitsmappe" ein, speichern und schliessen Sie dann die Arbeitsmappe. Beim nächsten Öffnen werden die Tasten geprüft.

Beispiel 1

```
Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer

Private Sub Workbook_Open()
  Dim strMessage As String
  strMessage = ""
  If Abs(GetKeyState(16) < 0) Then
    strMessage = strMessage & "Umschalt-Taste wurde gedrückt" & vbCrLf
  Else
    strMessage = strMessage & "Umschalt-Taste wurde nicht gedrückt" & vbCrLf
  End If
  If Abs(GetKeyState(17) < 0) Then
    strMessage = strMessage & "Strg-Taste wurde gedrückt" & vbCrLf
  Else
    strMessage = strMessage & "Strg-Taste wurde nicht gedrückt" & vbCrLf
  End If
  If Abs(GetKeyState(18) < 0) Then
    strMessage = strMessage & "Alt-Taste wurde gedrückt"
  Else
    strMessage = strMessage & "Alt-Taste wurde nicht gedrückt"
  End If
  MsgBox strMessage
End Sub
```

Beispiel 2

```
Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer

Private Sub Workbook_Open()
  If Abs(GetKeyState(17) < 0) And Abs(GetKeyState(66) < 0) Then
    MsgBox "Tastenkombination Strg+B wurde gedrückt"
  Else
    MsgBox "Tastenkombination Strg+B wurde nicht gedrückt"
  End If
End Sub
```

Optionsschaltflächen alle deaktivieren

Wichtig: sie müssen lückenlos aufsteigend existieren – hier von 1 bis 32

```
    For I = 1 To 32
        UserForm1.Controls("Optionbutton" & I).Value = False
```

```
    Next I
```

Optionsschaltflächen überprüfen, dass etwas angewählt ist

Unsere 32 Optionsschaltflächen verteilen sich auf 8 x 4 Gruppen und überall muss EINE Schaltfläche aktiviert sein – daher insgesamt braucht es 8 TRUE

```
    For I = 1 To 32
            if UserForm1.Controls("Optionbutton" & I).Value = True then Check=Check+1
    Next I

Msgbox Check
```

Position einer Userform festlegen

```
Private Sub UserForm_Activate()
    With Me
        '0: Manuell
        '1: Fenstermitte
        '2: Bildschirmmitte
        '3: Windows-Standard
        .StartUpPosition = 2
        .Top = 100
        .Left = 100
    End With
End Sub
```

Version 0a

```
Userform1.Top = Application.Top + AcitveWindow.Top + ActiveCell.Top + y
Userform1.Left = Application.Left + AcitveWindow.Left + ActiveCell.Left + ActiveCell.Width + x
```

Version 0b

```
Private Sub UserForm_Activate()
Me.Top = ActiveWindow.Top
Me.Left = ActiveWindow.Left + ActiveWindow.Width - Me.Width
End Sub
```

Version 0c

Userform positioniert starten 03.06.2002

Eine Userform muss nicht immer zentriert auf dem Bildschirm erscheinen, einige API-Aufrufe helfen uns dabei die Position selbst festzulegen:

```
'Bildschirmauflösung
Private Declare Function GetSystemMetrics Lib _
   "user32" (ByVal nIndex As Long) As Long
Private Const SM_CYSCREEN As Long = 1
Private Const SM_CXSCREEN As Long = 0
'Position
Private Declare Function FindWindow Lib "user32" Alias _
      "FindWindowA" (ByVal lpClassName As String, ByVal _
      lpWindowName As String) As Long
Private Declare Function MoveWindow Lib "user32" (ByVal hwnd _
      As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As
Long, _
      ByVal nHeight As Long, ByVal bRepaint As Long) As Long

'Starten einer Userform im Vollbildmodus
Private Sub UserForm_Activate()
    BildschirmBreite = GetSystemMetrics(SM_CXSCREEN)
    BildschirmHöhe = GetSystemMetrics(SM_CYSCREEN)
    wHandle = FindWindow(vbNullString, Me.Caption)
    MoveWindow wHandle, 0, 0, BildschirmBreite, BildschirmHöhe, 1
End Sub
```

Ein weiteres Beispiel, starten der Userform in der rechten oberen Bildschirmecke mit einer Breite von 350 und einer Höhe von 450 Pixel:

```
'Rechte obere Ecke des Bildschirms, 350x450 Pixel groß
Private Sub UserForm_Activate()
```

```
    BildschirmBreite = GetSystemMetrics(SM_CXSCREEN)
    BildschirmHöhe = GetSystemMetrics(SM_CYSCREEN)
    UserformBreite = 350
    UserformHöhe = 450
    wHandle = FindWindow(vbNullString, Me.Caption)
    MoveWindow wHandle, BildschirmBreite - UserformBreite, _
       0, UserformBreite, UserformHöhe, 1
End Sub
```

Version 1

```
Private Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" ( _
    ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long
Private Declare Function MoveWindow Lib "user32.dll" ( _
    ByVal hwnd As Long, _
    ByVal X As Long, _
    ByVal Y As Long, _
    ByVal nWidth As Long, _
    ByVal nHeight As Long, _
    ByVal bRepaint As Long) As Long
Private Declare Function GetSystemMetrics Lib "user32.dll" ( _
    ByVal nIndex As Long) As Long

Private Const SM_CXSCREEN = 0&
Private Const SM_CYSCREEN = 1&
Private Const GC_CLASSNAMEUSERFORM = "ThunderDFrame"

Private Sub UserForm_Activate()

    Dim lngTop As Long, lngLeft As Long, lngHwnd As Long
    Dim lngScreenWidth As Long, lngScreenHeight As Long
    Dim lngFormWidth As Long, lngFormHeight As Long

    lngScreenWidth = GetSystemMetrics(SM_CXSCREEN)
    lngScreenHeight = GetSystemMetrics(SM_CYSCREEN)

    lngFormWidth = Width / 0.75
    lngFormHeight = Height / 0.75

    With ActiveCell
        lngLeft = ActiveWindow.PointsToScreenPixelsX(.Left / 0.75)
        lngTop = ActiveWindow.PointsToScreenPixelsY(.Top / 0.75)
    End With

    lngHwnd = FindWindow(GC_CLASSNAMEUSERFORM, Caption)
```

```
    If lngLeft + lngFormWidth <= lngScreenWidth And lngTop + lngFormHeight <= lngScreenHeight Then
        Call MoveWindow(lngHwnd, lngLeft, lngTop, lngFormWidth, lngFormHeight, 1)
    ElseIf lngLeft + lngFormWidth <= lngScreenWidth And lngTop + lngFormHeight > lngScreenHeight Then
        Call MoveWindow(lngHwnd, lngLeft, lngTop - lngFormHeight, lngFormWidth, lngFormHeight, 1)
    ElseIf lngLeft + lngFormWidth > lngScreenWidth And lngTop + lngFormHeight <= lngScreenHeight Then
        Call MoveWindow(lngHwnd, lngLeft - lngFormWidth, lngTop, lngFormWidth, lngFormHeight, 1)
    Else
        Call MoveWindow(lngHwnd, lngLeft - lngFormWidth, lngTop - lngFormHeight, lngFormWidth, lngFormHeight, 1)
    End If

End Sub
```

Version 2

```
Private Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" ( _
    ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long

Private Declare Function MoveWindow Lib "user32.dll" ( _
    ByVal hwnd As Long, _
    ByVal X As Long, _
    ByVal Y As Long, _
    ByVal nWidth As Long, _
    ByVal nHeight As Long, _
    ByVal bRepaint As Long) As Long

Private Sub UserForm_Activate()
    Dim lngWidth As Long, lngHeight As Long, llngHwnd As Long
    llngHwnd = FindWindow("ThunderDFrame", Me.Caption)
    With ActiveCell
        lngWidth = ActiveWindow.PointsToScreenPixelsX((.Left + .Width) / 0.75)
        lngHeight = ActiveWindow.PointsToScreenPixelsY(.Top / 0.75)
    End With
    Call MoveWindow(llngHwnd, lngWidth, lngHeight, Me.Width / 0.75, Me.Height / 0.75, 1)
```

Version 3

Hier noch ein Vorschlag. Ragt die UF rechts über den Windowsrand des XL-Fensters, wird sie links von der Zelle angezeigt:

Code:

```
Option Explicit

Private Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" (
    ByVal lpClassName As String,
    ByVal lpWindowName As String) As Long

Private Declare Function MoveWindow Lib "user32.dll" (
```

```
    ByVal hwnd As Long,
    ByVal X As Long,
    ByVal Y As Long,
    ByVal nWidth As Long,
    ByVal nHeight As Long,
    ByVal bRepaint As Long) As Long


Private Sub UserForm Activate()
    Dim lLeft As Long, lTop As Long, llngHwnd As Long
    llngHwnd = FindWindow("ThunderDFrame", Me.Caption)
    With ActiveCell
        lLeft = ActiveWindow.PointsToScreenPixelsX((.Left + .Width) / 0.75)
        If (lLeft + Me.Width / 0.75) > ((Application.Left + ActiveWindow.VisibleRange.Width +
ActiveWindow.VisibleRange.Left) / 0.75) Then
            lLeft = ActiveWindow.PointsToScreenPixelsX((.Left - Me.Width) / 0.75)
        End If
        lTop = ActiveWindow.PointsToScreenPixelsY(.Top / 0.75)
    End With
    Call MoveWindow(llngHwnd, lLeft, lTop, Me.Width / 0.75, Me.Height / 0.75, 1)
End Sub
```

Form Positioner

Excel typically displays forms in the center of the screen. This is usually fine for data entry and dialog forms. However, in many cases it is desirable to display a form in

relation to a specific cell. This is not as simple a task as it seems because the top and left coordinates of a UserForm are not based on the same coordinate system as the Top and Left coordinates of a cell. To properly calculate the Top and Left coordinates of a UserForm, you have to take in to account the window state (normal or maximized) of the Excel application window, and the Workbook window, and their relative positions, in addition to whether the formula bar is visible, what command bars are displayed, and how they are positioned.

Needless to say, these calculation can get rather complicated. Fortunately, I've done the work for you. Download the FormPositioner workbook. NOTE: FormPositioner can be used only with VBA UserForms -- it will not work with VB Forms.

Note also that FormPositioner does *not* work on split windows or frozen panes. Support for split windows and frozen panes will be added in a later release.

Copy the module modFormPositioner in to your project. Then, do the following:

Set the StartupPosition property of your form to 0 - Manual. This is *very* important.

Declare a variable of type Positions:

```
Dim PS As Positions
```

Call the PositionForm function, passing it the following parameters. The PositionForm function returns a Positions structure.

WhatForm	The userform object
AnchorRange	The cell relative to which the form should be displayed.
NudgeRight	Optional: Number of points to nudge the for to the right. This is useful with bordered range. Typically, this should be 0, but may be positive or negative.
NudgeDown	Optional: Number of points to nudge the for downward. This is useful with bordered range. Typically, this should be 0, but may be positive or negative.
HorizOrientation:	Optional: One of the following values:

cstFhpNull = Left of screen
cstFhpAppCenter = Center of Excel screen
cstFhpAuto = Automatic (recommended and default)

cstFhpFormLeftCellLeft = left edge of form at left edge of cell
cstFhpFormLeftCellRight = left edge of form at right edge of cell
cstFhpFormLeftCellCenter = left edge of form at center of cell

cstFhpFormRightCellLeft = right edge of form at left edge of cell
cstFhpFormRightCellRight = right edge of form at right edge of cell
cstFhpFormRightCellCenter = right edge of form at center of cell

cstFhpFormCenterCellLeft = center of form at left edge of cell
cstFhpFormCenterCellRight = center of form at right edge of cell
cstFhpFormCenterCellCenter = center of form at center of cell

VertOrientation Optional: One of the following values:

cstFvpNull = Top of screen
cstFvpAppCenter = Center of Excel screen
cstFvpAuto = Automatic (recommended and default)

cstFvpFormTopCellTop = top edge of form at top edge of cell
cstFvpFormTopCellBottom = top edge of form at bottom edge of cell
cstFvpFormTopCellCenter = top edge of form at center of cell

cstFvpFormBottomCellTop = bottom edge of form at top of edge of cell
cstFvpFormBottomCellBottom = bottom edge of form at bottom edge of cell
cstFvpFormBottomCellCenter = bottom edge of form at center of cell

cstFvpFormCenterCellTop = center of form at top of cell
cstFvpFormCenterCellBottom = center of form at bottom of cell
cstFvpFormCenterCellCenter = center of form at center of cell

For example:

```
PS = PositionForm (UserForm1,Range("C12"),0,0,cstFvpAuto,cstFhpAuto)
```

Then, position the form using the values from PS:

```
UserForm1.Top = PS.FrmTop
UserForm1.Left = PS.FrmLeft
```

Finally, show the form:

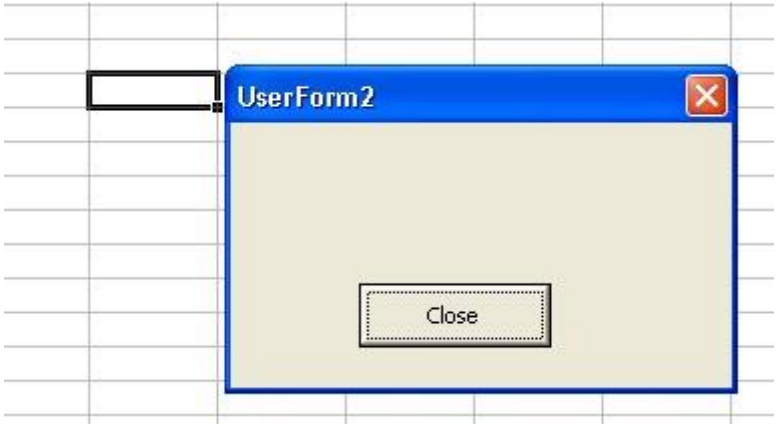
```
UserForm1.Show
```

In summary, the code would look like

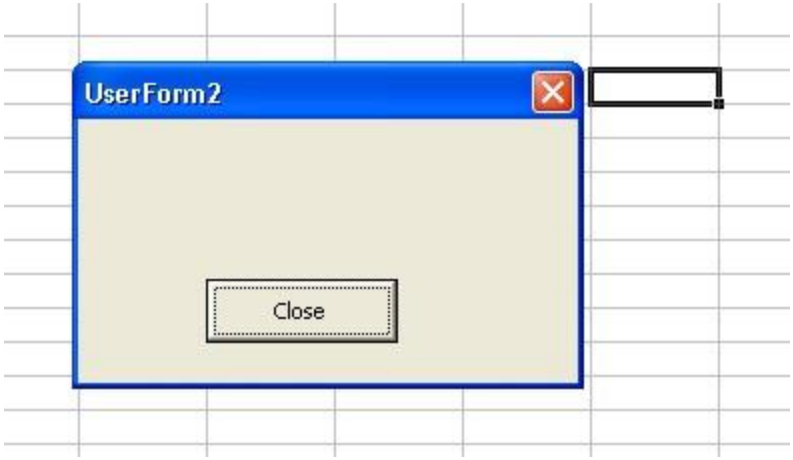
```
Dim PS As Positions
UserForm1.StartupPosition = 0
PS = PositionForm (UserForm1,ActiveCell,0,0,cstFvpAuto,cstFhpAuto)
UserForm1.Top = PS.FrmTop
UserForm1.Left = PS.FrmLeft
UserForm1.Show vbModal
```

The FormPositioner workbook includes a sample userform and a sample procedure called Test that you can use to see how the horizontal and vertical orientation parameters work.

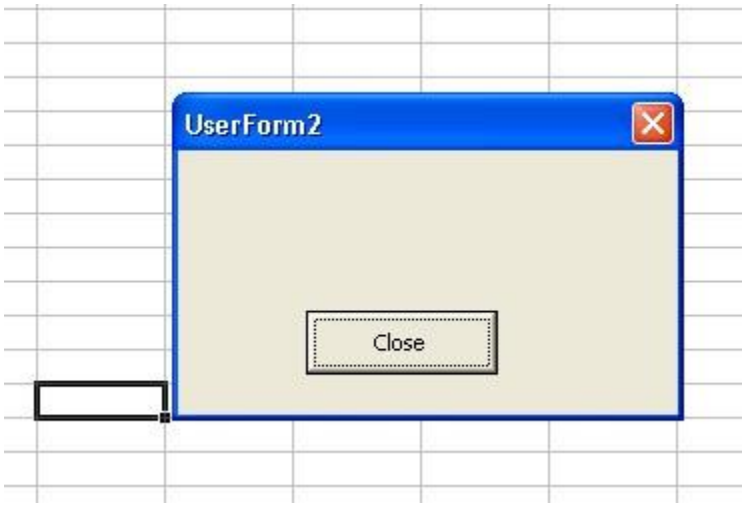
The following images show some of the many ways forms can be positioned:



And



And



The Positions structure is defined as follows, and is declared in the modFormPositioner module.

```
Public Type Positions
    FrmTop As Single ' Userform
    FrmLeft As Single
    FrmHeight As Single
    FrmWidth As Single

    RngTop As Single ' Passed in cell
    RngLeft As Single
    RngWidth As Single
    RngHeight As Single


    AppTop As Single 'Application
    AppLeft As Single
    AppWidth As Single
    AppHeight As Single
```

```
    WinTop As Single ' Window
    WinLeft As Single
    WinWidth As Single
    WinHeight As Single

    Cell1Top As Single ' 1st cell in visible range
    Cell1Left As Single
    Cell1Width As Single
    Cell1Height As Single

    LastCellTop As Single ' last visible cell in window
    LastCellLeft As Single
    LastCellWidth As Single
    LastCellHeight As Single

    BaseLeft As Single ' screen based coordinates for the upper left corner
    BaseTop As Single ' of cell.

    VComp As Single ' compensations for displayed object (toolbars, etc)
    HComp As Single

    NudgeDown As Single ' allow user to nudge positioning by a few pixels.
    NudgeRight As Single

#If VBA6 Then
    OrientationH As cstFormHorizontalPosition
    OrientationV As cstFormVerticalPosition
#Else
    OrientationH As Long
    OrientationV As Long
#End If

End Type
```

If you display the form modelessly, it will not move along with the application window and the sheet windows. It will retain its original position on the screen. See here for code to overcome this circumstance so that the form will move along with the windows and retain its position relative to the application window or sheet windows:

Userforms And The SetParent Function

This page describes how to use the SetParent API function to control how UserForms are displayed.

Introduction

Normally, a modeless userform floats above, and independently of, the Excel application window and the worksheet windows. This means that when you move the application window or a sheet window, the form does not move with the window -- it stays in its original location on the screen, disconnected from the Excel windows. Broadly speaking, this is not desirable. It is my experience that the users expect the userform to move along with the application window or the sheet window.

DEFINITION: A *modeless* form, or a form displayed *modelessly*, is where the form remains visible but does not have focus, and you can edit the worksheet or work with menus and command bars while the form is visible. The "Find" dialog is an example of a modeless form. You can modify cell values while the form is visible on the screen. A *modal* form, or a form shown *modally,* is when the form has focus and you can access only items on the form while it is visible. You cannot access anything that is not on the form until the form is hidden or closed. The "Options" dialog is an example of a modal form. You must close this dialog before doing anything else.

There is no "move" event of any Excel object, so you cannot detect when the user moves a window. To overcome this situation, you can use a series of Windows API system calls to set the form as a "child" window to the worksheet window. When you do this, the form will move along with the application window and the sheet window.

Note: This code applies only to Excel 2000 (VBA6) and later, and only for forms that are shown modelessly. It will not work in Excel97 or earlier, and will not work with modal forms. You show a form modelessly with code like the following:

```
UserForm1.Show vbModeless
```

You can also show a form modelessly by setting the userform's `ShowModal` property to `False`.

Download an example workbook illustrating the code here.

All of the following code should be placed in the userform's code module. At the top of the module, outside of and before any procedure, paste the code shown below.

Complete VBA Code

```
Private Declare Function SetParent Lib "user32" ( _
    ByVal hWndChild As Long, _
    ByVal hWndNewParent As Long) As Long

Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long

Private Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" _
    (ByVal hWnd1 As Long, _
    ByVal hWnd2 As Long, _
    ByVal lpsz1 As String, _
    ByVal lpsz2 As String) As Long

Private Declare Function GetAncestor Lib "user32.dll" ( _
    ByVal hwnd As Long, _
    ByVal gaFlags As Long) As Long
```

Note that the function declarations above must be `Private` not `Public` . You can't have `Public` declarations in an object module, such as a Sheet, Form or Class module.

Then, use the userform's `Initialize` event to set the form as a child to the worksheet window:

```
Private Sub UserForm_Initialize()

Const C_VBA6_USERFORM_CLASSNAME = "ThunderDFrame"
Dim AppHWnd As Long
Dim DeskHWnd As Long
Dim WindowHWnd As Long
Dim MeHWnd As Long
Dim Res As Long

' Get the window handle of the main Excel application window.
' Note, in Excel 2002 and later, you can use "Application.HWnd"
' rather than "FindWindow("XLMAIN", Application.Caption)" to get the
' handle of the main application window. In Excel 2002 and later,
' uncomment the line
' AppHWnd = Application.HWnd
' and remove the call to
' FindWindow("XLMAIN", Application.Caption)
'
'<<<
' Excel 2002 and later only
'AppHWnd = Application.HWnd
'<<<

' The following line of code is not necessary if you are in Excel 2002 or later
' and you are using "AppHWnd = Application.HWnd" to get the window handle to
' the Excel Application window.
AppHWnd = FindWindow("XLMAIN", Application.Caption)
```

```
If AppHWnd > 0 Then
   ' get the window handle of the Excel desktop
   DeskHWnd = FindWindowEx(AppHWnd, 0&, "XLDESK", vbNullString)
   If DeskHWnd > 0 Then
      ' get the window handle of the ActiveWindow
      WindowHWnd = FindWindowEx(DeskHWnd, 0&, "EXCEL7", ActiveWindow.Caption)
      If WindowHWnd > 0 Then
         ' ok
      Else
         MsgBox "Unable to get the window handle of the ActiveWindow."
      End If
   Else
      MsgBox "Unable to get the window handle of the Excel Desktop."
   End If
Else
   MsgBox "Unable to get the window handle of the Excel Application."
End If

' get the window handle of the userform
MeHWnd = FindWindow(C_VBA6_USERFORM_CLASSNAME, Me.Caption)

If (MeHWnd > 0) And (WindowHWnd > 0) Then
    ' make the userform a child window of the ActiveWindow
    Res = SetParent (MeHWnd, WindowHWnd)
    If Res = 0 Then
          ''''''''''''''''''''
```

```
        ' an error occurred.
        ''''''''''''''''''''
      MsgBox "The call to SetParent failed."
    End If
End If


End Sub
```

The code above makes the form a child of the active worksheet window. That means that if you switch to another window the form will disappear and will reappear when you activate the original window. If you want the form to remain visible in all windows, but still move with the application window, use the following simplified `UserForm_Initialize` procedure to make the form a child window of the main application window.

```
Private Sub UserForm_Initialize()

    Const C_VBA6_USERFORM_CLASSNAME = "ThunderDFrame"
    Const GA_ROOTOWNER As Long = 3&

    Dim AppHWnd As Long
    Dim UserFormHWnd As Long
    Dim Res As Long
      ''''''''''''''''''''''''''''''
      ' Get the HWnd of the UserForm
      ''''''''''''''''''''''''''''''
    UserFormHWnd = FindWindow(C_VBA6_USERFORM_CLASSNAME, Me.Caption)
    If UserFormHWnd > 0 Then
          ''''''''''''''''''''''''
          ' Get the ROOTOWNER HWnd
```

```
        ''''''''''''''''''''''''
      AppHWnd = GetAncestor(UserFormHWnd, GA_ROOTOWNER)
      If AppHWnd > 0 Then
            ''''''''''''''''''''''''''''''''
            ' Call SetParent to make the form
            ' a child of the application.
            ''''''''''''''''''''''''''''''''
        Res = SetParent(UserFormHWnd, AppHWnd)
        If Res = 0 Then
              ''''''''''''''''''''
              ' An error occurred.
              ''''''''''''''''''''
          MsgBox "The call to SetParent failed."
        End If
      End If
    End If
End Sub
```

Note: This code applies only to Excel 2000 and above, and applies only to forms that are displayed modelessly:

```
UserForm1.Show vbModeless
```

Schließen der Userform verhindern

Meine Variante

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
' Abfangen des X-Schließ-Knopfes der Registrierungsuserform
```

```
    '1.Variante
    If CloseMode = vbFormControlMenu Then
       MsgBox "DIESE FUNKTION IST VORÜBERGEHEN DEAKTIVIERT"
       Cancel = True
    End If

    ' 2. Variante: If CloseMode <> 1 Then Cancel = 1
End Sub
```

Variante 2

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
If CloseMode = 0 Then
    Cancel = 1
    MsgBox "Bitte verlassen Sie das Dialogfeld mit den Schaltflächen.", _
        vbOKOnly + vbInformation, "Bitte Schaltfläche betätigen."
End If
End Sub
```

Variante 3

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
If CloseMode = vbFormControlMenu Then
        MsgBox "Nix mehr mit schliessen über den (x) button"
        Cancel = True
End If
End Sub
```

Textbox soll nur Zahleneingaben erlauben

```
'  Title: Force numbers only in a text box
'



' ---------------------------------------------------------------------------

' Force numbers only in a text box
' place this in the 'Keypress' event of a text box

' -------------
' 1st Method
' -------------
Const Number$ = "0123456789." ' only allow these characters

If KeyAscii <> 8 Then
   If InStr(Number$, Chr(KeyAscii)) = 0 Then
      KeyAscii = 0
      Exit Sub
   End If
End If

' -------------
' 2nd Method
' -------------

' Force numbers only in a text box
If IsNumeric(Chr(KeyAscii)) <> True Then KeyAscii = 0
```

Userform ausdrucken

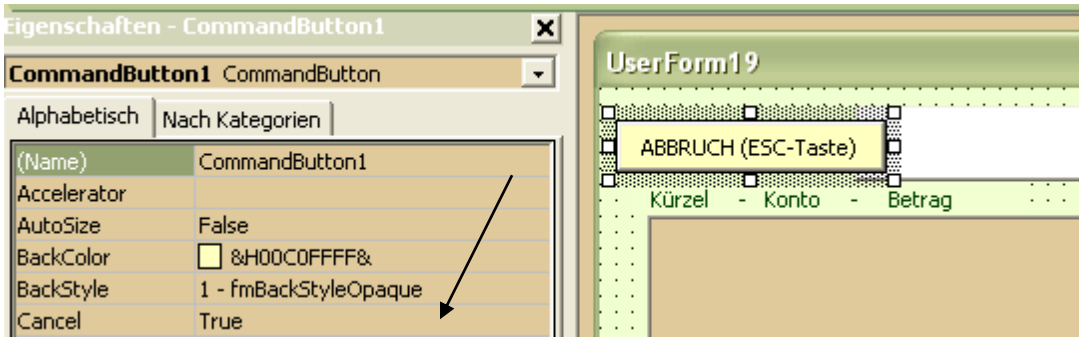
Die einfachste Form, die die Userform direkt ausdruckt mit dem Standarddrucker ist

Userform1.PrintForm

```
Private Sub CommandButton1_Click()

Dim sngHeight As Single, sngWidth As Single
'aktuelle Größe in Variablen merken
sngHeight = Me.Height
sngWidth = Me.Width
'eventuell an Deine benötigte Größe anpassen
Me.Height = 400
Me.Width = 860
Zoom = 69      'eventuell an Deine benötigte Größe anpassen
Me.PrintForm  'UF ausdrucken
'Form- Größe wieder zurückstellen
Me.Height = sngHeight
Me.Width = sngWidth
Zoom = 100 UserForm1.PrintForm  `Zoom wieder zurückstellen
End Sub
```

Userform mit ESC-Taste schließen

Einfach einen Button anlegen und im die CANCEL-Eigenschaft = True geben (daher: die ESC-Taste drückt automatisch diesen Button)



Der Button selbst muss z.B. mit diesem Code belegt sein:

```
Private Sub CommandButton1_Click()
   Unload UserForm19
End Sub
```

TIPP: wer die Taste unsichtbar machen möchte, macht sie einfach klein und transparent ohne Text …

Userform frei (ungebunden) aktivieren (vbModeless)

Damit eine Userform nicht den Focus fix behält, sondern das Arbeiten im Tabellenblatt erlaubt, öffne es vbModeless

UserForm1.Show vbModeless

DoEvents ' events unbeding anführen, damit Userform gerendert wird und Userforminhalt angezeigt wird

Siehe auch "Userform Fokus drauf setzen"

--- VERSION 2 ---

Jemand schrieb:

Erstellt man sich eine Excel Tabelle mit einer Userform, welche direkt beim Start geöffnet wird, so hat man ein Problem, wenn man auch noch andere Exceltabellen zur selben Zeit nutzen möchte. Im Normallfall ruft man eine Userform ja nur auf, wenn man diese auch benötigt. Das macht man mit dem einfachen Aufruf:

Userform1.show

Hat man dann seine Eingaben getätigt, dann schließt man sie wieder. In dem Moment wo wir aber eine Userform geöffnet haben, welche auch permanent offen bleiben soll, kann man nicht mehr zwischen Excel Tabellen switchen (umschalten). Es gibt da aber eine Abhilfe. Legt euch doch einmal diesen Code hinter *diese Arbeitsmappe*:

```
Private Sub Workbook_WindowActivate(ByVal Wn As Window)
   UserForm1.Show
End Sub

Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
 UserForm1.Hide
End Sub
```

Dann stellt ihr in den Eigenschaften der Userform noch *ShowModal* auf *false*. Speichert und schließt euere Excel Tabelle. Wenn ihr sie jetzt beim nächsten Mal startet, dann wird die Userform1 sofort gestartet. Ihr könnt aber, obwohl die Userform geöffnet ist, mit weiteren Excel Tabellen arbeiten.

Userform - Fokus darauf setzen - Fokus auf Userformelement setzen

Um beim bereits mit vbModeless geöffneten Userformular den Fokus wieder drauf zu setzen:

```
Public Sub test()
    AppActivate UserForm1.Caption, True
End Sub
```

Um gleich dort auf ein Tabellenelement den Fokus zu setzen:

UserForm1.ComboBox1.SetFocus

Manchmal hakt der SetFocus-Befehl und man setzt erst den Focus mit diesem Befehl auf ein anderes Element und erst dann auf das Gewünschte.

Siehe auch "Userform - Fokus aufs Tabellenblatt setzen"

Alles beides zusammen:

```
Sub showfrm()
  With UserForm1
    If .Visible Then
      AppActivate .Caption, True
      .TextBox1.SetFocus
      .ComboBox1.SetFocus
    End If
  End With

End Sub
```

Focus auf bestimmtes Element der Userform setzen

TextBox3.SetFocus ' setzt den Schreibcursor auf bestimmtes Element - hier Textbox

Inhalt eines Elements gleich markieren vor Ausfüllen

Möchte man den Text in der Textbox3 nach dem Setfocus-Vorgang gleich auch noch markieren, damit er beim Ausfüllen ersetzt wird, so geht das so:

```
With TextBox3
   .SelStart = 0
   .SelLength = Len(.TEXT)
End With
```

Die EnterFieldBehavior-Eigenschaft eines Feldes legt fest, ob beim Anspringen mit TAB automatisch der Text markiert werden soll. Dies gilt aber nicht für Anspringen mittels SETFOCUS und auch nicht, wenn man mit der Maus draufclickt.

Will man mit einem Mausclick auf ein Feld den Text markieren:

Private Sub TextBox1_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)

```
    With TextBox1
    .SelStart = 0
    .SelLength = Len(.Text)
    End With
End Sub
```

Eine andere Sache ist das Auffangen eines Ereignisses, wenn ein ActiveX-Element (Kombinationsfeld / Textfeld / Listfeld ...) direkt in einer Tabelle ist und man dort abfangen will, wenn es ausgewählt wird:

```
Private Sub Combobox24_GotFocus()
    [code]
End Sub
```

Eine Veriante, die meiner Erfahrung nach besser klappt als Gotfocus ist folgende. Das Problem ist ja immer, wie kam der User in das Feld: durch Mausclick, durch ENTER, durch Tab ...

```
Private Sub Combobox24_Enter()
    [code]
End Sub
```

Activecontrol.Name enthält den Namen des Formularelementes, das den Fokus enthält

ComboBox1.ListIndex = 1 ' setzt im Kombinationsfeld das vorausgewählte Element

Userform - Fokus aufs Tabellenblatt setzen

Variante 1

AppActivate "Microsoft Excel"

Variante 2

Auf UserForm-Schaltflächendruck Focus auf Tabelle setzen

Problem: Auf UserForm-Schaltflächendruck soll der Focus auf die Tabelle gesetzt werden. Nur ab XL9, da die Modal-Eigenschaft erst ab dieser Version eingestellt werden kann.

```
StandardModule: Modul1

Sub DialogAufruf()
    frmFocus.Show
End Sub
```

```
ClassModule: frmFocus

Private Declare Function FindWindow Lib "user32" _
   Alias "FindWindowA" (ByVal lpClassName As String, _
   ByVal lpWindowName As String) As Long
Private Declare Function SetForegroundWindow _
   Lib "user32" (ByVal hwnd As Long) As Long

Private m_hWndXl As Long

Private Sub cmdFocus_Click()
    SetForegroundWindow m_hWndXl
    ActiveCell.Select
End Sub

Private Sub UserForm_Initialize()
    Application.Caption = "My Unique Caption"
    m_hWndXl = FindWindow("XLMAIN", Application.Caption)
    Application.Caption = Empty
End Sub
```

Variante 3

```
'DieseArbeitsmappe
Option Explicit

Sub Workbook_Open()
LiveTabelle.Show vbModeless
'Call LiveTabelle_aktualisieren
Application.Goto Reference:=Worksheets("Tipprunde").Range("A4"), _
Scroll:=True
Test
End Sub

<hr>

'allgemeines Modul
Option Explicit

Public Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As Long) As Long

Sub Test()
SetForegroundWindow (Application.hwnd)
Application.WindowState = xlMaximized
End Sub
```

Userform maximieren

```
Public Sub UserForm_Initialize()
'Excel maximieren
Application.WindowState = xlMaximized
'Userform auf Excelgröße maximieren
With Me
.Height = Application.Height
.Width = Application.Width
End With
End Sub
```

Userform mit Liste von Einträgen zur Auswahl nutzen

Man möchte einen Wert aus einer Userform per Mausklick auswählen und diesen in einer Zelle einfügen.

Der Inhalt der Userform soll aus einem anderen Tabellenblatt (hier "Auswertung") ausgelesen werden.

(In der Simplefibu habe ich die Auswahl des Sach und des Personenkontos so gelöst).

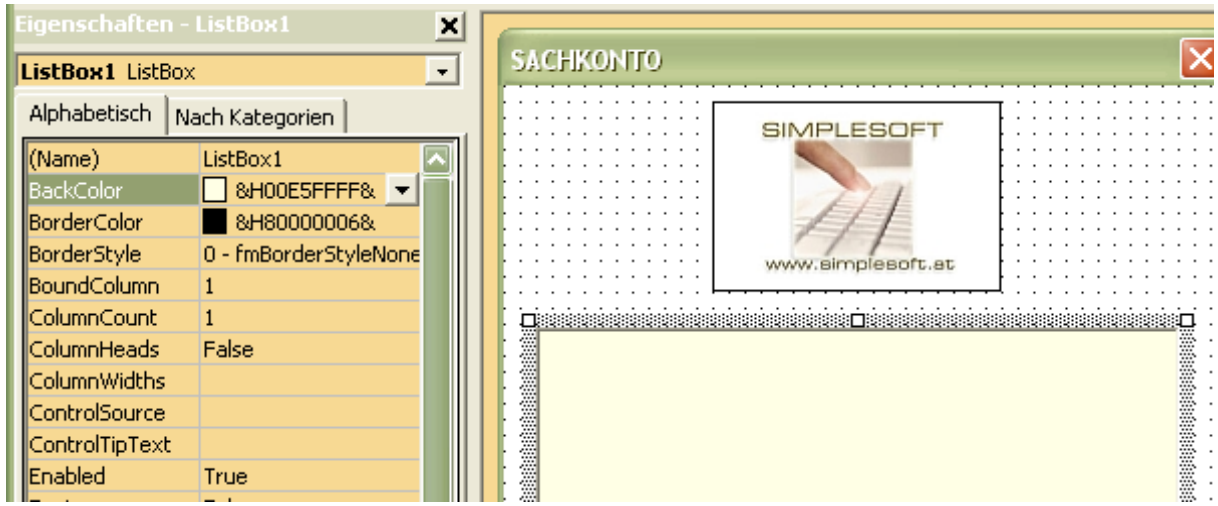
1.) Beim Klicken in eine bestimmte Zelle soll die Userform mit den Werten einer Tabelle befüllt und angezeigt werden.
(Zudem kann man in der Tabelle Auswertung mittels Optionsschaltfläche in Zelle N4 das automatische Aufrufen der Userform deaktivieren)

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
If Sheets("Auswertung").Range("N4") = 1 And ActiveCell.Row > 3 And ActiveCell.Column < 3 Then ' wenn man ab Zeile 4 entweder in die Spalte A oder B klickt
    UserForm17.ListBox1.RowSource = "Auswertung!$A$3:$A$35" ' Eintragen der auszuwählenden Daten in Tabelle Auswertung
    UserForm17.ListBox1.ListIndex = "0" ' Default-Auswahl ist das erste Listenfeld
    UserForm17.Show
End If
End Sub
```

2.) Direkt in der Userform frägt man zwei Ereignisse ab - das Loslassen der ENTER-Taste (KEYUP) und der Maustaste (MOUSEUP)

```
Private Sub ListBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
    If KeyCode = 13 Then KONTOEINFÜGEN1
End Sub
Private Sub ListBox1_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Single, ByVal Y As Single)
    KONTOEINFÜGEN1
End Sub

Sub KONTOEINFÜGEN1()
    LISTWAHL = ListBox1.ListIndex
    Unload UserForm17
    ActiveCell.Value = Sheets("Auswertung").Cells(3 + LISTWAHL, 1).Value ' Ab Zeile 3 sind die Werte, die eingefügt werden sollen
End Sub
```

Die Userform 17 selbst sieht ganz normal aus - die Listbox muss die Nummer 1 haben, damit obiger Code klappt



Userform ohne Titel anzeigen

```
Private Declare Function FindWindow Lib "user32" Alias
"FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As
String) As Long
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long,
ByVal wCmd As Long) As Long
Private Declare Function GetWindowRect Lib "user32" (ByVal hwnd As
```

```
Long, lpRect As RECT) As Long
Private Declare Function ReleaseCapture Lib "user32" () As Long
Private Declare Function SendMessage Lib "user32" Alias
"SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As
Long, lParam As Any) As Long
Private Declare Function CreateRectRgn Lib "gdi32" (ByVal X1 As Long,
ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long
Private Declare Function SetWindowRgn Lib "user32" (ByVal hwnd As Long,
ByVal hRgn As Long, ByVal bRedraw As Boolean) As Long

Private FensterRegion&, Region&
Private Hauptfensternummer&, Clientfensternummer&
Private dummy As Long
Private Type RECT
Left As Long
Top As Long
Right As Long
Bottom As Long
End Type
Private Const GW_CHILD = 5
Private Const WM_NCLBUTTONDOWN = &HA1
Private Const HTCAPTION = 2

Private Sub UserForm_Initialize()
Call FensterOhneKopf
End Sub

Sub FensterOhneKopf()
Dim Abmessung As RECT
Dim Abmessung1 As RECT
Dim Pos1x&, Pos1y&, Pos2x&, Pos2y&
If FensterRegion <> 0 Then Exit Sub
UserForm1.BorderStyle = fmBorderStyleSingle
Call Fensternummer(UserForm1, Abmessung, Abmessung1)
Pos1x = 0
Pos1y = (Abmessung1.Top - Abmessung.Top)
Pos2x = Abmessung.Right - Abmessung.Left
Pos2y = Abmessung.Bottom - Abmessung.Top
Region = CreateRectRgn(Pos1x, Pos1y, Pos2x, Pos2y)
FensterRegion = SetWindowRgn(Hauptfensternummer, Region, True)
End Sub

'Fensterhandles und Infos über Fenster holen
Private Sub Fensternummer(Form As Object, Abmessung As RECT, Abmessung1 As RECT)
Dim Fenstername$, Suchstring$
Suchstring = "UserForm ohne Titelzeile"
Fenstername = Form.Caption
Form.Caption = Suchstring
Hauptfensternummer = FindWindow(vbNullString, Suchstring)
Form.Caption = Fenstername
Clientfensternummer = GetWindow(Hauptfensternummer, GW_CHILD)
dummy = GetWindowRect(Hauptfensternummer, Abmessung)
dummy = GetWindowRect(Clientfensternummer, Abmessung1)
```

```
End Sub

'Folgendes ist notwendig, um die Form ohne Titelleiste zu verschieben
Private Sub UserForm_MouseDown(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
If Button = 1 Then
If Hauptfensternummer <> 0 Then
dummy = ReleaseCapture()
dummy = SendMessage(Hauptfensternummer, WM_NCLBUTTONDOWN, HTCAPTION, 0)
End If
Else
Unload UserForm1 ' Zum schließen, beim ausprobieren.
End If
End Sub

Private Sub CommandButton1_Click()
Unload Me
End Sub
```

VARIABLEN

-→ Grundlagen Variablen

GANZ WICHTIG: der END-Befehl löscht alle globalen Variablen ! Daher besser nur mit EXIT SUB arbeiten

Die gebräuchlichen Variablentypen:

Variablentyp	Namenskonvention	Res.Speicherplatz	Kurzbezeichnung	BEISPIELE	Dezimalstellen
Boolean	bln	16 Bit, 2 Bytes		True / False	-
Byte		8 Bit, 1 Byte		0 - 255	-
Integer (Ganzzahl)	int	16 Bit, 2 Bytes	%	-32.768 bis 32.767	-
Long (Ganzzahl)	lng	32 Bit, 4 Bytes	&	-2.147.483.648 bis 2.147.483.647	-
Currency	cur		@	-922 337 203 685 477,5808 bis 922 337 203 685 477,5807	32
Single (Kommazahl)	sng	32 Bit, 4 Bytes	!	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte	8
Double	dbl	64 Bit, 8 Bytes	#	-1,79769313486232E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte	16
Date	dat	64 Bit, 8 Bytes			
String	str		$	Anzahl Zeichen: 0 bis ungefähr 2 Billionen	
Object	obj	32 Bit, 4 Bytes			
Variant	var	128 Bit, 16 Bytes			
benutzerdefinierter Typ	typ				

Decimal	14 Bytes	+/-79 228 162 514 264 337 593 543 950 335 ohne Dezimalpunkt; +/-7,9228162514264337593543950335 mit 28 Stellen vor dem Komma; die kleinste Nicht-Null-Zahl ist +/- 0,0000000000000000000000000001

Gültigkeit von Variablen und Konstanten

Die Gültigkeit:

Variablen sind Platzhalter für Zeichenfolgen, Werte und Objekte. Sie können Werte oder Objekte enthalten. Abhängig vom Ort und der Art ihrer Deklaration werden ihre Gültigkeit und die Lebensdauer ihrer Werte festgelegt.

- **Deklaration innerhalb einer Prozedur**
  Die Variable hat ihre Gültigkeit ausschließlich für diese Prozedur und kann aus anderen Prozeduren nicht angesprochen werden.
- **Deklaration im Modulkopf**
  Die Variable gilt für alle Prozeduren dieses Moduls, eine Weitergabe als Parameter ist nicht notwendig.
- **Deklaration im Modulkopf eines Standardmoduls als *Public***
  Die Variable gilt für alle Prozeduren der Arbeitsmappe, soweit das die Prozedur enthaltene Modul nicht als *Private* deklariert ist.

Empfehlenswert ist die grundsätzliche Vermeidung von Public-Variablen und der Verzicht auf Variablen auf Modulebene. Es ist nicht immer einfach zu beurteilen, wann diese öffentlichen Variablen ihren Wert verlieren oder wo er geändert wird. Die sauberste Lösung ist die Deklariation innerhalb der Prozeduren und die Weitergabe als Parameter.

Wenn Sie mit öffentlichen Variablen arbeiten, sollten Sie Ihre Variablennamen gemäß den Programmier-Konventionen vergeben und sie so als öffentlich kennzeichnen. Ein vorangestelltes **g** könnte darauf hinweisen, dass es sich um eine Public-Variable, ein kleines **m**, dass es sich um eine Variable auf Modulebene handelt.

In den nachfolgenden Beispielen wird Deklaration und Verhalten von Variablen demonstriert.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-2**Die Beispiele**
http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-3**Deklaration auf Prozedurebene**

Eine Variable ist innerhalb einer Prozedur deklariert und nur in dieser Prozedur gültig.

- Prozedur: varA
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Wert an Integer-Variable übergeben
    - Wert melden
- Code:

```
Sub VarA()
   Dim iValue As Integer
   iValue = 10 + 5
   MsgBox "Variablenwert: " & iValue
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-4

Deklaration auf Modulebene

Eine Variable ist innerhalb eines Moduls in jeder Prozedur gültig und wird im Modulkopf deklariert.

- Prozedur: varB und ProcedureA
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:

    - Variablendeklaration im Modulkopf
    - Wert an Double-Variable übergeben
    - Unterprogramm ohne Parameter aufrufen
    - Variablenwert melden
- Code:

```
Dim mdModul As Double

Sub VarB()
   mdModul = 23 / 14
   Call ProcedureA
End Sub

Private Sub ProcedureA()
   MsgBox "Variablenwert: " & mdModul
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-5**Statische Variable**

Eine Variable ist innerhalb einer Prozedur als statisch deklariert und behält bei neuen Prozeduraufrufen ihren Wert.

- Prozedur: varC
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Aufrufzähler hochzählen
    - Wert melden
    - Wert hochzählen

- Code:

```
Sub VarC()
   Static iValue As Integer
   Static iCount As Integer
   iCount = iCount + 1
   MsgBox iCount & ". Aufruf: " & iValue
   iValue = iValue + 100
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-6**Public-Variable**

Eine Variable ist in der Arbeitsmappe in jedem Modul gültig und im Modulkopf eines Moduls als Public deklariert.

- Prozedur: varD und varE für den Folgeaufruf
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration im Modulkopf
    - Arbeitsblatt an Objektvariable übergeben
    - Arbeitsblattnamen melden
- Im zweiten Aufruf:
    - Wenn die Objekt-Variable nicht initialisiert ist...
    - Warnton
    - Negativmeldung
    - Sonst...
    - Arbeitsblattnamen melden
- Code:

```
Public gwksMain As Worksheet

Sub VarD()
   Set gwksMain = Worksheets("Tabelle1")
   MsgBox "Blattname: " & gwksMain.Name
End Sub

Sub varE()
   If gwksMain Is Nothing Then
      Beep
      MsgBox "Bitte zuerst über Beispiel D initialisieren!"
   Else
      MsgBox "Blattname: " & gwksMain.Name
   End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-7**Übergabe von Variablen an eine Funktion**

Variablen an eine Funktion übergeben und den Rückgabewert melden.

- Prozedur: varF und Funktion GetCbm
- Art: Sub/Funktion
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Funktions-Rückgabewert in eine Double-Variable einlesen
    - Ergebnis melden
- Die Funktion:
    - Rückgabewert berechnen
- Code:

```
Sub varF()
```

```
   Dim dCbm As Double
   dCbm = GetCbm(3.12, 2.44, 1.58)
   MsgBox "Kubikmeter: " & Format(dCbm, "0.00")
End Sub

Private Function GetCbm( _
   dLength As Double, _
   dWidth As Double, _
   dHeight As Double)
   GetCbm = dLength * dWidth * dHeight
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-8**ByRef-Verarbeitung in einem Unterprogramm**

Variable ByRef an ein Unterprogramm übergeben und den veränderten Rückgabewert melden.

- Prozedur: varG und Unterprogramm ProcedureB
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Variable für Rückgabewert initialisieren
    - Unterprogramm mit Parametern aufrufen
    - Ergebnis melden
- Das Unterprogramm:
    -
        - Rückgabewert berechnen
- Code:

```
Sub varG()
   Dim dCbm As Double
   dCbm = 0
   Call ProcedureB(3.12, 2.44, 1.58, dCbm)
   MsgBox "Kubikmenter: " & dCbm
End Sub

Private Sub ProcedureB( _
   ByVal dLength As Double, _
   ByVal dWidth As Double, _
   ByVal dHeight As Double, _
   ByRef dErgebnis As Double)
   dErgebnis = dLength * dWidth * dHeight
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-9**Übergabe von Variablen an eine andere Arbeitsmappe**

Variable an eine Funktion einer anderen Arbeitsmappe übergeben und den Rückgabewert melden.

- Prozedur: varH und Funktion in anderer Arbeitsmappe
- Art: Sub/Funktion
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Pfad und Dateinamen der Test-Arbeitsmappe an String-Variable übergeben
    - Wenn die Test-Arbeitsmappe nicht gefunden wurde...
    - Negativmeldung
    - Sonst...
    - Bildschirmaktualisierung ausschalten
    - Wert an Long-Variable übergeben

- Test-Arbeitsmappe öffnen
- Funktion in der Text-Arbeitsmappe aufrufen und Ergebnis in Long-Variable einlesen
- Test-Arbeitsmappe schließen
- Bildschirmaktualisierung einschalten
- Rückgabewert melden

- Code:

```
Sub varH()
   Dim lValue As Long
   Dim sFile As String
   sFile = ThisWorkbook.Path & "\vb04_test.xls"
   If Dir(sFile) = "" Then
      MsgBox "Die Testdatei " & sFile & " fehlt!"
   Else
      Application.ScreenUpdating = False
      lValue = 12345
      Workbooks.Open sFile
      lValue = Application.Run("vb04_test.xls!Berechnung", lValue)
      ActiveWorkbook.Close savechanges:=False
      Application.ScreenUpdating = True
      MsgBox "Ergebnis: " & lValue
   End If
End Sub

Function Berechnung(lWert As Long)
   Berechnung = lWert * 54321
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-10**Variablen füllen und zurücksetzen**

Variablenwerte werden belegt und zurückgesetzt.

- Prozedur: varI
- Art: Sub
- Modul: Standardmodul

- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Variablendeklaration
    - Aktives Arbeitsblatt an eine Objekt-Variable übergeben
    - Schleife bilden
    - Array mit Werten füllen
    - Meldung mit Arbeitsblattnamen, Array-Inhalt und Wert der Zählvariablen
    - Meldung, dass die Werte zurückgesetzt werden
    - Objektvariable zurücksetzen
    - Array zurücksetzen
    - Zählvariable zurücksetzen
    - Fehlerroutine initialisieren
    - Arbeitsblattnamen melden (führt zum Fehler)
    - Wert des ersten Datenfeldes melden (leer)
    - Wert der Zählvariablen melden (0)
    - Prozedur verlassen
    - Fehlerroutine
    - Wenn es sich um die Fehlernummer 91 handelt...
    - Meldung mit Fehlernummer und Fehlertext
    - Nächste Programmzeile abarbeiten

- Code:

```
Sub varI()
   Dim wks As Worksheet
   Dim arr(1 To 3) As String
   Dim iCounter As Integer
   Set wks = ActiveSheet
   For iCounter = 1 To 3
      arr(iCounter) = Format(DateSerial(1, iCounter, 1), "mmmm")
   Next iCounter
   MsgBox "Name des Objeks Arbeitsblatt:" & vbLf & _
      "   " & wks.Name & vbLf & vbLf & _
      "Inhalt des Arrays:" & vbLf & _
      "   " & arr(1) & vbLf & _
      "   " & arr(2) & vbLf & _
      "   " & arr(3) & vbLf & vbLf & _
      "Inhalt der Zählvariablen:" & vbLf & _
      "   " & iCounter
   MsgBox "Jetzt werden die Variablen zurückgesetzt!"
   Set wks = Nothing
   Erase arr
   iCounter = 0
   On Error GoTo ERRORHANDLER
   MsgBox wks.Name
   MsgBox "Wert des ersten Datenfeldes: " & arr(1)
   MsgBox "Wert der Zählvariablen: " & iCounter
   Exit Sub
ERRORHANDLER:
   If Err = 91 Then
      MsgBox "Fehler Nr. " & Err & ": " & Error
      Resume Next
   End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-11**Konstanten auf Prozedurebene**

Konstante auf Prozedurebene als Endpunkt einer Schleife.

- Prozedur: varJ
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration

- Ablaufbeschreibung:
    - Konstantendeklaration
    - Variablendeklaration
    - Schleife bilden
    - Schleife beenden
    - Zählvariable melden
- Code:

```
Sub varJ()
   Const ciLast As Integer = 100
   Dim iCounter As Integer
   For iCounter = 1 To ciLast
   Next iCounter
   MsgBox "Zähler: " & iCounter
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-12**Public-Konstanten**

Public-Konstante für alle Prozeduren der Arbeitsmappe.

- Prozedur: varK
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Konstantendeklaration im Modulkopf
    - Meldung mit der Public-Konstanten
- Code:

```
Public Const gciDecember  As Integer = 12

Sub varK()
   MsgBox "Monat Dezember hat den Index " & gciDecember
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_G%C3%BCltigkeit_von_Variablen_und_Konstanten&action=edit&section=T-13**Übergabe eines variablen Wertes an eine Konstante**

Variabler Wert als Konstante. Gegen Versuche, einen variablen Wert an eine Konstante zu übergeben, wehrt sich VBA vehement. Das Beispiel zeigt eine Möglichkeit, das Problem zu umgehen.

- Prozedur: varL
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
    - Konstantendeklaration
    - Meldung mit der variablen Konstanten
- Code:

```
Sub varL()
   Const cDay As String = "Day(Now())"
   MsgBox "Tageskonstante: " & Evaluate(cDay)
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-1# Zu ByRef und ByVal

Variablen können an Funktionen oder Unterprogramme übergeben, dort zu Berechnungen verwendet und mit geänderten Werten zurückgegeben werden. Entscheidend hierfür ist das Schlüsselwort der Parameter-Definition des aufnehmenden Unterprogramms.

VBA kennt die Parameterübergaben **ByRef** und **ByVal**. Im ersten Fall - das ist die Standardeinstellung, d.h. wenn keine Vorgabe erfolgt, wird der Parameter als **ByRef** behandelt - wird der Wert des Parameters weiterverarbeitet; änderungen sind auch für das aufrufende Programm wirksam. Im zweiten Fall wird eine Kopie des Parameters übergeben; die Wirksamkeit beschränkt sich auf das aufgerufene Unterprogramm und der Parameter im aufrufenden Programm behält seinen ursprünglichen Wert.

Dies gilt nicht für Objekt-Variablen. Diese behalten auch bei der Verwendung des Schlüsselwortes **ByRef** in der aufrufenden Prozedur ihren ursprünglichen Wert.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-2# Die Beispiele

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-3## Aufruf einer benutzerdefinierten Funktion ohne ByRef/ByVal-Festlegung

Die Funktion errechnet anhand der übergebenen Parameter den Wert und gibt diesen an das aufrufende Programm zurück, wobei die übergebenen Parameter nicht geändert werden.

```
Sub CallFunction()
   Dim dQM As Double
   dQM = fncQM( _
      Range("A2").Value, _
      Range("B2").Value, _
      Range("C2").Value)
   MsgBox "Quadratmeter Außenfläche: " & _
      Format(dQM, "0.000")
End Sub

Private Function fncQM( _
   dLong As Double, dWidth As Double, dHeight As Double)
   fncQM = 2 * (dLong * dWidth + _
      dLong * dHeight + _
      dWidth * dHeight)
End Function
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-4## Aufruf eines Unterprogramms ohne ByRef/ByVal-Festlegung

Das Unterprogramm wird mit den für die Berechnung notwendigen Parametern und zusätzlich mit einer 0-Wert-Double-Variablen, die als Container für das Berechnungsergebnis dient, aufgerufen. Alle Parameter gelten als **ByRef**, da kein Schlüsselwort verwendet wurde.

```
Sub CallMacro()
   Dim dQM As Double
   Call GetQm( _
      dQM, _
      Range("A2").Value, _
      Range("B2").Value, _
      Range("C2").Value)
   MsgBox "Quadratmeter Außenfläche: " & _
      Format(dQM, "0.000")
End Sub
```

```
Private Sub GetQm( _
   dValue As Double, dLong As Double, _
   dWidth As Double, dHeight As Double)
   dValue = 2 * (dLong * dWidth + _
      dLong * dHeight + _
      dWidth * dHeight)
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-5**Aufruf mit einer Integer-Variablen bei Anwendung von *ByVal***

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```
Sub AufrufA()
   Dim iRow As Integer, iStart As Integer
   iRow = 2
   iStart = iRow
   Call GetRowA(iRow)
   MsgBox "Ausgangszeile: " & iStart & _
      vbLf & "Endzeile: " & iRow
End Sub

Private Sub GetRowA(ByVal iZeile As Integer)
   Do Until IsEmpty(Cells(iZeile, 1))
      iZeile = iZeile + 1
   Loop
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-6**Aufruf mit einer Integer-Variablen bei Anwendung von *ByRef***

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, damit auch der Wert der Variablen im aufrufenden Programm.

```
Sub AufrufB()
   Dim iRow As Integer, iStart As Integer
   iRow = 2
   iStart = iRow
   Call GetRowB(iRow)
   MsgBox "Ausgangszeile: " & iStart & _
      vbLf & "Endzeile: " & iRow
End Sub
```

```
Private Sub GetRowB(ByRef iZeile As Integer)
   Do Until IsEmpty(Cells(iZeile, 1))
      iZeile = iZeile + 1
   Loop
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-7

Aufruf mit einer String-Variablen bei Anwendung von *ByVal*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```
Sub CallByVal()
   Dim sPath As String, sStart As String
   sPath = ThisWorkbook.Path
   sStart = sPath
   Call GetByVal(sPath)
   MsgBox "Vorher: " & sStart & _
      vbLf & "Nachher: " & sPath
End Sub

Private Sub GetByVal(ByVal sDir As String)
   If Right(sDir, 1) &lt;&gt; "\" Then
      sDir = sDir & "\"
   End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-8

Aufruf mit einer String-Variablen bei Anwendung von *ByRef*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablauf des Unterprogramms, damit auch der Wert der Variablen im aufrufenden Programm.

```
Sub CallByRef()
   Dim sPath As String, sStart As String
   sPath = ThisWorkbook.Path
   sStart = sPath
   Call GetByRef(sPath)
   MsgBox "Vorher: " & sStart & _
      vbLf & "Nachher: " & sPath
End Sub

Private Sub GetByRef(ByRef sDir As String)
   If Right(sDir, 1) &lt;&gt; "\" Then
      sDir = sDir & "\"
```

```
   End If
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-9

Aufruf mit einer Objekt-Variablen bei Anwendung von *ByVal*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```
Sub CallObjectA()
   Dim rngA As Range, rngB As Range
   Set rngA = Range("A1:A10")
   Set rngB = rngA
   Call GetObjectA(rngA)
   MsgBox "Vorher: " & rngB.Address(False, False) & _
      vbLf & "Nachher: " & rngA.Address(False, False)
End Sub

Private Sub GetObjectA(ByVal rng As Range)
   Set rng = Range("F1:F10")
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_ByRef_und_ByVal&action=edit&section=T-10

Aufruf mit einer Objekt-Variablen bei Anwendung von *ByRef*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```
Sub CallObjectB()
   Dim rngA As Range, rngB As Range
   Set rngA = Range("A1:A10")
   Set rngB = rngA
   Call GetObjectB(rngA)
   MsgBox "Vorher: " & rngB.Address(False, False) & _
      vbLf & "Nachher: " & rngA.Address(False, False)
End Sub

Private Sub GetObjectB(ByRef rng As Range)
   Set rng = Range("F1:F10")
End Sub
```

Grundlagen 2

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-1

Grundlegendes

Was sind Variablen?

Variablen sind eine Art von Platzhalter für Zeichenfolgen, Werte und Objekte. So können beispielsweise mehrfach anzuzeigende Meldungen, bei Berechnungen mehrfach einzusetzende Werte oder in einer Schleife anzusprechende Objekte in Variablen gespeichert werden.

Wann sind Variablen einzusetzen?

Der Einsatz von Variablen ist immer dann sinnvoll, wenn das Element mehrfach angesprochen wird. Sinnvoll eingesetzt, beschleunigen Variablen die Ausführung eines VBA-Programms erheblich. Wird das Element im Code nur einmal angesprochen – wie zum Beispiel eine Msg-Meldung – ist das Speichern dieser Zeichenfolge in eine String-Variable überflüssig und verwirrend. Ausnahmen bilden Fälle, in denen auch bei einmaligem Vorkommen die Übersichtlichkeit des Codes verbessert wird. Dies kann beispielsweise bei langen Objektnamen der Fall sein.

Sind Variablen zu deklarieren?

Eine Deklaration der Variablen sollte immer erfolgen. Dazu sollte in der Entwicklungsumgebung im Menü **Extras / Optionen** die CheckBox **Variablendeklaration erforderlich** aktiviert sein. VBA-Anweisungen zur Dimensionierung sind:

- `Dim`
  - In einer `Function` oder `Sub` Anweisung. Die Deklaration sollte am Anfang stehen
  - Zu Beginn eines (Standard-)Moduls oder Klassenmoduls, ist gleichwertig mit `Public Dim`
- `Private`: Am Anfang eines (Standard-)Moduls oder Klassenmoduls, bedeutet `Private Dim` (nicht zulässig)
- `Global` entspricht `Public`, aus Gründen der Abwärtskompatibilität unterstützt

Empfehlenswert ist ein Kommentar in der Zeile vor der Variablendeklaration oder in der Zeile der Deklaration am Ende, um den Zweck der Variablen zu erklären. Beispiel:

```
Private i As Integer ' Schleifenzähler
```

Wo sind Variablen zu deklarieren?

Variablen, die nur für die Prozedur gelten sollen, sind innerhalb der Prozedur, in der Regel am Prozeduranfang zu deklarieren. Variablen, die außerhalb einer Prozedur deklariert werden, gelten für das ganze Modul, werden sie als `Public` deklariert, für das gesamte Projekt. Zu einem sauberen

Programmierstil gehört es, Variablen soweit irgend möglich nur auf Prozedurebene zu deklarieren und an Unterprogramme als Parameter zu übergeben.

Sind Variablen zu dimensionieren?

Wenn Variablen als Array deklariert wurden, z.B. `Dim MitgliedsNr() As Long` können sie entweder mit der Deklaration dimensioniert werden (`Dim MitgliedsNr(1001) As Long` oder `Dim MitgliedsNr(1 To 1000) As Long` oder nachträglich mit der `ReDim`-Anweisung

Sind Objekttyp-Variablen bestimmten Objekten zuzuweisen?

Zur Referenzierung von Objekten durch Variable kann stets der allgemeine Typ `Variant` (nicht empfehlenswert), als auch der allgemeine Objekttyp `Object` verwendet werden. Wenn die Bibliothek des Objekts über das Menü 'Extras' 'Verweise' eingebunden ist, kann auch der spezielle Objekttyp deklariert werden. Zu bevorzugen ist immer eine möglichst genaue Deklaration, die Deklaration des spezifischen Objekttyps bietet vor allem diese Vorteile:

- Schnellerer Programmablauf
- Weniger Speicherbedarf als bei `Variant`
- In der Entwicklungsumgebung werden während der Programmierphase - wenn im obigen Dialog die CheckBox Elemente automatisch auflisten aktiviert ist - beim Eintippen des Punktes nach einem Objektnamen alle Methoden und Eigenschaften automatisch aufgelistet, was Fehler vermeidet und Schreibarbeit erspart.
- Fehlermeldungen schon beim Kompilieren (falls beispielsweise Argumente fehlerhaft sind), genauere Fehlerbeschreibungen

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-2

Konstanten

Konstanten werden hier der Vollständigkeit halber erwähnt. Weisen Sie immer dann, wenn ein Wert vom Programmstart bis zum Programmende unverändert bleibt, diesen einer Konstanten, keiner Variablen zu. Konstanten werden in VBA-Programmen schneller berechnet als Variablen. Konstanten werden generell im Allgemein-Abschnitt von Modulen deklariert, Private-Konstanten in Klassen- und Standard-, Public-Konstanten nur in Standardmodulen. Beispiel für eine Konstanten-Deklaration:

```
Private Const cintStart As Integer = 5
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-3

Variablentypen

Die gebräuchlichen Variablentypen:

Variablentyp	Namenskonvention	Res.Speicherplatz	Kurzbezeichnung	Dezimalstellen
Boolean	bln	16 Bit, 2 Bytes		-
Byte		8 Bit, 1 Byte		-
Integer Ganzzahl	int	16 Bit, 2 Bytes	%	-
Long Ganzzahl	lng	32 Bit, 4 Bytes	&	-
Currency	cur		@	32
Single Kommazahl	sng	32 Bit, 4 Bytes	!	8
Double	dbl	64 Bit, 8 Bytes	#	16
Date	dat	64 Bit, 8 Bytes		
String	str		$	
Object	obj	32 Bit, 4 Bytes		
Variant	var	128 Bit, 16 Bytes		
benutzerdefinierter Typ	typ			
Objekttyp				

Variablentyp	Beschreibung
Boolean	WAHR (-1) oder FALSCH (0)
Byte	0 ... +255
Integer	-32.768 ... +32.767
Long	-2.147.483.648 ... +2.147.483.647
Currency	-922.337.203.685.477,5808 ... +922.337.203.685.477,5807
Single	±3,402823E38 ... ±1,401298E-45 und 0
Double	-1.79769313486231E308 bis -4,94065645841247E-324 für negative Werte und von 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte und 0
Date	Datum und Zeit

String	Zeichenfolgen (Text)
Object	Objekte
Variant	Alle Typen, Voreinstellung
benutzerdefinierter Typ	ein oder mehrere Elemente jeden Datentyps. Der Aufbau wird mit einer `Type`-Anweisung deklariert
Objekttyp	Objekte wie Workbook, Range

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-4

Anmerkungen zu den Variablentypen

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-5

Boolean

Dieser Datentyp speichert eigentlich nur ein Bit, aus Gründen der Speicherorganisation wird jedoch stets ein Byte belegt. Die Werte von Boolean werden als 8-Bit Zahl dargestellt, wobei nur -1 (= alle Bits gesetzt bei Darstellung der -1 als Zweierkomplement) als WAHR gilt, jeder andere Wert aber als FALSCH. Speziell bei Vergleichen wird das Ergebnis FALSCH als 0 (= kein Bit gesetzt) zurückgegeben.

In Kenntnis dieser Interpretation kann der Programmierer Vergleiche auch direkt auf Zahlenwerte in Long-, Integer- und Byte-Datentypen (bei letzteren setzt der Wert 255 alle Bits) anwenden. Aus Gründen der Lesbarkeit des Codes sollte das aber vermieden werden.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-6

Byte

Bei diesem Variablentyp ist in speziellen Fällen Vorsicht geboten, beispielsweise kann bei

```
For i = 10 To 0 Step -1
```

dieser Schleifenkonstruktion ein Unterlauf-Fehler auftreten, wenn i als Byte dimensioniert wird, weil in der internen Berechnung auch noch -1 berechnet wird. Wird als Endwert der Schleife 1 statt 0 angegeben oder wird beispielsweise der Datentyp Integer für i verwendet, gibt es kein Problem.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-7

Date

Der Typ speichert das Datum in zwei Teilen:

- Vor dem Komma steht die fortlaufende Tagesnummer. Tag 0 dieser Zählung ist der 31.12.1899; Bei der Anzeige wird es in die vom System eingestellte Darstellung von Tag, Monat und Jahr umgerechnet.
- Nach dem Komma stehen die Anteile des Tages. 0,25 steht für 6 Stunden, 0,5 für 12 h usw.

Vom Wert her ist der Inhalt dieses Datentyps nicht von einem Fließkommawert zu unterscheiden. Entsprechend einfach können Tage und Stunden addiert werden, hier einige Beispiele:

- Um zu einem Datum h Stunden zu addieren, rechnet man Datum + h/24
- Um zu einem Datum h Stunden und m Minuten zu addieren, rechnet man Datum + h/24 + m/(24*60) oder Datum + (h + m/60)/24
- Um zu einem Datum h Stunden und m Minuten und s Sekunden zu addieren, rechnet man Datum + (h + (m + s/60)/60)/24

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-8

Currency

Der Datentyp ist ein Festkommaformat mit vier Nachkommastellen. Daher wird er intern wie eine Ganzzahl berechnet. Wenn die Genauigkeit ausreicht, kann mit der Wahl dieses Datentyps gegenüber Single und Double die Berechnung schneller erfolgen. Bei Kettenrechnungen mit langen oder periodischen Dezimalteilen ist allerdings mit einem Genauigkeitsverlust zu rechnen.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-9

String

Der Datentyp speichert Zeichen mit variabler Länge von maximal 2^{31} Zeichen.

Für bestimmte Zwecke können auch Strings mit fester Länge sinnvoll sein. Sie können mit einem * definiert werden, Beispiel String mit der festen Länge 3:

```
Public Sub Demo_StringMitFesterLänge()
    Dim ZeichenKette As String * 3
    ZeichenKette = "A"
    MsgBox ">" & ZeichenKette & "<"
End Sub
```

Bei der Zuweisung von "A" wird der String von links belegt, die übrigen Zeichen werden mit einem Leerzeichen aufgefüllt. Die Strings mit fester Länge unterliegen gewissen Einschränkungen, so können sie max. 2^{16} Zeichen speichern und nicht mit dem Attribut Attribut Public in Klassenmodulen verwendet werden.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-10

Benutzerdefinierte Typen

Diese Typen werden aus den Grundtypen mit Hilfe der Type-Anweisung zusammengesetzt. Das folgende Beispiel zeigt, wie die Typdeklaration für komplexe Zahlen aussehen könnte. Neben dem Real- und Imaginärteil wird in dem benutzerdefinierten Typ auch gespeichert, ob die komplexe Zahl in kartesischen Koordinaten (FALSE) oder in Polarkoordinaten (TRUE) abgelegt wurde.

Das Beispiel des komplexen Multiplikationsprogramms `cMult` wurde nur für den Fall ausgeführt, in dem beide Variablen in kartesischen Koordinaten vorliegen.

```
Type Komplex            ' Komplexe Zahl
   Re      As Double    ' Realteil
   Im      As Double    ' Imaginärteil
   Winkel  As Boolean   ' FALSE = Kartesisch, TRUE = Polar
End Type
```

```
' ** Funktion zur Multiplikation zweier komplexer Zahlen
Public Function cMult(a As Komplex, b As Komplex) As Komplex
    If (a.Winkel = b.Winkel) Then
        ' Beide Zahlen liegen im gleichen Koordinatensystem vor
        If Not a.Winkel Then
            ' Beide Zahlen liegen in kartesischen Koordinaten vor
            ' Multiplikation in kartesischen Koordinaten
            cMult.Re = a.Re * b.Re - a.Im * b.Im
            cMult.Im = a.Im * b.Re + a.Re * b.Im
            cMult.Winkel = a.Winkel
       End If
    End If
End Function
```

Das folgende Beispiel zeigt zwei Möglichkeiten, um die Variablen `Faktor1` und `Faktor2` mit Werten zu belegen und wie man das Ergebnis der Funktion `cMult` im weiteren Programmlauf verwerten kann:

```
Public Sub Demo_KomplexeMultiplikation()
    Dim Faktor1  As Komplex  ' Erster Faktor
    Dim Faktor2  As Komplex  ' Zweiter Faktor
    Dim Ergebnis As Komplex  ' Komplexes Produkt

    ' Möglichkeit 1.1: Variable mit Hilfe der With-Anweisung belegen
    With Faktor1
        .Re = 2
        .Im = 3
        .Winkel = False
    End With

    ' Möglichkeit 1.2: Direkt belegen
    Faktor2.Re = 5
    Faktor2.Im = 7
    Faktor2.Winkel = False

    ' Möglichkeit 2.1: Ergebnis einer Variablen vom Typ Komplex zuweisen
    Ergebnis = cMult(Faktor1, Faktor2)

    ' Ausgabe ins Direktfenster
    Debug.Print Ergebnis.Re, Ergebnis.Im, Ergebnis.Winkel

    ' Möglichkeit 2.2: Alle Werte einzeln aus dem Rückgabewert der Funktion holen
    With cMult(Ergebnis, Faktor2)
        MsgBox Iif(.Winkel, "R: ", "x-Koordinate: ") & .Re
        MsgBox Iif(.Winkel, "Winkel: ", "y-Koordinate: ") & .Im
    End With
End Sub
```

Der Einfachheit halber wurden die Rückgabewerte mit `Debug.Print` in das Direktfenster geschrieben.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-11

Variablendeklaration

Wie schon erwähnt, sind Variablen generell zu deklarieren und zu dimensionieren. Werden sie nicht deklariert oder nicht dimensioniert, handelt es sich beim Programmstart in jedem Fall um den Variablentyp Variant, der zum einen mit 16 Bytes den größten Speicherplatz für sich beansprucht, zum anderen während des Programmablaufes seinen Typ mehrmals wechseln kann, was möglicherweise zu unerwarteten Verhalten und damit Fehlern führen kann. Außerdem benötigen Variant-Variablen erheblich längere Berechnungszeiten als andere.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-12

Einsatz von String-Variablen

Im nachfolgenden Beispiel wird eine String-Variable deklariert und zum Finden und Ersetzen einer Zeichenfolge eingesetzt:

```
Sub Ersetzen()
   Dim rngCell As Range
   Dim strText As String
   strText = "Kasse "
   strYear = CStr(Year(Date))
   For Each rngCell In Range("A1:F15")
      If rngCell.Value = strText & Year(Date) - 1 Then
         rngCell.Value = strText & Year(Date)
      End If
   Next rngCell
End Sub
```

Im vorgegebenen Bereich werden alle Zellen darauf überprüft, ob ihr Text aus der Zeichenfolge Kasse und der Jahreszahl des Vorjahres besteht. Wenn ja, wird die Vorjahreszahl durch die aktuelle Jahreszahl ersetzt. String-Variablen sollten mit dem &-Zeichen verknüpft werden. Strings können auch mit + verknüpft werden. Dies funktioniert aber nur zuverlässig, wenn beide Variablen oder Ausdrücke strings sind. Falls ein Ausdruck numerisch ist und der andere ein String, der als Zahl interpretierbar ist, nimmt Excel eine Typumwandlung um und liefert als Ergebnis die algebraische Summe der beiden Ausdrucke. Wenn in einem Ausdruck & mit + gemischt wird, berechnet VBA zuerst + (und alle anderen algebraischen Operationen wie -*/) dann erst &;

Beispiele:

- Aus "2" + "3" wird "23"
- Aus "2" + 3 wird 5
- Aus "2" & 3 wird "23"

- Aus "2" & 3 + 4 & "5" wird 275
- Aus "2" & 3 & 4 & "5" wird 2345
- Aus "2" + 3 & 4 + "5" wird 59

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-13

Einsatz von Variant-Variablen

Es gibt Fälle, in denen eine Variable ihren Typ ändert oder unterschiedliche Typen entgegennehmen muss. In diesem Fall können Variant-Variablen eingesetzt werden. Dies ist besonders dann notwendig, wenn eine Funktion unterschiedliche Datentypen zurückgeben kann, wie z.B. GetOpenFilename. Diese liefert entweder einen String als Pfadangabe oder den booleschen Wert FALSE, wenn in dem von ihr geöffneten Dialog die Schaltfläche 'Abbrechen' betätigt wurde:

```
Sub Oeffnen()
   Dim varFile As Variant
   varFile = Application.GetOpenFilename("Excel-Dateien (*.xls), *.xls")
   If varFile = False Then Exit Sub
   Workbooks.Open varFile
End Sub
```

Ein anderes Beispiel ist die Funktion IsMissing, mit der geprüft werden kann, ob einer Funktion ein optionales Argument übergeben wurde:

```
Public Sub EingabeMöglich(Optional Wert As Variant)
    If IsMissing(Wert) Then
        MsgBox "Kein Argument übergeben"
    Else
        MsgBox Wert
    End If
End Sub
```

Falls das übergebene Argument in (Optional Wert As String) geändert wird, funktioniert IsMissing() nicht mehr und das Programm durchläuft immer den Else-Zweig.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-14

Einsatz von Public-Variablen

Im nachfolgenden Beispiel wird in einem Standardmodul eine Public-String-Variable deklariert. Diese wird in der Prozedur AufrufenMeldung mit einem Wert belegt; danach wird das Unterprogramm Meldung aufgerufen. Da die Variable außerhalb der Prozeduren deklariert wurde, ist der Wert nicht verlorengegangen und kann weiterverwertet werden.

```
Public strMsg As String
```

```
Sub AufrufenMeldung()
    strMsg = "Hallo!"
    Call Meldung
End Sub

Sub Meldung()
    MsgBox strMsg
End Sub
```

Auch wenn sich die Prozedur Meldung in einem anderen Modul befindet, funktioniert der Aufruf. Erfolgt jedoch die Deklaration mit Dim oder als Private, gilt sie nur für das jeweilige Modul.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-15

Übergabe von String-Variablen

Eine Vorgehensweise wie im vorhergehenden Beispiel ist zu meiden und eine Übergabe der Variablen als Parameter ist vorzuziehen:

```
 Sub AufrufenMeldung()
     Dim strMsg As String
     strMsg = "Hallo!"
     Call Meldung(strMsg)
 End Sub

 Sub Meldung(strMsg As String)
     MsgBox strMsg
 End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-16

Variablen in Funktionen

Funktionen werden eingesetzt, wenn Werte zurückgeliefert werden müssen. Eine Alternative wäre (neben einer ByRef-Variablenübergabe) der Einsatz von Public-Variablen, die wir ja meiden wollen. Bei den Parametern einer Funktion handelt es sich ebenfalls um Variablen. Der Deklarationsbereich liegt innerhalb der Klammern der Funktion. Diese Parameter müssen beim Aufruf der Funktion - aus einem Tabellenblatt oder aus einer anderen Prozedur - übergeben werden. In der nachfolgenden Funkion wird die Kubatur errechnet:

```
Function Kubatur( _
   dblLaenge As Double, _
   dblBreite As Double, _
   dblHoehe As Double) As Double
   Kubatur = dblLaenge * dblBreite * dblHoehe
End Function
```

Die Eingabesyntax einer solchen Prozedur in einem Tabellenblatt ist, wenn die Werte in den Zellen A1:C1 stehen:

```
=kubatur(A1;B1;C1)
```

Wird die Funktion aus einer anderen Prozedur zur Weiterverarbeitung aufgerufen, sieht das wie folgt aus:

```
Sub ErrechneGewicht()
   Dim dblSpezGewicht As Double, dblKubatur As Double
   dblSpezGewicht = 0.48832
   dblKubatur = Kubatur(Range("A1"), Range("B1"), Range("C1"))
   Range("E1").Value = dblKubatur * dblSpezGewicht
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-17

Hierarchische Anordnung der Objekttyp-Variablen

Über die Objekttypvariablen kann ein Typengerüst aufgebaut werden, indem die jeweils aktuelle Ebene referenziert wird:

```
Sub NeueSymbolleiste()
   Dim objCmdBar As CommandBar
   Dim objPopUp As CommandBarPopup
   Dim objButton As CommandBarButton
   Dim intMonth As Integer, intDay As Integer
   On Error Resume Next
   Application.CommandBars("Jahr " & Year(Date)).Delete
   On Error GoTo 0
   Set objCmdBar = Application.CommandBars.Add("Jahr " & Year(Date), msoBarTop)
   For intMonth = 1 To 12
      Set objPopUp = objCmdBar.Controls.Add(msoControlPopup)
      objPopUp.Caption = Format(DateSerial(1, intMonth, 1), "mmmm")
      For intDay = 1 To Day(DateSerial(Year(Date), intMonth + 1, 0))
         Set objButton = objPopUp.Controls.Add
         With objButton
            .Caption = Format(DateSerial(Year(Date), intMonth, intDay), _
               "dd.mm.yy - dddd")
            .OnAction = "MeldenTag"
            .Style = msoButtonCaption
         End With
      Next intDay
   Next intMonth
   objCmdBar.Visible = True
End Sub
```

Mit vorstehendem Code wird eine neue Symbolleiste mit dem Namen des aktuellen Jahres angelegt und im Symbolleistenbereich als nächstuntere platziert. Der Leiste wird für jeden Monat ein Menü und diesem Menü wird für jeden Tag eine Schaltfläche hinzugefügt.

Das Auslesen der betätigten Schaltfläche und die Datumsberechnungen erfolgen anhand einer Datumsvariablen:

```
Private Sub MeldenTag()
    Dim datAC As Date
```

```
    datAC = DateSerial(Year(Date), Application.Caller(2), Application.Caller(1))
    Select Case datAC
        Case Is < Date
            MsgBox Date - datAC & " Tage vergangen"
        Case Is = Date
            MsgBox "Heute"
        Case Is > Date
            MsgBox "Noch " & datAC - Date & " Tage"
    End Select
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Variablen_und_Arrays&action=edit&section=T-18

Collections von Objekttyp-Variablen

Das Objekt UserForm1.Controls stellt alle Steuerelemente dar, die in der UserForm1 enthalten sind. Nicht ganz so einfach ist es, auf alle CheckBoxes dieser UserForm zuzugreifen, um sie über eine Schleife zu , denn die CheckBox ist kein gültiges Objekt, das heißt Controls. Liest man die CheckBoxes in ein Collection-Objekt ein, lassen Sie sich später problemlos ansprechen und in Schleifen einbinden:

```
Public colChBox As New Collection

Private Sub UserForm_Initialize()
   Dim cnt As Control, intMonth As Integer
   For Each cnt In Controls
      If TypeName(cnt) = "CheckBox" Then
         intMonth = intMonth + 1
         colChBox.Add cnt
         cnt.Caption = Format(DateSerial(1, intMonth, 1), "mmmm")
      End If
   Next cnt
End Sub
```

Das Collection-Objekt wird - damit es seinen Wert nicht verliert - als Public außerhalb einer Prozedur deklariert und im Initialisierungscode der UserForm mit den Einzelobjekten - den 12 CheckBoxes der UserForm - belegt. Beim Klick auf die Schaltfläche Meldung werden alle aktivieren CheckBoxes in einer MsgBox ausgegeben:

```
Private Sub cmdMeldung_Click()
   Dim intCounter As Integer
   Dim strMsg As String
   strMsg = "Aktiviert:" & vbLf
   For intCounter = 1 To 12
      If colChBox(intCounter).Value Then
         strMsg = strMsg & colChBox(intCounter).Caption & vbLf
      End If
   Next intCounter
   MsgBox strMsg
```

```
End Sub
```

Grundlagen 3 (englisch)

Using Variables in Excel VBA Macro Code

VARIABLES

A Variable is used to store temporary information that is used for execution within the Procedure, Module or Workbook. Before we go into some detail of Variables, there are a few important rules that you must know about.

1) A Variable name must Start with a letter and not a number. Numbers can be included within the name, but not as the first character.
2) A Variable name can be no longer than 250 characters.
3) A Variable name cannot be the same as any one of Excel's key words. By this, I mean you cannot name a Variable with such names as Sheet, Worksheet etc.
4) All Variables must consist of one continuous string of characters only. You can separate words by either capitalising the first letter of each word, or by using the underscore characters if you prefer.

You can name variables with any valid name you wish. For Example you could name a variable "David" and then declare it as any

one of the data types shown below. However, it is **good practice to formalize some sort of naming convention**. This way when reading back your code you can tell at a glance what data type the variable is. An example of this could be the system I use! If you were to declare a variable as a Boolean (shown in table below) I may use: **bIsOpen** I might then use this Boolean variable to check if a Workbook is open or not. The "b" stands for Boolean and the "IsOpen" will remind me that I am checking if something is open.

You may see code that uses letters only as variables, **this is bad programming and should be avoided**. Trying to read code that has loads of single letters only can (and usually does) cause grief.

Variables can be declared as any one of the following data types:

Byte data type

A data type used to hold positive integer numbers ranging from 0 to 255. Byte variables are stored as single, unsigned 8-bit (1-byte) numbers.

Boolean data type

A data type with only two possible values, True (-1) or False (0). Boolean variables are stored as 16-bit (2-byte) numbers.

Integer data type

A data type that holds integer variables stored as 2-byte whole numbers in the range -32,768 to 32,767. The Integer data type is also used to represent enumerated values. The percent sign (%) type-declaration character represents an Integer in Visual Basic.

Long data type

A 4-byte integer ranging in value from -2,147,483,648 to 2,147,483,647. The ampersand (&) type-declaration character represents a Long in Visual Basic.

Currency data type

A data type with a range of -922,337,203,685,477.5808 to 922,337,203,685,477.5807. Use this data type for calculations involving money and for fixed-point calculations where accuracy is particularly important. The at sign (@) type-declaration character represents Currency in Visual Basic.

Single data type

A data type that stores single-precision floating-point variables as 32-bit (2-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values, and 1.401298E-45 to 3.402823E38 for positive values. The exclamation point (!) type-declaration character represents a Single in Visual Basic.

Double data type

A data type that holds double-precision floating-point numbers as 64-bit numbers in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values. The number sign (#) type-declaration character represents the Double in Visual Basic.

Date data type

A data type used to store dates and times as a real number. Date variables are stored as 64-bit (8-byte) numbers. The value to the left of the decimal represents a date, and the value to the right of the decimal represents a time.

String data type

A data type consisting of a sequence of contiguous characters that represent the characters themselves rather than their numeric values. A String can include letters, numbers, spaces, and punctuation. The String data type can store fixed-length strings ranging in length from 0 to approximately 63K characters and dynamic strings ranging in length from 0 to approximately 2 billion characters. The dollar sign ($) type-declaration character represents a String in Visual Basic.

Object data type

A data type that represents any Object reference. Object variables are stored as 32-bit (4-byte) addresses that refer to objects. Variant data type A special data type that can contain numeric, string, or date data as well as the special values Empty and Null. The Variant data type has a numeric storage size of 16 bytes and can contain data up to the range of a Decimal, or a character

storage size of 22 bytes (plus string length), and can store any character text. The VarType function defines how the data in a Variant is treated. All variables become Variant data types if not explicitly declared as some other data type.

Why we use variables

Excel will still allow us to run our code without using variables, it is not a must! But having said this it is very bad programming to not use variables. You could quite easily just assign a value, string or whatever each time you need it, but it would mean:

1) Your code would become hard to follow (even for yourself)

2) Excel would constantly need to look for the value elsewhere.

3) Editing your code would become awkward.

Let's use an example to highlight the above

```
Sub NoVariable()

          Range("A1").Value = Range("B2").Value

          Range("A2").Value = Range("B2").Value * 2

          Range("A3").Value = Range("B2").Value * 4

          Range("B2").Value = Range("B2").Value * 5

End Sub
```

In the above code, Excel would need to retrieve the value from cell **B2 five times**. It would also mean if we had many other procedures using the same value i.e **B2**, it would need to retrieve it's value even more times.

There is a lot of editing to be done if we were to change from wanting **B2** value to say, **B5** value. It is messy code.

Let's now use a variable to store the value of cell **B2!**

```
Sub WithVariable()
Dim iMyValue as Integer
          iMyValue = Range("B2").Value
          Range("A1").Value = iMyValue
          Range("A2").Value = iMyValue * 2
          Range("A3").Value = iMyValue * 4
          Range("B2").Value = iMyValue * 5
End Sub
```

In the above code Excel only needs to retrieve the value of cell **B2 once**.

To edit our code we only need to change it in one place.

It is easier to read.

You might be thinking that there is no big difference in the above 2 examples, and to a point you would be correct. But what you must realize is, most VBA projects will consist of hundreds (if not thousands) of lines of code. They would also contain a lot more than one procedure. If you had 2 average size VBA projects, one using variables and one without, the one using variables would run far more efficiently!

Declaring Variables

To declare a variable we use the word "Dim" (short for Dimension) followed by our chosen variable name then the word "As" followed by the variable type. So a variable dimmed as a String could look like:

Dim sMyWord As String

You will notice that as soon as we type the word As, Excel will display a drop-down list of all variables.

The default value for any Numeric type Variable is zero.
The default value for any String type variable is "" (empty text).
The default value for an Object type Variable is Nothing. While the default value for an Object type Variable is Nothing, Excel will still reserve space in memory for it.

To assign a value to a Numeric or String type Variable, you simply use your Variable name, followed by the equals sign (=) and then the String or Numeric type. eg:

```
Sub ParseValue()
Dim sMyWord as String
Dim iMyNumber as Integer
```

```
    sMyWord = Range("A1").Text
    iMyNumber = Range("A1").Value
End Sub
```

To assign an Object to an Object type variable you must use the key word "Set". eg:

```
Sub SetObJect()
Dim rMyCell as Range
    Set rMyCell = Range("A1")
End Sub
```

In the example immediately above, we have set the Object variable to the range **A1**. So when we have finished using the Object Variable "rMyCell" it is a good idea to Set it back to it's default value of Nothing. eg:

```
Sub SetObjectBack()
Dim rMyCell as Range
    Set rMyCell = Range("A1")

    Set rMyCell = Nothing
End Sub
```

This will mean Excel will not be reserving unnecessary memory.

In the first example above (Sub ParseValue()) we used 2 lines to declare our 2 variables ie

Dim sMyWord as String

Dim iMyNumber as Integer

We can, if we wish just use:

Dim sMyWord as String, iMyNumber as Integer

There is no big advantage to this, but you may find it easier.

Not Declaring Variables

There is a difference between using variables and correctly declaring them. You can if you wish not declare a variable and still use it to store a Value or Object. Unfortunately this comes at a price though! If you are using variables which have not been dimensioned Excel (by default) will store them as the Variant data type. This means that Excel will need to decide each time it (the variable) is assigned a value what data type is should be. The price for this is slower running of code! My advise is do it right and form the good habit early!

There is also another advantage to correctly declaring variables and that is Excel will constantly check to ensure you have spelt the variable name correctly. It does this by capitalizing the all lower case letters that are capitalized at the point it was dimensioned. Let's assume you you use:

Dim iMyNumber As Integer

At the top of your procedure. You then intend to use this variable in other parts of the procedure. Each time you type **imynumber** and then push the Space bar or Enter Excel will capitalize it for you i.e **imynumber** will become **iMyNumber**. This is a very simple and easy way to ensure you have used the correct spelling.

While we are on this subject, it is very good practice to type all code in lower case, because not only will Excel do this for variables but also for all Keywords!

There may be times when you will actually need to use a Variant data type as you cannot be certain what will be parsed to it, say from a cell. It might be text, it maybe a very low or high number etc. In these circumstances you can use:

Dim vUnKnown As Variant

Or, simply:

Dim vUnKnown

Both are quite valid! The reason we do not have to explicitly declare a Variant is because the default for a variable is a Variant.

The Scope and Lifetime of Excel VBA Variables

In Excel, when coding in VBA, we can use what are know as variables to store information. These variables (as the name suggests) can be varied and changed to store different data information. As soon as a variable loses scope it loses its stored value.

Excel VBA Variables Levels

There are 3 levels at which we can dimension (Dim) variables. These are;

1) Procedure-Level

2) Module-Level

3) Project-Level, Workbook Level, or Public Module-Level

Each of these levels differ in scope and lifetime. This is discussed below

Procedure-Level Variables

These are probably the best known and widely used variables. They are dimensioned (Dim) inside the Procedure itself. See Example below;

```
Sub MyMacro ()
Dim lRows as Long
          'Code Here
End Sub
```

All variables dimensioned at this level are only available to the Procedure that they are within. As soon as the Procedure finishes, the variable is destroyed.

Module-Level Variables

These are variables that are dimensioned (Dim) outside the Procedure itself at the very top of any Private or Public Module. See Example below;

```
Dim lRows as Long


Sub MyMacro ()
    'Code Here
End Sub
```

All variables dimensioned at this level are available to all Procedures that they are within the same Module the variable is dimensioned in. Its value is retained unless the Workbook closes or the **End Statement** is used.

Project-Level, Workbook Level, or Public Module-Level

These variables are dimensioned at the top of any standard public module, like shown below;

```
Public lRows as Long
```

All variables dimensioned at this level are available to all Procedures in all Modules. Its value is retained unless the Workbook closes or the **End Statement** is used.

Arrays

Da die EDV immer von 0 weg zu zählen beginnt, beginnen Arrays, wenn man keinen Anfang definiert, immer mit der Indizes-Positionsnummer 0.

Möchte man lieber mit der Position 1 beginnen, gibt es dazu zwei Lösungen

1.) Lösung: beim Definieren des jeweiligen Arrays wird die Untergrenze angegeben

```
Dim WOCHENTAG (1 to 7) as string

WOCHENTAG=Array("Montag", "Dienstag", "Mittwoch","Donnerstag","Freitag","Samstag","Sonntag")
```

2.) Lösung: Auf Modulebene VOR allen Funktionen und Prozeduren KANN man mit Option Base 1 für alle Arrays in diesem Modul festlegen, dass ihre Indizes nicht mit der Position 0, sondern mit der Position 1 beginnen.

```
Option Base 1

Sub MEINEPROZEDUR

Dim WOCHENTAG (7) as String  ' eindimensionales String-Array, das mit den Indizes von (1) bis (7) befüllt ist (wegen Option Base 1)
WOCHENTAG(1)="Montag"
WOCHENTAG(2)="Dienstag"
…
WOCHENTAG(7)="Sonntag"

End Sub
```

Mehrdimensionales Array

```
Dim MEINARRAY (10,12,2) as string ' dreidimensionales String-Array

' oder auch

Dim Monatstage As Variant: ' Feld mit Anzahl der Tage für die 12 Monaten

Monatstage = Array(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

XY=Monatstage(0) : '  => XY=31
```

Hinweis: den ersten Eintrag (31) spricht man mit Position 0 an

Möchte man, dass die Arrayzählung nicht bei 0 beginnt, dann verwendet man OPTION BASE 1

```
'Wichtig damit die Array Zählung bei 1 startet und nicht wie sonst bei 0
Option Base 1
```

Arrays neu dimensionieren

Gerade wenn nicht absehbar ist, wie groß ein Array benötigt werden wird und es um mehr als 100 Elemente geht, kann es Sinn machen die Größe des Arrays dynamisch anzupassen.

Beim folgenden Beispiel werden die Tabellenblattnamen in ein Array geschrieben und das Array dynamisch angepasst mit dem Befehl ReDim

```
Public Sub SheetArray()

Private Sub Build_Worksheet_Array()
ArraySize = 0
For j = 1 To Sheets.Count              ' Provides a count of all worksheets
   If TypeName(Sheets(j)) = "Worksheet" Then    ' Gives the type of worksheet
      MyWorkSheetName = Sheets(j).Name                ' Provides the name for the worksheet
      ArraySize = ArraySize + 1
      ReDim Preserve SubsheetArray(ArraySize)
      SubsheetArray(ArraySize) = MyWorkSheetName
   End If
Next
End Sub
```

Die **ReDim**-Anweisung dient normalerweise zum Festlegen oder Ändern der Größe eines dynamischen Datenfeldes, das bereits formal mit einer **Private**-, **Public**- oder **Dim**-Anweisung und einem leeren Klammernpaar (ohne Indizes für die Dimensionen) deklariert wurde.
Sie können die **ReDim**-Anweisung mehrmals verwenden, um die Anzahl der Elemente und Dimensionen in einem Datenfeld zu ändern. Sie können allerdings nicht ein Datenfeld mit einem Datentyp deklarieren und später den Datentyp mit **ReDim** ändern. Dies ist nur möglich, wenn sich das Datenfeld in einem **Variant**-Wert befindet. In diesem Fall können Sie den Datentyp der Elemente mit einem **As** *Typ*-Abschnitt ändern, bis Sie das **Preserve**-Schlüsselwort verwenden, das keine Änderungen der Datentypen mehr zuläßt.

Die Anzahl der Dimensionen kann nicht verändert werden.

Mit dem Schlüsselwort **Preserve** können Sie nur die Größe der letzten Datenfelddimension ändern. Wenn das Datenfeld zum Beispiel nur eine Dimension hat, können Sie die Größe dieser Dimension ändern, weil es die letzte und einzige Dimension ist. Hat das Datenfeld jedoch mindestens zwei Dimensionen, so können Sie lediglich die Größe der letzten Dimension ändern, wenn der Inhalt des Datenfeldes erhalten bleiben soll. Das folgende Beispiel verdeutlicht, wie Sie die letzte Dimension eines dynamischen Datenfeldes vergrößern können, ohne bereits bestehende Daten im Datenfeld zu löschen.

```
ReDim X(10, 10, 10)
. . .
```

ReDim Preserve X(10, 10, 15)

Syntax
ReDim [**Preserve**] *VarName***(***Indizes***)** [**As** *Typ*] [**,** *VarName***(***Indizes***)** [**As** *Typ*]] **. . .**
Die Syntax der **ReDim**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Preserve	Optional. Schlüsselwort, das zum Erhalt der Daten in einem bestehenden Datenfeld verwendet wird, wenn die Größe der letzten Dimension geändert wird.
VarName	Erforderlich. Name der Variablen gemäß den Standardkonventionen für Namen von Variablen.
Indizes	Erforderlich. Dimensionen einer Datenfeldvariablen. Bis zu 60 Dimensionen sind zulässig. Die Syntax für das Argument *Indizes* ist: [*Untergrenze* **To**] *Obergrenze* [**,**[*Untergrenze* **To**] *Obergrenze*] **. . .** Wenn nicht explizit in *Untergrenze* angegeben, wird die Untergrenze eines Datenfeldes durch die **Option Base**-Anweisung gesteuert. Die Untergrenze ist Null, wenn keine **Option Base**-Anweisung vorhanden ist.
Typ	Optional. Datentyp der Variablen. Zulässige Typen sind: Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (zur Zeit nicht unterstützt), Date, String (für Zeichenfolgen variabler Länge), **String** * *Länge* (für Zeichenfolgen fester Länge), Object, Variant, ein benutzerdefinierter Typ oder ein Objekttyp. Verwenden Sie für jede deklarierte Variable einen separaten **As** *Typ*-Abschnitt. Bei **Variant**-Werten, die ein Datenfeld enthalten, beschreibt *Typ* den Typ der Elemente im Datenfeld, ändert den **Variant**-Wert aber nicht in einen anderen Typ.

Arrays Grundlagen

Matrizen in VBA werden als Arrays bezeichnet. Grundsätzlich gibt es mehrere Möglichkeiten, ein Array zu erzeugen:

- Über Dim als Datenfeld, z.B. ergibt die Anweisung `Dim Matrix(1 To 3, 1 To 3)` eine 3 × 3-Matrix mit der mathematisch richtigen Indizierung der Zeilen und Spalten jeweils von 1..3
- An eine Variable vom Typ Variant kann ein Array aus einer anderen Variablen zugewiesen werden

- Über die Anweisung array() kann an eine Variable vom Typ Variant ein Array zugewiesen werden, z.B. mit `Var1D = array(11,12,13)`; Auf diese Art ist es auch möglich, ein zweidimensionales Array anzulegen, z.B. durch `Var2D = array(array(11, 12), array(21, 22))`; Arrays höherer Dimensionen lassen sich auf vergleichbare Weise anlegen.

Arrays können auch als Rückgabewert einer benutzerdefinierten Funktion definiert werden. Wenn eine benutzerdefinierte Funktion eine 2 × 2-Matrix in ein Tabellenblatt zurückgeben soll, muss auf dem Tabellenblatt zuerst ein Bereich mit 2 × 2 Zellen markiert werden, dann tippt man die Funktion ein und schließt die Eingabe wie bei einer Matrixformel mit Umschalt+Strg+Eingabe ab.

Das Array lässt sich leider nicht als Konstante (über `Const`) speichern - weder in einer Prozedur/Funktion noch im Deklarationsteil eines Moduls.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen_und_Matrizen&action=edit&section=T-1Arrays in VBA

Das erste Beispiel prüft, ob eine Zahl durch eine Gruppe von anderen Zahlen teilbar ist - falls nicht, wird die Zahl selbst zurückgegeben. Der Vorteil bei dieser Schreibweise mit einem array() ist, dass das Programm zu einem späteren Zeitpunkt ohne besondere Kenntnisse des Codes erweitert werden kann, indem man der `TeilerListe` einfach noch ein paar Zahlen anhängt:

```
Public Function TeilerGefunden(Zahl As Long) As Long
    Dim TeilerListe As Variant  ' Liste der Primteiler
    Dim Teiler      As Variant  ' Schleifenvariable

    TeilerListe = Array(2, 3, 5, 7, 11, 13)
    TeilerGefunden = Zahl

    For Each Teiler In TeilerListe
        If Zahl Mod Teiler = 0 Then
            TeilerGefunden = Teiler
            Exit Function
        End If
    Next Teiler
End Function
```

Das nächste Beispiel nutzt folgende Eigenschaften in Excel: Tabellenblätter haben nicht nur einen Namen (Eigenschaft `.Name`), der auf der Registerkarte sichtbar ist, sondern auch einen Objektnamen (Eigenschaft `.CodeName`), der nur im Projekt-Explorer des VBA-Editors sichtbar ist und auch dann unverändert bleibt, wenn der Benutzer das Blatt umbenennt. Das deutsche Excel legt diesen Namen (`.CodeName`) standardmäßig wie den Blattnamen (`.Name`) an, aber -wie geschrieben- ändert er sich `.CodeName` nicht mehr bei einer Umbenennung des Blattes.

In diesem Falle enthält die Arbeitsmappe zwei Blätter, die als Objekte mit `Tabelle1` und `Tabelle2` angesprochen werden können. Die Prozedur bestimmt die Anzahl der benutzten Zellen in jedem Blatt und zeigt sie an:

```
Public Sub BelegungTabellenblätter()
```

```
    Dim ListeAllerTabellen  As Variant ' Liste aller Tabellen
    Dim Tabelle             As Variant ' Schleifenvariable

    ListeAllerTabellen = Array(Tabelle1, Tabelle2) ' Zuweisung des Objektarrays

    For Each Tabelle In ListeAllerTabellen
        MsgBox "Tabelle " & Tabelle.Name & " hat " & _
                Tabelle.UsedRange.Cells.Count & " belegte Zellen"
    Next Tabelle
End Sub
```

Dieses Beispiel zeigt also, dass das `array()` auch Objekte aufnehmen kann. Auch hier bietet sich wieder die einfache Möglichkeit, den Code später einfach von Hand zu ergänzen.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen_und_Matrizen&action=edit&section=T-2

Eindimensionale vordimensionierte Matrix füllen

Eine dimensionierte eindimensionale Matrix wird mit der Zählvariablen gefüllt und danach werden die Werte per MsgBox ausgegeben.

```
Sub FuellenMatrixEinfach()
   Dim arrNumbers(1 To 3) As Integer
   Dim intCounter As Integer
   For intCounter = 1 To 3
      arrNumbers(intCounter) = intCounter
   Next intCounter
   For intCounter = 1 To UBound(arrNumbers)
      MsgBox arrNumbers(intCounter)
   Next intCounter
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen_und_Matrizen&action=edit&section=T-3

Eindimensionale Matrix mit vorgegebenem Wert dimensionieren und füllen

Die Matrix wird auf die Hälfte der Anzahl der Zeilen der mit A1 verbundenen Zellen dimensioniert. Danach werden die Zellinhalte jeder zweiten Zelle der ersten Spalte in die Matrix eingelesen und über eine MsgBox wieder ausgegeben.

```
Sub FuellenMatrixSingle()
   Dim arrCells() As String
   Dim intCounter As Integer, intCount As Integer, intArr As Integer
   Dim strCell As String
   intCount = Range("A1").CurrentRegion.Rows.Count / 2
   ReDim arrCells(1 To intCount)
   For intCounter = 1 To intCount * 2 Step 2
      intArr = intArr + 1
```

```
      arrCells(intArr) = Cells(intCounter, 1)
   Next intCounter
   For intCounter = 1 To UBound(arrCells)
      MsgBox arrCells(intCounter)
   Next intCounter
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Schleifen_und_Matrizen&action=edit&section=T-4

Mehrdimensionale Matrix füllen

Der mit der Zelle A1 zusammengehörige Bereich wird in eine Matrix ein- und eine einzelne Zelle über MsgBox wieder ausgelesen.

```
Sub FuellenMatrixMulti()
   Dim arrJahr As Variant
   arrJahr = Range("A1").CurrentRegion
   MsgBox arrJahr(3, 2)
End Sub
```

Das folgende Beispiel zeigt, wie man den markierten Bereich im aktiven Tabellenblatt ausliest. Die Funktion geht die Auswahl im Tabellenblatt Zeile für Zeile und dann Spalte für Spalte durch. Jeder gefundene Wert wird in ein Element der Variablen Matrix gespeichert. Diese ist dann der Rückgabewert der Funktion MatrixFüllen():

```
Public Function MatrixFüllen() As Double()
    Dim ZeileNr     As Long     ' Zeilenzähler
    Dim SpalteNr    As Long     ' Spaltenzähler

    Dim Matrix()    As Double   ' Matrix

    ' Matrix auf Zeilen- und Spaltenzahl der Auswahl bringen
    ' Dabei soll jeder Index bei 1 beginnen
    ReDim Matrix(1 To Selection.Rows.Count, 1 To Selection.Columns.Count)

    ' Auswahl zeilenweise lesen
    For ZeileNr = 1 To Selection.Rows.Count
        ' Auswahl spaltenweise lesen
        For SpalteNr = 1 To Selection.Columns.Count
            With Selection.Cells(ZeileNr, SpalteNr)
                If IsNumeric(.Value) Then
                    ' Matrix elementweise füllen
                    Matrix(ZeileNr, SpalteNr) = .Value
                Else
                    ' Fehlermeldung ausgeben
                    MsgBox "Zelle " & .Address & " enthält keine Zahl"
                    Exit Function
                End If
            End With
```

```
        Next SpalteNr
    Next ZeileNr

    ' Rückgabewert der Funktion
    MatrixFüllen = Matrix
End Function
```

Die Funktion MatrixFüllen() erstellt die Größe der Matrix anhand der Markierung dynamisch und weist den Inhalt der Matrix dem Rückgabewert der Funktion zu. Zur dynamischen Dimensionierung gehört im Beispiel auch, dass der Index der Matrix mit 1 beginnend definiert wird (mathematische Notation), ohne diese Angabe würde Excel gewohnheitsmäßig die Indizes bei 0 beginnen lassen. Falls eine Zelle keine Zahl enthält, erscheint eine Fehlermeldung. Leere Zellen werden als 0 interpretiert.

Coder vor der ersten Prozedur

Option Explicit: ' jede Variable muss vor erster Verwendung deklariert werden

Public Variablenname as Variablentyp
(veraltet "Global") definiert Variablen für alle Module/Userforms

Public Sub PROZEDURNAME
... solche Prozeduren sind aus allen Modulen / Userforms aufrufbar - normale Prozeduren kann man ja nur innerhalb des aktuellen Moduls / Userforms aufrufen

Dim Variablenname as Variablentyp
… VOR dem Code der SUB's definiert allgemeine Variablen, die in allen SUB's dieses Moduls gelten

Übergabe von Variablen an eine Prozedur

Der Hauptteil ist erklärt im Bereich FUNKTIONEN - PROZEDUREN ALLGEMEIN

Die Prozedur sei:

Sub CALCULATING_EMPLOYERS_WORKING_DAYS(FIRSTDAY As Integer, LASTDAY As Integer)

Der Aufruf dieser Prozedur aus einer anderen Prozedur und die Übergabe der Werte 1 und 2 erfolgt auf 3 verschiedene Möglichkeiten

If FIRSTDAY <> 0 Then Call CALCULATING_EMPLOYERS_WORKING_DAYS(1, 2)
If FIRSTDAY <> 0 Then CALCULATING_EMPLOYERS_WORKING_DAYS 1, 2
If FIRSTDAY <> 0 Then CALCULATING_EMPLOYERS_WORKING_DAYS FIRSTDAY:=1, LASTDAY:=2

Ein klassisches Beispiel ist meine Prozedur SORTIERE AUFSTEIGEN (ABSTEIGEND) im Bereich ZELLEN - SPALTEN - ZEILEN

Im Gegensatz zu der hier übergebenen Variablen an eine Prozedur steht die Übergabe von Variablen an eine Funktion. Diese ergeben immer ein Ergebnis und die Funktion muss entsprechend auch aufgerufen etwa: SPALTENBUCHSTABE = SPALTENTEXTUMWANDELN(1)

Siehe auch FUNKTIONEN ALLGEMEIN - BENUTZERDEFINIERTE FUNKTIONEN FÜR VBA

Übergabe von Variablen an eine Prozedur in anderer Arbeitsmappe

Der Hauptteil ist erklärt im Bereich FUNKTIONEN - PROZEDUREN ALLGEMEIN

Man kann auch Variablen übergeben an Prozedur in einer anderen Arbeitsmappe:

Dokument1:

Code:

```
Public Sub test()
DasIstEinTest = Application.Run("Dokument2.xls'!testSub", DasIstEinTest)
End Sub
```

Dokument2:

Code:

```
Public Function testSub(ByRef blabla As Integer)
testSub = blabla +5
End Function
```

VBA-Code

VBA-Prozedur in anderer, offenen Arbeitsmappe aufrufen

Wir haben zwei Mappen - in der Datei Mappe2.xlsm gibt es einen VBA-Code, der die Prozedur "Sub TEST" in der bereits offenen Datei Mappe1.xlsm aufrufen soll.
So sieht der Copde in der Mappe2 aus, um die Prozedur TEST in der offenen Mappe1.xlsm aufzurufen

Application.Run ("'Mappe1.xlsm'!TEST")

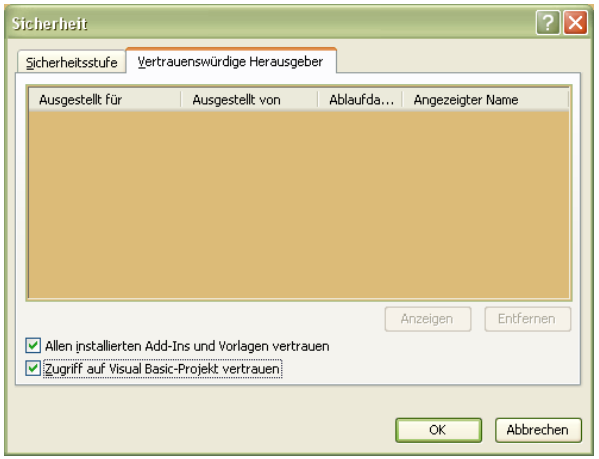
Probleme gabs, wenn meine Datei im Dateinamen schon mit einem Hochstrich begann - die Datei ' Buchhaltung.xlsm war so nicht erreichbar und die Prozedur in ihr nicht aufrufbar

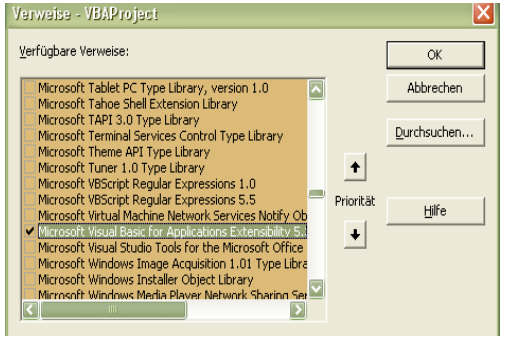
VBA-Code direkt anspringen

Application.goto reference:="[Sub-Name]" öffnet die VBA-Umgebung und springt direkt zum entsprechenden Code der übergebenden Prozedur

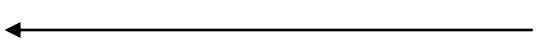
0.) Voraussetzungen

For full manipulation of VBA-Code in other Excel workbooks, the reference to the Microsoft VBA Extensibility library must be set in the VBA-environment.





The second requirement is the macro-security-parameter in Excel that hinders/allows the manipulation of VBA-Code.



Usually a once set parameter stays. If he resets every new start of the PC a group policy in the network can be the reason.

A second hint can be found in the Internet: https://groups.google.com/forum/?fromgroups#!topic/microsoft.public.excel.programming/aYKg9PErA4w

OK, I found the answer. If anyone is interested. I had changed the key HKLM\Software\Microsoft\Office\11\Excel\Security\AccessVBOM to 1.

I expected this to make the UI check box a non-greyed-out option, but it left it greyed out. But the box was not checked instead of unchecked. So here is how it works. If you leave the key there and it is a zero the option will be greyed out and unchecked. If you leave the key there and make it a 1, the option will be greyed out but checked. If you remove the key entirely the option will no longer be greyed out and the user can chose to check it or not.

http://microsoft.public.de.excel.narkive.com/XACfT3QS/zugriff-auf-visual-basic-projekt-bleibt-nicht-dauerhaft

Frage: In Excel 2003 und Word 2003 ist nach einem Neustart die Option "Zugriff auf Visual Basic-Projekt vertrauen" wieder inaktiv. Ebenfalls bleibt die Makrosicherheit immer auf Mittel.

Lösung: Wenn ich den Key Security unter HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Excel lösche dann funktioniert es wieder!

0.) Mein Beispiel-Code

Wichtig: die beiden Voraussetzungen im obigen Bereich VORAUSSETZUNGEN müssen erfüllt sein

Achtung: der Code lief problemlos unter deutschem Excel 2003, und englischem Excel 2010, bei Seibersdorf englischem Excel 2003 stürzte Excel beim Schreiben von Zeilen in ThisWorksheet, in ein einzelnes Sheet und in das ActiveX-Commandbutton immer ab.

Ich habe dann einen Code hinbekommen, der funktionierte – (er schreibt zwar in Chart-Sheets hinein – sollte aber auch bei normalen Sheets funktionieren). Der entsprechende Code ist dann im BSP 2

Das Problem trat auch bei anderen, englisch-sprachigen Usern auf und es wurde keine Lösung gefunden im Internet.

Wir haben den Code, der in die neue Excelmappe übertragen wird übrigens nicht in die VBA-Umgebung eingefügt (durch den .Insertlines Linenum – Befehl vor jeder Zeile wird der Code etwas schwerer lesbar), sondern haben den Code in ein (danach ausgeblendetes) Arbeitsblatt direkt eingefügt und schaufeln von dort (siehe hier Beispiel 2) den Code raus direkt in die Ziel-Arbeitsmappe.

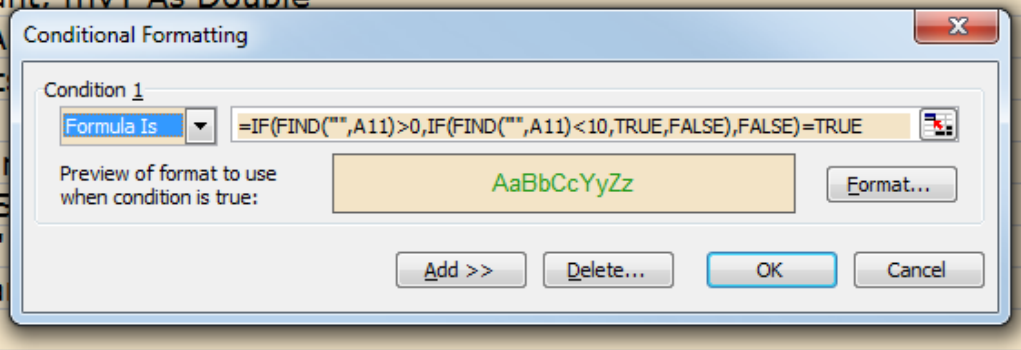
Um den Code farblich ein bisschen zu „highlighten" habe ich in Spalte A den Code mit der folgenden bedingten Formatierung grün getextet, sobald innerhalb der ersten 10 Zeichen ein ' ist (daher: wenn der Kommentar erst nach der 10 Stelle beginnt, weil davor noch Code in dieser Zeile ist, dann bleibt die ganze Zeile schwarz).

=IF(FIND("",A11)>0,IF(FIND("",A11)<10,TRUE,FALSE),FALSE)=TRUE

A11 Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)

	A
1	This page is only for editing the VBA-Code for the button in the Countrate-Overview-File
2	The first 9 lines are for comment purposes - the first code-line starts in line 10
10	VBA-Code (Empty Lines are allowed !)
11	Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
12	Dim ElementID As Long, Arg1 As Long, Arg2 As Long
13	Dim myX As Variant, myY As Double
14	Dim l As Long, c A
15	Dim wks As Sheet
16	Dim ch As Chart
17	Dim wbCO As Stri
18	Dim wbPsWin As S
19	Dim ID As String '
20	Dim ID_cell As Ra
21	Dim ANSWER
22	
23	wbCO = ActiveWorkbook.Name ' reading filename of this countrates-overview-workbook
24	wbPsWin = Sheets("PsWin_Filename").Range("A1").Value ' reading the filename of the PsWin-workbook
25	
26	' --- leave routine if right mouse-button was clicked
27	
28	If Button = 2 Then ' a pressed down right mousebutton has the value 2
29	Exit Sub ' don't execute procedure when right mousekey is pressed down for context-menu
30	End If
31	
32	' --- get the data of the element clicked on
33	

Hide this sheet



BEISPIEL 1 (läuft unter dt.Excel2003, dt./eng.Excel 2010)

WICHTIG: wenn man sich davon nur Teile herausholt, dann braucht es die eine oder andere Definitionszeile von einem vorigen VBA-Generier-Bereich

z.B. wenn man ein leeres Workbook bereits erzeugt hat (mit Workbooks.Add**) dann kann man** Set NewWkb = Activeworkbook **einfügen, um die NewWkb-Variable richtig zu setzen.**

WICHTIG2: es gab beim Erzeugen eines Formularbuttons Probleme, wenn dieser in einer Datei erzeugt werden sollte, der - und # im Namen hatte. Dann gab es eine Fehlermeldung bei der zeile:

NewButton.OnAction = NewWkb.Name & "!Modul4Button.MyButtonProc"

Wichtig3: Wir hatten den Fehler, dass beim Zuweisen von VBA-Code die folgende Zeile

Set NewComp = NewProj.VBComponents(Chart_Codename)

immer einen Fehler brachte, weil Chart_Codename leer war.

Die Fehlerbehebung war ganz seltsam: wenn der VBA-Editor offen war, lief der Code problemlos – war er zu, kam es zum Fehler.

Also haben wir unseren Code für den Button Countrates_Overview so umgebaut, dass zuvor VBA geöffnet wurde und danach wieder zugemacht wurde

```
' Zeige VBA an
Application.VBE.MainWindow.SetFocus
Application.VBE.MainWindow.Visible = True
Application.VBE.MainWindow.WindowState = vbext_ws_Minimize

' Führe unsere Prozeduren aus
Save_Workbook_Name
Countrates_overview_code

' hier unsere Sorgenkind-Prozedur
```

Create_VBA_Code

' wieder Ausblenden von VBA
Application.VBE.MainWindow.Visible = False

Das wars :o)

Und hier mein Codebeispiel

```
Sub Code_Generating()

    ' setting variables
    Dim ThisWkb As Workbook ' the active workbook with this code
    Dim NewWkb As Workbook ' the new workbook in which VBA-code will be generated
    Dim VBComp As VBComponent ' the VB-Component (in our case: a new modul) that will be generated
    Dim VBCodeMod As CodeModule ' the newly added module
    Dim LineNum As Long ' the line-number for the newly added code-lines

    Dim NewProj As VBIDE.VBProject ' this is for generating code in worksheets of the new workbook
    Dim NewComp As VBIDE.VBComponent ' this is for generating code in worksheets of the new workbook

    Dim ObjSh As Object ' this is for changing Worksheet_Codename into Worksheet_Name
    Dim WkShName As String ' name of the worksheet that gets VBA-Code

    Dim NewButton As Object ' for creation of a new formular-button
    Dim ActXbutton As OLEObject ' for creation of a new active-x-command-button

    ' saving the name of the active workbook
    Set ThisWkb = ActiveWorkbook

    ' creating a new workbook
    Workbooks.Add
    ActiveWindow.Caption = "test.xls"

    ' Interim-Save-Location (otherwise Workbooks("test.xls").-Command would not work
    Application.DisplayAlerts = False ' for the next line: no "Save and Replace"-warning shall be displayed
```

```
    ActiveWorkbook.SaveAs "C:\test.xls"
    Application.DisplayAlerts = True ' re-activating Excel-warnings

    ' Returning to this workbook
    ThisWkb.Activate

    ' defining the workbook into which the VBA-Code will be generated (it must be open)
    Set NewWkb = Workbooks("test.xls")

' ----------------------------------------
' 2 examples for generating standard modules
' ----------------------------------------

    ' adding a new module in that workbook
    Set VBComp = NewWkb.VBProject.VBComponents.Add(vbext_ct_StdModule) 'vbext_ct_Document would be for code in worksheets, workbooks or diagrammsheets / vbext_ct_MSForm would be for Code in userforms / vbext_ct_ClassModule would be for classmoduls

    ' naming the new module
    VBComp.Name = "MyNewModul"

    ' referenzing this new module
    Set VBCodeMod = NewWkb.VBProject.VBComponents("MyNewModul").CodeModule

    ' inserting new code lines
    With VBCodeMod
        LineNum = .CountOfLines + 1 ' .countoflines seeks the last used linenumber in the module (in our new module it is of course 0)
        .InsertLines LineNum, "Sub MyNewProcedure()" & Chr(13) & " Msgbox ""Here is my new procedure"" " & Chr(13) & "End Sub"
    End With

    ' alternative for adding code
    With VBCodeMod
        LineNum = .CountOfLines + 1
        .InsertLines LineNum, "Sub MyNewProcedure2()"
        .InsertLines LineNum + 1, " Msgbox ""Here is my new procedure 2"" "
        .InsertLines LineNum + 2, "End Sub"
    End With

    ' we could start this new procedure immediately
    'Application.Run "test.xls!MyNewProcedure"

' ---------------------------------------------------------------------------------------------------
' example for generating code for the 'ThisWorkbook'-area in the new workbook
' ---------------------------------------------------------------------------------------------------
```

```
        ' for German Excel
        If Application.LanguageSettings.LanguageID(msoLanguageIDUI) = 1031 Then
            With NewWkb.VBProject.VBComponents("DieseArbeitsmappe").CodeModule
                .InsertLines 1, "Private Sub Workbook_Open()"
                .InsertLines 2, " Msgbox ""Workbook has successfully opened"" "
                .InsertLines 3, "End Sub"
            End With
        End If

        ' for Englisch Excel
        If Application.LanguageSettings.LanguageID(msoLanguageIDUI) = 1033 Then
            With NewWkb.VBProject.VBComponents("ThisWorkbook").CodeModule
                .InsertLines 1, "Private Sub Workbook_Open()"
                .InsertLines 2, " Msgbox ""Workbook has successfully opened"" "
                .InsertLines 3, "End Sub"
            End With
        End If

' ---------------------------------------------
' example for generating code for a worksheet
' ---------------------------------------------

    ' examing name of the active worksheet (English Excel: Sheet1 / German Excel: Tabelle1)
    WkShName = NewWkb.ActiveSheet.Name

    ' renaming the code-name of the active worksheet with our new name "Testsheet"
    NewWkb.VBProject.VBComponents(WkShName).Properties(5).Value = "Testsheet"

    ' renaming the worksheet-name of the active worksheet with our new name "Testsheet"
    NewWkb.ActiveSheet.Name = "Testsheet"

    ' creating code for the worksheet "Testsheet"
    Set NewProj = NewWkb.VBProject
    Set NewComp = NewProj.VBComponents("Testsheet")
    With NewComp.CodeModule
        .DeleteLines 1, .CountOfLines ' deleting all existing code-lines (here: not needed, for the sheet and its code is brandnew)
        .InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
        .InsertLines 2, " Msgbox ""You've clicked on a different cell"",vbOkOnly"
        .InsertLines 3, "End Sub"
    End With

' ---------------------------------------------------------------------------------------------------------------------
' example for generating code for a formular-button created in a worksheet and the new vba-code-lines is located in the worksheet "Sheet1"
' ---------------------------------------------------------------------------------------------------------------------
```

```
    ' --- creating a new module with a procedure for the new button ---

    ' adding a new module in that workbook
    Set VBComp = NewWkb.VBProject.VBComponents.Add(vbext_ct_StdModule)

    ' naming the new module
    VBComp.Name = "Modul4Button"

    ' referenzing this new module
    Set VBCodeMod = NewWkb.VBProject.VBComponents("Modul4Button").CodeModule

    ' inserting new code lines
    With VBCodeMod
       LineNum = .CountOfLines + 1
       .InsertLines LineNum, "Sub MyButtonProc()"
       .InsertLines LineNum + 1, " Msgbox ""You have clicked on the formular-button"" "
       .InsertLines LineNum + 2, "End Sub"
    End With

    ' --- creating the new button ---

    Set NewButton = NewWkb.ActiveSheet.Buttons.Add(255, 40, 180, 40)
    NewButton.Caption = "Click on me (formular)"
    NewButton.Font.Bold = True
    NewButton.OnAction = NewWkb.Name & "!Modul4Button.MyButtonProc"

' ------------------------------------------------------------------------------------------------------------------------
' example for generating code for an ActiveX-command-button created in a worksheet
' ------------------------------------------------------------------------------------------------------------------------

    NewWkb.Activate ' new workbook must be activated

    ' creating the button
    Set ActXbutton = ActiveSheet.OLEObjects.Add(ClassType:="Forms.CommandButton.1", Link:=False, DisplayAsIcon:=False, Left:=285.75, Top:=140.25,
Width:=110.25, Height:=24.75)
    ActXbutton.Name = "myButton"
    ActXbutton.Object.Caption = "Click on me (Active X)"


    ' creating the VBA-Code for clicking on this button
    With NewProj.VBComponents(ActiveSheet.CodeName).CodeModule
       ' .DeleteLines 1, .CountOfLines ' damit würde man ev. existierende Programmzeilen löschen
       .InsertLines 1, "Private Sub myButton_Click()"
```

```
        .InsertLines 2, "Msgbox ""You've clicked the ActiveX-Commandbutton"""
        .InsertLines 3, "End Sub"
    End With



End Sub
```

BEISPIEL 2 (läuft auch unter engl. Excel 2003)

```
Sub Create_VBA_Code()

' This routine was programmed in 2012-11 by Stefan Wenninger

    ' --- Declaring Variables ---

    Dim NewWkb As Workbook ' the new workbook in which VBA-code will be generated
    Dim VBComp As VBComponent ' the VB-Component (in our case: a new modul) that will be generated
    Dim VBCodeMod As CodeModule ' the newly added module
    Dim LineNum As Long ' the line-number for the newly added code-lines
    Dim SourceLineNum As Long ' the line-number from the VBA-Code-Source-Page CO-Code

    ' --- Setting Countrates_Overview-Workbook ---

    Set NewWkb = ActiveWorkbook ' this is the now already opened Countrates_Overview.xls-file

    ' --- Creating a new sheet and force a certain sheet-name ---

    NewWkb.Activate
    NewWkb.Sheets.Add
    NewWkb.ActiveSheet.Name = "PsWin_Filename"
    NewWkb.ActiveSheet.Range("A1") = ThisWkb.Name

' ---------------------------------------------
' example for generating code for a worksheet
' ---------------------------------------------

    ' examing name of the active worksheet (English Excel: Sheet1 / German Excel: Tabelle1)
    WkShName = NewWkb.ActiveSheet.Name

'    ' renaming the code-name of the active worksheet with our new name "Testsheet"
'    NewWkb.VBProject.VBComponents(WkShName).Properties(5).value = "Testsheet"
```

```
'    ' renaming the worksheet-name of the active worksheet with our new name "Testsheet"
'    NewWkb.ActiveSheet.Name = "Testsheet"

    ' creating code for the worksheet "Testsheet"
    Set NewProj = NewWkb.VBProject
    Set NewComp = NewProj.VBComponents("Chart4")

    With NewComp.CodeModule
        .DeleteLines 1, .CountOfLines ' deleting all existing code-lines (here: not needed, for the sheet and its code is brandnew)
         LineNum = .CountOfLines + 1
        For SourceLineNum = 11 To LastLineNum
            .InsertLines LineNum, ThisWkb.Sheets("CO-Code").Cells(SourceLineNum, 1)
            LineNum = LineNum + 1
        Next SourceLineNum
    End With

End Sub
```

Variante BSP.2-Code, die durch alle existierenden Charts durchläuft:

```
Sub Create_VBA_Code()

' This routine was programmed in 2012-11 by Stefan Wenninger
' it generates VBA-Code in the Countrates_Overview.xls-Files

    ' --- Declaring Variables ---

    Dim NewWkb As Workbook ' the new workbook in which VBA-code will be generated
    Dim VBComp As VBComponent ' the VB-Component (in our case: a new modul) that will be generated
    Dim VBCodeMod As CodeModule ' the newly added module
    Dim LineNum As Long ' the line-number for the newly added code-lines
    Dim SourceLineNum As Long ' the line-number from the VBA-Code-Source-Page CO-Code
    Dim Chart_Codename As String ' Codename of the chart-sheet

    ' --- Setting Countrates_Overview-Workbook ---

    Set NewWkb = ActiveWorkbook ' this is the now already opened Countrates_Overview.xls-file

    ' --- Creating a new sheet and force a certain sheet-name ---

    NewWkb.Activate
    NewWkb.Sheets.Add After:=Worksheets(Worksheets.Count)
    NewWkb.ActiveSheet.Name = "PsWin_Filename"
    NewWkb.ActiveSheet.Range("A1") = ThisWkb.Name
```

```
    NewWkb.ActiveSheet.Visible = False
    NewWkb.Sheets(1).Activate

    ' --- loop through all chart-sheets in Countrates_Overview.xls ---

    For xx = 1 To NewWkb.Charts.Count

        Chart_Codename = NewWkb.Charts(xx).CodeName

        ' creating code for the actual chart
        Set NewProj = NewWkb.VBProject
        Set NewComp = NewProj.VBComponents(Chart_Codename)

        With NewComp.CodeModule
            .DeleteLines 1, .CountOfLines ' deleting all existing code-lines (here: not needed, for the sheet and its code is brandnew)
            LineNum = .CountOfLines + 1
            For SourceLineNum = 11 To LastLineNum
                .InsertLines LineNum, ThisWkb.Sheets("CO-Code").Cells(SourceLineNum, 1)
                LineNum = LineNum + 1
            Next SourceLineNum
        End With

    Next xx

End Sub
```

BEISPIELE 3 (erzeugt neuen Formular-Button - läuft auch engl. 2003 Excel)

VERSION 1 gekürzt Version von Seibersdorf

```
Sub test()

    ' setting variables
    Dim ThisWkb As Workbook ' the active workbook with this code
    Dim NewWkb As Workbook ' the new workbook in which VBA-code will be generated
    Dim VBComp As VBComponent ' the VB-Component (in our case: a new modul) that will be generated
    Dim VBCodeMod As CodeModule ' the newly added module
    Dim LineNum As Long ' the line-number for the newly added code-lines

    Dim NewProj As VBIDE.VBProject ' this is for generating code in worksheets of the new workbook
    Dim NewComp As VBIDE.VBComponent ' this is for generating code in worksheets of the new workbook
```

```
    Dim ObjSh As Object ' this is for changing Worksheet_Codename into Worksheet_Name
    Dim WkShName As String ' name of the worksheet that gets VBA-Code

    Dim NewButton As Object ' for creation of a new formular-button
    Dim ActXbutton As OLEObject ' for creation of a new active-x-command-button

    ' saving the name of the active workbook
    Set ThisWkb = ActiveWorkbook

    ' creating a new workbook
    Workbooks.Add

    Set NewWkb = ActiveWorkbook

' ------------------------------------------------------------------------------------------------------------------------
' example for generating code for a formular-button created in a worksheet and the new vba-code-lines is located in the worksheet "Sheet1"
' ------------------------------------------------------------------------------------------------------------------------

    ' --- creating a new module with a procedure for the new button ---

    ' adding a new module in that workbook
    Set VBComp = NewWkb.VBProject.VBComponents.Add(vbext_ct_StdModule)

    ' naming the new module
    VBComp.Name = "Modul4Button"

    ' referenzing this new module
    Set VBCodeMod = NewWkb.VBProject.VBComponents("Modul4Button").CodeModule

    ' inserting new code lines
    With VBCodeMod
        LineNum = .CountOfLines + 1
        .InsertLines LineNum, "Sub MyButtonProc()"
        .InsertLines LineNum + 1, " Msgbox ""You have clicked on the formular-button"" "
        .InsertLines LineNum + 2, "End Sub"
    End With

    ' --- creating the new button ---

    Set NewButton = NewWkb.ActiveSheet.Buttons.Add(255, 40, 180, 40)
    NewButton.Caption = "Click on me (formular)"
    NewButton.Font.Bold = True
    NewButton.OnAction = NewWkb.Name & "!Modul4Button.MyButtonProc"
```

```
End Sub
```

VERSION 2 mit original Seibersdorfcode

```
Sub Create_VBA_Button_2()

' This routine was programmed in 2012-11 by Stefan Wenninger

    ' --- Declaring Variables ---

    Dim NewWkb As Workbook ' the new workbook in which VBA-code will be generated
    Dim VBComp As VBComponent ' the VB-Component (in our case: a new modul) that will be generated
    Dim VBCodeMod As CodeModule ' the newly added module
    Dim LineNum As Long ' the line-number for the newly added code-lines
    Dim SourceLineNum As Long ' the line-number from the VBA-Code-Source-Page CO-Code

    ' --- Setting Countrates_Overview-Workbook ---

    Set NewWkb = ActiveWorkbook ' this is the now already opened Countrates_Overview.xls-file

    ' --- Activating empty sheet 2 for pasting new button ---

    NewWkb.Activate
    NewWkb.Sheets("Sheet2").Activate

    ' --- creating the new button ---

    Set NewButton = NewWkb.ActiveSheet.Buttons.Add(255, 40, 180, 40)
    NewButton.Caption = "Click on me"
    NewButton.Font.Bold = True
    NewButton.OnAction = NewWkb.Name & "!Modul4Button.CO_Button"

    ' --- creating a new module with a procedure for the new button ---

    ' adding a new module in that workbook
    Set VBComp = NewWkb.VBProject.VBComponents.Add(vbext_ct_StdModule)

    ' naming the new module
    VBComp.Name = "Modul4Button"

    ' referenzing this new module
    Set VBCodeMod = NewWkb.VBProject.VBComponents("Modul4Button").CodeModule
```

```
    ' --- inserting new code lines ---
    With VBCodeMod

        LineNum = .CountOfLines + 1
        For SourceLineNum = 11 To LastLineNum
            .InsertLines LineNum, ThisWkb.Sheets("CO-Code").Cells(SourceLineNum, 1)
            LineNum = LineNum + 1
        Next SourceLineNum

    End With

    ' as the VBA-Code in Countrates-Overview needs the Name of the opened PsWin-File
    NewWkb.ActiveSheet.Range("A1") = ThisWkb.Name

End Sub
```

--- GRUNDLAGEN ---

Programmierungen am Visual Basic Editor

Der folgende Beitrag stammt von einem meiner US-MVP-Kollegen: **Chip Pearson**
Er hat mir erlaubt, einen seiner veröffentlichten Beiträge zu übersetzen und hier zu publizieren.
Ein herzliches Dankeschön!

Den englische Originaltext findest Du hier:
http://www.cpearson.com/excel/vbe.htm

http://www.cpearson.com/excel/vbe.aspx

Übersetzt am: 5. September 2004 - Monika Weber - http://www.jumper.ch

---

Inhaltsverzeichnis

1.) Einführung

Der Visual Basic Editor (VBE) wird verwendet, um Visual Basic for Applications (VBA) Prozeduren und Module zu erstellen, verändern und verwalten. VBA gibt Dir die Möglichkeit, Mappen und Tabellenblätter zu verändern, indem Excel als Schnittstelle dient. VBA erlaubt zudem, VBA Komponenten und Code Module zu bearbeiten, indem VBE das Interface darstellt. Diese Seite bezieht sich lediglich auf Excel 97 und höhere Versionen. Ältere Versionen werden nicht unterstützt.

Diese Seite beschreibt einige Objekt, Methoden und Eigenschaften von VBE, die mittels VBA manipuliert werden können. In Excel 97 sind diese Objekte, Methoden und Eigenschaften nicht in der VBA-Hilfe-Datei beschrieben. Es muss die Datei VEENOB3.hlp verwendet werden. Diese Datei wurde möglicherweise nicht auf Deinem System installiert, als Du Office 97 installiert hast. Ab Office 2000 sind diese Themen in der Standard VBA Hilfe enthalten.

Bevor Du mit dem Programmieren durch die VBE beginnen kannst, musst Du den Verweis auf die Bibliothek "Extensibility" setzen. Du findest Sie im VBA Editor unter dem Menü "Extras/Verweise". Aktiviere das Kontrollkästchen "Microsoft Visual Basic for Applications Extensibility". Dies ermöglicht VBA die Definitionen der Objekte zu finden. Wenn Excel 97 verwendet wird, erscheint die Bibliothek in der Referenzliste ohne Versionsnummer. Wenn Du mit einer neueren Version arbeitest, ist es die Version 5.3 ("Microsoft Visual Basic for Applications Extensibility 5.3"). Es ist sehr wichtig, dass die richtige Bibliothek referenziert wird. Wenn die Falsche oder gar keine verwendet wird, wird ein Fehler beim Ausführen der nachfolgenden Codes erzeugt.

Informationen zur Programmierung von VBE Menüs findest Du unter Adding Menus To The VBA Editor (englisch).

Hinweis: In der Version Excel 2002 wurde ein zusätzlicher Sicherheitslevel eingeführt. Um VBE Projekt Objekte zu manipulieren, so wie es hier beschrieben ist, müsst Du Deine Sicherheitseinstellungen ändern. Verwende in Excel das Menü "Extras/Makro/Sicherheit". Aktiviere das Register "Vertrauenswürdige Quellen". Aktiviere darin das Kontrollkästchen "Zugriff auf Visual Basic-Projekt vertrauen".

Hinweis: Für alle Excel-Versionen gilt, dass das Projekt nicht geschützt werden darf. Wenn dies der Fall ist, schlägt die Prozedur fehl.

Zusätzliche Fehler können entstehen, wenn versucht wird aus einem Modul her Veränderungen im selben Modul vorzunehmen. Damit ist der Zugriff von z.B. Modul1 auf Modul1 gemeint. Es ist zu empfehlen, dies nicht zu tun.


Zurück zum Anfang

2.) VBE-Objekte

Wir werden drei Objekte in unseren Codes verwenden:

VBProject
Das ist der gesamte Satz von VBA Modulen und Referenzen, die mit einer Mappe verbunden sind.

VBComponent
Das sind die individuellen Komponenten innerhalb von VBProject. Zum Beispiel ein UserForm und ein Standard-Modul sind VBComponenten. Die VBComponents Kollektion enthält alle existierenden VBComponent Objekte.

CodeModule
Dieses Objekt repräsentiert den aktuellen Code innerhalb von VBComponent. Wenn Du beispielsweise einen Code in Modul1 eintippst, gibst Du den Code in das CodeModule Objekt ein in der VBComponenten mit dem Namen "Modul1".

Wir werden programmiertechnisch durch das Workbook Objekt zu diesen Komponenten "navigieren". Du kannst diese Komponenten auch über den `Application.VBE` Objekt-Pfad ansteuern, aber das werden wir hier nicht tun.

Es gibt verschiedene Typen an VBComponents, identifiziert durch die Typen-Eigenschaft des Objekt VBComponent.

Typen Konstanten	Beschreibung
`vbext_ct_ClassModule`	Dies ist ein Klassenmodul, das verwendet werden kann, um eigene Objekte zu erzeugen. Wir werden das hier nicht benutzen.
`vbext_ct_Document`	Dies ist die Komponente für ein Tabellenblatt, Diagrammblatt oder eine Mappe (ThisWorkbook).
`vbext_ct_MSForm`	Dies ist die Komponente für ein UserForm.
`vbext_ct_StdModule`	Dies ist die Komponente für ein Standard Code Modul. Die meisten unserer

	Prozeduren werden diesen Typ verwenden.

Zurück zum Anfang

3.) Ein Objekt referenzieren

Der erste Schritt in der Programmierung mit VBE ist, das Objekt, das für die Arbeit benötigt wird, zu referenzieren.

VBProject

```
Dim VBProj As VBProject
Set VBProj = ThisWorkbook.VBProject
```

VBComponent

```
Dim VBComp As VBComponent
Set VBComp = ThisWorkbook.VBProject.VBComponents("Modul1")
```

CodeModule

```
Dim VBCodeMod As CodeModule
Set VBCodeMod = ThisWorkbook.VBProject.VBComponents("Modul1").CodeModule
```

In allen Beispielen auf dieser Seite werden wir mit dem `ThisWorkbook` Objekt arbeiten -- das bedeutet, wir werden die VBA Komponenten über die Mappe ansprechen. Natürlich können jegliche Arten an offenen Mappen verwendet werden `AktiveWorkbook` oder `Workbooks(IrgendeineMappe.xls)`.

Zurück zum Anfang

4.) Ein Modul in eine Mappe einfügen

Die untenstehende Prozedur fügt ein neues Modul mit dem Namen "NeuesModul" in ThisWorkbook ein.

```
Sub AddModule()
   Dim VBComp As VBComponent
   Set VBComp = ThisWorkbook.VBProject.VBComponents. _
                Add(vbext_ct_StdModule)

   VBComp.Name = "NeuesModul"
   Application.Visible = True
End Sub
```

Wenn dieser Code von Excel her ausgeführt wird, während der VBE offen ist, wirst Du direkt zum neuen Modul geführt und die Prozedur wird beendet. Wenn der Code ausgeführt wird, wenn der VBE geschlossen ist, wird der Fokus an die Excel Applikation zurückgegeben. Der VBE wird nicht geöffnet.

Zurück zum Anfang

5.) Ein Modul aus einer Mappe entfernen

Die folgende Prozedur löscht das Modul "NeuesModul" aus ThisWorkbook.

```
Sub DeleteModule()
   Dim VBComp As VBComponent
   Set VBComp = ThisWorkbook.VBProject. _
                VBComponents("NeuesModul")

   ThisWorkbook.VBProject.VBComponents.Remove VBComp
End Sub
```

Ein ThisWorkbook Code Modul oder ein Sheet Code Modul oder ein Chart Code Modul kann nicht gelöscht werden.

Zurück zum Anfang

6.) Eine Prozedur in ein Modul einfügen

Die nächste Prozedur fügt eine neue Prozedur mit dem Namen "MeineNeueProzedur" in das Modul mit dem Namen "NeuesModul" in ThisWorkbook ein. Das Modul "NeuesModul" muss vorhanden sein, ansonsten entsteht eine Fehlermeldung.

```
Sub AddProcedure()
   Dim VBCodeMod As CodeModule
   Dim LineNum As Long
   Set VBCodeMod = ThisWorkbook.VBProject. _
                   VBComponents("NeuesModul").CodeModule

   With VBCodeMod
       LineNum = .CountOfLines + 1
       .InsertLines LineNum, _
   "Sub MeineNeueProzedur()" & Chr(13) & _
   " Msgbox ""Hier ist die neue Prozedur"" " & Chr(13) & _
   "End Sub"
   End With
   Application.Run "MeineNeueProzedur"
End Sub
```

Beachte den Weg, wie die `.InsertLines` Methode verwendet wird. Die gesamte Prozedur wird als ein Argument aufbereitet. Mit dem `Chr(13)` werden die Zeilenumbrüche erzeugt. Die Anweisung

```
Application.Run "MeineNeueProzedur"
```

bewirkt, dass die Prozedur gleich gestartet wird. Anstatt die Prozedur direkt aufzurufen (`Call`), musst Du die Anweisung `Application.Run` verwenden. Damit kann ein eventueller Compile-Time Error vermieden werden. Die Methode `Call` wird nur funktionieren, wenn Du Code in ein anderes Code Modul einfügst. Wenn Code im selben Modul eingefügt wird, musst Du ein `Application.OnTime` einsetzen. Auf diese Weise wird die Kontrolle an Excel zurückgegeben und das Modul kann kompiliert und geladen werden. Mit der Benutzung von `Application.OnTime` können unter Umständen Synchronisierungsprobleme entstehen. Du solltest vermeiden, eine Prozedur aufzurufen, die gerade erst ins selbe Modul eingefügt wurde ohne, dass zuvor alle VBA Prozeduren die Möglichkeit hatten beendet zu werden.

```
Application.OnTime Now, "NeueProzedurName"
```


Zurück zum Anfang

7.) Eine Ereignis-Prozedur erzeugen

Das CodeModul Objekt verfügt über eine Methode mit dem Namen `CreateEventProc`, die benutzt werden kann, um eine Prozedur in einem Document-Modul zu erstellen. Z.B. Ein Tabellenmodul, oder das DieseArbeitsmappe-Modul. Der Vorteil der Benutzung von `CreateEventProc` über `InsertLines` besteht darin, dass `CreateEventProc` automatisch die gesamte Prozedur Deklaration, inklusive aller korrekten Parameter einfügt.

`CreateEventProc` gibt die Zeilennummer zurück, in welcher die Prozedur beginnt. Wenn Du einem `CreateEventProc` aufrufst, addierte eins zum Ergebnis und verwende dies mit `InsertLines` um den Kern der Ereignisprozedur einzufügen. Zum Beispiel: der untenstehende Code erzeugt eine `Workbook_Open`-Prozedur mit einer `MsgBox`. Der Code wird in `DieseArbeitsmappe` eingefügt.

```
Sub CreateEvent()
   Dim StartLine As Long

   With ActiveWorkbook.VBProject. _
        VBComponents("DieseArbeitsmappe").CodeModule

       StartLine = .CreateEventProc("Open", "Workbook") + 1
       .InsertLines StartLine, _
       "Msgbox ""Hallo Welt"",vbOkOnly"
   End With
End Sub
```

Zurück zum Anfang

8.) Eine Prozedur aus einem Modul entfernen

Die nachfolgende Prozedur löscht die Prozedur mit dem Namen "MeineNeueProzedur" aus dem Modul "NeuesModul" in ThisWorkbook.

```
Sub DeleteProcedure()
   Dim VBCodeMod As CodeModule
   Dim StartLine As Long
   Dim HowManyLines As Long
   Set VBCodeMod = ThisWorkbook.VBProject. _
                   VBComponents("NeuesModul").CodeModule

   With VBCodeMod
       StartLine = .ProcStartLine("MeineNeueProzedur", vbext_pk_Proc)
       HowManyLines = .ProcCountLines("MeineNeueProzedur", vbext_pk_Proc)
       .DeleteLines StartLine, HowManyLines
   End With
End Sub
```

Zurück zum Anfang

9.) Alle Prozeduren aus einem Modul entfernen

Die untenstehende Prozedur löscht sämtlichen Code aus dem Modul mit dem Namen "NeuesModul".

```
Sub DeleteAllCodeInModule()
   Dim VBCodeMod As CodeModule
   Dim StartLine As Long
   Dim HowManyLines As Long
   Set VBCodeMod = ThisWorkbook.VBProject. _
                   VBComponents("NeuesModul"). _
                   CodeModule
   With VBCodeMod
       StartLine = 1
       HowManyLines = .CountOfLines
      .DeleteLines StartLine, HowManyLines
   End With
End Sub
```

Zurück zum Anfang

10.) Alle Module einer Mappe auflisten

Die nächste Prozedur listet, in einer MsgBox, alle Module in ThisWorkbook auf. Es wird eine Funktion mit dem Namen `CompTypeToName` verwendet, um eine Textbeschreibung vom Typ des Modul zu erhalten. Die Funktion ist ebenfalls nachfolgend zu finden.

```
Sub ListModules()
   Dim VBComp As VBComponent
   Dim Msg As String
   For Each VBComp In ThisWorkbook.VBProject.VBComponents
       Msg = Msg & VBComp.Name & " Type: " & _
             CompTypeToName(VBComp) & Chr(13)
   Next VBComp
   MsgBox Msg
End Sub
Function CompTypeToName(VBComp As VBComponent) As String
   Select Case VBComp.Type
       Case vbext_ct_ActiveXDesigner
           CompTypeToName = "ActiveX Designer"
       Case vbext_ct_ClassModule
           CompTypeToName = "Class Module"
       Case vbext_ct_Document
           CompTypeToName = "Document"
       Case vbext_ct_MSForm
           CompTypeToName = "MS Form"
       Case vbext_ct_StdModule
           CompTypeToName = "Standard Module"
       Case Else
   End Select
End Function
```

Zurück zum Anfang

11.) Alle Prozeduren eines Moduls auflisten

Die nächste Prozedur listet, in einer MsgBox, alle Prozeduren eines Standard Moduls auf. Das Standard Modul hat den Namen "NeuesModul" in ThisWorkbook. Die Prozeduren werden in der Reihenfolge aufgelistet, in der sie im Modul enthalten sind.

```
Sub ListProcedures()
   Dim VBCodeMod As CodeModule
   Dim StartLine As Long
   Dim Msg As String
   Dim ProcName As String
   Set VBCodeMod = ThisWorkbook.VBProject. _
                   VBComponents("NeuesModul").CodeModule
   With VBCodeMod
       StartLine = .CountOfDeclarationLines + 1
       Do Until StartLine >= .CountOfLines
           Msg = Msg & .ProcOfLine(StartLine, vbext_pk_Proc) & _
                 Chr(13)
                 StartLine = StartLine + _
```

```
                .ProcCountLines(.ProcOfLine _
                (StartLine, vbext_pk_Proc), _
                vbext_pk_Proc)
        Loop
    End With
    MsgBox Msg
End Sub
```

Beachte den folgenden Hyperlink für mehr Informationen zu `CodeName` Eigenschaften von `VBComponents`: Code Modules And Code Names (englisch).

Zurück zum Anfang

12.) Alle Module in ein Projekt exportieren

Die nachfolgende Prozedur exportiert alle Module in den Ordner, in dem die Mappe abgespeichert ist. Damit kann eine schnelle Sicherung der VBA Prozeduren erfolgen, ohne dass jedes einzelne Modul exportiert werden muss.

```
Sub ExportAllVBA()
    Dim VBComp As VBIDE.VBComponent
    Dim Sfx As String
    For Each VBComp In ActiveWorkbook.VBProject.VBComponents
        Select Case VBComp.Type
            Case vbext_ct_ClassModule, vbext_ct_Document
                Sfx = ".cls"
            Case vbext_ct_MSForm
                Sfx = ".frm"
            Case vbext_ct_StdModule
                Sfx = ".bas"
            Case Else
                Sfx = ""
        End Select

        If Sfx <> "" Then
           VBComp.Export _
              Filename:=ActiveWorkbook.Path & "\" & _
              VBComp.Name & Sfx
        End If
    Next VBComp
End Sub
```

Zurück zum Anfang

13.) Sämtlichen VBA-Code eines Projekts löschen

Die folgende Prozedur löscht sämtliche Codezeilen aus dem gesamten Projekt. Du solltest diese Prozedur mit Vorsicht genießen, da der Code nicht wieder hergestellt werden kann. Es werden Standardmodule, UserForms und Klassenmodule entfernt. Ebenso wird der Code in ThisWorkbook-Modulen gelöscht. Zur Sicherheit könntest Du den vorangegangenen Code verwenden, um vor dem Löschen eine Sicherungskopie der Codes zu erstellen.

```
Sub DeleteAllVBA()
   Dim VBComp As VBIDE.VBComponent
   Dim VBComps As VBIDE.VBComponents
   Set VBComps = ActiveWorkbook.VBProject.VBComponents
   For Each VBComp In VBComps
      Select Case VBComp.Type
         Case vbext_ct_StdModule, vbext_ct_MSForm, _
              vbext_ct_ClassModule
              VBComps.Remove VBComp
         Case Else
            With VBComp.CodeModule
               .DeleteLines 1, .CountOfLines
            End With
      End Select
   Next VBComp
End Sub
```

 Zurück zum Anfang

14.) Module zwischen Projekten kopieren

Es gibt nicht einfach eine Methode, um Module zwischen Projekten zu kopieren. Der Code eines Projektes muss zuerst exportiert werden. Danach kann dieser in ein anderes Projekt importiert werden. Die folgende Prozedur exportiert das "Modul1" von der "Mappe2" in die "Mappe1".

```
Sub CopyOneModule()
   Dim FName As String
   With Workbooks("Mappe2")
       FName = .Path & "\code.txt"
       .VBProject.VBComponents("Modul1").Export FName
   End With
   Workbooks("Mappe1").VBProject.VBComponents.Import FName
End Sub
```

Ändere lediglich "Modul1" in den Namen des Moduls, das Du tatsächlich kopieren möchtest. Wenn Du alle Module kopieren möchtest (mit Ausnahme von DieseArbeitsmappe und den Tabellenmodulen), kannst Du den folgenden Code benutzen.

```
Sub CopyAllModules()
   Dim FName As String
   Dim VBComp As VBIDE.VBComponent
   With Workbooks("Mappe2")
       FName = .Path & "\code.txt"
       If Dir(FName) <> "" Then
```

```
            Kill FName
        End If
        For Each VBComp In .VBProject.VBComponents
            If VBComp.Type <> vbext_ct_Document Then
               VBComp.Export FName
               Workbooks("Mappe1").VBProject.VBComponents. _
                  Import FName
               Kill FName
            End If
        Next VBComp
    End With
End Sub
```

Zurück zum Anfang

15.) Prüfen, ob ein Modul oder eine Prozedur existiert

Du kannst das VBA Extensibility Tool nutzen, um zu prüfen, ob ein Modul existiert, oder ob eine bestimmte Prozedur in einem Modul.

```
Function ProcedureExists(ProcedureName As String, _
        ModuleName As String) As Boolean
    On Error Resume Next
    If ModuleExists(ModuleName) = True Then
        ProcedureExists = ThisWorkbook.VBProject. _
                          VBComponents(ModuleName) _
           .CodeModule.ProcStartLine _
            (ProcedureName, vbext_pk_Proc) <> 0
    End If
End Function
```

Type Property (VBA Add-In Object Model)

See Also Example Applies To Specifics

Returns a numeric or string value containing the type of object. Read-only.

Return Values

The **Type** property settings for the **Window** object are described in the following table:

Constant	Value	Description
vbext_wt_CodeWindow	0	**Code** window

vbext_wt_Designer	1	Designer
vbext_wt_Browser	2	**Object Browser**
vbext_wt_Immediate	5	**Immediate** window
vbext_wt_ProjectWindow	6	Project window
vbext_wt_PropertyWindow	7	Properties window
vbext_wt_Find	8	**Find** dialog box
vbext_wt_FindReplace	9	**Search and Replace** dialog box
vbext_wt_LinkedWindowFrame	11	Linked window frame
vbext_wt_MainWindow	12	Main window
vbext_wt_Watch	3	**Watch** window
vbext_wt_Locals	4	**Locals** window
vbext_wt_Toolbox	10	**Toolbox**
vbext_wt_ToolWindow	15	**Tool** window

The **Type** property settings for the **VBComponent** object are described in the following table:

Constant	Value	Description

vbext_ct_StdModule	1	Standard module
vbext_ct_ClassModule	2	Class module
vbext_ct_MSForm	3	Microsoft Form
vbext_ct_ActiveXDesigner	11	ActiveX Designer
vbext_ct_Document	100	Document Module

The **Type** property settings for the **Reference** object are described in the following table:

Constant	Value	Description
vbext_rk_TypeLib	0	Type library
vbext_rk_Project	1	Project

--- The original workshop in English

Programming The VBA Editor

This page describes how to write code that modifies or reads other VBA code.

Introduction

You can write code in VBA that reads or modifies other VBA projects, modules, or procedures. This is called *extensibility* because *extends* the editor -- you can use VBA code to create new VBA code. You can use these features to write custom procedures that create, change, or delete VBA modules and code procedures.

In order to use the code on this page in your projects, you must change two settings.

- First, you need to set an reference to the VBA Extensibility library. The library contains the definitions of the objects that make up the VBProject. In the VBA editor, go the the *Tools* menu and choose *References*. In that dialog, scroll down to and check the entry for *Microsoft Visual Basic For Applications Extensibility 5.3*. If you do not set this reference, you will receive a *User-defined type not defined* compiler error.

- Next, you need to enable programmatic access to the VBA Project. In Excel 2003 and earlier, go the *Tools* menu (in Excel, not in the VBA editor), choose *Macros* and then the *Security* item. In that dialog, click on the *Trusted Publishers* tab and check the *Trust access to the Visual Basic Project* setting.

  In Excel 2007, click the *Developer* item on the main Ribbon and then click the *Macro Security* item in the *Code* panel. In that dialog, choose *Macro Settings* and check the *Trust access to the VBA project object model*.

The VBA Project that you are going to change with these procedures must be unlocked. There is no programmatic way to unlock a VBA project (other than using `SendKeys`). If the project is locked, you must manually unlock. Otherwise, the procedures will not work.

CAUTION: Many VBA-based computer viruses propagate themselves by creating and/or modifying VBA code. Therefore, many virus scanners may automatically and without warning or confirmation delete modules that reference the VBProject object, causing a permanent and irretrievable loss of code. Consult the documentation for your anti-virus software for details.

For information about using creating custom menu items in the Visual Basic Editor, see Menus In The VBA Editor.

Operations Described On This Page

Objects In The VBA Extensibility Model

The following is a list of the more common objects that are used in the VBA Extensibilty object model. This is not a comprehensive list, but will be sufficient for the tasks at hand.

`VBIDE`
The `VBIDE` is the object library that defines all the objects and values that make up VBProject and the Visual Basic Editor. You must reference this library to use the VBA Extensibility objects. To add this reference, open the VBA editor, open your VBProject in the editor, and go to the *Tools* menu. There, choose *References* . In the *References* dialog, scroll down to *Microsoft Visual Basic for Applications Extensibility 5.3* and check that item in the list. You can add the reference programmatically with code like:

```
ThisWorkbook.VBProject.References.AddFromGuid _
    GUID:="{0002E157-0000-0000-C000-000000000046}", _
    Major:=5, Minor:=3
```

VBE
The VBE refers to the Visual Basic Editor, which includes all the windows and projects that make up the editor.

VBProject
A VBProject contains all the code modules and components of a single workbook. One workbook has exactly one VBProject. The VBProject is made up of 1 or more VBComponent objects.

VBComponent
A VBComponent is one object within the VBProject. A VBComponent is a regular code module, a UserForm, a class module, any one of the Sheet modules, or the ThisWorkbook module (together, the Sheet modules and the ThisWorkbook module are called *Document Type* modules). A VBComponent is of one of the following types, identified by the Type property. The following constants are used to identify the Type. The numeric value of each constant is shown in parentheses.

- vbext_ct_ClassModule (2): A class module to create your own objects. See Class Modules for details about classes and objects.
- vbext_ct_Document (100): One of the Sheet modules or the ThisWorkbook module.
- vbext_ct_MSForm (3): A UserForm. The visual component of a UserForm in the VBA Editor is called a *Designer*.
- vbext_ct_StdModule (1): A regular code module. Most of the procedures on this page will work with these types of components.

CodeModule
A CodeModule is the VBA source code of a VBComponent. You use the CodeModule object to access the code associated with a VBComponent. A VBComponent has exactly one CodeModule which contains all the code for that component.

CodePane
A CodePane is an open editing window of a CodeModule. When you are typing code, you are entering code into the CodePane.

Referencing VBIDE Objects

The code below illustrate various ways to reference Extensibility objects.

```
Dim VBAEditor As VBIDE.VBE
Dim VBProj As VBIDE.VBProject
Dim VBComp As VBIDE.VBComponent
Dim CodeMod As VBIDE.CodeModule
```

```
Set VBAEditor = Application.VBE
''''''''''''''''''''''''''''''''''''''''''''
Set VBProj = VBAEditor.ActiveVBProject
' or
Set VBProj = Application.Workbooks("Book1.xls").VBProject
''''''''''''''''''''''''''''''''''''''''''''
Set VBComp = ActiveWorkbook.VBProject.VBComponents("Module1")
' or
Set VBComp = VBProj.VBComponents("Module1")
''''''''''''''''''''''''''''''''''''''''''''
Set CodeMod = ActiveWorkbook.VBProject.VBComponents("Module1").CodeModule
' or
Set CodeMod = VBComp.CodeModule
```

In the code and descriptions on this page, the term *Procedure* means a `Sub`, `Function`, `Property Get`, `Property Let`, or `Property Set` procedure. The Extensibility library defines four procedures types, identified by the following constants. The numeric value of each constant is shown within parentheses.

- `vbext_pk_Get` (3). A `Property Get` procedure.
- `vbext_pk_Let` (1). A `Property Let` procedure.
- `vbext_pk_Set` (2). A `Property Set` procedure.
- `vbext_pk_Proc` (0). A `Sub` or `Function` procedure.

The rest of this page describes various procedures that modify the various objects of a `VBProject`.

Ensuring The Editor In Synchronized

The VBA editor is said to be "in sync" if the `ActiveVBProject` is the same as the VBProject that contains the `ActiveCodePane`. If you have two or more projects open within the VBA editor, it is possible to have an active code pane open from Project1 and have a component of Project2 selected in the Project Explorer window. In this case, the `Application.VBE.ActiveVBProject` is the project

that is selected in the Project window, while `Application.VBE.ActiveCodePane` is a different project, specifically the project referenced by `Application.VBE.ActiveCodePane.CodeModule.Parent.Collection.Parent`.

You can test whether the editor in in sync with code like the following.

```
Function IsEditorInSync() As Boolean
'=====================================================================
' IsEditorInSync
' This tests if the VBProject selected in the Project window, and
' therefore the ActiveVBProject is the same as the VBProject associated
' with the ActiveCodePane. If these two VBProjects are the same,
' the editor is in sync and the result is True. If these are not the
' same project, the editor is out of sync and the result is True.
'=====================================================================
    With Application.VBE
    IsEditorInSync = .ActiveVBProject Is _
        .ActiveCodePane.CodeModule.Parent.Collection.Parent
    End With
End Function
```

You can force synchronization with code like the following. This will set the `ActiveVBProject` to the project associated with the `ActiveCodePane`.

```
Sub SyncVBAEditor()
'=====================================================================
' SyncVBAEditor
' This syncs the editor with respect to the ActiveVBProject and the
' VBProject containing the ActiveCodePane. This makes the project
' that conrains the ActiveCodePane the ActiveVBProject.
'=====================================================================
With Application.VBE
If Not .ActiveCodePane Is Nothing Then
    Set .ActiveVBProject = .ActiveCodePane.CodeModule.Parent.Collection.Parent
End If
End With
End Sub
```

Adding A Module To A Project

This code will add new code module named `NewModule` to the VBProject of the active workbook. The type of `VBComponent` is specified by the value of the parameter passed to the `Add` method.

```
Sub AddModuleToProject()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents.Add(vbext_ct_StdModule)
    VBComp.Name = "NewModule"
End Sub
```

Adding A Procedure To A Module

Creating a procedure via VBA code is really quite simple. Build up a text string of the code, using `vbCrLf` to create new lines, and then insert that text with the `InsertLines` method, passing to it the line number and the text string. The following code will add a simple "Hello World" procedure named `SayHello` to the end of the module named `Module1`.

```
Sub AddProcedureToModule()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim LineNum As Long
    Const DQUOTE = """" ' one " character

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("Module1")
    Set CodeMod = VBComp.CodeModule
```

```
        With CodeMod
            LineNum = .CountOfLines + 1
            .InsertLines LineNum, "Public Sub SayHello()"
            LineNum = LineNum + 1
            .InsertLines LineNum, "    MsgBox " & DQUOTE & "Hello World" & DQUOTE
            LineNum = LineNum + 1
            .InsertLines LineNum, "End Sub"
        End With

    End Sub
```

Copy A Module From One Project To Another

There is no direct way to copy a module from one project to another. To accomplish this task, you must export the module from the Source VBProject and then import that file into the Destination VBProject. The code below will do this. The function declaration is:

```
Function CopyModule(ModuleName As String, _
    FromVBProject As VBIDE.VBProject, _
    ToVBProject As VBIDE.VBProject, _
    OverwriteExisting As Boolean) As Boolean
```

`ModuleName` is the name of the module you want to copy from one project to another.

`FromVBProject` is the VBProject that contains the module to be copied. This is the source VBProject.

`ToVBProject` is the VBProject in to which the module is to be copied. This is the destination VBProject.

`OverwriteExisting` indicates what to do if `ModuleName` already exists in the `ToVBProject`. If this is `True` the existing VBComponent will be removed from the `ToVBProject`. If this is `False` and the VBComponent already exists, the function does nothing and returns `False`.

The function returns `True` if successful or `False` is an error occurs. The function will return `False` if any of the following are true:

- FromVBProject is nothing.
- ToVBProject is nothing.
- ModuleName is blank.
- FromVBProject is locked.
- ToVBProject is locked.
- ModuleName does not exist in FromVBProject.
- ModuleName exists in ToVBProject and OverwriteExisting is False.

The complete code is shown below:

```
Function CopyModule(ModuleName As String, _
    FromVBProject As VBIDE.VBProject, _
    ToVBProject As VBIDE.VBProject, _
    OverwriteExisting As Boolean) As Boolean
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' CopyModule
    ' This function copies a module from one VBProject to
    ' another. It returns True if successful or  False
    ' if an error occurs.
    '
    ' Parameters:
    ' -------------------------------
    ' FromVBProject         The VBProject that contains the module
    '                       to be copied.
    '
    ' ToVBProject           The VBProject into which the module is
    '                       to be copied.
    '
    ' ModuleName            The name of the module to copy.
    '
```

```
' OverwriteExisting     If True, the VBComponent named ModuleName
'                       in ToVBProject will be removed before
'                       importing the module. If False and
'                       a VBComponent named ModuleName exists
'                       in ToVBProject, the code will return
'                       False.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim VBComp As VBIDE.VBComponent
Dim FName As String
Dim CompName As String
Dim S As String
Dim SlashPos As Long
Dim ExtPos As Long
Dim TempVBComp As VBIDE.VBComponent

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Do some housekeeping validation.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
If FromVBProject Is Nothing Then
    CopyModule = False
    Exit Function
End If

If Trim(ModuleName) = vbNullString Then
    CopyModule = False
    Exit Function
End If

If ToVBProject Is Nothing Then
    CopyModule = False
    Exit Function
End If
```

```
If FromVBProject.Protection = vbext_pp_locked Then
    CopyModule = False
    Exit Function
End If

If ToVBProject.Protection = vbext_pp_locked Then
    CopyModule = False
    Exit Function
End If

On Error Resume Next
Set VBComp = FromVBProject.VBComponents(ModuleName)
If Err.Number <> 0 Then
    CopyModule = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' FName is the name of the temporary file to be
' used in the Export/Import code.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
FName = Environ("Temp") & "\" & ModuleName & ".bas"
If OverwriteExisting = True Then
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' If OverwriteExisting is True, Kill
    ' the existing temp file and remove
    ' the existing VBComponent from the
    ' ToVBProject.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    If Dir(FName, vbNormal + vbHidden + vbSystem) <> vbNullString Then
        Err.Clear
        Kill FName
        If Err.Number <> 0 Then
            CopyModule = False
            Exit Function
```

```
        End If
    End If
    With ToVBProject.VBComponents
        .Remove .Item(ModuleName)
    End With
Else
    '''''''''''''''''''''''''''''''''''''''''
    ' OverwriteExisting is False. If there is
    ' already a VBComponent named ModuleName,
    ' exit with a return code of False.
    '''''''''''''''''''''''''''''''''''''''''
    Err.Clear
    Set VBComp = ToVBProject.VBComponents(ModuleName)
    If Err.Number <> 0 Then
        If Err.Number = 9 Then
            ' module doesn't exist. ignore error.
        Else
            ' other error. get out with return value of False
            CopyModule = False
            Exit Function
        End If
    End If
End If

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Do the Export and Import operation using FName
' and then Kill FName.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
FromVBProject.VBComponents(ModuleName).Export Filename:=FName

''''''''''''''''''''''''''''''''''''''''
' Extract the module name from the
' export file name.
''''''''''''''''''''''''''''''''''''''''
SlashPos = InStrRev(FName, "\")
```

```
    ExtPos = InStrRev(FName, ".")
    CompName = Mid(FName, SlashPos + 1, ExtPos - SlashPos - 1)

    '''''''''''''''''''''''''''''''''''''''''''''''
    ' Document modules (SheetX and ThisWorkbook)
    ' cannot be removed. So, if we are working with
    ' a document object, delete all code in that
    ' component and add the lines of FName
    ' back in to the module.
    '''''''''''''''''''''''''''''''''''''''''''''''
    Set VBComp = Nothing
    Set VBComp = ToVBProject.VBComponents(CompName)

    If VBComp Is Nothing Then
        ToVBProject.VBComponents.Import Filename:=FName
    Else
        If VBComp.Type = vbext_ct_Document Then
            ' VBComp is destination module
            Set TempVBComp = ToVBProject.VBComponents.Import(FName)
            ' TempVBComp is source module
            With VBComp.CodeModule
                .DeleteLines 1, .CountOfLines
                S = TempVBComp.CodeModule.Lines(1, TempVBComp.CodeModule.CountOfLines)
                .InsertLines 1, S
            End With
            On Error GoTo 0
            ToVBProject.VBComponents.Remove TempVBComp
        End If
    End If
    Kill FName
    CopyModule = True
End Function
```

Creating An Event Procedure

This code will create a `Workbook_Open` event procedure. When creating an event procedure, you should use the `CreateEventProc` method so that the correct procedure declaration and parameter list is used. `CreateEventProc` will create the declaration line and the end of procedure line. It returns the line number on which the event procedure begins.

```
Sub CreateEventProcedure()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim LineNum As Long
    Const DQUOTE = """" ' one " character

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("ThisWorkbook")
    Set CodeMod = VBComp.CodeModule

    With CodeMod
        LineNum = .CreateEventProc("Open", "Workbook")
        LineNum = LineNum + 1
        .InsertLines LineNum, "    MsgBox " & DQUOTE & "Hello World" & DQUOTE
    End With
End Sub
```

Creating A Procedure

You can use code to create code in a module. The code below creates a simple "Hello World" Sub procedure. You can either create a new VBComponent to hold the procedure or you can use an existing module. Comment out the appropriate lines of code.

```
Sub CreateProcedure()
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim S As String
```

```
    Dim LineNum As Long

    ' Use the next two lines to create a new module for the code
    'Set VBComp = ThisWorkbook.VBProject.VBComponents.Add(vbext_ct_StdModule)
    'VBComp.Name = "NewModule"
    ' OR use the following line to use an existing module for the code
    'Set VBComp = ThisWorkbook.VBProject.VBComponents("Module2")

    Set CodeMod = VBComp.CodeModule
    LineNum = CodeMod.CountOfLines + 1
    S = "Sub HelloWorld()" & vbCrLf & _
        "    MsgBox ""Hello, World""" & vbCrLf & _
        "End Sub"
    CodeMod.InsertLines LineNum, S
End Sub
```

This code creates the procedure:

```
    Sub HelloWorld()
        MsgBox "Hello, World"
    End Sub
```

Deleting A Module From A Project

This code will delete `Module1` from the VBProject. Note that you cannot remove any of the Sheet modules or the ThisWorkbook module. In general, you cannot delete a module whose Type is `vbext_ct_Document`.

```
    Sub DeleteModule()
        Dim VBProj As VBIDE.VBProject
        Dim VBComp As VBIDE.VBComponent

        Set VBProj = ActiveWorkbook.VBProject
        Set VBComp = VBProj.VBComponents("Module1")
        VBProj.VBComponents.Remove VBComp
```

```
End Sub
```

Renaming A Module

You can manually rename a module by displaying the Properties window (press F4) for the module and changing the Name property. You can do this programmatically with

```
ActiveWorkbook.VBProject.VBComponents("OldName").Name = "NewName"
```

Deleting A Procedure From A Module

This code will delete the procedure `DeleteThisProc` from the `Module1`. You must specify the procedure type in order to differentiate between `Property Get`, `Property Let`, and `Property Set` procedure, all of which have the same name.

```
Sub DeleteProcedureFromModule()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim StartLine As Long
    Dim NumLines As Long
    Dim ProcName As String

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("Module1")
    Set CodeMod = VBComp.CodeModule

    ProcName = "DeleteThisProc"
    With CodeMod
        StartLine = .ProcStartLine(ProcName, vbext_pk_Proc)
        NumLines = .ProcCountLines(ProcName, vbext_pk_Proc)
        .DeleteLines StartLine:=StartLine, Count:=NumLines
    End With
```

```
End Sub
```

Deleting All VBA Code In A Project

This code will delete ALL VBA code in a VBProject.

```
Sub DeleteAllVBACode()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule

    Set VBProj = ActiveWorkbook.VBProject

    For Each VBComp In VBProj.VBComponents
        If VBComp.Type = vbext_ct_Document Then
            Set CodeMod = VBComp.CodeModule
            With CodeMod
                .DeleteLines 1, .CountOfLines
            End With
        Else
            VBProj.VBComponents.Remove VBComp
        End If
    Next VBComp
End Sub
```

Eliminating Screen Flicker During VBProject Code

When you used the Extensibility code, the VBA Editor window will flicker. This can be reduced with the code:

```
Application.VBE.MainWindow.Visible = False
```

This will hide the VBE window, but you may still see it flicker. To prevent this, you must use the `LockWindowUpdate` Windows API function.

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal ClassName As String, ByVal WindowName As String) As Long

Private Declare Function LockWindowUpdate Lib "user32" _
    (ByVal hWndLock As Long) As Long


Sub EliminateScreenFlicker()
    Dim VBEHwnd As Long

    On Error GoTo ErrH:

    Application.VBE.MainWindow.Visible = False

    VBEHwnd = FindWindow("wndclass_desked_gsk", _
        Application.VBE.MainWindow.Caption)

    If VBEHwnd Then
        LockWindowUpdate VBEHwnd
    End If

    ''''''''''''''''''''''''''
    ' your code here
    ''''''''''''''''''''''''''

    Application.VBE.MainWindow.Visible = False
ErrH:
    LockWindowUpdate 0&
End Sub
```

Exporting A VBComponent Code Module To A Text File

You can export an existing VBComponent CodeModule to a text file. This can be useful if you are archiving modules to create a library of useful module to be used in other projects.

```
Public Function ExportVBComponent(VBComp As VBIDE.VBComponent, _
            FolderName As String, _
            Optional FileName As String, _
            Optional OverwriteExisting As Boolean = True) As Boolean
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This function exports the code module of a VBComponent to a text
' file. If FileName is missing, the code will be exported to
' a file with the same name as the VBComponent followed by the
' appropriate extension.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Extension As String
Dim FName As String
Extension = GetFileExtension(VBComp:=VBComp)
If Trim(FileName) = vbNullString Then
    FName = VBComp.Name & Extension
Else
    FName = FileName
    If InStr(1, FName, ".", vbBinaryCompare) = 0 Then
        FName = FName & Extension
    End If
End If

If StrComp(Right(FolderName, 1), "\", vbBinaryCompare) = 0 Then
    FName = FolderName & FName
Else
    FName = FolderName & "\" & FName
End If

If Dir(FName, vbNormal + vbHidden + vbSystem) <> vbNullString Then
    If OverwriteExisting = True Then
```

```
            Kill FName
        Else
            ExportVBComponent = False
            Exit Function
        End If
    End If

    VBComp.Export FileName:=FName
    ExportVBComponent = True

    End Function

    Public Function GetFileExtension(VBComp As VBIDE.VBComponent) As String
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' This returns the appropriate file extension based on the Type of
    ' the VBComponent.
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        Select Case VBComp.Type
            Case vbext_ct_ClassModule
                GetFileExtension = ".cls"
            Case vbext_ct_Document
                GetFileExtension = ".cls"
            Case vbext_ct_MSForm
                GetFileExtension = ".frm"
            Case vbext_ct_StdModule
                GetFileExtension = ".bas"
            Case Else
                GetFileExtension = ".bas"
        End Select

    End Function
```

Listing All Modules In A Project

This code will list all the modules and their types in the workbook, starting the listing in cell **A1**.

```
Sub ListModules()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim WS As Worksheet
    Dim Rng As Range

    Set VBProj = ActiveWorkbook.VBProject
    Set WS = ActiveWorkbook.Worksheets("Sheet1")
    Set Rng = WS.Range("A1")

    For Each VBComp In VBProj.VBComponents
        Rng(1, 1).Value = VBComp.Name
        Rng(1, 2).Value = ComponentTypeToString(VBComp.Type)
        Set Rng = Rng(2, 1)
    Next VBComp
End Sub


Function ComponentTypeToString(ComponentType As VBIDE.vbext_ComponentType) As String
    Select Case ComponentType
        Case vbext_ct_ActiveXDesigner
            ComponentTypeToString = "ActiveX Designer"
        Case vbext_ct_ClassModule
            ComponentTypeToString = "Class Module"
        Case vbext_ct_Document
            ComponentTypeToString = "Document Module"
        Case vbext_ct_MSForm
            ComponentTypeToString = "UserForm"
        Case vbext_ct_StdModule
            ComponentTypeToString = "Code Module"
        Case Else
            ComponentTypeToString = "Unknown Type: " & CStr(ComponentType)
    End Select
```

```
End Function
```

Listing All Procedures In A Module

This code will list all the procedures in `Module1`, beginning the listing in cell **A1**.

```
Sub ListProcedures()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim LineNum As Long
    Dim NumLines As Long
    Dim WS As Worksheet
    Dim Rng As Range
    Dim ProcName As String
    Dim ProcKind As VBIDE.vbext_ProcKind

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("Module1")
    Set CodeMod = VBComp.CodeModule

    Set WS = ActiveWorkbook.Worksheets("Sheet1")
    Set Rng = WS.Range("A1")
    With CodeMod
        LineNum = .CountOfDeclarationLines + 1
        Do Until LineNum >= .CountOfLines
            ProcName = .ProcOfLine(LineNum, ProcKind)
            Rng.Value = ProcName
            Rng(1, 2).Value = ProcKindString(ProcKind)
            LineNum = .ProcStartLine(ProcName, ProcKind) + _
                      .ProcCountLines(ProcName, ProcKind) + 1
            Set Rng = Rng(2, 1)
        Loop
```

```
    End With

End Sub


Function ProcKindString(ProcKind As VBIDE.vbext_ProcKind) As String
    Select Case ProcKind
        Case vbext_pk_Get
            ProcKindString = "Property Get"
        Case vbext_pk_Let
            ProcKindString = "Property Let"
        Case vbext_pk_Set
            ProcKindString = "Property Set"
        Case vbext_pk_Proc
            ProcKindString = "Sub Or Function"
        Case Else
            ProcKindString = "Unknown Type: " & CStr(ProcKind)
    End Select
End Function
```

General Infomation About A Procedure

The code below returns the following information about a procedure in a module, loaded into the `ProcInfo` Type. The function `ProcedureInfo` takes as input then name of the procedure, a `VBIDE.vbext_ProcKind` procedure type, and a reference to the CodeModule object containing the procedure.

```
Public Enum ProcScope
    ScopePrivate = 1
    ScopePublic = 2
    ScopeFriend = 3
    ScopeDefault = 4
End Enum
```

```
Public Enum LineSplits
    LineSplitRemove = 0
    LineSplitKeep = 1
    LineSplitConvert = 2
End Enum

Public Type ProcInfo
    ProcName As String
    ProcKind As VBIDE.vbext_ProcKind
    ProcStartLine As Long
    ProcBodyLine As Long
    ProcCountLines As Long
    ProcScope As ProcScope
    ProcDeclaration As String
End Type

Function ProcedureInfo(ProcName As String, ProcKind As VBIDE.vbext_ProcKind, _
    CodeMod As VBIDE.CodeModule) As ProcInfo

    Dim PInfo As ProcInfo
    Dim BodyLine As Long
    Dim Declaration As String
    Dim FirstLine As String


    BodyLine = CodeMod.ProcStartLine(ProcName, ProcKind)
    If BodyLine > 0 Then
        With CodeMod
            PInfo.ProcName = ProcName
            PInfo.ProcKind = ProcKind
            PInfo.ProcBodyLine = .ProcBodyLine(ProcName, ProcKind)
            PInfo.ProcCountLines = .ProcCountLines(ProcName, ProcKind)
            PInfo.ProcStartLine = .ProcStartLine(ProcName, ProcKind)
```

```
                FirstLine = .Lines(PInfo.ProcBodyLine, 1)
                If StrComp(Left(FirstLine, Len("Public")), "Public", vbBinaryCompare) = 0 Then
                    PInfo.ProcScope = ScopePublic
                ElseIf StrComp(Left(FirstLine, Len("Private")), "Private", vbBinaryCompare) = 0 Then
                    PInfo.ProcScope = ScopePrivate
                ElseIf StrComp(Left(FirstLine, Len("Friend")), "Friend", vbBinaryCompare) = 0 Then
                    PInfo.ProcScope = ScopeFriend
                Else
                    PInfo.ProcScope = ScopeDefault
                End If
                PInfo.ProcDeclaration = GetProcedureDeclaration(CodeMod, ProcName, ProcKind,
LineSplitKeep)
            End With
        End If

        ProcedureInfo = PInfo

    End Function


    Public Function GetProcedureDeclaration(CodeMod As VBIDE.CodeModule, _
        ProcName As String, ProcKind As VBIDE.vbext_ProcKind, _
        Optional LineSplitBehavior As LineSplits = LineSplitRemove)
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' GetProcedureDeclaration
    ' This return the procedure declaration of ProcName in CodeMod. The LineSplitBehavior
    ' determines what to do with procedure declaration that span more than one line using
    ' the "_" line continuation character. If LineSplitBehavior is LineSplitRemove, the
    ' entire procedure declaration is converted to a single line of text. If
    ' LineSplitBehavior is LineSplitKeep the "_" characters are retained and the
    ' declaration is split with vbNewLine into multiple lines. If LineSplitBehavior is
    ' LineSplitConvert, the "_" characters are removed and replaced with vbNewLine.
    ' The function returns vbNullString if the procedure could not be found.
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        Dim LineNum As Long
```

```
    Dim S As String
    Dim Declaration As String

    On Error Resume Next
    LineNum = CodeMod.ProcBodyLine(ProcName, ProcKind)
    If Err.Number <> 0 Then
        Exit Function
    End If
    S = CodeMod.Lines(LineNum, 1)
    Do While Right(S, 1) = "_"
        Select Case True
            Case LineSplitBehavior = LineSplitConvert
                S = Left(S, Len(S) - 1) & vbNewLine
            Case LineSplitBehavior = LineSplitKeep
                S = S & vbNewLine
            Case LineSplitBehavior = LineSplitRemove
                S = Left(S, Len(S) - 1) & " "
        End Select
        Declaration = Declaration & S
        LineNum = LineNum + 1
        S = CodeMod.Lines(LineNum, 1)
    Loop
    Declaration = SingleSpace(Declaration & S)
    GetProcedureDeclaration = Declaration


End Function

Private Function SingleSpace(ByVal Text As String) As String
    Dim Pos As String
    Pos = InStr(1, Text, Space(2), vbBinaryCompare)
    Do Until Pos = 0
        Text = Replace(Text, Space(2), Space(1))
        Pos = InStr(1, Text, Space(2), vbBinaryCompare)
    Loop
```

```
        SingleSpace = Text
    End Function
```

You can call the `ProcedureInfo` function using code like the following:

```
    Sub ShowProcedureInfo()
        Dim VBProj As VBIDE.VBProject
        Dim VBComp As VBIDE.VBComponent
        Dim CodeMod As VBIDE.CodeModule
        Dim CompName As String
        Dim ProcName As String
        Dim ProcKind As VBIDE.vbext_ProcKind
        Dim PInfo As ProcInfo

        CompName = "modVBECode"
        ProcName = "ProcedureInfo"
        ProcKind = vbext_pk_Proc

        Set VBProj = ActiveWorkbook.VBProject
        Set VBComp = VBProj.VBComponents(CompName)
        Set CodeMod = VBComp.CodeModule

        PInfo = ProcedureInfo(ProcName, ProcKind, CodeMod)

        Debug.Print "ProcName: " & PInfo.ProcName
        Debug.Print "ProcKind: " & CStr(PInfo.ProcKind)
        Debug.Print "ProcStartLine: " & CStr(PInfo.ProcStartLine)
        Debug.Print "ProcBodyLine: " & CStr(PInfo.ProcBodyLine)
        Debug.Print "ProcCountLines: " & CStr(PInfo.ProcCountLines)
        Debug.Print "ProcScope: " & CStr(PInfo.ProcScope)
        Debug.Print "ProcDeclaration: " & PInfo.ProcDeclaration
    End Sub
```

Searching For Text In A Module

The CodeModule object has a Find method that you can use to search for text within the code module. The Find method accepts ByRef Long parameters. Upon input, these parameters specify the range of lines and column to search. On output, these values will point to the found text. To find the second and subsequent occurence of the text, you need to set the parameters to refer to the text following the found line and column. The Find method returns True or False indicating whether the text was found. The code below will search all of the code in Module1 and print a Debug message for each found occurrence. Note the values set with the SL, SC, EL, and EC variables. The code loops until the Found variable is False.

```
Sub SearchCodeModule()
    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim FindWhat As String
    Dim SL As Long ' start line
    Dim EL As Long ' end line
    Dim SC As Long ' start column
    Dim EC As Long ' end column
    Dim Found As Boolean

    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("Module1")
    Set CodeMod = VBComp.CodeModule

    FindWhat = "findthis"

    With CodeMod
        SL = 1
        EL = .CountOfLines
        SC = 1
        EC = 255
        Found = .Find(target:=FindWhat, StartLine:=SL, StartColumn:=SC, _
            EndLine:=EL, EndColumn:=EC, _
            wholeword:=True, MatchCase:=False, patternsearch:=False)
        Do Until Found = False
            Debug.Print "Found at: Line: " & CStr(SL) & " Column: " & CStr(SC)
            EL = .CountOfLines
```

```
                SC = EC + 1
                EC = 255
                Found = .Find(target:=FindWhat, StartLine:=SL, StartColumn:=SC, _
                    EndLine:=EL, EndColumn:=EC, _
                    wholeword:=True, MatchCase:=False, patternsearch:=False)
            Loop
        End With
    End Sub
```

Testing If A VBComponent Exists

This code will return True or False indicating whether the VBComponent named by `VBCompName` exists in the project referenced by `VBProj`. If `VBProj` is omitted, the `VBProject` of the ActiveWorkbook is used.

```
    Public Function VBComponentExists(VBCompName As String, Optional VBProj As VBIDE.VBProject =
Nothing) As Boolean
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' This returns True or False indicating whether a VBComponent named
    ' VBCompName exists in the VBProject referenced by VBProj. If VBProj
    ' is omitted, the VBProject of the ActiveWorkbook is used.
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        Dim VBP As VBIDE.VBProject
        If VBProj Is Nothing Then
            Set VBP = ActiveWorkbook.VBProject
        Else
            Set VBP = VBProj
        End If
        On Error Resume Next
        VBComponentExists = CBool(Len(VBP.VBComponents(VBCompName).Name))

    End Function
```

Total Code Lines In A Component Code Module

This function will return the total code lines in a VBComponent. It ignores blank lines and comment lines. It will return -1 if the project is locked.

```
Public Function TotalCodeLinesInVBComponent(VBComp As VBIDE.VBComponent) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This returns the total number of code lines (excluding blank lines and
' comment lines) in the VBComponent referenced by VBComp. Returns -1
' if the VBProject is locked.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Dim N As Long
    Dim S As String
    Dim LineCount As Long

    If VBComp.Collection.Parent.Protection = vbext_pp_locked Then
        TotalCodeLinesInVBComponent = -1
        Exit Function
    End If

    With VBComp.CodeModule
        For N = 1 To .CountOfLines
            S = .Lines(N, 1)
            If Trim(S) = vbNullString Then
                ' blank line, skip it
            ElseIf Left(Trim(S), 1) = "'" Then
                ' comment line, skip it
            Else
                LineCount = LineCount + 1
            End If
        Next N
    End With
    TotalCodeLinesInVBComponent = LineCount
End Function
```

Total Lines In A Project

This code will return the count of lines in all components of the project referenced by `VBProj`. If `VBProj` is omitted, the VBProject of the ActiveWorkbook is used. The function will return -1 if the project is locked.

```
Public Function TotalLinesInProject(Optional VBProj As VBIDE.VBProject = Nothing) As Long
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This returns the total number of lines in all components of the VBProject
' referenced by VBProj. If VBProj is missing, the VBProject of the ActiveWorkbook
' is used. Returns -1 if the VBProject is locked.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Dim VBP As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim LineCount As Long

    If VBProj Is Nothing Then
        Set VBP = ActiveWorkbook.VBProject
    Else
        Set VBP = VBProj
    End If

    If VBP.Protection = vbext_pp_locked Then
        TotalLinesInProject = -1
        Exit Function
    End If

    For Each VBComp In VBP.VBComponents
        LineCount = LineCount + VBComp.CodeModule.CountOfLines
    Next VBComp

    TotalLinesInProject = LineCount
End Function
```

Total Code Lines In A Component

This function will return the total number of code lines in a VBComponent. It ignores blank lines and comment lines. It will return -1 if the project is locked.

```
Public Function TotalCodeLinesInVBComponent(VBComp As VBIDE.VBComponent) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This returns the total number of code lines (excluding blank lines and
' comment lines) in the VBComponent referenced by VBComp. Returns -1
' if the VBProject is locked.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Dim N As Long
    Dim S As String
    Dim LineCount As Long

    If VBComp.Collection.Parent.Protection = vbext_pp_locked Then
        TotalCodeLinesInVBComponent = -1
        Exit Function
    End If

    With VBComp.CodeModule
        For N = 1 To .CountOfLines
            S = .Lines(N, 1)
            If Trim(S) = vbNullString Then
                ' blank line, skip it
            ElseIf Left(Trim(S), 1) = "'" Then
                ' comment line, skip it
            Else
                LineCount = LineCount + 1
            End If
        Next N
    End With
    TotalCodeLinesInVBComponent = LineCount
```

```
End Function
```

Total Code Lines In A Project

This function will return the total number of code lines in all the components of a VBProject. It ignores blank lines and comment lines. It will return -1 if the project is locked.

```
Public Function TotalCodeLinesInProject(VBProj As VBIDE.VBProject) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This returns the total number of code lines (excluding blank lines and
' comment lines) in all VBComponents of VBProj. Returns -1 if VBProj
' is locked.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Dim VBComp As VBIDE.VBComponent
    Dim LineCount As Long
    If VBProj.Protection = vbext_pp_locked Then
        TotalCodeLinesInProject = -1
        Exit Function
    End If
    For Each VBComp In VBProj.VBComponents
        LineCount = LineCount + TotalCodeLinesInVBComponent(VBComp)
    Next VBComp

    TotalCodeLinesInProject = LineCount
End Function
```

Workbook Associated With A VBProject

The Workbook object provides a property named `VBProject` that allows you to reference to the VBProject associated with a workbook. However, the reverse is not true. There is no direct way to get a reference to the workbook that contains a specific VBProject. However, it can be done with some fairly simple code. The following function, `WorkbookOfVBProject`, will return a reference to the Workbook object that contains the VBProject indicated by the `WhichVBP` parameter. This parameter may be a `VBIDE.VBProject` object, or a string containing the name of the VBProject (the project name, not the workbook name), or a numeric index, indicating the ordinal index

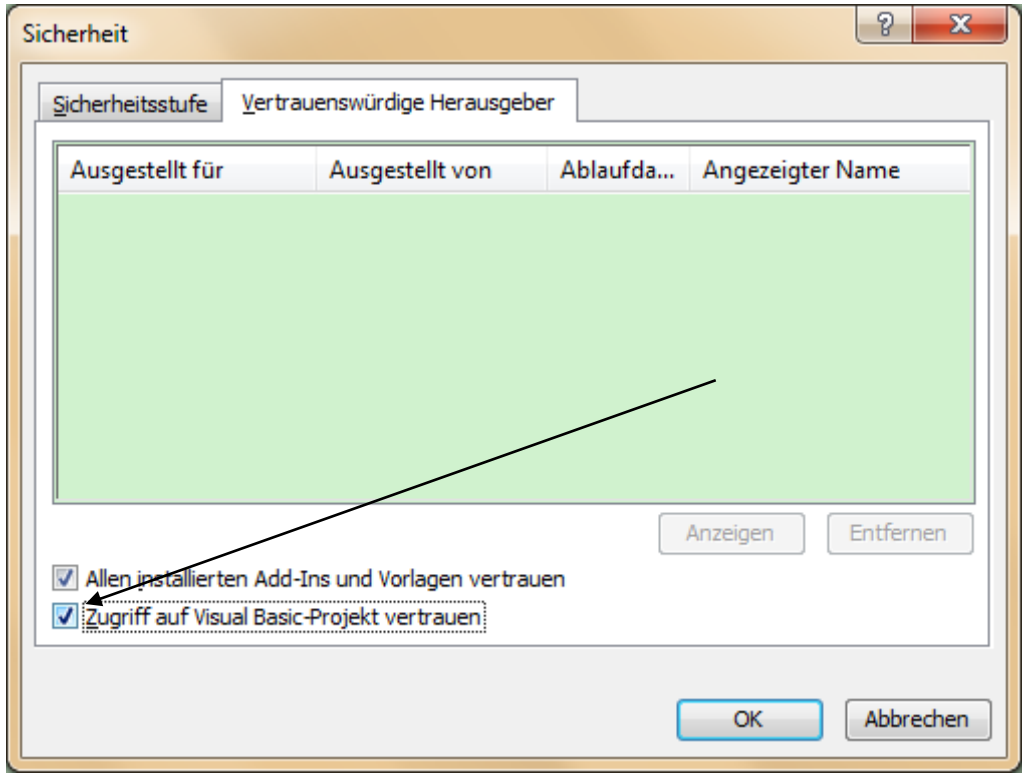
of the VBProject (its position in the list of VBProjects in the Project Explorer window). If the parameter is any object other than VBIDE.VBProject, the code raises an error 13 (type mismatch). If the parameter does not name an existing VBProject, the code raises an error 9 (subscript out of range). If you have more than one VBProject with the default name VBAProject, the code will return the first VBProject with that name.

```
Function WorkbookOfVBProject(WhichVBP As Variant) As Workbook
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' WorkbookOfVBProject
' This returns the Workbook object for a specified VBIDE.VBProject.
' The parameter WhichVBP can be any of the following:
'   A VBIDE.VBProject object
'   A string containing the name of the VBProject.
'   The index number (ordinal position in Project window) of the VBProject.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim WB As Workbook
Dim AI As AddIn
Dim VBP As VBIDE.VBProject

If IsObject(WhichVBP) = True Then
    ' If WhichVBP is an object, it must be of the
    ' type VBIDE.VBProject. Any other object type
    ' throws an error 13 (type mismatch).
    On Error GoTo 0
    If TypeOf WhichVBP Is VBIDE.VBProject Then
        Set VBP = WhichVBP
    Else
        Err.Raise 13
    End If
Else
    On Error Resume Next
    Err.Clear
    ' Here, WhichVBP is either the string name of
    ' the VBP or its ordinal index number.
```

```
    Set VBP = Application.VBE.VBProjects(WhichVBP)
    On Error GoTo 0
    If VBP Is Nothing Then
        Err.Raise 9
    End If
End If

For Each WB In Workbooks
    If WB.VBProject Is VBP Then
        Set WorkbookOfVBProject = WB
        Exit Function
    End If
Next WB
' not found in workbooks, search installed add-ins.
For Each AI In Application.AddIns
    If AI.Installed = True Then
        If Workbooks(AI.Name).VBProject Is VBP Then
            Set WorkbookOfVBProject = Workbooks(AI.Name)
            Exit Function
        End If
    End If
Next AI

End Function
```

Neue Exceldatei mit Commandbutton und VBA-Code

Das Wichtigste: möchte man VBA-Code generieren, muss dies bei den Makro-Sicherheitseinstellungen aktiviert sein:



Und hier der eigentliche Code:

```
Sub Erzeuge_Vorlage()

    Workbooks.Add

    ' den nachfolgenden Steuerelement-Toolbox-Button erzeugt man als OLE-Object:

    Set MY_OO = ActiveSheet.OLEObjects.Add(ClassType:="Forms.CommandButton.1", Link:=False, DisplayAsIcon:=False, Left:=100, Top:=100, Width:=100, Height:=40)

    MY_OO.Object.Caption = "Drück mich"

    ' Nun tragen wir den VBA-Code für diesen Button in das betreffende Tabellenblatt ein, in dem wir den Button erzeugt haben

    With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.CodeName).CodeModule
```

```
    ' .DeleteLines 1, .CountOfLines ' damit würde man ev. existierende Programmzeilen löschen
    .InsertLines 1, "Private Sub CommandButton1_Click()"
    .InsertLines 2, "Msgbox ""Es hat geklappt"""
    .InsertLines 3, "End Sub"
  End With

End Sub
```

Adding a CommandButton with code

Hallo, liebe Mituser!

In meinem VBA-Programm generiere ich Tabellenblätter. Diese sollen auch eine bestimmte Befehlsschaltfläche bekommen. Auch per VBA. Der Endnutzer kann sie nicht erzeugen, also soll der Code das tun.

Wie lautet der Befehl zum Einfügen einer Befehlsschaltfläche auf einem Tabellenblatt und wie weist man Namen, Caption und Eigenschaften der Befehlsschaltfläche zu? Und natürlich: wie weist man der Befehlsschaltfläche Ereignis-Code zu?

Lösung 1

Hinzufügen von Steuerelementen mit Visual Basic

In Microsoft Excel werden ActiveX-Steuerelemente durch OLEObject-Objekte in der OLEObjects-Auflistung dargestellt (alle OLEObject-Objekte befinden sich ebenfalls in der Shapes-Auflistung). Um ein ActiveX-Steuerelement per Programm zu einem Tabellenblatt hinzuzufügen, verwenden Sie die Add-Methode der OLEObjects-Auflistung. In dem folgenden Beispiel wird eine Befehlsschaltfläche zu der ersten Arbeitsmappe hinzugefügt.

```
Worksheets(1).OLEObjects.Add "Forms.CommandButton.1", Left:=10, Top:=10, Height:=20, Width:=100
```

Verwenden von Steuerelementeigenschaften mit Visual Basic

Am häufigsten verweist Visual Basic-Code über den Namen auf ActiveX-Steuerelemente. In dem folgenden Beispiel wird die Beschriftung des Steuerelements mit dem Namen "CommandButton1" geändert.

```
Sheet1.CommandButton1.Caption = "Run"
```

Beachten Sie, dass Sie beim Verwenden eines Steuerelementnamens außerhalb des Klassenmoduls für das Tabellenblatt, das das Steuerelement enthält, den Steuerelementnamen zusammen mit dem Blattnamen angeben müssen.

Zum Ändern des Steuerelementnamens, der in Visual Basic-Code verwendet wird, wählen Sie das Steuerelement aus und legen Sie die (Name)-Eigenschaft im Eigenschaftenfenster fest.

Da ActiveX-Steuerelemente auch durch OLEObject-Objekte in der OLEObjects-Auflistung dargestellt werden, können Sie Steuerelementeigenschaften mit Hilfe der Objekte in der Auflistung festlegen. In dem folgenden Beispiel wird die linke Position des Steuerelements mit dem Namen "CommandButton1" festgelegt.

```
Worksheets(1).OLEObjects("CommandButton1").Left = 10
```

Lösung 2

HAllo

Lösung 3 beschreibt das Vorgehen wenn der Button aus der Symbolleiste Formular stammt.
Willst du einen echten CommandButton (ActiveX-Steuerelement) einfügen ist das Vorgehen ähnlich, aber nicht genauso.

Bau dir das mal nach:
Im Codemodul von Tabelle1 liegt der Code der ins neue Blatt soll.

```
' ************************************************************************
' Modul: Tabelle1 Typ: Element der Mappe(Sheet, Workbook, ...)
' ************************************************************************

Option Explicit


Private Sub CommandButton1_Click()
MsgBox "Test"
MsgBox "...und noch ein Text"
End Sub
```

Damit das Ganze auch automatisch angeschubst wird, diesen Code ins CodeModul von DieseArbeitsmappe:

```
' ************************************************************************
' Modul: DieseArbeitsmappe Typ: Element der Mappe(Sheet, Workbook, ...)
' ************************************************************************

Option Explicit
```

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
Dim CB As Object
Dim sh_New As Object
Dim sh_mit_Code As Object 'Tabelle1. Hier liegt der Code der ins neue Blatt soll.
Dim i As Integer


If Sh.Type <> xlWorksheet Then Exit Sub


Set CB = Sh.OLEObjects.Add("Forms.CommandButton.1") 'Button einfügen
```

```
With CB 'Button formatieren
    .PrintObject = False ' Button nicht ausdruckbar
    .Object.Caption = "Lalala..."
    .Top = 123
    .Left = 456
    .Height = 40
    .Width = 150
End With

Set sh_mit_Code = Application.VBE.ActiveVBProject.VBComponents(Sheets("Tabelle1").CodeName)
Set sh_New = Application.VBE.ActiveVBProject.VBComponents(Sh.CodeName)

With sh_New.CodeModule
    .DeleteLines 1, .countoflines 'Alte Zeilen löschen
    For i = 1 To sh_mit_Code.CodeModule.countoflines 'Zeilen aus Tabelle1 "abschreiben"
        .InsertLines i, sh_mit_Code.CodeModule.Lines(i, 1)
    Next
End With

End Sub
```

ransi

LÖSUNG 3

Neue Schaltfläche mit zugehörigem Code in Tabellenblatt

Problem: Wie kann ich auf in ein Tabellenblatt eine neue Schaltfläche mit zugehörigem Code einfügen lassen? Nach betätigung dieser Schaltfläche soll sie einschließlich Code wieder gelöscht werden.

```
ClassModule: Tabelle1

Private Sub cmdCreateButton_Click()
   Dim oBtn As Button
   Dim sCode As String
   sCode = "Sub cmdNew_Click" & vbLf
   sCode = sCode & "   Msgbox ""Hat funktioniert!""" & vbLf
   sCode = sCode & _
      "   ActiveSheet.Buttons(Application.Caller).Delete " & vbLf
   sCode = sCode & "   With ThisWorkbook.VBProject.VBComponents" & _
      "(""basMain"").CodeModule" & vbLf
   sCode = sCode & "      .DeleteLines 1, .CountOfLines" & vbLf
   sCode = sCode & "   End With" & vbLf
   sCode = sCode & "End Sub"
   Set oBtn = Buttons.Add(100, 100, 70, 20)
   With oBtn
      .Caption = "Meldung"
      .OnAction = "cmdNew_Click"
   End With
   With ThisWorkbook.VBProject.VBComponents("basMain").CodeModule
      .AddFromString sCode
   End With
End Sub
```

Problem

Hi!!!

Ich bin am verzweifeln und brauche dringend Hilfe.

Bevor ich meinen Code poste, hier ne kurze Erklärung:

VB erzeugt im Exel ein neues Sheet und darin einen CommandButton.
Dieser Commandbutton wird auch erzugt und lässt sich auch die Caption ändern.
Allerdings löst er beim klicken nicht wie gewünscht die MSGBox aus.

Ich hoffe mir kann jemand helfen! Hier mein Code:

Option Explicit

```
Sub Neu()

    Dim Button As Object

    ActiveWorkbook.Sheets.Add
    ActiveSheet.Name = "OPG"
    ActiveSheet.Move Before:=Sheets(9)

    Range("A1") = "Produktnummer"
    With ActiveCell.Characters(Start:=1, Length:=13).Font
        .Name = "Arial"
        .FontStyle = "Fett"
        .Size = 10
    End With
```

```
    Columns("A:A").EntireColumn.AutoFit

    Range("B1") = "Einkaufspreis"
    With ActiveCell.Characters(Start:=1, Length:=13).Font
        .Name = "Arial"
        .FontStyle = "Fett"
        .Size = 10
    End With
    Columns("B:B").EntireColumn.AutoFit

    MsgBox "Bitte fügen Sie die Produktnummern und Einkaufspreise aus dem OPG ein. Klicken Sie danach den Button >Weiter< um fortzufahren"

    Dim oo As OLEObject
    Set oo = ActiveSheet.OLEObjects.Add(ClassType:="Forms.Commandbutton.1", Link:=False _
        , DisplayAsIcon:=False, Left:=350, Top:=10, Width:=87.75, Height _
        :=29.25)
    Set ActiveWorkbook.buttonOPG = oo.Object
    ActiveWorkbook.buttonOPG.Caption = "Hallo"

End Sub
```

```
Public Sub callme()
    MsgBox "gna"
End Sub
```

```
Public WithEvents buttonOPG As CommandButton
```

Private Sub buttonOPG_Click()

Call ModulOPG.callme

End Sub

Danke mal für Eure Hilfe
Nali

Antwort

Hallo Nali,
das funktioniert so nicht. Warum? Weil Excel beim einfügen eines ActiveX - Controls, auch per Makro, in den Bearbeitungsmodus wechselt, der erst nach dem beenden der Prozedur wieder verlassen wird. Dabei geht dein Verweis auf die Klasse flöten. Ich behelfe mir damit, dass ich die Zuweisung an die Klasse über die OnTime - Methode starte. Da ist die Prozedur zum erstellen des Buttons schon durchgelaufen. Reicht der Hinweis, oder benötigst du ein Beispiel?

Gruß
Nepumuk

Adding a VBA CommandButton with its respective the code

Paste these two sections of code into a general module (Module1 for example).

```
Sub CreateButton()

Dim Obj As Object
Dim Code As String

Sheets("Sheet1").Select
```

```
'create button
    Set Obj = ActiveSheet.OLEObjects.Add(ClassType:="Forms.CommandButton.1", _
    Link:=False, DisplayAsIcon:=False, Left:=200, Top:=100, Width:=100, Height:=35)
    Obj.Name = "TestButton"
'buttonn text
    ActiveSheet.OLEObjects(1).Object.Caption = "Test Button"

'macro text
    Code = "Sub ButtonTest_Click()" & vbCrLf
    Code = Code & "Call Tester" & vbCrLf
    Code = Code & "End Sub"
'add macro at the end of the sheet module
    With ActiveWorkbook.VBProject.VBComponents(ActiveSheet.Name).CodeModule
        .insertlines .CountOfLines + 1, Code
    End With
End Sub

Sub Tester()
    MsgBox "You have click on the test button"
End Sub
```

Macro to Create Command Button Click Event Code

I used this to add a button and code to an existing spreadsheet I had issued. Rather than trying to add the entire code under the button, I export a module "Update1" containing the procedure "PrepareMacro" and then import it back into the workbook being updated.

VB:

```
Sub UpdateWeeklyFigures()

    Dim vbCodeMod As CodeModule
    Dim oleButton As OLEObject
    Dim wbUpdate As Workbook
    Dim shtControls As Worksheet
    Dim strFileName As String
    Dim lngLines As Long
    Dim bolSkip As Boolean
    Dim strPath As String

     'Get the name of the file to open
    strFileName = Application.GetOpenFilename(Title:="Select File To Update")

     'Turn screen off
    Application.ScreenUpdating = False

     'If file cannot be opened or the sheet Controls does not exist then drop to error handler
```

```
    On Error Goto errhandler
     'Open the file
    Set wbUpdate = Application.Workbooks.Open(strFileName)
     'Reference shtControls
    Set shtControls = wbUpdate.Worksheets("Controls")
    On Error Goto 0

     'Check that button not already entered
    For Each oleButton In shtControls.OLEObjects
        If oleButton.Name = "Prepare" Then bolSkip = True
    Next

     'If button not entered
    If Not bolSkip Then
         'Add the button to the sheet
        Set oleButton = shtControls.OLEObjects.Add(ClassType:="Forms.CommandButton.1", Link:=False _
        , DisplayAsIcon:=False, Left:=49.5, Top:=194.5, Width:=169.5, Height _
        :=34.5)
         'Rename the button, so that the code added later refers to it
        oleButton.Name = "Prepare"
        oleButton.Object.Caption = "Click to prepare for Sending"
         'Add code to trigger action
        With wbUpdate.VBProject.VBComponents(shtControls.CodeName).CodeModule
             'Add event procedure for prepare click
            lngLines = .CreateEventProc("click", "prepare") + 1
             'Now add the code
            .InsertLines lngLines, _
            "PrepareMacro"
        End With

         'Copy the code module to the selected workbbok
        With ThisWorkbook
             'Name for temp file to store code
            strPath = .Path & "\code.txt"
             'Export code to temp file
            .VBProject.VBComponents("Update1").Export strPath
        End With
         'Import code from temp file
        wbUpdate.VBProject.VBComponents.Import strPath
         'Delete the temp file
        Kill strPath

         'Add some comments next to new button
        shtControls.Range("F15").Value = "Hides or displays Actuals and Forecast"

         'Save the updated workbook witn the changes
        wbUpdate.Close savechanges:=True

        MsgBox "Update Finished"

    Else
        MsgBox "Button already exists"
```

```
    End If

     'Turn screen back on
    Application.ScreenUpdating = True

    Exit Sub

errhandler:

    MsgBox "Incorrect/No file selected for update" & Chr(13) & Chr(13) & "Please try again"

End Sub
```

create a button with macro from code module programmatically

Version tested with	2003
Submitted by:	Paleo
Description:	Insert a button in your spreadsheet and set a macro for it programmatically
Discussion:	Sometimes you need to create a button in a spreadsheet you have imported or made some modifications and you must do it several times without knowing each sheet name. So you need a code that creates this button and its code programmaticaly. This is what I do here.

Code: **instructions for use**

```
Sub AddButtonAndCode()
     ' Declare variables
    Dim i As Long, Hght As Long
    Dim Name As String, NName As String
     ' Set the button properties
    i = 0
    Hght = 305.25
```

```
 ' Set the name for the button
NName = "cmdAction" & i
 ' Test if there is a button already and if so, increment its name
For Each OLEObject In ActiveSheet.OLEObjects
    If Left(OLEObject.Name, 9) = "cmdAction" Then
        Name = Right(OLEObject.Name, Len(OLEObject.Name) - 9)
        If Name >= i Then
            i = Name + 1
        End If
        NName = "cmdAction" & i
        Hght = Hght + 27
    End If
Next
 ' Add button
Dim myCmdObj As OLEObject, N%
Set myCmdObj = ActiveSheet.OLEObjects.Add(ClassType:="Forms.CommandButton.1", _
Link:=False, DisplayAsIcon:=False, Left:=52.5, Top:=Hght, _
Width:=202.5, Height:=26.25)
 ' Define buttons name
myCmdObj.Name = NName
 ' Define buttons caption
myCmdObj.Object.Caption = "Click for action"
 ' Inserts code for the button
With ThisWorkbook.VBProject.VBComponents(ActiveSheet.CodeName).CodeModule
```

```
        N = .CountOfLines
        .InsertLines N + 1, "Private Sub " & NName & "_Click()"
        .InsertLines N + 2, vbNewLine
        .InsertLines N + 3, vbTab & "MsgBox(" & """" & "Button Clicked!" & """" & " & vbCrLf &"
& _
        """" & "Put your code here!" & """" & " & vbCrLf & " & """" & "This is " & """" & _
        "& " & """" & NName & """" & ")"
        .InsertLines N + 4, vbNewLine
        .InsertLines N + 5, "End Sub"
    End With
End Sub
```

How to use:	1. Copy the above code. 2. Open any workbook. 3. Press Alt + F11 to open the Visual Basic Editor (VBE). 4. From the Menu, choose Insert-Module. 5. Paste the code into the right-hand code window. 6. Close the VBE, save the file if desired. 7. Tools | Macro | Security | Trusted Sources. 8. Check the box for "Trust access to Visual Basic Project".
Test the code:	1. Go to Tools-Macro-Macros or simply press Alt + F8.

2. Choose "AddButtonAndCode" and then choose "Execute" or simply double-click "AddButtonAndCode".
3. A new button will appear and if you click it you will see a message box, telling you which button you have clicked (in case you have more than one).

Auf VBA-Module zugreifen

http://www.vb-fun.de/cgi-bin/loadframe.pl?ID=vb/tipps/tip0071.shtml

Excel - Komponenten/Prozeduren ermitteln

Autor/Einsender:	**Angie**	Entwicklungsumgebung:	**Excel 97**
Datum:	**17.08.2001**		

Über die **VBE-Programmierung** ist es u.a. auch möglich, direkt auf die in einer Arbeitsmappe enthaltenen **Komponenten** (Excel-Objekte, UserForms, Module und Klassenmodule) und den darin enthaltenen **Prozeduren** zuzugreifen.

Mit der ***ProcStartLine***-Eigenschaft wird die Zeile zurückzugeben, an der die jeweilige Prozedur in einem bestimmten Code-Bereich beginnt. Die erste Code-Zeile wird mit der ***ProcBodyLine***-Eigenschaft und die Anzahl der Zeilen mit der ***ProcCountLines***-Eigenschaft ermittelt. Wie in diesem Download-Beispiel ersichtlich, ist die Anzahl der Zeilen einer Prozedur nicht unbedingt identisch mit der Anzahl der Code-Zeilen. Die hier ermittelten Angaben werden auf dem aktiven Arbeitsblatt aufgelistet.

```
Option Explicit

Public Sub Start_KomponentenErmitteln()

  Dim vbaProjekt As Object
  Dim vbaKomponente As Object
  Dim anzKomponenten As Integer

  Dim vbaProzedur As Object
  Dim prozedurName As String
  Dim anzProzeduren As Integer
  Dim zEndSub As String

  Dim startZ As Long
  Dim bodyZ As Long
```

```
  Dim anzZ As Long
  Dim anzC As Long

  Dim z As Integer
  Dim i As Integer
  Dim j As Integer

  anzKomponenten = 0
  anzProzeduren = 0
  z = 2

  Set vbaProjekt = ActiveWorkbook.VBProject.VBComponents

  For Each vbaKomponente In vbaProjekt
    z = z + 1

    Select Case vbaKomponente.Type
      Case 1: Cells(z, 1) = "Modul"
      Case 2: Cells(z, 1) = "Klassenmodul"
      Case 3: Cells(z, 1) = "Formular"
      Case 100: Cells(z, 1) = "Microsoft Excel Objekt"
    End Select

    Set vbaProzedur = vbaProjekt(vbaKomponente.Name).CodeModule
    Cells(z, 2) = vbaKomponente.Name
    anzKomponenten = anzKomponenten + 1

    With vbaProzedur

      For i = 1 To .CountOfLines
        If .ProcOfLine(i, vbext_pk_Proc) <> "" Then
          prozedurName = .ProcOfLine(i, vbext_pk_Proc)
          anzProzeduren = anzProzeduren + 1

          startZ = .ProcStartLine(prozedurName, vbext_pk_Proc)
          bodyZ = .ProcBodyLine(prozedurName, vbext_pk_Proc)
          anzZ = .ProcCountLines(prozedurName, vbext_pk_Proc)

          For j = bodyZ To .CountOfLines
            zEndSub = Trim(.Lines(j, 1))
            If (Left(zEndSub, 7)) = "End Sub" Or _
```

```
                (Left(zEndSub, 12)) = "End Function" Then
               anzC = j - bodyZ + 1
               Exit For
             End If
           Next

           Cells(z, 3) = prozedurName
           Cells(z, 4) = startZ
           Cells(z, 5) = bodyZ
           Cells(z, 6) = anzC
           Cells(z, 7) = anzZ
           z = z + 1

           i = i + anzZ
         End If
       Next
     End With
     z = z + 1
   Next

   ActiveSheet.Columns("A:C").AutoFit

   MsgBox "In der Arbeitsmappe '" & ActiveWorkbook.Name & _
          "' sind " & vbCrLf & _
          anzKomponenten & " Komponenten und " & _
          anzProzeduren & " Prozeduren enthalten !", _
          vbOKOnly + vbInformation, Title:="Komponenten ermitteln"

End Sub
```

Hinweis

Für die VBE-Programmierung ist das Einbinden der Objektbibliothek ***Visual Basic For Applications Extensibility 5.3*** notwendig.

VBA-Module Exportieren und Importieren

Excel - Komponenten/Prozeduren ermitteln

Autor/Einsender:	**Angie**	Entwicklungsumgebung:	**Excel 97**
Datum:	**27.05.2001**		

Über die **VBE-Programmierung** ist es u.a. möglich direkt auf Module, den darin enthaltenen Makros und auf Userforms zuzugreifen.

In diesem Beispiel werden verschiedene Einsatzmöglichkeiten aufgezeigt, z.B. wie Module exportiert, importiert und gelöscht werden können. Auch die Erstellung eines neuen Moduls sowie das Hinzufügen eines Makros aus einer Textdatei wird hier demonstriert.

```
Option Explicit

'-- Vorhandenes Modul 'modModulExpImp' als bas-Datei exportieren --
Sub ModulExportieren()
  Dim pfad As String
  Dim modulName As String

  pfad = ThisWorkbook.Path
  modulName = "modModulExpImp"

  ThisWorkbook.VBProject.VBComponents(modulName). _
      Export pfad & "\" & modulName & ".bas"

End Sub

'-- Vorhandenes Modul 'modModulExpImp' löschen --
Sub ModulLoeschen()
  Dim modulName As String
  Dim codeModul As Object 'VBComponent

  modulName = "modModulExpImp"

  Set codeModul = ThisWorkbook.VBProject.VBComponents _
      (modulName)
  ThisWorkbook.VBProject.VBComponents.Remove codeModul

  Set codeModul = Nothing
End Sub

'-- Modul-Datei 'modModulExpImp.bas' importieren --
Sub ModulImportieren()
  Dim pfad As String
  Dim modulName As String
```

```
  pfad = ThisWorkbook.Path
  modulName = "modModulExpImp"

  ThisWorkbook.VBProject.VBComponents.Import _
      pfad & "\" & modulName & ".bas"
End Sub

'-- Makro aus Textdatei (*.txt) in ein vorhandenes Modul einlesen --
Sub MakroAusTextDateiImportieren()
  Dim modulName As String
  Dim neuesMakro As CodeModule
  Dim pfad As String
  Dim makroImportDatei As String

  modulName = "modModulMakroHinzufuegen"
  pfad = ThisWorkbook.Path
  makroImportDatei = pfad & "\" & "MakroImportDatei.txt"

  Set neuesMakro = ThisWorkbook.VBProject.VBComponents _
      (modulName).CodeModule
  neuesMakro.AddFromFile makroImportDatei

  Set neuesMakro = Nothing
End Sub

'-- Neues Modul hinzufügen und Makro aus Textdatei einlesen --
Sub NeuesModulMitMakroHinzufuegen()
  Dim neuesModul As Object 'VBComponent
  Dim neuesModulName As String
  Dim pfad As String
  Dim makroImportDatei As String

  neuesModulName = "modNeuesModul"
  pfad = ThisWorkbook.Path
  makroImportDatei = pfad & "\" & "MakroImportDatei.txt"

  Set neuesModul = ThisWorkbook.VBProject.VBComponents.Add _
      (vbext_ct_StdModule)
  neuesModul.Name = neuesModulName
  ThisWorkbook.VBProject.VBComponents(neuesModulName).CodeModule. _
```

```
      AddFromFile (makroImportDatei)

  Set neuesModul = Nothing
End Sub

'-- 'Neues Modul 'modNeuesModul' löschen
Sub NeuesModulLoeschen()
  Dim neuesModul As Object 'VBComponent
  Dim neuesModulName As String

  neuesModulName = "modNeuesModul"

  Set neuesModul = ThisWorkbook.VBProject.VBComponents _
      (neuesModulName)
  ThisWorkbook.VBProject.VBComponents.Remove neuesModul

  Set neuesModul = Nothing
End Sub
```

Hinweis

Im Download-Beispiel werden zur Demo die entsprechenden Codes auf einer Userform angezeigt. Wegen der umfangreichen Programmierung ist hier nur ein Teil des Codes dargestellt.

Für die VBE-Programmierung ist das Einbinden der Objektbibliothek ***Visual Basic For Applications Extensibility 5.3*** notwendig.

Neue Excelmappe mit VBA-Code versehen und speichern

Hallo VBAler,

ich bin mal wieder am Ende meiner Möglichkeiten. Ich möchte in einer Routine ein Worksheet von einer Vorlage neu anlegen, da ich VBA-Module mit darin haben will. Es klappt auch alles wunderbar, nur wenn ich es dann speichern will, verweigert Excel das, weil es die VBA-Module nicht mitspeichern will. Bei dem dann folgenden Excel-Dialog fliegt mein Code dann immer aus der Kurve (es sei denn, ich speichere ohne VBA, aber das ist gerade nicht Sinn der Sache):

Code:

```
...
    If strAuswahl = vbYes Then
            'Set Newbook = Workbooks.Add
```

```
            Workbooks.Add Template:="C:\Users\Alexander\AppData\Roaming\Microsoft\Templates\LLB-Daten Export-Import-Datei.xlt"

            Sheets("Tabelle1").Select
            Range("A1").Select
            ActiveSheet.Paste
            Application.CutCopyMode = False
            Application.DisplayAlerts = False
            For Each blatt In Worksheets
            Select Case blatt.Name
                Case "Tabelle1"
                ' Diese Blätter sollen nicht gelöscht werden!
                Case Else
                blatt.Delete
            End Select
            Next blatt
            Application.DisplayAlerts = True
            ActiveSheet.Name = strLocation & Format(Date, "YYYY-MM-DD")
            strName = Application.GetSaveAsFilename(InitialFileName:=
            "LLB-Daten_Export-Import-Datei" & ".xls",
            FileFilter:="Microsoft Excel-Arbeitsmappe (*.xls), *.xls")
            Select Case strName
                Case False
                ActiveWorkbook.Close SaveChanges:=False
                Dateipfade.Hide
                Sheets("Nebenrechnungen").Visible = False
                Worksheets("Marschtabelle").Select
                Exit Sub
            Case Else
   '----- hier fliege ich dann raus:
                ActiveWorkbook.SaveAs FileName:=strName
                MsgBox "Es wurde die Export-Import-Datei " _
                & vbLf & vbLf
                & "=>    " & ActiveWorkbook.Name
                & vbLf & vbLf
                & "angelegt. Sie befindet sich im Verzeichnis:"
                & vbLf & vbLf
                & ActiveWorkbook.Path
                ActiveWorkbook.Close SaveChanges:=True
            End Select
    Else
...
```

Was mache ich falsch?

Hallöchen,

versuch es so:

Code:

```
ActiveWorkbook.SaveAs Filename:=strName, FileFormat:=xlExcel8
```

_____

LG Isi

Hi Isabelle,
du bist 'ne Wucht: Das war's.

In neues Tabellenblatt VBA-Code einfügen

Hallo zusammen,

ich habe das Problem, das ich neue Worksheets mit Hilfe von VBA erstellen muss und diese auf das Ereignis "SelectionChange(ByVal Target As Range)" reagieren sollen.

Gibt es eine Möglichkeit im VBA-Code den Code bzw. das Ereignis für das neu erstellte Worksheet einzufügen?

Code:

```
Sub Code_einfügen()
    With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.CodeName).CodeModule
        .DeleteLines 1, .CountOfLines
        .InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
        .InsertLines 2, "Msgbox ""Treffer"""
        .InsertLines 3, "End Sub"
    End With
End Sub
```

```
Sub Code_einfügen()
Dim x As Long
Worksheets.Add
With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.Name).CodeModule
For x = .CountOfLines To 1 Step -1
.DeleteLines (x)
Next
.InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
.InsertLines 2, "Msgbox ""Treffer"""
.InsertLines 3, "End Sub"
End With

End Sub
```

Hallo Forum,
ich stehe vor folgendem Problem: Per VBA erstelle ich in meinem Excel-Projekt neue Tabellenblätter. Per Makro füge ich diesen Tabellenblättern einen Button hinzu. Gleichzeitig soll dieser Button auch ein "Click-Event" haben. Dazu generiere ich Code. Allerdings stürzt mir Excel jedes Mal beim Code-generieren ab, mit einer Meldung von wegen "Problembericht an Microsoft senden".

Der Code sieht wie folgt aus:

Code:

```
Sub Code_einfuegen(sheetName As String)
On Error GoTo Err_Code_Einfuegen

    With Application.VBE.ActiveVBProject.VBComponents(Worksheets(sheetName).CodeName).CodeModule
        .DeleteLines 1, .CountOfLines
        .InsertLines 1, "Private Sub CommandButton1_Click"
        .InsertLines 2, "Msgbox ""Treffer"""
        .InsertLines 3, "Workbooks.Open Filename:=strFile"
```

```
        .InsertLines 4, "End Sub"
    End With

Exit Sub

Err_Code_Einfuegen:
    RC = 12
    Cancel = MsgBox("Bei der Erstellung des Codes ist folgender Fehler aufgetreten: " & Err.Description _
    & vbCrLf & vbCrLf & "Der Fehler trat in folgender Funktion auf: " _
    & Chr(34) & "Code_einfuegen" & Chr(34), _
    vbCritical, "Fehler")
    Exit Sub
End Sub
```

IN VISUAL BASIC EINE EXCELDATEI MIT VBA HINTERLEGEN

http://support.microsoft.com/kb/194611/de

1. Erstellen Sie ein Standard-EXE-Projekt in Visual Basic. Form1 wird standardmäßig erstellt.
2. Klicken Sie auf Verweise im Menü Projekt, und überprüfen Sie "Microsoft Visual Basic for Applications-Erweiterbarkeit".
3. Fügen Sie eine Befehlsschaltfläche zu Form1 hinzu.
4. Kopieren Sie und fügen Sie folgenden Code zum Code-Fenster des Formulars:

```
5.        Private Sub Command1_Click()
6.         ' Start Excel
7.         Dim xlapp As Object 'Excel.Application
8.         Set xlapp = CreateObject("Excel.Application")
9.
10.        ' Make it visible...
11.        xlapp.Visible = True
12.
13.        ' Add a new workbook
14.        Dim xlbook As Object 'Excel.Workbook
15.        Set xlbook = xlapp.Workbooks.Add
16.
17.        ' Add a module
18.        Dim xlmodule As Object 'VBComponent
19.        Set xlmodule = xlbook.VBProject.VBComponents.Add(1) 'vbext_ct_StdModule
20.
```

```
        ' Add a macro to the module...
        Dim strCode As String
        strCode = _
           "sub MyMacro()" & vbCr & _
           "   msgbox ""Inside generated macro!!!"" " & vbCr & _
           "end sub"
        xlmodule.CodeModule.AddFromString strCode


        ' Run the new macro!
        xlapp.Run "MyMacro"

        ' ** Create a new toolbar with a button to fire macro...
        ' Add a new toolbar...
        Dim cbs As Object 'CommandBars
        Dim cb As Object 'CommandBar
        Set cbs = xlapp.CommandBars
        Set cb = cbs.Add("MyCommandBar", 1, , True) '1=msoBarTop
        cb.Visible = True

        ' Make it visible & add a button...
        Dim cbc As Object 'CommandBarControl
        Set cbc = cb.Controls.Add(1) '1=msoControlButton

        ' Assign our button to our macro
        cbc.OnAction = "MyMacro"

        ' Set text...
        cbc.Caption = "Call MyMacro()"

        ' Set Face image...
        ' 51 = white hand
        ' 25 = glasses
        ' 34 = ink dipper
        ' etc...
        cbc.FaceId = 51

        ' Pause so you can inspect results...
        MsgBox "All done, click me to continue...", vbMsgBoxSetForeground

        ' Remember to release module
        Set xlmodule = Nothing

        ' Clean up
        xlbook.Saved = True
```

```
66.         xlapp.Quit
67.       End Sub
```

68. Führen Sie die Anwendung. Sie sollten sehen, dass Microsoft Excel starten, gefolgt von der Meldung im Feld "innen generiert Makro!!!." An dieser Stelle werden Sie Code innerhalb des generierten Makros ausgeführt. Klicken Sie auf "OK", um dieses Dialogfeld zu schließen und Sie sollten dann ein Dialogfeld reporting "Auf Alles, was getan werden, klicken Sie mich um den Vorgang fortzusetzen." Lassen Sie dieses, und wechseln Sie zu Excel. Es sollte eine neue Symbolleiste mit einer Schaltfläche mit einem weiß-Handsymbol angezeigt. Der obige Code Visual Basic-diese Schaltfläche, der das Makro MyMacro(), über die OnAction-Eigenschaft zugeordnet ist. Wenn Sie auf diese Schaltfläche klicken, wird MyMacro() aufgerufen. Klicken Sie darauf, um es funktioniert. Klicken Sie auf zurück, das Formular in der Visual Basic, und klicken Sie auf OK im Meldungsfeld "Alles getan, click me weiterhin".
69.

Arbeitsmappe anlegen und Workbook_Open-Prozedur schreiben

Problem: Wie kann ich in XL8 über VBA-Code eine Arbeitsmappe anlegen, im Klassenmodul der Arbeitsmappe und eine Workbook_Open-Prozedur schreiben?

```
StandardModule: basMain

Sub OpenProzedurAnlegen()
   Dim wkb As Workbook
   Dim sFile As String
   Application.ScreenUpdating = False
   sFile = Application.Path & "\testwkb.xls"
   Set wkb = Workbooks.Add(1)
   With wkb.VBProject.VBComponents(wkb.CodeName).CodeModule
      .InsertLines 3, "Private Sub Workbook_Open()"
      .InsertLines 4, "    Msgbox ""Bin jetzt da!"""
      .InsertLines 5, "    ActiveWorkbook.Close Savechanges:=False"
      .InsertLines 6, "End Sub"
   End With
   Application.DisplayAlerts = False
   ActiveWorkbook.SaveAs sFile
   Application.DisplayAlerts = True
   ActiveWorkbook.Close savechanges:=False
   Workbooks.Open sFile
   Application.ScreenUpdating = True
End Sub
```

NEUE ARBEITSMAPPE ERZEUGEN UND CODE HINTERLEGEN

http://www.ms-office-forum.net/forum/showthread.php?t=139216

VBA - Blatt samt VBA-Code generieren

Hallo allerseits,

ich suche nach einer Möglichkeit, nachdem man per VBA ActiveWorkbook.Worksheet.Add einArbeitsblatt generiert hat, soll dieses Blatt mit VBA-Code ausgestattet werden.

Z.B. soll folgender Code im automatisch generierten Blatt automatisch hinterlegt werden:

Code:

```
Private Sub Worksheet_Change(ByVal Target As Range)
If Target.Row Mod 3 = 0 Then
If Not Intersect(Target, Range("B" & Target.Row & ":K" & Target.Row)) Is Nothing Then
Target.Offset(-1, 0).Value = Target.Value + Target.Offset(1, 0).Value
End If
Else
Exit Sub
End If
End Sub
```

Habe ich da eine Chance?

Dank vielmals vorab

Gruß, Otto

jinx
MS-Office-Forum Team

Moin, Otto,

Registrierung: 28.02.2001
Ort: Northern Hemisphere
Beiträge: 30.012
Karma:
Modifikator: 45

vielleicht siehst Du Dir für ein Vorgehen einmal Arbeitsmappe anlegen und Workbook_Open-Prozedur schreiben an - die andere Methode schreibt den Code nicht direkt, sondern importiert eine Textdatei: Module und UserForms austauschen (Klassenmodul Tabelle hat den Wert 100).

_____

cu
jinx

per 31.12.2010 ausgeschiedener User und ehemaliger Excel-MODERATOR

Folgende Tools werden zur optischen Aufbereitung eingesetzt: Code Converter; Excel Jeanie Html
Für die allgemeinen Hinweise: Netiquette; Fragen und Antworten in Foren

eingesetzte Betriebssysteme: XP Home, XP Media Center Edition, XP Professional, Vista Ultimate, 7/Seven
verwendete Programme: Excel97 SR-2, Office 2000 SP-3, Office2002/XP SP-3, Office 2003 Professional SP-3, Office 2007 Home & Student SP-3, Office 2007 Professional SP-3

16.09.2004, 22:49 #3

otto-mueller
Threadstarter
MOF User

Erste Lösung interessant aber leider funktioniert noch nicht...

Hallo Jinx,

vielen Dank für die Links.

Die erste Lösung gefält mir ganz gut. Ich habe es auch natürlich gleich getestet aber es kommt leider imm zwei Fehlermeldungen:

Registrierung: 15.12.2002
Ort: Deutschland
Beiträge: 93
Karma:
Modifikator: 10

Code:

```
Laufzeitfehler: 1004
Der programmatische Zugriff auf das Visual-Basic Projekt ist nicht sicher.
```

Gleich nach dieser Fehlermeldung kommt diese Meldung:

Code:

```
Laufzeitfehler: 1004
Die Methode 'VBProject' für das Object '_Workbook' ist fehlgeschlagen
```

Ich habe den Code fast 1 zu 1 zum Testen in einem Modul übernommen

Code:

```
Sub ProcAnlegen()
Dim wkb As Workbook

Set wkb = Workbooks.Add(1)
Debug.Print wkb.CodeName
With wkb.VBProject.VBComponents(wkb.CodeName).CodeModule
    .InsertLines 3, "Private Sub Workbook_Open()"
    .InsertLines 4, "   Msgbox ""Bin jetzt da!"""
    .InsertLines 5, "   ActiveWorkbook.Close Savechanges:=False"
    .InsertLines 6, "End Sub"
End With

End Sub
```

Mein Excel: Excel 2003, Win2K

Danke nochmal für die Hilfe

17.09.2004, 05:38 #4

jinx
MS-Office-Forum Team

Registrierung: 28.02.2001
Ort: Northern Hemisphere
Beiträge: 30.012
Karma:
Modifikator: 45

Moin, Otoo,

den Code hätte ich vor dem Hinweis testen sollen - bei einer Überprüfung stürzt mir Excel97 bzw. Excel2002 und auf dem Notebook XP und 2003 gnadenlos ab...

Zur Zeit kann ich das Problem nur notieren und mich später darum kümmern - ich brauche den Rechner jetzt, um etwas für einen Kunden vorzubereiten, und ich habe keinen Ansatz, was ich im Code ändern muss (Schulterzucken ist angesagt - leider).

_____

cu
jinx

jinx
MS-Office-Forum Team

Registrierung: 28.02.2001
Ort: Northern Hemisphere
Beiträge: 30.012

Moin, Otto,

`Tschuldigung wegen der "Fehlschreibung". Auch heute noch möchten miene Versionen von Excel nicht in eine andere Mappe einen Code in DieseArbeitsmappe einfügen. Da es Dir aber um einen Code für eine Tabelle ging, könnte vielleicht der folgende Weg (Erstellung in der aktuellen Mappe und anschließendes Kopieren) eine Lösung sein. Dazu bitte in der VBE unter Verweise die *Microsoft Visual Basic Applications Extensibility 5.3* aktivieren:

Code:

Karma:

Modifikator: 45

```
Sub CodeHinterTabelleSchreiben_Otto()
Dim wks As Worksheet
Set wks = Sheets.Add
With ThisWorkbook.VBProject.VBComponents(wks.Name).CodeModule
 .InsertLines 1, _
   "Private Sub Worksheet_Change(ByVal Target as Range)" & Chr(13) & _
   " If Target.Row Mod 3 = 0 Then" & Chr(13) & _
   "  If Not Intersect(Target, Range(""B"" & Target.Row & "":K"" & Target.Row)) Is Nothing Then" & Chr(13) & _
   "   Target.Offset(-1, 0).Value = Target.Value + Target.Offset(1, 0).Value" & Chr(13) & _
   "  End If" & Chr(13) & _
   " Else" & Chr(13) & _
   "  Exit Sub" & Chr(13) & _
   " End If" & Chr(13) & _
   "End Sub"
End With
wks.Move
ActiveSheet.Name = "Sheet" & Worksheets.Count
ActiveWorkbook.SaveAs "Test_" & Format(Now, "YYYYMMDD-hhmmss") & ".xls"
ActiveWorkbook.Close
Set wks = Nothing
End Sub
```

Solltest Du den Code nicht in Zeile 1 der Tabelle beginnen lassen, ersetze bitte die Codezeile wie folgt:

Code:

```
 .InsertLines .CountOfLines + 1, _
```

Und trotzdem erhalte ich ab und zu bei der Überprüfung noch Abstürze - so richtig glücklich macht mich das nicht

---

cu

otto-mueller

Threadstarter

kein Unterschied :-(

MOF User  Registrierung: 15.12.2002 Ort: Deutschland Beiträge: 93 Karma: Modifikator: 10	Hallo Jinx, kein Problem. ;-) ich konnte keinen Unterschied zwischen dem ersten und diesem Code erkennen. Ich habe die Bibliothek Microsoft Visual Basic Applications Extensibility 5.3 auch vorher aktiviert, leider immer noch die selbe Fehlermeldungen, sobald der Debugger in folgende Zeile ankommt: **Code:** `With ThisWorkbook.VBProject.VBComponents(wks.Name).CodeModule` Danke nochmal Gruß Otto

Nepumuk	Hallo Otto, an Stelle von wks.Name einfach wks.CodeName verwenden.

Arbeitsmappe anlegen und Workbook_Open-Prozedur schreiben

Problem: Wie kann ich in XL8 über VBA-Code eine Arbeitsmappe anlegen, im Klassenmodul der Arbeitsmappe und eine Workbook_Open-Prozedur schreiben?

```
StandardModule: basMain

Sub OpenProzedurAnlegen()
   Dim wkb As Workbook
   Dim sFile As String
   Application.ScreenUpdating = False
   sFile = Application.Path & "\testwkb.xls"
   Set wkb = Workbooks.Add(1)
   With wkb.VBProject.VBComponents(wkb.CodeName).CodeModule
      .InsertLines 3, "Private Sub Workbook_Open()"
      .InsertLines 4, "    Msgbox ""Bin jetzt da!"""
      .InsertLines 5, "    ActiveWorkbook.Close Savechanges:=False"
      .InsertLines 6, "End Sub"
   End With
   Application.DisplayAlerts = False
   ActiveWorkbook.SaveAs sFile
   Application.DisplayAlerts = True
   ActiveWorkbook.Close savechanges:=False
   Workbooks.Open sFile
   Application.ScreenUpdating = True
End Sub
```

Modul anlegen per VBA?

von: Erich M.

Geschrieben am: 24.09.2004 09:57:26

Hallo EXCEL-Freunde,

ich habe mit dem Makrorecorder versucht aufzuzeichnen, wie ein neues

Modul mit dem Namen "Muster" angelegt wird.

Das Makro hat allerdings kein Ergebnis angezeigt??

Geht sowas nicht?

Besten Dank für eine Hilfe!

mfg

Erich

Betrifft: AW: Modul anlegen per VBA?

von: K.Rola

Geschrieben am: 24.09.2004 10:30:53

Hallo,

```
Option Explicit
Sub Modul_anlegen()
Dim m As Object
Set m = Application.VBE.ActiveVBProject.VBComponents.Add(1)
m.Name = "Mein_neues_Modul"
End Sub
```

Gruß K.Rola

Betrifft: AW: Modul anlegen per VBA?
von: Worti
Geschrieben am: 24.09.2004 10:38:26

Hallo Erich,
so geht's:

```
Sub ModulEinfügen()

    Dim VBComp As VBComponent

    Set VBComp = ThisWorkbook.VBProject.VBComponents.Add(vbext_ct_StdModule)

    With VBComp.CodeModule

     .InsertLines 1, "Sub MeinModul()"
     .InsertLines 2, "'Durch Code eingefügt"
     .InsertLines 3, "     MsgBox ""Heureka"" "
     .InsertLines 4, "End Sub"

    End With
```

```
    Set VBComp = Nothing

End Sub
```

Worti

Betrifft: AW: Modul anlegen per VBA?
von: WernerB.
Geschrieben am: 24.09.2004 10:39:43

Hallo Erich,

wie gefällt Dir das?

```
Sub StandardModulHinzu()
Dim mdlWB As Object
    Set mdlWB = ThisWorkbook.VBProject.VBComponents.Add(vbext_ct_StdModule)
    Set mdlWB = Nothing
End Sub
```

Vorher in der Entwicklungsumgebung über Menü "EXTRAS / VERWEISE" die "Microsoft Visual Basic for Application Extensibility" aktivieren.

```
Option Explicit

Sub Modul_anlegen()
   Dim m As Object ' VBComponent
   Dim strCode As String

   Set m = Application.VBE.ActiveVBProject.VBComponents.Add(1)
   m.Name = "Mein_neues_Modul"

   strCode = "Sub Hallo()" & vbLf & _
      "   MsgBox ""Hallo""" & vbLf & _
      "End Sub"
   m.CodeModule.AddFromString strCode
End Sub
```

Modul per VBA einfügen

von: Peter Feustel
Geschrieben am: 29.04.2003 - 09:42:12

Hallo Excel Gemeinde,

ich möchte einem aktiven Tabellenblatt („Rg-Formular“) per VBA ein Modul hinzufügen.
Obwohl sowohl die Mappe (ThisWokbook) als auch das Tabellenblatt (AcitveSheet) bekannt sind, die MsgBox zeigt sie mir an, meint der Set Befehl

Laufzeitfehler ‚9'

Index außerhalb des gültigen Bereichs

Was ist an meinem Set-Befehl falsch?

```
Sub Modul_generieren()
Dim CodeModul     As CodeModule

 MsgBox ThisWorkbook.Name & " / " & ActiveSheet.Name

    Set CodeModul = _
    ThisWorkbook.VBProject.VBComponents _
       ("Rg-Formular").CodeModul

    With CodeModul .....
```

Fehlt hier irgendwo ein .Add oder warum mag mich Excel nicht? Es gibt zum Tabellenblatt noch kein Modul!

Danke für jeden guten Tipp im voraus.

Gruß, Peter

Re: Modul per VBA einfügen
von: Nike
Geschrieben am: 29.04.2003 - 09:54:01

Hi,
eine Tabelle hat immer bereits einen eigenen Codebereich,
da muß kein separates Modul für eingefügt werden...

Schau mal hier rein...

Bye

Nike

Re: Modul per VBA einfügen
von: Peter Feustel
Geschrieben am: 29.04.2003 - 11:22:00

Hallo Nike,

danke für den Tipp, aber...

Ich möchte mein Modul dem Tabellenblatt "Rg-Formular" zuordnen, nicht "basMain". Setze ich anstelle basMain "Tabelle1" ein, läuft es in einer Test Mappe einwandfrei.
In meiner Excel-Mappe bekomme ich immer noch Laufzeitfehler '9' und Index außerhalb des gültigen Bereichs. Was mache ich falsch, wo ist mein Denkfehler?
Auch die Erstattung von Rg-Formular durch "basMain" bring bei mir o. a. Fehler.

```
Dim sCode     As String

   sCode = "Private Sub Worksheet_SelectionChange(ByVal Target As Range)" _
         & vbLf & vbLf _
         & "If Target.Row < 30 Or Target.Row > 179 Then Exit Sub" _
         & vbLf & vbLf _
         & "If Target.Column <> 6 Then Exit Sub" _
         & vbLf & vbLf _
         & "If IsNumeric(Cells(Target.Row, 5)) And IsNumeric(Target) Then" _
         & vbLf _
         & "   Cells(Target.Row, 7) = Cells(Target.Row, 5) * Target" _
         & vbLf _
         & "End If" _
         & vbLf & vbLf _
         & "End Sub"
    MsgBox ThisWorkbook.Name & " / " & ActiveSheet.Name
    With ThisWorkbook.VBProject.VBComponents("basMain").CodeModule
       .AddFromString sCode
    End With
```

Gruß, Peter

Re: Nachtrag mit halber Lösung
von: Peter Feustel
Geschrieben am: 29.04.2003 - 11:33:27

Hallo Excels,

zu meinem Problem:

Mein Tabellenblatt heißt "Rg-Formular", ist intern zur Zeit aber "Tabelle5". Setzte ich im obigen Code Tabelle5 ein, läuft er.
Wie bekomme ich immer den aktuellen Tabellex Namen zu meinem Blatt "Rg-Formular" ?

Gruß, Peter

Re: Nachtrag mit halber Lösung
von: Nike
Geschrieben am: 29.04.2003 - 11:59:24

Hi,

hier mal stöbern?

Bye

Nike

Re: danke Nike
von: Peter Feustel
Geschrieben am: 29.04.2003 - 15:52:33

Hallo Nike,

ich bin fündig geworden. ActiveSheet.CodeName liefert den Namen Tabellexx.

Danke für Deine Hilfe.

Gruß, Peter

Modul mittels VBA hinzufügen

von: PeterA

Geschrieben am: 14.08.2003 11:48:51

Hallo !

Kann mir jemand sagen, wie ich ein Modul, dass ich mit VBA eingefügt habe (siehe Code unten) umbenennen kann bzw. einen neuen Namen geben kann. Der Name "Modul1" reicht mir leider nicht.

Vielen Dank

Peter

Workbooks("Code erstellen mittels VBA Quelle.xls").VBProject.VBComponents.Add vbext_ct_StdModule

Betrifft: AW: Modul mittels VBA hinzufügen

von: Nepumuk

Geschrieben am: 14.08.2003 11:54:47

Hallo Peter,

so:

Public

```
Sub test()
    ThisWorkbook.VBProject.VBE.ActiveCodePane.CodeModule.Parent.Name = "MeinModul1"
End Sub
```

Gruß
Nepumuk

Betrifft: AW: Modul mittels VBA hinzufügen
von: PeterA
Geschrieben am: 14.08.2003 12:02:29

Hallo Nepumuk

Ich habe folgendes geändert an deinem Code.

Anstatt ThisWorkbook habe ich einen Namen einer anderen Datei angegeben.
Nach Ausführung deines Codes wird ein Modul mit dem Namen Modul1 eingefügt und der Name der Tabelle1 in Mein Modul umgewandelt.?!? Komisch oder?

Workbooks("Code erstellen mittels VBA Quelle.xls").VBProject.VBE.ActiveCodePane.CodeModule.Parent.Name = "MeinModul"

Hast du noch eine Idee dazu?

Danke Peter

Betrifft: AW: Modul mittels VBA hinzufügen
von: Nepumuk
Geschrieben am: 14.08.2003 12:09:56

Hallo Peter,
Ich habe jetzt nur keine Zeit mehr. Ich melde mich heute Nacht nochmal.
Gruß
Nepumuk

Betrifft: AW: Modul mittels VBA hinzufügen
von: PeterA
Geschrieben am: 14.08.2003 13:03:22

Hallo Nepumuk

Ich hab es jetzt doch noch hinbekommen.

Dim WB_Quelle As Workbook

Set WB_Quelle = Workbooks("Code erstellen mittels VBA Quelle.xls")

On Error GoTo Fehlermeldung

WB_Quelle.VBProject.VBComponents.Add vbext_ct_StdModule
WB_Quelle.VBProject.VBComponents("Modul1").CodeModule.Parent.Name = "Javascript"
WB_Quelle.VBProject.VBComponents("Javascript").CodeModule.InsertLines 4, "
```
Sub Test()"
WB_Quelle.VBProject.VBComponents("Javascript").CodeModule.InsertLines 5, "Msgbox"
WB_Quelle.VBProject.VBComponents("Javascript").CodeModule.InsertLines 6, "End Sub
```

"

Aber eines bleibt ein Problem. Falls "Javascript" schon vorhanden ist dann geb ich eine Fehlernummer aus und bereche das Programm ab. Das Problem ist das ich dann "Modulx" schon angelegt ist. Aber eigentlich nicht in dem Projekt bleiben sollte

Kann man den namen des gerade mit

WB_Quelle.VBProject.VBComponents.Add vbext_ct_StdModule

angelegten Moduls irgendwie temproär speichern?

Vielen dank Peter

Code aus DieserArbeitsmappe in neue Mappe kopieren

AUFTRAGSPROGRAMMIERUNG | EXCEL-CD

- **Code aus DieserArbeitsmappe in neue Mappe kopieren** von Peter W vom 14.05.2005 23:09:14
  - AW: Code aus DieserArbeitsmappe in neue Mappe kopieren - von **Beate Schmitz** *am 14.05.2005 23:44:22*
    - AW: Code aus DieserArbeitsmappe in neue Mappe kopieren - von **Peter W** *am 14.05.2005 23:51:51*
      - AW: Code aus DieserArbeitsmappe in neue Mappe kopieren - von **Beate Schmitz** *am 15.05.2005 00:05:40*
        - AW: Code aus DieserArbeitsmappe in neue Mappe kopieren - von **Peter W** *am 15.05.2005 00:41:49*
  - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Ralf (Schwabenland)** *am 15.05.2005 00:39:40*
    - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Peter W** *am 15.05.2005 00:45:36*
      - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Reinhard** *am 15.05.2005 00:49:29*
      - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Ralf (Schwabenland)** *am 15.05.2005 00:51:31*
        - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Peter W** *am 15.05.2005 01:16:08*
          - Danke für die Rückmeldung. Auch schöne Pfingsten. - von **Ralf (Schwabenland)** *am 15.05.2005 01:20:48*
          - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Reinhard** *am 15.05.2005 01:48:08*
            - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Peter W** *am 15.05.2005 02:11:09*
              - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Peter W** *am 15.05.2005 02:29:51*
                - AW: Code aus DieserArbeitsmappe in neue Mappe kopi - von **Reinhard** *am 15.05.2005 03:12:43*

Betrifft: Code aus DieserArbeitsmappe in neue Mappe kopieren
von: Peter W
Geschrieben am: 14.05.2005 23:09:14

Servus,

hab noch mal ne Frage, ich hab in der Recherche die Möglichkeit gefunden Module in eine neue Mappe zu kopieren. Das klappt soweit auch gut, doch wie kann ich DieseArbeitsmappe(Klassmodul(cls)) in die neue Mappe kopieren, so das auch das Workbook_Open ereignis funzt, mit unten stehenden Code funktionierts leider nicht.

```
Dim sPath As String
Dim sPath2 As String
sPath = Application.DefaultFilePath & "\temp.bas"
sPath2 = Application.DefaultFilePath & "\temp.cls"
ThisWorkbook.VBProject.VBComponents("Modul1").Export sPath
ActiveWorkbook.VBProject.VBComponents.Import sPath
Kill sPath
ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").Export sPath2
ActiveWorkbook.VBProject.VBComponents.Import sPath2
Kill sPath2
```

Vieleicht hat jemand nen Tip für mich, danke.

MfG Peter

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopieren
von: Beate Schmitz
Geschrieben am: 14.05.2005 23:44:22

Hallo Peter,

warum setzt du die Datei mit den Makros nicht als Vorlagendatei (*.xlt) ein? Dann hätte doch jede daraus generierte Datei automatisch alle Makros bei Fuß?

Von Bernd Strohhaecker hatte ich mal nachstehenden Code, der sollte eigentlich alle VBA Komponenten umkopieren. Tipp: Vorher - UNBEDINGT - sichern !

Dateinamen natürlich anpassen. Ich habe das eben mal ausprobiert. Betreffend "dieseArbeitsmappe" wird durch den Code in der Zielmappe Mappe2.xls ein Klassenmodul "dieseArbeitsmappe1" angelegt. Also zusätzlich zur Bestehenden. Aber für dich vielleicht ein Ausgangspunkt zum Ausprobieren.

Gruß,
Beate

--------aus:http://www.excel-center.de/forum/read.php?f=1&i=47565&t=47437#reply_47565 ---

Option Explicit

```
Sub CopyModules(SourceWB As Workbook, TargetWB As Workbook)
Dim strTempFile As String
Dim i As Integer

strTempFile = "c:\temp\xx.vba"
For i = 1 To SourceWB.VBProject.VBComponents.Count
SourceWB.VBProject.VBComponents(i).Export strTempFile
TargetWB.VBProject.VBComponents.Import strTempFile
' Kill strTempFile
Next
End Sub


Sub test()
Call CopyModules(Workbooks("MappemitVBA.xls"), Workbooks("Mappe2.xls"))
End Sub
```

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopieren
von: Peter W
Geschrieben am: 14.05.2005 23:51:51

Servus,

und danke aber das mit DieseArbeitsmappe1 macht mein Code auch und dann funktioniert das Workbook_Open Ereignis eben nicht.
Mit der Vorlage ist keine schlechte Idee, das Problem ist nur das es sich um ein Kassenbuch handelt und der entsprechende Code quasi ein neues Jahr in einer neuen Arbeitsmappe generiert. Ich poste mal den ganzen Code vieleicht wirds dann deutlicher.
Der Code wird über eine selbst erstellte Symbolleiste ausgelöst. @Beate: Danke.

```
Sub neuesjahr()
Dim y As Variant
Dim i%
Dim m As Variant
Dim wrkbk As String
Dim wrkbk2 As String
Dim sPath As String
Dim sPath2 As String
Application.DisplayAlerts = False
Application.ScreenUpdating = False
m = Range("D25")
wrkbk = ActiveWorkbook.name
y = InputBox("Bitte geben Sie das Jahr an:", "Kassenbuch-Jahr")
sPath = Application.DefaultFilePath & "\temp.bas"
sPath2 = Application.DefaultFilePath & "\temp.cls"
Workbooks.Add
wrkbk2 = ActiveWorkbook.name
For i = 1 To 12
With Workbooks(wrkbk).Worksheets("Vorlage")
.Range("D25").Value = i
.Range("E25").Value = y
.Copy before:=Workbooks(wrkbk2).Sheets(1)
End With
Workbooks(wrkbk2).Worksheets("Vorlage").name = "KB" & i
Next
Sheets(Array("Tabelle1", "Tabelle2", "Tabelle3")).Delete
ThisWorkbook.VBProject.VBComponents("Modul1").Export sPath
ActiveWorkbook.VBProject.VBComponents.Import sPath
Kill sPath
ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").Export sPath2
ActiveWorkbook.VBProject.VBComponents.Import sPath2
Kill sPath2
Workbooks(wrkbk2).SaveAs ("Kasse" & y)
Application.DisplayAlerts = True
Application.ScreenUpdating = True
```

```
End Sub
```

MfG Peter

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopieren
von: Beate Schmitz
Geschrieben am: 15.05.2005 00:05:40

Hallo Peter,

ein Kassenbuch wird aber nur einmal jährlich neu angelegt, soweit mir bekannt. Warum machst du dann nicht (natürlich nach Sicherung des alten Jahres) einfach speichern unter...., könntest ja noch ein Makro einfügen, welches den Endbestand zum Anfangsbestand macht und die restlichen Buchungen löscht? Wärst du deine Probleme los und hättest alle Makros erhalten.

Gruß,
Beate

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopieren
von: Peter W
Geschrieben am: 15.05.2005 00:41:49

Servus,

erstmal Danke das du dich um Mitternächtliche Stunde noch meinem Problem annimmst :)
Hast eigentlich recht, werd ich mal so Versuchen, ärgere mich nur , das ich ewig für den zuvor gepostеten Code gebraucht hab. (zwecks suchen, testen, anfänglich nichts funktioniert, usw.)
Interisieren wie das funktioniert das man Klassmodule wie DieseArbeitsmappe lauffähig kopiert würds mich ja schon, da ich in der Recherche auch nichts gefunden habe, vieleicht weiss jemand ja noch was dazu? Danke.

MfG Peter

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopi
von: Ralf (Schwabenland)
Geschrieben am: 15.05.2005 00:39:40

Hallo Peter,

das ist auch aus dem Archiv und erfolgreich bei mir im Einsatz:

http://www.herber.de/bbs/user/22693.xls

Hoffe es hilft.

Gruß
Ralf

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopi
von: Peter W
Geschrieben am: 15.05.2005 00:45:36

Servus,

ich sag jetzt mal Danke?
In deiner Datei war ein Virus (NortonAV), deshalb hab ich leider keinen Code (wurde entfernt).

MfG Peter

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopi
von: Reinhard
Geschrieben am: 15.05.2005 00:49:29

Hallo Peter,
habe keinen Norton, deshalb den Code*g:
in Modul1 (in Diesearbeitsmappe steht nur was von columsautofir bei wb-open):

```
Option Explicit
```

```
Sub copy_code()
Dim i As Integer
Dim scode As String
Dim scode1 As String
Dim myVBComponents As Object
Dim neuesModul As Object
  With ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule
    scode = .Lines(1, .CountOfLines)
  End With
  With ThisWorkbook.VBProject.VBComponents("Modul1").CodeModule
    scode1 = .Lines(1, .CountOfLines)
  End With
Workbooks.Add
  With ActiveWorkbook.VBProject
    For Each myVBComponents In .VBComponents
      Select Case myVBComponents.Type
      Case 1, 2, 3
        .VBComponents.Remove .VBComponents(myVBComponents.Name)
      Case 100
        With myVBComponents.CodeModule
          .DeleteLines 1, .CountOfLines
        End With
      End Select
    Next
  End With
ActiveWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule.AddFromString scode
Set neuesModul = ActiveWorkbook.VBProject.VBComponents.Add(1)
neuesModul.Name = "Modul1"
  With ActiveWorkbook.VBProject.VBComponents("Modul1").CodeModule
    .DeleteLines 1, .CountOfLines
  End With
ActiveWorkbook.VBProject.VBComponents("Modul1").CodeModule.AddFromString scode1
Application.VBE.MainWindow.Visible = False
End Sub
```

Gruß
Reinhard

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopi
von: Ralf (Schwabenland)
Geschrieben am: 15.05.2005 00:51:31

Hallo Peter,

ich sage jetzt mal - sorry, aber mit dem Norton hatte ich auch massive Probleme. Deshalb habe ich ihn rausgeschmissen. Aber das ist ein anderes Thema. Hier der Code (der vom Norton und vom AntiVir angemeckert wurde):

```
Sub copy_code()
Dim i As Integer
Dim scode As String
Dim scode1 As String
Dim myVBComponents As Object
Dim neuesModul As Object
  With ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule
    scode = .Lines(1, .CountOfLines)
  End With
  With ThisWorkbook.VBProject.VBComponents("Modul1").CodeModule
    scode1 = .Lines(1, .CountOfLines)
  End With
Workbooks.Add
  With ActiveWorkbook.VBProject
    For Each myVBComponents In .VBComponents
      Select Case myVBComponents.Type
      Case 1, 2, 3
        .VBComponents.Remove .VBComponents(myVBComponents.Name)
      Case 100
        With myVBComponents.CodeModule
          .DeleteLines 1, .CountOfLines
        End With
      End Select
    Next
  End With
ActiveWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule.AddFromString scode
Set neuesModul = ActiveWorkbook.VBProject.VBComponents.Add(1)
neuesModul.Name = "Modul1"
  With ActiveWorkbook.VBProject.VBComponents("Modul1").CodeModule
    .DeleteLines 1, .CountOfLines
  End With
ActiveWorkbook.VBProject.VBComponents("Modul1").CodeModule.AddFromString scode1
Application.VBE.MainWindow.Visible = False
End Sub
```

Gruß
Ralf

Betrifft: AW: Code aus DieserArbeitsmappe in neue Mappe kopi
von: Peter W
Geschrieben am: 15.05.2005 01:16:08

Servus,

@Ralf: brauchst dich nicht entschuldigen wahr eigentlich klar das das an NortonAV liegt :) , Code funzt wunderbar, hab Ihn jetzt angepasst, einfach ge.. :) Danke
@Reinhardt: Danke für die Übersetzung ;)
@alle: poste den fertigen Code jetzt nochmal für´s Archiv, wenn Ihr NortonAV benutzt, werdet Ihr zwar ne Viruswarnung bekommen, warum keine Ahnung? Ist aber kein Virus.

```
Sub neuesjahr()
Dim y As Variant
Dim i%
Dim m As Variant
Dim wrkbk As String
Dim wrkbk2 As String
Dim sPath As String
Dim sPath2 As String
Application.DisplayAlerts = False
Application.ScreenUpdating = False
m = Range("D25")
wrkbk = ActiveWorkbook.name
y = InputBox("Bitte geben Sie das Jahr an:", "Kassenbuch-Jahr")
sPath = Application.DefaultFilePath & "\temp.bas"
Workbooks.Add
wrkbk2 = ActiveWorkbook.name
For i = 1 To 12
With Workbooks(wrkbk).Worksheets("Vorlage")
.Range("D25").Value = i
.Range("E25").Value = y
.Copy before:=Workbooks(wrkbk2).Sheets(1)
End With
Workbooks(wrkbk2).Worksheets("Vorlage").name = "KB" & i
Next
Sheets(Array("Tabelle1", "Tabelle2", "Tabelle3")).Delete
ThisWorkbook.VBProject.VBComponents("Modul1").Export sPath
ActiveWorkbook.VBProject.VBComponents.Import sPath
```

```
Kill sPath
With ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule
    sPath2 = .Lines(1, .CountOfLines)
End With
ActiveWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule.AddFromString sPath2
Workbooks(wrkbk2).SaveAs ("Kasse" & y)
Application.DisplayAlerts = True
Application.ScreenUpdating = True
End Sub
```

Module und UserForms austauschen

Problem: Wie kann ich alle Module und UserForms einer Arbeitsmappe löschen und danach die Module und UserForms des aktuellen Unterverzeichnisses laden?

```
StandardModule: basMain

Sub DeleteAndImport()
   Dim col As New Collection
   Dim vbc As VBComponent
   Dim iCounter As Integer
   Dim sPath As String
   sPath = Application.Path & "\"
   For Each vbc In ThisWorkbook.VBProject.VBComponents
      If vbc.Type <> 3 And _
         vbc.Type < 100 And _
         vbc.Name <> "basMain" Then
         vbc.Export sPath & vbc.Name & ".bas"
         col.Add sPath & vbc.Name & ".bas"
         ThisWorkbook.VBProject.VBComponents.Remove vbc
      ElseIf vbc.Type = 3 Then
         vbc.Export sPath & vbc.Name & ".frm"
         col.Add sPath & vbc.Name & ".frm"
         ThisWorkbook.VBProject.VBComponents.Remove vbc
      End If
   Next vbc
   For iCounter = 1 To col.Count
      ThisWorkbook.VBProject.VBComponents.Import col(iCounter)
      Kill col(iCounter)
   Next iCounter
   MsgBox "Ex- und Import abgeschlossen!"
End Sub

ClassModule: Tabelle1

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
```

```
End Sub
```

VBA-Code generieren

VBA-Code generieren

Hallo zusammen,

ich habe das Problem, das ich neue Worksheets mit Hilfe von VBA erstellen muss und diese auf das Ereignis "SelectionChange(ByVal Target As Range)" reagieren sollen.

Gibt es eine Möglichkeit im VBA-Code den Code bzw. das Ereignis für das neu erstellte Worksheet einzufügen?

Min Schlupp,

versuchs mal hiermit:

```
Sub Code_einfügen()
Dim x As Long
Worksheets.Add
With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.Name).CodeModule
For x = .CountOfLines To 1 Step -1
.DeleteLines (x)
Next
.InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
.InsertLines 2, "Msgbox ""Treffer"""""
.InsertLines 3, "End Sub"
End With

End Sub
```

Danke,

der Code hat mir schon viel geholfen, jetzt brauch ich nur noch die Umsetzung des VBComponenten-Namens auf den Namen des aktuellen Sheets, da in der Komponente der Name der Tabelle gespeichert ist (z.B. "Tabelle1") und nicht "newitem", wie das Sheet heißt. Habe das per Debugger herausgefunden, aber leider kein Attribut gefunden, welches den angezeigten Namen enthält.

Hast Du noch irgendeine Idee?

Zitat:

```
Sub Code_einfügen()
Dim x As Long
Worksheets.Add
With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.Name).CodeModule
For x = .CountOfLines To 1 Step -1
.DeleteLines (x)
Next
.InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
.InsertLines 2, "Msgbox ""Treffer"""
.InsertLines 3, "End Sub"
End With

End Sub
```

Moin, Schlupp,

wozu - das verstehe ich nicht. Den Namen der Tabelle kann man jeder User ohne Probleme ändern...

@Marc:
Extrem vorschtig, bei einem neuen Tabellenblatt erst den vorhandenen Code zu löschen... 

_____

cu

Hallo Schlupp,
versuch es mal so:
Code:

```
Sub Code_einfügen()
    With Application.VBE.ActiveVBProject.VBComponents(ActiveSheet.CodeName).CodeModule
        .DeleteLines 1, .CountOfLines
        .InsertLines 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"
        .InsertLines 2, "Msgbox ""Treffer"""
        .InsertLines 3, "End Sub"
    End With
End Sub
```

Moin, Du siehst aber auch jede Kleinigkeit!
Habe den Code natürlich aus einem Projekt von mir übernommen - da war es notwendig - und somit wieder einmal ein Opfer der eigenen Faulheit!
Hey,

danke euch, der letzte Code ist genau das, was mir noch gefehlt hat.

thx an alle Beteiligten!

VBA-CODE EINFÜGEN (einzeln oder aus Datei)

Hallo Andy,

es geht auf diverse Arten, z. B. kannst du das Makro in eine Text-Datei schreiben, die du dann einliest und in das Tabellenblatt einfügst z. B. so:

With ThisWorkbook.VBProject.VBComponents(ActiveSheet.CodeName).CodeModule

.AddFromFile ImportDatei

End With

oder auch direkt aus einem Makro, wie im Beispiel:

Option Explicit

```
Sub WB_Code_via_VBA_Kapitel()

Const WS      As String = "Kapitel"

Dim VBC       As Object
Dim LineNr    As Integer

   With ThisWorkbook.VBProject.VBComponents(Worksheets(WS).CodeName).CodeModule

      LineNr = .CreateEventProc("Change", "Worksheet")
         .InsertLines LineNr + 1, " "
         .InsertLines LineNr + 2, "Dim r    As Long"
         .InsertLines LineNr + 3, "Dim wsB  As Worksheet"
         .InsertLines LineNr + 4, "Dim wsC  As Worksheet"
         .InsertLines LineNr + 5, " "
         .InsertLines LineNr + 6, "   If Target.Column = 7 And Target.Row > 1 Then"
         .InsertLines LineNr + 7, "      If Target.Count = 1 Then"
         .InsertLines LineNr + 8, "         Set wsB = Sheets(""Lotus Notes"")"
         .InsertLines LineNr + 9, "         Set wsC = Sheets(""Binf neu (2)"")"
         .InsertLines LineNr + 10, "         r = Target.Row - 1"
         .InsertLines LineNr + 11, "         Application.EnableEvents = False"
         .InsertLines LineNr + 12, "         On Error GoTo ERRH"
         .InsertLines LineNr + 13, "         Range(Cells(r, 1), Cells(r, 6)).Copy Range(Cells(r + 1, 1), Cells(r + 1, 6))"
         .InsertLines LineNr + 14, "         wsB.Range(wsB.Cells(r, 1), wsB.Cells(r, 6)).Copy wsB.Range(wsB.Cells(r + 1, 1), wsB.Cells(r + 1, 6))"
         .InsertLines LineNr + 15, "         wsC.Range(wsC.Cells(r, 1), wsC.Cells(r, 9)).Copy wsC.Range(wsC.Cells(r + 1, 1), wsC.Cells(r + 1, 9))"
         .InsertLines LineNr + 16, "         Range(Cells(r, 8), Cells(r, 9)).Copy Range(Cells(r + 1, 8), Cells(r + 1, 9))"
         .InsertLines LineNr + 17, "         wsB.Range(wsB.Cells(r, 7), wsB.Cells(r, 12)).Copy wsB.Range(wsB.Cells(r + 1, 7), wsB.Cells(r + 1, 12))"
         .InsertLines LineNr + 18, "         wsC.Range(wsC.Cells(r, 22), wsC.Cells(r, 26)).Copy wsC.Range(wsC.Cells(r + 1, 22), wsC.Cells(r + 1, 26))"
         .InsertLines LineNr + 19, "         Set wsB = Nothing"
         .InsertLines LineNr + 20, "         Set wsC = Nothing"
         .InsertLines LineNr + 21, "      End If"
         .InsertLines LineNr + 22, "   End If"
         .InsertLines LineNr + 23, " "
         .InsertLines LineNr + 24, "ERRH:"
         .InsertLines LineNr + 25, " "
         .InsertLines LineNr + 26, "   Application.EnableEvents = True"

   End With

End
Sub
```

Viele Grüße Peter

Eine kurze Nachricht, ob es läuft, wäre nett - danke.

Betrifft: AW: VBA Code per Makro in Arbeitsblatt einfügen

von: Kurt

Geschrieben am: 21.08.2006 21:29:08

Hi,

du hast vergessen zu erwähnen, dass ein Verweis auf die VB6EXT.OLB erforderlich ist.

mfg Kurt

Betrifft: AW: VBA Code per Makro in Arbeitsblatt einfügen

von: Andy

Geschrieben am: 22.08.2006 08:43:29

Oh super! das klappt wie geschmiert :-)
So hab ich mir das vorgestellt :-)

Achso, einen Hinweis noch für diejenigen die das auch brauchen..

Falls ein Laufzeitfehler 1004 vorkommt, einfach bei Extra, Makro, dann Sicherheit und dann Vertrauenswürdige Quellen das Häkchen bei "Zugriff auf Visual Basic Projekte vertrauen" setzen..

VBA code durch VBA ändern

AUFTRAGSPROGRAMMIERUNG | EXCEL-CD

VBA code durch VBA ändern 2

von: Chris b
Geschrieben am: 20.01.2004 07:30:13

VBA Code durch VBA ändern.

Hallo Excel Profis,
habe eine frage und zwar weiß ich das ich mit VBA den VB Code ändern kann dazu habe ich diesen unten stehenden Code:
Probelm ist aber dieser Code ändert nur das VBA Projekt in dem er auch steht wegen dem Activecodepane glaube ich.
Ist es Möglich wenn z.B der code der geändert werden soll im Modul1 steht und der VBA code der das ändern aufruft im Modul2 das da auch geht?
Für eure Hilfe wäre ich euch sehr dankbar Christian

'Code soll im Modul1 sein (Momentan sind beide Codes im Modul1)
Const SuchZeile = " MsgBox ""VBA macht Spaß !"""
Const NeueZeile = " MsgBox ""VBA macht großen Spaß !"""

```
Sub VBAZeileÄndern()

    Set VBE = Application.VBE.ActiveCodePane.CodeModule

    With VBE
        For x = 1 To .countoflines
         i = .countoflines
            If .Lines(x, 1) = NeueZeile Then
```

```
                .ReplaceLine x, SuchZeile
                Exit Sub
            End If
            If .Lines(x, 1) = SuchZeile Then
                .ReplaceLine x, NeueZeile
                Exit Sub
            End If
        Next x
    End With
End Sub
```

```
'Code Soll im Modul 2 geändert werden (Momentan sind beide Codes im Modul1)
Sub Testen()
    MsgBox "VBA macht Spaß !"
End Sub
```

Betrifft: AW: VBA code durch VBA ändern
von: Detlev
Geschrieben am: 20.01.2004 08:19:48

Hallo Chris,

ich hab Deinen Code etwas geändert, danach lief es bei mir.

Modul 1

```
Const SuchZeile = "MsgBox ""VBA macht Spaß !"""
Const NeueZeile = "MsgBox ""VBA macht großen Spaß !"""
Sub VBAZeileÄndern()

    Set VBE = Application.VBE.VBProjects(2).VBComponents.Item(4).CodeModule

    With VBE
        For x = 1 To .countoflines
         i = .countoflines
            If Trim(.Lines(x, 1)) = NeueZeile Then
                .ReplaceLine x, SuchZeile
                Exit Sub
            End If
            If Trim(.Lines(x, 1)) = SuchZeile Then
                .ReplaceLine x, NeueZeile
                Exit Sub
            End If
        Next x
    End With
End Sub
```

Modul 2

```
Sub Testen()
MsgBox "VBA macht Spaß !"
End Sub
```

Gruss aus Hannover
Detlev

www.Bieler.org

Betrifft: leider nicht das was ich wollte
von: chris b
Geschrieben am: 20.01.2004 08:32:49

Is leider nicht das was ich wollte.
dacht mir ich kann irgendwo angeben modul2 oder so.Also hier meine ich

Set VBE = Application.VBE.ActiveCodePane.CodeModule
Set VBE = Application.VBE.Modul2.CodeModule

Betrifft: AW: Hinbekommen
von: chris b
Geschrieben am: 20.01.2004 09:18:41

Habs jetzt doch noch hinbekommen.
folgende CodeZeile geändert !
Set VBE = Application.VBE.VBProjects(2).VBComponents.Item(4).CodeModule
in ->
Set VBE = Application.VBE.VBProjects(1).VBComponents.Item("Modul2").CodeModule

Betrifft: AW: Hinbekommen
von: Detlev
Geschrieben am: 20.01.2004 10:32:24

Gratz,

auf die Anführungszeichen hätte ich auch kommen können!

Gruss
Detlev

www.Bieler.org

VBA Code löschen in verschiedenen Bereichen

There are times when you may find it useful to delete your VBA. First you will need to select your project in the Visual Basic Editor, then from the top menu, select Tools, References, then tick Microsoft Visual Basic for Applications Extensibility 5.3. Now you are ready to run one of the below examples of deletion code. (Be careful doing this, okay?)

Deleting all code in the Active Workbook

```
Sub DeleteAllVBACode()
    Dim oVBComp As Object

    For Each oVBComp In ActiveWorkbook.VBProject.VBComponents

        If oVBComp.Type = 100 Then
            With oVBComp.CodeModule
                .DeleteLines 1, .CountOfLines
            End With
        Else
            ActiveWorkbook.VBProject.VBComponents.Remove oVBComp
        End If

    Next oVBComp

End Sub
```

Deleting a module in the Active Workbook

```
Sub DeleteSingleCodeModule()
    Dim oVBComp As Object

    For Each oVBComp In ActiveWorkbook.VBProject.VBComponents

        If oVBComp.Name = "Module1" Then
            ActiveWorkbook.VBProject.VBComponents.Remove oVBComp
        End If

    Next oVBComp
End Sub
```

Deleting ThisWorkbook code in the Active Workbook

```
Sub DeleteThisWorkbookCode()

    With ActiveWorkbook.VBProject.VBComponents("ThisWorkbook").CodeModule
        .DeleteLines 1, .CountOfLines
    End With

End Sub
```

Deleting a Worksheet's code in the Active Workbook

```
Sub DeleteSheetCode()
    Dim ws As Worksheet

    With ActiveWorkbook
        For Each ws In .Worksheets
            If ws.Name = "Sheet1" Then
                With .VBProject.VBComponents(ws.CodeName).CodeModule
                    .DeleteLines 1, .CountOfLines
                End With
            End If
        Next ws
    End With
End Sub
```

Note that the code to delete a single module will also work with class modules and userforms as well as normal code modules. Night!

VBA-Code der aktiven Exceldatei entfernen

Nach Rückfrage jeden VBA-Code sowie Module, Klassenmodule und UserFormen der **aktiven** Exceldatei entfernen.

```
Sub VBA_Code_entfernen()
Dim Ding As Object
Dim Zeile As Long
Dim Antwort As Integer

Antwort = MsgBox("Wollen Sie wirklich alle VBA-Elemente in:" _
        & vbNewLine & ActiveWorkbook.Name & vbNewLine _
        & "komplett löschen?", vbOKCancel, _
        "Achtung!")

If Antwort <> 1 Then
  MsgBox "Keine Änderungen durchgeführt", , ""
  Exit Sub
End If
```

```
  For Each Ding In ActiveWorkbook.VBProject.vbcomponents

   'Type 100 = DieseArbeitsmappe und alle Tabellen
    If Ding.Type = 100 Then

      With ActiveWorkbook.VBProject.vbcomponents(Ding.Name).CodeModule
        For Zeile = 1 To .CountOfLines
          .DeleteLines 1
        Next Zeile
      End With

   'Type 1 = Modul, Type 2 = Klassenmodul, Type 3 = UserForm
    Else
      ActiveWorkbook.VBProject.vbcomponents.Remove Ding
    End If

  Next

MsgBox "Fertig", , ""
End Sub
```

VBA-Code löschen

Hallo,

ich möchte den **Code** eines *Worksheets* löschen. Geht das? Falls ja - wie?

Hintergrund: Ich kopiere ein Register mit Buttons und Grafiken in eine neue Mappe. Alle *anderen* Register der neuen Mappe werden gelöscht (neue Mappe enthält damit genau *ein* Register). Die Buttons werden gelöscht. Die neue Mappe wird gespeichert. So weit, so gut - die neue Mappe enthält (optisch) keine Schalter mehr.

Die CommandButtons (aus der Symbolleiste *Steuerelemente*) sind in der ursprünglichen Mappe erforderlich. Ihre Funktionalität wird in der neuen Mappe nicht mehr benötigt. Die neue Mappe ist eine Art "Archiv", das nur noch gelesen werden muss. (Die Option, ein pdf zu drucken, ist zwar schön - einem pdf kann *ich* allerdings keinen Pfad etc. zuweisen, also bleibt's bei Excel und VBA. 41 kB (mit Makros) bzw. 34 kB (ohne Makros) Dateigröße kann ich verkraften.)

Wenn ich den Code des Worksheets per Hand lösche, wird die Datei kleiner - und es liegen keine "toten" Makros herum. Je nach Einstellung bleibt außerdem die "Makros-Aktivieren"-Meldung aus.

Danke im Voraus,
Frederik

Hallo Frederik,
wäre DAS nicht was für Dich?

*Der Code ist mit Vorsicht zu behandeln, da er in einer Mappe auch **sich selbst** wegputzt. Das überrascht mich ein wenig, denn es erscheint die MsgBox "Fertig" (siehe link), obwohl es eigentlich kein Modul mehr gibt..? Faszinierend! Einzelschrittmodus is' nich - naja. Zum Testen am besten in eine **neue, leere Mappe** kopieren, da kann man nichts kaputt machen.*

Diesen Teil habe ich übernommen:

Code:

```
Private Sub VBA_Code_entfernen(objWB As Workbook)
'löscht VBA-Code der Mappe objWB
'Quelle: http://vba1.de/vba/080vbe_clean.php
Dim obj As Object
Dim lgLine As Long

'Debug.Print objWB.FullName

    With objWB.VBProject.vbcomponents
        For Each obj In objWB.VBProject.vbcomponents
          'Type 100 = DieseArbeitsmappe und alle Tabellen
          If obj.Type = 100 Then
              With objWB.VBProject.vbcomponents(obj.Name).CodeModule
                  For lgLine = 1 To .CountOfLines
                      .DeleteLines 1
                  Next lgLine
              End With
          End If
        Next
    End With

End Sub
```

Danke, Danke, Danke - es läuft bei mir soweit und erfüllt seinen Zweck. Du weißt nicht *zufällig*, was genau es mit "Type 100" auf sich hat? Die Hilfe ist etwas dünn

WORD/POWERPOINT/ANDERE EXTERNE PRG

Alle Windows-Fenster minimieren

```
' Alle Programmfenster minimieren
    Set objShell = CreateObject("Shell.Application")
    objShell.ToggleDesktop
```

Batch-Datei starten

Variante 0, die bei mir klappte

```
' Wichtig - erst als ich nachfolgende beide Befehle CHDRIVE und CHDIR eingab, klappt der Aufruf der Batchdatei

VONPFAD= "L:\bh10\22700"
ChDrive Left(VONPFAD, 1)
ChDir (VONPFAD)

Shell "L:\bh10\22700\AUSMISTN.bat"
```

Variante 1

The /c closes the DOS prompt when finished.

```
Private Sub Workbook_Open()

Call Shell(Environ$("COMSPEC") & " /c C:\Path.bat", vbNormalFocus)

End Sub
```

Variante 2

```
Shell "C:\My Files\Junk\MyBat.bat"
```

Variante 3

```
x = SHELL("command /c c:\xyz.bat", 1)
```

Variante 4

```
Private Sub Run_Prog_Bttn_Click()
    Dim Test As Double
    Dim Response As VbMsgBoxResult
    Dim sYourCommand As String
    sYourCommand = "C:\1.bat"
    Test = Shell("cmd /c " & sYourCommand, vbNormalFocus)
    If Test = 0 Then
        Response = MsgBox("The Specified Batch File Does Not Exist.", vbOKOnly, "File Error")
        Exit Sub
    ElseIf Test > 0 Then
        Response = MsgBox("Successful call to Batch File.", vbOKOnly, "File Run Report")
    End If
End Sub
```

Variante 5

```
x = Shell("cmd.exe /c C:\x.bat", 1)
```

Externes Programm starten + anzeigen

das folgende klappt - wobei, wenn Programm schon gestartet, dann wird es aktiviert, sonst wird es gestartet

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Sub FORMEL_F_STARTEN()

   Dim FENSTER
   Dim ANZEIGE
   Dim FENSTERNAME
   Dim PROGRAMMNAME

   ANZEIGE = 1
   FENSTERNAME = "Formel-f"
   PROGRAMMNAME = "H:\Programme\FormelF\FormelF.exe"
```

```
    FENSTER = FindWindow(vbNullString, FENSTERNAME)
    If (FENSTER = 0) Then
          ' Programm starten
        Shell PROGRAMMNAME, vbMaximizedFocus
    Else
        'FENSTER AKTIVIEREN
        AppActivate FENSTERNAME
    End If

End Sub

Sub Start_Programm()
' erst prüfen, ob Prg schon aktiv (mit Namen des Prg's in Taskleiste bzw Titelleiste)
If Tasks.Exists("Formel-f") = False Then
' wenn nicht, dann starten
    Shell " H:\Programme\FormelF\FormelF.exe"
    Else
' sonst aktivieren
    Tasks("Formel-f").Activate
End If
Tasks("Formel-f ").WindowState = wdWindowStateNormal
End Sub
```

IngGi
20.11.2008, 10:48
Hallo nerif,

möglicherweise lässt sich das Problem auch an der Wurzel packen und dadurch einfacher lösen. Ich gehe mal davon aus, dass du ein externes Programm von Excel-VBA aus per Shell startest, also

```
Shell("LW:\Pfad\Programmname.exe")
```

Das Problem besteht dann darin, dass VBA nicht wartet, bis das Programm ausgeführt ist, sondern sofort selbst weiter ausgeführt wird. Das lässt sich verhindern, wenn man das externe Programm statt dessen folgendermaßen startet:

```
CreateObject("WScript.Shell").Run "LW:\Pfad\Programmname.exe", 1, True ' oder Null statt 1 siehe unten
```

Damit wartet VBA, bis das externe Programm fertig ist und fährt erst dann selbst fort.

Gruß Ingolf

gibt es auch eine möglichkeit die shell auszublenden sone art visible(false) ?^^

IngGi
20.11.2008, 13:39
Hallo nerif,

setze dafür den vorletzten Parameter auf 0 an Stelle von 1.

--- Access-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallAccess
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einer Access-Datenbank aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Name der Access-Datenbank an String-Variable übergeben
    - Wenn die Datei nicht existiert...
    - Negativmeldung
    - Sonst...
    - Acess-Instanz bilden
    - Access-Datenbank öffnen
    - Access-Makro aufrufen
    - Access-Instanz schließen
    - Objektvariable zurücksetzen
- Code:

```
Sub CallAccess()
   Dim accApp As Object
```

```
   Dim sFile As String
   ' Pfad, wenn die Access-MDB im gleichen Verzeichnis wie die XLS-Datei liegt
   sFile = ThisWorkbook.Path & "\vb07_AccessTest.mdb"
   If Dir(sFile) = "" Then
      Beep
      MsgBox "Access-Datenbank wurde nicht gefunden!"
   Else
      With CreateObject("Access.Application")
          .OpenCurrentDatabase sFile
          .Run "AcMeldung"
          .CloseCurrentDatabase
      End With
   End If
End Sub
```

--- POWERPOINT ---

Tabellenbereich nach Powerpoint exportieren

Exportiert einen definierten Bereich in eine definierte Powerpoint Präsentation auf eine bestimmte Folie

```
Sub Excel_Range_an_PPT()
    Dim ppApp As Object
    Dim ppFile As Object
    Dim ppPres As String
    Dim picObj As Object, picName As String
    'Dateiname
    ppPres = "C:\Demo.ppt"
    'Object referenzieren
    Set ppApp = CreateObject("Powerpoint.Application")
    'Bereich kopieren
    Worksheets("Tabelle1").Range("A1:E6").Copy
    'Object initialisieren
    ppApp.Visible = msoTrue
    'PPT öffnen
    Set ppFile = ppApp.Presentations.Open(ppPres)
    'Foliennummer angeben
    ppApp.ActivePresentation.Slides(1).Select
    'Bereich einfügen und OLE Verknüpfung herstellen = Link
    With ppApp.ActiveWindow
        .ViewType = ppViewSlide
        .View.PasteSpecial DataType:=ppPasteDefault, link:=msoTrue
```

```
    End With
    'Eingefügte Tabelle skalieren
    With ppApp.ActiveWindow.Selection.ShapeRange
        'Oberer Rand 1 cm unter Standardtitel
        .Top = 150
        'Linker Rand 1.5 cm von linkem Folienrand
        .Left = 35
        'Eingefügte Tabelle auf Links und rechts 1,5 cm Rand skalieren
        .Width = 650
        'Bei Bedarf Höhe noch einstellen
        'Hier ist jedoch zu beachten, dass das Object skaliert wird !!!
        'Die Breite verändert sich dann
        '.Height = 300
    End With
End Sub
```

Diagramm in existierende Präsentation einfügen

Ein Chart in eine Powerpoint Präsentation exportieren und einfügen

```
Sub Excel_Chart_an_PPT()
    Dim ppApp As Object
    Dim ppFile As Object
    Dim ppPres As String
    'Dateiname
    ppPres = "C:\Demo.ppt"
    'Object referenzieren
    Set ppApp = CreateObject("Powerpoint.Application")
    'Diagramm kopieren : Name bitte anpassen
    ActiveSheet.ChartObjects("Diagramm 1").Chart.ChartArea.Copy
    'Object initialisieren
    ppApp.Visible = msoTrue
    'PPT öffnen
    Set ppFile = ppApp.Presentations.Open(ppPres)
    'Foliennummer angeben
    ppApp.ActivePresentation.Slides(1).Select
    'Bereich einfügen und OLE Verknüpfung herstellen = Link
    With ppApp.ActiveWindow
        .ViewType = ppViewSlide
        .View.PasteSpecial DataType:=ppPasteOLEObject, link:=msoTrue
    End With
```

```
    'Eingefügtes Diagramm skalieren
    With ppApp.ActiveWindow.Selection.ShapeRange
        'Oberer Rand 1 cm unter Standardtitel
        .Top = 150
        'Linker Rand 1.5 cm von linkem Folienrand
        .Left = 35
        'Eingefügte Tabelle auf Links und rechts 1,5 cm Rand skalieren
        .Width = 650
        'Bei Bedarf Höhe noch einstellen
        'Hier ist jedoch zu beachten, dass das Object skaliert wird !!!
        'Die Breite verändert sich dann
        '.Height = 300
    End With
End Sub
```

Diagramme exportieren von Excel nach PowerPoint

```
Private Sub cmdExport_Click()
Dim pApp As PowerPoint.Application
Dim pPres As PowerPoint.Presentation
Dim pSlide As PowerPoint.Slide
Dim i As Integer
Dim wks As Worksheet

Application.ScreenUpdating = False

    With Application.FileDialog(msoFileDialogOpen)

        .Filters.Clear
        .Filters.Add "Powerpoint-Präsentation", "*.ppt"
        .AllowMultiSelect = False
        .Title = "PowerPoint wählen"
        .ButtonName = "Exportieren"

        If .Show = -1 Then

            Set pApp = New PowerPoint.Application
            pApp.Visible = True
            Set pPres = pApp.Presentations.Open(.SelectedItems(1))
            i = 1

            For Each wks In Worksheets

                If IsNumeric(wks.Name) = False And wks.Name <> Me.Name Then
```

```
                    Set pSlide = pPres.Slides.Add(1 + (i * 2), ppLayoutTitleOnly)
                    wks.ChartObjects(1).Chart.CopyPicture
                    pSlide.Shapes.Paste '39.62, 101.12, 714.5, 406.38 (LTWH)
                    wks.Range("A1:E5").CopyPicture
                    pSlide.Paste '39.62, 10.12, 714.5, 406.38 (LTWH)
                    pSlide.Shapes(1).TextFrame.TextRange.Characters.Text = wks.Name
                    With pSlide.Shapes(2)
                    .LockAspectRatio = False
                        .Height = 405
                        .Width = 715
                        .Left = 40
                        .Top = 100
                    End With
                    i = i + 1

                End If

            Next wks

            pPres.SaveAs ActiveWorkbook.Path & "\" & Format(Date, "yyyymmdd") & "_Auswertung_Webservicenutzung.ppt"

        End If

    End With

Application.ScreenUpdating = True

End Sub
```

Hier einige Infos zum Code:

Code:

```
    With Application.FileDialog(msoFileDialogOpen)

        .Filters.Clear
        .Filters.Add "Powerpoint-Präsentation", "*.ppt"
        .AllowMultiSelect = False
        .Title = "PowerPoint wählen"
        .ButtonName = "Exportieren"

        If .Show = -1 Then
```

Bei dem part kann man eine Powerpoint datei wählen, in die das ganze integriert wird. Du kannst aber auch einfach den Part weglassen und im nächsten Part eine Powerpoint-Datei fest vorgeben:

Code:

```
            Set pApp = New PowerPoint.Application
            pApp.Visible = True
            Set pPres = pApp.Presentations.Open(.SelectedItems(1))
```

Hier wird die Powerpoint-Datei geladen. Wenn du ".Open(.SelectedItems(1))" ersetzt durch ".Add msoTrue" erzeugst du eine neue Powerpoint-Datei. Diese kannst du dann über pPres im Code ansprechen!

Code:

```
            For Each wks In Worksheets

                If IsNumeric(wks.Name) = False And wks.Name <> Me.Name Then

                        '.....................


                End If

            Next wks
```

hier werden alle deine Worksheets durchlaufen und wenn es nicht das aktuelle ist, wird es exportiert (Ich habe den Ausruf aus einem Worksheet herausgemacht. Dieses sollte nicht exportiert werden). Erfolgt dein Ausruf über ein modul oder eine Userform, dann kannst das "wks.Name <> Me.Name" raus. "IsNumeric(wks.Name) = False" habe ich nur benutzt, da meine Worksheets mit Ziffern beschrieben waren die später auch für den Folienverweis genutzt wurden. Kannst du auch rausnehmen.

Code:

```
Set pSlide = pPres.Slides.Add(1 + (i * 2), ppLayoutTitleOnly)
```

So fügst du eine Slide zur Präsentation hinzu." i+ (i*2)" musst du du durch die Stelle an der die Slide eingefügt werden soll ersetzen. Bei mir sollte das eben nach dem Titel jede zweite Folie sein (Ich hatt ebereits Folien in der Präsi und konnte dementsprechend beim Einfügen Indizes überspringen.) ppLayoutTitleOnly ist eine konstante, die das Format der Slide angibt.

Code:

```
wks.ChartObjects(1).Chart.CopyPicture
                    pSlide.Shapes.Paste '39.62, 101.12, 714.5, 406.38 (LTWH)
                    wks.Range("A1:E5").CopyPicture
                    pSlide.Paste '39.62, 10.12, 714.5, 406.38 (LTWH)
```

Hier kopiere ich ein Diagramm und eine Range als Bild und füge sie in die Slide ein.

Code:

```
                    pSlide.Shapes(1).TextFrame.TextRange.Characters.Text = wks.Name
                    With pSlide.Shapes(2)
                    .LockAspectRatio = False
                        .Height = 405
                        .Width = 715
                        .Left = 40
                        .Top = 100
                    End With
                    i = i + 1
```

An dieser Stelle greife ich auf das erste Shape (Der Titel) zu und ändere den Inhalt. Shape(2) ist in meinem Fall das Diagramm gewesen. An dieser Stelle positioniere und skaliere ich es.!

Code:

```
            pPres.SaveAs ActiveWorkbook.Path & "\" & Format(Date, "yyyymmdd") & " Auswertung Webservicenutzung.ppt"
```

Hier wird die präsi gespeichert. Der name setzt sich bei mir zusammen aus dem Excel-Dateinamen, dem Datum und einem festen String.

Zuletzt noch eine Variante des Codes

```
Set pSlide = pPres.Slides.Add(0 + i, ppLayoutTitleOnly)
                    wks.ChartObjects(1).Chart.CopyPicture
                    pSlide.Shapes.Paste '9.62, 1.12, 4.5, 6.38 (LTWH)
                    With pSlide.Shapes(5)
                        .LockAspectRatio = True
                        .Height = 280
                        .Width = 550
                        .Left = 50
                        .Top = 140
                    End With
```

```
wks.Range("A1:d4").CopyPicture
pSlide.Shapes.Paste '39.62, 10.12, 714.5, 406.38 (LTWH)
With pSlide.Shapes(6)
    .LockAspectRatio = True
    .Height = 280
    .Width = 350
    .Left = 50
    .Top = 20
End With
```

Diagramme in eine Powerpoint-Vorlage einfügen

```
Sub ChartObjectsNachPowerpoint()
Dim pptApp As Object, pptPres As Object
Dim chtObj As Object, shp As Object, i
Set pptApp = CreateObject("PowerPoint.Application")

' neue Präsentation erzeugen
' Set pptPres = pptApp.Presentations.Add(msoTrue)

' oder bestehende Vorlage öffnen
Set pptPres = pptApp.Presentations.Open "D:\test\präs.ppt"

For Each chtObj In ActiveSheet.ChartObjects
chtObj.Chart.ChartArea.Copy
i = i + 1
Set pptslide = pptPres.Slides.Add(i, 12) '12 = ppLayoutBlank
Set shp = pptslide.Shapes.Paste
shp.Top = 0
shp.Left = 0
shp.Width = 400
shp.Height = 400
Next
pptApp.Visible = True
End Sub
```

Jemand schrieb:

ich will auch noch kurz was sagen ;)

damit das mit dem Presentations.Open funktioniert, muss das ppt-Object sichtbar sein, also muss pptApp.Visible weiter nach oben im Code.

```
Set pptApp = CreateObject("PowerPoint.Application")
pptApp.Visible = True
Set pptPres = pptApp.Presentations.Open("d:\test.ppt")
```

Diagramme (mehrere pro Exceltabellenblatt) nach Powerpoint

bei nachfolgendem Code muss die Powerpoint-Datei bereits geöffnet sein!!!

```
Sub ChartObjectsNachPowerpoint()

Dim pptApp As Object, pptPres As Object
Dim chtObj As Object, shp As Object, i

Set pptApp = CreateObject("PowerPoint.Application")
Set pptPres = pptApp.Presentations.Open("c:\Dokumente und Einstellungen\User\Desktop\Vorlage.ppt")
'Set pptPres = pptApp.Presentations.Add(msoTrue) 'erzeugt neue .ppt
iter = 1
While iter <= Sheets.Count
Sheets(iter).Select
For Each chtObj In ActiveSheet.ChartObjects
    chtObj.CopyPicture xlScreen, xlBitmap
    i = i + 1
    Set pptSlide = pptPres.Slides.Add(i, 12) ' neues Blatt in ppt, 12 ist ohne Textfelder
    Set shp = pptSlide.Shapes.Paste
    shp.Top = 0
    shp.Left = 0
    shp.Height = 530
    shp.ScaleWidth 1.23, msoFalse, msoScaleFromTopLeft
Next
iter = iter + 1
Wend
pptApp.Visible = True
End Sub
```

Ein User schreibt dazu

ich habe es jetzt in excel 2007 probiert...

es wird nicht beschnitten und die beschriftung ist auch da

leider gibt es noch ein kleines aber ich habe die von stefan genannte variante mit tabellen benennen genommen. allerdings zeigt er mit nur die diagramme einer der (als test) genannten drei arbeitsblätter an. insgesammt hab ich 20 arbeitsblätter mit je 4 grafiken.

ich hänge mal meinen code (zu 95% von schokomonster) hier rein, vielleicht kann mir jemand damit helfen:

Code:

```
Sub ChartObjectsNachPowerpoint()

Dim pptApp As Object, pptPres As Object
Dim chtObj As Object, shp As Object, i

Set pptApp = CreateObject("PowerPoint.Application")
Set pptPres = pptApp.Presentations.Open("d:\test.ppt")
'Set pptPres = pptApp.Presentations.Add(msoTrue) 'erzeugt neue .ppt
arrSheets = Array("Dia.name1", "Dia.name2", "Dia.name3")
For i = 1 To UBound(arrSheets)
For Each chtObj In Sheets(arrSheets(i)).ChartObjects
    chtObj.CopyPicture xlScreen, xlBitmap
    i = i + 1
    Set pptSlide = pptPres.Slides.Add(i, 12) ' neues Blatt in ppt, 12 ist ohne Textfelder
    Set shp = pptSlide.Shapes.Paste
    shp.Top = 100
    shp.Left = 30
    shp.Height = 330
    shp.ScaleWidth 1.11, msoFalse, msoScaleFromTopLeft
Next
Next
pptApp.Visible = True
End Sub
```

Antwort:

Arg, da sind ja noch einige Probleme. *grrrr*

1. Einfügen hinter den bereits bestehenden Folien:

Code:

```
ii = pptPres.Slides.Count  ' zählt wieviele Folien schon da sind
For Each chtObj In ActiveSheet.ChartObjects
    'chtObj.CopyPicture xlPrinter, xlBitmap
```

```
    chtObj.CopyPicture xlScreen, xlBitmap
    i = i + 1
    Set pptSlide = pptPres.Slides.Add(i + ii, 12)
```

2. Schmeiß mal diese beiden Zeilen heraus:

Code:

```
shp.Height = 530
    shp.ScaleWidth 1.23, msoFalse, msoScaleFromTopLeft
```

Mit etwas Glück siehst dann zumindest die linke Seite der Diagramme wieder. Zumindest hoffe ich, dass das Diagramm zu groß gezogen wurde.

3. Das mit den über 30 leeren Diagrammen ist echt merkwürdig!
Ich würde fast behaupten wollen, dass da bei deiner Datei noch irgend welche Diagramm-Leichen im Keller liegen. Magst dir nicht mal kurz eine Spieldatei basteln, damit du zum testen des Makros nicht gleich mit ganz so vielen Seiten und Diagrammen hantieren musst? Dann kann zumindest mein Verdacht ausgeschlossen werden.

Wäre doch gelacht wenn wir das Makro nicht so hin biegen können wie es gebraucht wird.
Stück Schoki als Nervennahrung rüber reich

Gruß Schokomonster

edit: du hast bei beiden for-Schleifen die Variable i. Das muss Probleme geben. Bennene eines in ii, oder so.
Und diese Variable musst dann auch hoch zählen, genau wie das i = i + 1

REAKTION: Schokomonster du bist super!!! es läuft!

Folgewunsch:

jetzt noch eine frage dazu..... 

ist es auch möglich einem bestehenden diagramm in der excel-file eine bestimmte seite der existierenden powerpoint-datei zuzuweisen?

als beispiel: diagramm 1 (in excel sind die ja benannt) soll auf seite 4, diagramm 2 soll auf seite 4, etc. ?

Antwort:

Wenn die einzelnen Blätter in ppt schon existieren, brauchst nur eine Zeile ändern.

Code:

```
Set pptSlide = pptPres.Slides.Add(i, 12)
'''''''''''' änderst in ''''''''''''''
Set pptSlide = pptPres.Slides(i + 6)
```

Dann sollte Diagramm 1 im bestehenden Blatt 7 landen, D.2 in Blatt 8, ...

Reaktion des Fragestellers des Folgewunsches:

ja cool! ihr seit super!!!

das ist genau das was ich brauche. eine zeile, kurz und knackig ;)

ich dachte anfangs, dass ich jetzt für jedes einzelne diagramm eine einzelne codezeile schreiben müsste.
(dann wäre man zwar flexibler welches diagramm wo genau hin soll)

... aber sollte ja auch zeit und nutzen betrachten ;)

ich denke mit dieser lösung werden mein chef und ich uns einig werden.

ansonsten weiß ich ja wo es die kompetentesten lösungen gibt

vielen dank nochmal an schokomonster und stefan!!!

Jemand andere hatte folgende Lösung:

Hi Sub-Zero,

Ich hätte hier auch noch einen Vorschlag.

Code:

```
Sub ChartObjectsNachPowerpoint()

    Dim pptApp As Object, pptPres As Object, pptSlide As Object
    Dim shp As Object
```

```
    Dim shListe
    Dim chObjListe
    Dim chPosListe
    Dim vTmp
    Dim chObj As ChartObject
    Dim iX As Integer
    Dim iCX As Integer
    Dim iSX As Integer
    Dim bFlipped As Boolean

    Set pptApp = CreateObject("PowerPoint.Application")
    pptApp.Visible = True
    Set pptPres = pptApp.Presentations.Open("d:\Report.ppt")
    'oeffnet bestehende .ppt-Vorlage
    Sheets("Overview").Range("B2:K20").CopyPicture xlScreen, xlBitmap
    iSX = 4
    Set pptSlide = pptPres.Slides(iSX) 'Overview auf Seite 4
    Set shp = pptSlide.Shapes.Paste
    shp.Top = 80 'Ausrichtung: 80 pixel von oben
    shp.Left = 130 'Ausrichtung: 130 pixel von links
    shp.Height = 330
    shp.ScaleWidth 1.18, msoFalse, msoScaleFromTopLeft  'Skalierung von 118%

    shListe = Array("Dia.Cust&Segm", "Dia.Cust&Segm (2)")

    For iX = 0 To UBound(shListe)
        iCX = Worksheets(shListe(iX)).ChartObjects.Count - 1
        ReDim chObjListe(iCX)
        ReDim chPosListe(iCX)
        iCX = 0
        For Each chObj In Worksheets(shListe(iX)).ChartObjects
            chObjListe(iCX) = chObj.Name
            chPosListe(iCX) = Format(chObj.TopLeftCell.Row, "00000") & Format(chObj.TopLeftCell.Column,
"00000")
            iCX = iCX + 1
        Next chObj
        Do
            bFlipped = False
            For iCX = 0 To UBound(chObjListe) - 1
                If chPosListe(iCX) > chPosListe(iCX + 1) Then
                    bFlipped = True
                    vTmp = chPosListe(iCX + 1)
                    chPosListe(iCX + 1) = chPosListe(iCX)
                    chPosListe(iCX) = vTmp
                    vTmp = chObjListe(iCX + 1)
                    chObjListe(iCX + 1) = chObjListe(iCX)
                    chObjListe(iCX) = vTmp
```

```
                End If
            Next iCX
        Loop While bFlipped = True

        For iCX = 0 To UBound(chObjListe)
            Worksheets(shListe(iX)).ChartObjects(chObjListe(iCX)).CopyPicture xlScreen, xlBitmap
            iSX = iSX + 1
            Set pptSlide = pptPres.Slides(iSX)
            Set shp = pptSlide.Shapes.Paste
            shp.Top = 100 'Ausrichtung: 100 pixel von oben
            shp.Left = 25 'Ausrichtung: 25 pixel von links
            shp.Height = 330
            shp.ScaleWidth 1.1, msoFalse, msoScaleFromTopLeft 'Skalierung von 110%
        Next iCX
    Next iX
    pptApp.Visible = True
End Sub
```

Der Code sortiert die Charts automatisch in der Reihenfolge von Links-Nach-Rechts und von Oben-Nach-Unten. Die Anzahl Charts wird automatisch festgelegt.
(Natürlich ungetestet)

Gruess Hansueli

Seine zweite Version für einen anderen User

```
Sub ChartObjectsNachPowerpoint()

    Dim pptApp As Object, pptPres As Object, pptSlide As Object
    Dim shp As Object
    Dim shListe
    Dim iX As Integer
    Dim iSX As Integer

    Set pptApp = CreateObject("PowerPoint.Application")
    pptApp.Visible = True
    Set pptPres = pptApp.Presentations.Open("D:\Excelstuff\DiaVorlage.ppt")
    'oeffnet bestehende .ppt-Vorlage
    shListe = Array("Sheet1", "Sheet2", "Sheet3")
    For iX = 0 To UBound(shListe)
        Worksheets(shListe(iX)).ChartObjects(1).CopyPicture xlScreen, xlBitmap
        iSX = iSX + 1
        If pptPres.Slides.Count < iSX Then
                Set pptSlide = pptPres.Slides.Add(iSX, 12)
```

```
        Else
            Set pptSlide = pptPres.Slides(iSX)
        End If
        Set shp = pptSlide.Shapes.Paste
        shp.Top = 100 'Ausrichtung: 100 pixel von oben
        shp.Left = 25 'Ausrichtung: 25 pixel von links
        shp.Height = 330
        shp.ScaleWidth 1.1, msoFalse, msoScaleFromTopLeft 'Skalierung von 110%
    Next iX
    pptApp.Visible = True
End Sub
```

Sheet-Array und PPT.Vorlage anpassen (Vorlage braucht nur 1 Slide, die restlichen werden je nach Anzahl dazugefügt). Es wird davon ausgegangen, das jedes Blatt im Array ein ChartObject (Diagramm) hat !

Diagramme exportieren nach Powerpoint Seibersdorf 1

Hier ist mein Code für die IAEA.

Zuerst die beiden wichtigsten Problemstellen

Problemstelle 1: Charts sind abgeschnitten (kommt nur in 2003 vor, nicht Powerpoint 2010)

ich schrieb dazu an den IAEA Mitarbeiter:

wenn ich den Code, den du in PsWin bereits einsetzt, verwende, werden die Charts immer abgeschnitten und die Legende ist nicht vollständig (siehe "Präsentation mit PsWin-Code.ppt" im Anhang). Natürlich fand ich im Internet 20 verschiedene Code-Schnippsel für den Chart-Transfer - aber da diese alle mit Diagrammen arbeiten, die in Tabellenblätter eingefügt sind und nicht mit Diagrammen wie sie bei euch sind (also als eigene Sheets), darum war auch deren Ergebnis immer zu vergessen. Die Lösung fand ich dann rein zufällig: das eingefügte-Chart-Sheet ist kein klassisches Excel-Chart, sondern muss als ein gruppiertes Shape behandelt werden und wenn man den Befehl Shape.Selection.Ungroup darauf anwendet, dann wird das Chart korrekt dargestellt. Hier der Code vom Einfügen des Diagramms und dem vollständig anzeigen:

```
' pptSlide.Shapes.Paste
 pptSlide.Shapes.Paste.Select
```

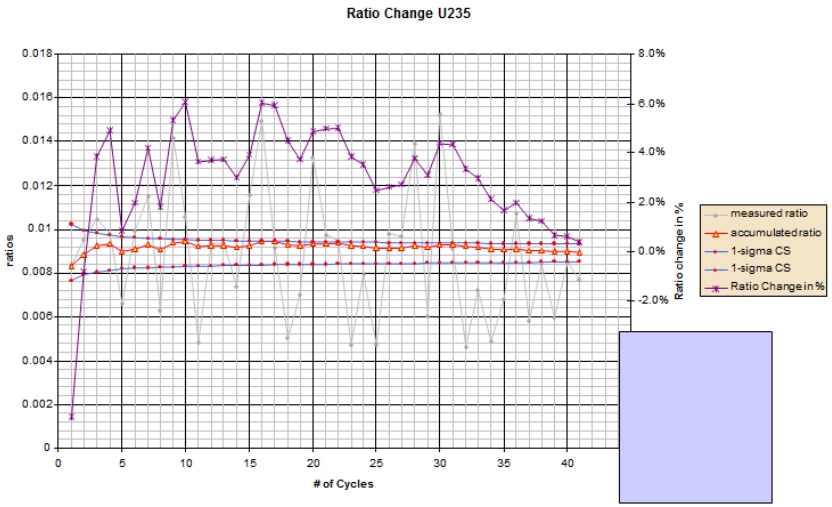
Lösung manuell (bzw durch untenstehenden Ungroup-Befehl):

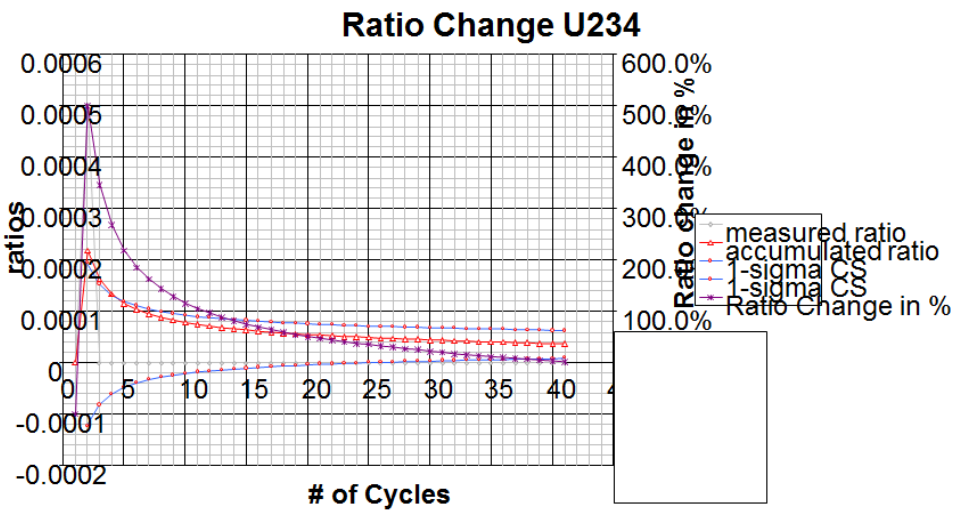

```
' Optimizing chart which often is not seen as whole - by converting it to picture
pptApp.ActiveWindow.Selection.ShapeRange.Ungroup.Select
```

Warum schrieb ich "converting it to picture": ich fand den code zufällig, als ich in Powerpoint mit dem Makrorecoder und rechten Mausklick auf "Bild bearbeiten" ging. Powerpoint erkannte automatisch, dass es sich um ein gruppiertes Shape handelt und fragte, ob es das Objekt in ein Zeichenobjekt umwandeln dürfe. Wenn ich da auf JA klickte, wurde der Ungroup-Befehl generiert*, der meine Lösung darstellte.

(* Unter 2010 gibts ja keinen Makrorecorder mehr in Powerpoint 2010 – aber in 2007 gibts ihn noch und man kann dorthin ausweichen)

Achtung: damit lief der Code sehr gut unter Powerpoint 2003 – jedoch nicht mehr so gut unter Powerpoint 2010. Dort wird nämlich durch den Ungroup-Befehl das Bild wieder rückgewandelt in das original Chart und wenn man dann das Chart verkleinern möchte (um zB 4 Charts auf eine Folie zu pasten), dann hat man das Problem, dass die Schriftgröße des Textes nicht mit verkleinert wird:





Ergebnisse in Powerpoint 2010 nach dem Resizen (verkleinern): Links ohne Ungroup-Befehl und rechts mit Ungroup-Befehl

Problemstelle 2: Die Fehlermeldung (unkown member): Invalid request. To select a shape its view must be active

Siehe auch den Punkt THE ‚TO SELECT A SHAPE, ITS VIEW MUST BE ACTIVE"-ERROR hier bei meinen Powerpoint-Unterkapiteln !!!

Ursache: ich fügte per VBA jeweils ein neues Slide ein - wodurch es auch zum aktuellen Slide (Folie) wird, aber noch nicht automatisch angezeigt wird - und fügte danach immer auch den Befehl ein, dieses neue Slide anzuzeigen.

Nur beim allerersten Slide-Einfügen hatte ich das nicht gemacht. Beim ersten Durchlauf der Schleife führte das auch zu keinem Fehler, denn bei einer neuen, leeren Präsentation wird durch das Erzeugen eines neuen Slildes dieses auch immer auch angezeigt.

Aber beim zweiten Durchlauf, bestand schon eine Präsentation und ein neues Slide wurde zwar erzeugt, aber wegem dem fehlenden Anzeigen-Befehl war es nicht sichtbar. Zu Problemen führt das erst, wenn man in dieser neuen Folie was SELECTEN will.

Daher die golden rule: immer nach dem Erzeugen eines neuen Slides, dieses auch anzeigen, damit man darin selektieren kann.

```
' VBA-Code für Erzeugen
Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutBlank) ' dies erzeugt ein neues Slide und aktiviert es
' VBA-Code für Anzeigen
pptApp.ActiveWindow.View.GotoSlide Index:=pptSlideNr ' dies zeigt das neue und bereits aktive Slide nun auch an
```

PS: diese Fehlermeldung kommt übrigens auch, wenn man in der falschen Powerpointansicht ist ! Nur ppViewSlide oder ppViewNormal gehen:

```
' VBA-Code für ppViewSlide
pptApp.ActiveWindow.ViewType = ppViewSlide

' ****************************************
' ----------------------------------------
' defining public varibales
' ----------------------------------------
' ****************************************

    ' Filevariables

'    Public File_PsWin As String ' Filename of PsWin

    ' Variables for / in MS Powerpoint

'    Public PPTtemplate As String ' Path and filename of the Word-template
    Public pptApp As PowerPoint.Application ' the Powerpoint-Application
    Public pptPres As PowerPoint.Presentation ' the actual open Powerpoint-Presentation
    Public pptSlide As PowerPoint.Slide ' the actual Powerpoint-Slide
    Public pptSlideNr As Integer ' the Number of the actual Powerpoint-Slide
    Public pptFile As String ' the filename of the actual Powerpoint-Presentation
    Public pptShapeRange  ' the shape of the chart in Powerpoint

    ' Public MyShape As Shape
    ' Public TextTransfer As String ' for transferring text like sample-number to Powerpoint-Report
    ' Public Y_Axis_Maximum As Double ' for setting maximum of y-axis in charts - especially to reset it to original value after removing the
backup-data

' ----------------------------------------------

' ****************************************
```

```
' ------------------------------------
 Sub Print_Charts_on_PPT()
' ------------------------------------
' ************************************

' This is the procedure that is called-up by pressing the PRINT-CHARTS-ON-PPT-Button in Sheet PPT-REPORTS
' important: the reference to "Microsoft PowerPoint x.x Object Library" must be set in the VBA-environment

' -------------------
' Defining variables
' -------------------

   Dim Line_Nr As Integer ' counter for all to be reported particles in sheet Report
   Dim File_PsWin As String ' active PsWIN-Excel-File-Name
   Dim File_MP As String ' name of the MP_File

' -------------------
' Setting variables
' -------------------

   File_PsWin = ActiveWorkbook.Name
   TEXT_TRANSFER = Worksheets("Report").Range("C6") ' sample-number

' defining the variable with full path and filename for the Powerpoint-Template (actually not needed)

'    PPTtemplate = Sheets("").Range("") & "\" & Sheets("").Range("") ' f.e.: C:\Analysis IAEA\Reports\PPT_template.ppt

' -------------------------
' opening Powerpoint
' -------------------------

   Set pptApp = CreateObject("Powerpoint.Application") ' starts Powerpoint
   pptApp.Visible = True
   Set pptPres = pptApp.Presentations.Add
   pptApp.ActiveWindow.ViewType = ppViewSlide

' -------------------------
' loop through all particles
' -------------------------
```

```
' all particles in sheet "Report"

For Line_Nr = 21 To 1000

    ' activating PsWin-Excel-file
    AppActivate (File_PsWin)
    Workbooks(File_PsWin).Sheets("Report").Activate

    ' checking if there is another particle to be reported
    If Sheets("Report").Cells(Line_Nr, 3) = "" Then Exit For ' when no more particle is left, leave the loop

    ' marking the first particle
    Sheets("Report").Cells(Line_Nr, 4).Select

    ' ---------------------------------------------------
    ' "pressing" the Open MP-File-Button Commandbutton29
    ' ---------------------------------------------------
    Dim x As Variant
    Dim y As Variant
    x = ActiveCell.Row
    y = ActiveCell.Column
    If Worksheets("Toolbox").AllwaysopenMP_file_with_all_charts Then AllwaysopenMP_file_with_all_charts_flag = True
    Dim ID As Variant
    Dim series As Variant
    Dim answer As Variant
    ID = Cells(x, 3)
    If y = 4 Or y = 5 Then Specimen = "U234"
    If y = 6 Or y = 7 Then Specimen = "U235"
    If y = 8 Or y = 9 Then Specimen = "U236"
    filen = Cells(x, 251)
    If x <= 20 Or Cells(x, 251) = "" Then
    MsgBox ("Please select a cell with a MP entry to open the respecive File!" & Chr(13) & Chr(13) & "Chose: Part.ID Row for Results page" &
Chr(13) & "Chose an isotope of interest to display this isotopes properties only" & Chr(13) & "Chose any other Row to display all charts")
    Exit Sub
    End If
    Application.ScreenUpdating = False
    open_a_MP_file_and_add_charts_if_necessary (filen)
    namepswin = ActiveWorkbook.Name
```

```
    If contains(namepswin, "PsWIN") Then
    MsgBox ("File could not be opened, check Defaultpath in Report Sheet!")
    Exit Sub
    End If
    If y = 3 Then mode_of_display = "results"
    answer = Display_only_needed_charts(Specimen, mode_of_display)
    Add_a_button_in_results_chart
    If AllwaysopenMP_file_with_all_charts_flag = True Then
         i = Charts.Count
        For x = 1 To i
          Charts(x).Visible = True
        Next x
    End If
    SortWorksheetsByColorAndName
    ' ----------------------------------------------------
    ' end of code pressing Commandbutton29
    ' ----------------------------------------------------

' saving the MP-Filename

    File_MP = ActiveWorkbook.Name

' -------------------------
' Transferring first chart - if activated in Sheets PPT_Reports
' -------------------------

    If Workbooks(File_PsWin).Worksheets("PPT_Reports").Countrates_Charts = True Then

        ' increasing the Slide-Nr.:
        pptSlideNr = pptSlideNr + 1

        ' copying chart
        ActiveWorkbook.Sheets("Countrates_Chart").Activate
        ActiveChart.ChartArea.Select
        ' alternative 1: copy as Picture
        ActiveChart.CopyPicture xlScreen
        ' alternative 2: copy as full chart (bis size)
        ' ActiveChart.ChartArea.Copy
```

```
        ' Adding new slide
        Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutBlank)
        pptApp.Visible = True
        pptApp.Activate

        ' pptApp.ActiveWindow.View.GotoSlide Index:=pptSlideNr

        ' Pasting Chart Version 1
        ' SendKeys ("^v")

        ' Pasting Chart Version 2
        ' pptSlide.Shapes.Paste
        pptSlide.Shapes.Paste.Select

        ' Optimizing chart which often is not seen as whole - by converting it to picture
        pptApp.ActiveWindow.Selection.ShapeRange.Ungroup.Select

        ' sizing and positioning the chart
        Set pptShapeRange = pptApp.ActiveWindow.Selection.ShapeRange
        With pptShapeRange
            .Fill.Transparency = 0#
            .Left = 0#
            .Top = 30#
            .Height = 442.12
            .Width = 710#
        End With

    End If


' -------------------------
' Transferring second chart - if activated in Sheets PPT_Reports
' -------------------------

    If Workbooks(File_PsWin).Worksheets("PPT_Reports").Atom_Chart = True Then

        ' increasing the Slide-Nr.:
        pptSlideNr = pptSlideNr + 1
```

```
    ' copying chart
    ActiveWorkbook.Sheets("Atom Percent Chart").Activate
    ActiveChart.ChartArea.Select
    ' alternative 1: copy as Picture
    ActiveChart.CopyPicture xlScreen
    ' alternative 2: copy as full chart (bis size)
    ' ActiveChart.ChartArea.Copy

    ' Adding new slide
    Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutBlank)
    pptApp.Visible = True
    pptApp.Activate

    pptApp.ActiveWindow.View.GotoSlide Index:=pptSlideNr

    ' Pasting Chart Version 1
    ' SendKeys ("^v")

    ' Pasting Chart Version 2
    ' pptSlide.Shapes.Paste
    pptSlide.Shapes.Paste.Select

    ' Optimizing chart which often is not seen as whole - by converting it to picture
    pptApp.ActiveWindow.Selection.ShapeRange.Ungroup.Select

    ' sizing and positioning the chart
    Set pptShapeRange = pptApp.ActiveWindow.Selection.ShapeRange
    With pptShapeRange
       .Fill.Transparency = 0#
       .Left = 0#
       .Top = 30#
       .Height = 442.12
       .Width = 710#
    End With

End If
```

```
End

Next Line_Nr ' End of the loop trhough all particles


    ActiveWorkbook.Close

End Sub
```

Diagramme exportieren nach Powerpoint Seibersdorf 2 mehrere auf 1 Chart

```
Sub Print_Multiple_Charts_on_PPT()


' This is the procedure that is called-up by pressing the PRINT-MULTIBLE-CHARTS-ON-ONE-PPT-SLIDE-Button in Sheet PPT-REPORTS
' important: the reference to "Microsoft PowerPoint x.x Object Library" must be set in the VBA-environment

' --------------------
' Defining variables
' --------------------

   Dim Line_Nr As Integer ' counter for all to be reported particles in sheet Report
   Dim Particle_Nr As Integer ' number of particles in sheet report (for loops from 1 to x)
   Dim File_PsWin As String ' active PsWIN-Excel-File-Name
   Dim File_MP(100) As String  ' names of the MP_Files (max 100)
   Dim File_PP As String ' name of the PPT-File
   Dim PP_has_started As Integer ' remember the first start of Powerpoint
   Dim File_CO As String ' name of the CO-File
   Dim Slide_Pos As Integer ' as up to 4 charts are pasted into a Powerpointslide, Slide_Pos remembers where the last of these 4 charts has
been pasted: 1=upper left, 2= upper right, 3= bottom left, 4=bottom right
   Dim Pos_Left As Single ' contains the X-Axis of the upper left corner of the to be inserted chart for the chart-positioning
   Dim Pos_Top As Single ' contains the Y-Axis of the upper left corner of the to be inserted chart for the chart-positioning
   Dim Selected_Nr As Integer ' number of selected charts by user (max 4 are allowed)
   Dim Default_Label As String ' The User can choose, wether he wants the default label (Sample-Nr and particle ID) pasted to PPT or not - if
so, this contains the Sample-Nr and Particle-ID
   Dim User_Label As String ' The User can add and activate an user-label in Shets PPT_Reports to be pasted in the title of each slide
```

Dim ch As Chart ' for renaming the second data-series in CO-File (when the CO-Charts are transferred to Powerpoint for each particle, the particle is highlighted / marked pink as it is the only element in the second data-series - and its particle-number is also becoming the name of the second data-serie)

```
' --------------------
' Setting variables
' --------------------

    File_PsWin = ActiveWorkbook.Name
    TEXT_TRANSFER = Worksheets("Report").Range("C6") ' sample-number
    If Sheets("PPT_Reports").addUserLabel = True Then User_Label = Sheets("PPT_Reports").UserLabelForCharts

    ' resetting ppt_slide_nr
    pptSlideNr = 0


' defining the variable with full path and filename for the Powerpoint-Template (actually not needed)

'    PPTtemplate = Sheets("").Range("") & "\" & Sheets("").Range("") ' f.e.: C:\Analysis IAEA\Reports\PPT_template.ppt

' --- Checking number of selected charts (not more than 4 are allowed)

    Selected_Nr = 0
    If Sheets("PPT_Reports").CheckBox23 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox24 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox25 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox26 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox27 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox28 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox31 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox34 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox17 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox18 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox19 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox29 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox30 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox20 = True Then Selected_Nr = Selected_Nr + 1
    If Sheets("PPT_Reports").CheckBox21 = True Then Selected_Nr = Selected_Nr + 1
```

```
    If Sheets("PPT_Reports").CheckBox22 = True Then Selected_Nr = Selected_Nr + 1

    If Selected_Nr > 4 Then
        MsgBox "You've selected more than 4 charts - Please unselect " & Selected_Nr - 4 & " chart."
        End
    End If

' ----------------------------------------------------
' opening MP-Files of all particles in sheet "report"
' ----------------------------------------------------

Line_Nr = 21
Particle_Nr = 0

While Workbooks(File_PsWin).Sheets("Report").Cells(Line_Nr, 3) <> ""

    Particle_Nr = Particle_Nr + 1

    ' activating PsWin-Excel-file
    AppActivate (File_PsWin)
    Workbooks(File_PsWin).Sheets("Report").Activate

    ' marking the particle
    Sheets("Report").Cells(Line_Nr, 4).Select

    ' ----------------------------------------------------
    ' "pressing" the Open MP-File-Button Commandbutton29
    ' ----------------------------------------------------
    Dim x As Variant
    Dim y As Variant
    x = ActiveCell.Row
    y = ActiveCell.Column
    If Worksheets("Toolbox").AllwaysopenMP_file_with_all_charts Then AllwaysopenMP_file_with_all_charts_flag = True
    Dim ID As Variant
    Dim series As Variant
    Dim answer As Variant
    ID = Cells(x, 3)
    If y = 4 Or y = 5 Then Specimen = "U234"
    If y = 6 Or y = 7 Then Specimen = "U235"
```

```
    If y = 8 Or y = 9 Then Specimen = "U236"
    filen = Cells(x, 251)
    If x <= 20 Or Cells(x, 251) = "" Then
    MsgBox ("Please select a cell with a MP entry to open the respecive File!" & Chr(13) & Chr(13) & "Chose: Part.ID Row for Results page" &
Chr(13) & "Chose an isotope of interest to display this isotopes properties only" & Chr(13) & "Chose any other Row to display all charts")
    Exit Sub
    End If
    Application.ScreenUpdating = False
    open_a_MP_file_and_add_charts_if_necessary (filen)
    namepswin = ActiveWorkbook.Name
    If contains(namepswin, "PsWIN") Then
    MsgBox ("File could not be opened, check Defaultpath in Report Sheet!")
    Exit Sub
    End If
    If y = 3 Then mode_of_display = "results"
    answer = Display_only_needed_charts(Specimen, mode_of_display)
    Add_a_button_in_results_chart
    If AllwaysopenMP_file_with_all_charts_flag = True Then
         i = Charts.Count
        For x = 1 To i
          Charts(x).Visible = True
        Next x
    End If
    SortWorksheetsByColorAndName
    ' ----------------------------------------------------
    ' end of code pressing Commandbutton29
    ' ----------------------------------------------------

' saving the MP-Filename

    File_MP(Particle_Nr) = ActiveWorkbook.Name
    Line_Nr = Line_Nr + 1
Wend

' ----------------------------------------------------
' opening CO-File
' ----------------------------------------------------

    ' returning to PsWin
```

```
    Workbooks(File_PsWin).Activate

    ' --- code of CommandButton27_Click ---

    ' Save_Workbook_Name ' remember the name of this Workbook (PsWin-file) for reading VBA-Code-lines from here (Sheet CO-Code) when
creating a VBA-button in the excel-file Countrates_Overview.xls
        'Dim ID As Variant
        'Dim series As Variant
        'Dim answer As Variant

        If Worksheets("ToolBox").Ignore_Errors = True Then
            On Error Resume Next
        End If
        ID = ""
        series = Worksheets("Report").Range("C6")

        Countrates_overview_new_1280
        answer = Assign_Chart_Titles(ID, series)
    ' Create_VBA_Code ' routine to generate VBA-Code in all charts of the Countrates_Overview-File

    ' --- end of code of CommandButton27_Click ---

    File_CO = ActiveWorkbook.Name ' remember CO-File name


    Application.DisplayAlerts = False

' -------------------------
' opening Powerpoint
' -------------------------

    Set pptApp = CreateObject("Powerpoint.Application") ' starts Powerpoint
    pptApp.Visible = True
    Set pptPres = pptApp.Presentations.Add
    pptApp.ActiveWindow.ViewType = ppViewSlide
    ' bring it to front
    pptApp.WindowState = wdWindowStateMinimize
    pptApp.WindowState = wdWindowStateMaximize
```

```
' activating Powerpoint

    pptApp.Visible = msoTrue
    Set pptPres = pptApp.ActivePresentation
    pptApp.ActiveWindow.ViewType = ppViewSlide
    pptSlideNr = 0

' ------------------------------------------------------------------------------------------------------------------------------
' loop through all particles and transferring their charts A) form the MP-Files and B) from the Countrates-Overview-File to Powerpoint
' ------------------------------------------------------------------------------------------------------------------------------

For ii = 1 To Particle_Nr

    ' resetting slide-position
    Slide_Pos = 0

    ' increasing the Slide-Nr.:
    pptSlideNr = pptSlideNr + 1

    ' Adding new slide
    Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutBlank)
    pptApp.Visible = True
    pptApp.Activate
    pptApp.ActiveWindow.View.GotoSlide Index:=pptSlideNr

    ' insert textframe in PPT

    If Workbooks(File_PsWin).Sheets("PPT_Reports").add_default_label = True Then
        Default_Label = Workbooks(File_PsWin).Sheets("PSearch").Range("C9") & " - " & Workbooks(File_PsWin).Sheets("Report").Cells(20 + ii,
3) & " - "
    End If

    With pptSlide.Shapes.AddShape(msoShapeRectangle, 10, 10, 690, 30).TextFrame
        .TextRange.Text = Default_Label & User_Label
        .TextRange.Font.Size = 12
        .MarginBottom = 10
        .MarginLeft = 10
        .MarginRight = 10
        .MarginTop = 10
```

```
    End With

' --- error handling ---

    On Error GoTo Chart_Not_Found_MP


' ***********************************
' ***  A) TRANSFER OF MP-CHARTS   ***
' ***********************************

' ------------------------------------------------------------------------------------------------
' Transferring first INDIVIDUAL MP chart "Ratio Chart U235" - if activated in Sheets PPT_Reports
' ------------------------------------------------------------------------------------------------

    If Workbooks(File_PsWin).Worksheets("PPT_Reports").CheckBox23 = True Then

        ' selecting chart
        Workbooks(File_MP(ii)).Sheets("Ratio Change U235").Activate

        ' copying chart
        ActiveChart.ChartArea.Select
        ' alternative 1: copy as Picture
        ActiveChart.CopyPicture ' xlScreen
        ' alternative 2: copy as full chart (big size)
        ' ActiveChart.ChartArea.Copy

        ' Adding new slide
        pptApp.Activate

        ' Pasting Chart Version 1
        ' SendKeys ("^v")

        ' checking the next position for the chart
        Slide_Pos = Slide_Pos + 1
        If Slide_Pos = 1 Then
            Pos_Left = 0
            Pos_Top = 60
        End If
```

```
      If Slide_Pos = 2 Then
         Pos_Left = 360
         Pos_Top = 60
      End If
      If Slide_Pos = 3 Then
         Pos_Left = 0
         Pos_Top = 300
      End If
      If Slide_Pos = 4 Then
         Pos_Left = 360
         Pos_Top = 300
      End If

      If Slide_Pos < 5 Then

         ' Pasting Chart Version 2
         ' pptSlide.Shapes.Paste
         pptSlide.Shapes.Paste.Select

         ' Optimizing chart which often is not seen as whole - by converting it to picture
         'pptApp.ActiveWindow.Selection.ShapeRange.Ungroup.Select

         ' sizing and positioning the chart
         Set pptShapeRange = pptApp.ActiveWindow.Selection.ShapeRange

         With pptShapeRange
            .Fill.Transparency = 0#
            .Left = Pos_Left
            .Top = Pos_Top
            .Height = 240
            .Width = 360
         End With

      End If
   End If

' ------------------------------------------------------------------------------------------------
' Transferring second INDIVIDUAL MP chart "Ratio Chart U234" - if activated in Sheets PPT_Reports
' ------------------------------------------------------------------------------------------------
```

```
Continue_After_Error:

Next ii

' --- End of loop through all particles ---

    ' Closing the MP_Files
    For ii = 1 To Particle_Nr
        Workbooks(File_MP(ii)).Close
    Next ii

    ' Closing the CO_File
    Workbooks(File_CO).Close

    Exit Sub

Chart_Not_Found_MP:

    MsgBox "One of Your selected MP-charts is missing for the current particle " & Workbooks(File_PsWin).Sheets("Report").Cells(20 + ii, 3)
    GoTo Continue_After_Error

Chart_Not_Found_CO:

    MsgBox "One of Your selected CO-charts is missing for the current particle " & Workbooks(File_PsWin).Sheets("Report").Cells(20 + ii, 3)
    GoTo Continue_After_Error

End Sub
```

Diagramme exportieren nach Powerpoint Seibersdorf 3 Axel

Hier der Code vom Kollegen Axel

```
Sub Print_PB_Fit_Report(a)
    end_MP = determine_last_entry_by_bottom

    'check for minorpresence
MP_end = determine_last_entry_by_bottom
```

```
U234_present = False
U236_present = False
For X = 21 To MP_end
If Cells(X, 4) <> "" Then U234_present = True
If Cells(X, 8) <> "" Then U236_present = True

Next X


    Worksheets("Report").Activate
    savepath = Worksheets("Psearch").Defaultpath
        Dim PPApp As PowerPoint.Application
    Dim PPPres As PowerPoint.Presentation
    Dim PPSlide As PowerPoint.Slide


'_____________________________________________________________________________________________________________________end U234
If Worksheets("Toolbox").CheckBox7 And U234_present Then
    Specimen = 234

        '''PPT handler

    Set PPApp = CreateObject("Powerpoint.Application")
    PPApp.Visible = True
    Set PPPres = PPApp.Presentations.Add
    PPApp.ActiveWindow.ViewType = ppViewSlide
         '''PPT handler


    counter = 0
    For X = 21 To end_MP
    ID = Worksheets("Report").Cells(X, 3)
    Worksheets("Report").Cells(X, 4).Activate

    If Worksheets("Report").Cells(X, 4) <> "" Then
        atompercent = Worksheets("Report").Cells(X, 4)
        F_stat_value = Worksheets("Report").Cells(X, 29)
        teststring = Worksheets("Report").Cells(X, 27)
        totalcounts = Worksheets("Report").Cells(X, 35)
        minor_to_U235_ratio = Worksheets("Report").Cells(X, 6) / Worksheets("Report").Cells(X, 4)
        U235_atompercent = Worksheets("Report").Cells(X, 6)
        total_error_forcast = Worksheets("Report").Cells(X, 19) / Worksheets("Report").Cells(X, 4)

        gradient = True
```

```
    ratio_scatter = True
    If Len(teststring) > Len(Replace(teststring, "GG4", "")) Then gradient = False
    If Len(teststring) > Len(Replace(teststring, "GGG4", "")) Then gradient = False
    If Len(teststring) > Len(Replace(teststring, "FF4", "")) Then ratio_scatter = False
    If Len(teststring) > Len(Replace(teststring, "FFF4", "")) Then ratio_scatter = False



    pb_fit_call
    If a = 0 Then Sheets("Signal").Activate
    If a = 1 Then Sheets("Fit_Results").Activate
    answer = add_text_box_and_standardize_scale(atompercent, F_stat_value, gradient, ratio_scatter, totalcounts, minor_to_U235_ratio,
U235_atompercent, total_error_forcast)
    counter = counter + 1


'''PPT handler
ActiveChart.ChartArea.Select
ActiveChart.CopyPicture
Set PPSlide = PPPres.Slides.Add(counter, ppLayoutText)
PPApp.Visible = True
PPApp.ActiveWindow.View.GotoSlide Index:=counter
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
   .Fill.Transparency = 0#
   .Left = 0#
   .Top = 0#
   .Height = 442.12
   .Width = 719.75
End With
'''PPT handler

Application.DisplayAlerts = False
Windows("Transient_uncertainties_" & ID & ".xls").Close (False)
Application.DisplayAlerts = True
If a = 0 And gradient = False Or ratio_scatter = False Then
        'include fstat-graphs
   answer = include_fstat_graphs(Specimen, Defaultpath, ID)
ActiveSheet.ChartObjects("Chart 3").Activate
ActiveChart.ChartArea.Select
```

```
    ActiveChart.CopyPicture
    PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Left = 0#
    .Top = 417.12
    .Height = 123#
    .Width = 200.38
End With
    ActiveSheet.ChartObjects("Chart 2").Activate
ActiveChart.ChartArea.Select



    ActiveChart.CopyPicture
    PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Height = 123#
    .Width = 200.12
    .Left = 519.62
    .Top = 416.75
End With
    ActiveWindow.Close (True)


End If
End If
Next X

With PPPres
If a = 0 Then
    .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Signal"
Else
    .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Pb-Fit"
End If
End With
```

```
    End If

'_____________________________________________________________________________________________________end U234

'_____________________________________________________________________________________________________Start U235

If Worksheets("Toolbox").CheckBox8 Then
    Specimen = 235
        '''PPT handler

    Set PPApp = CreateObject("Powerpoint.Application")
    PPApp.Visible = True
    Set PPPres = PPApp.Presentations.Add
    PPApp.ActiveWindow.ViewType = ppViewSlide
        '''PPT handler


    counter = 0
    For X = 21 To end_MP
    ID = Worksheets("Report").Cells(X, 3)
    Worksheets("Report").Cells(X, 6).Activate

    If Worksheets("Report").Cells(X, 6) <> "" Then
        atompercent = Worksheets("Report").Cells(X, 6)
        F_stat_value = Worksheets("Report").Cells(X, 30)
        teststring = Worksheets("Report").Cells(X, 27)
        totalcounts = Worksheets("Report").Cells(X, 36)
        minor_to_U235_ratio = 1
        U235_atompercent = Worksheets("Report").Cells(X, 6)
        total_error_forcast = Worksheets("Report").Cells(X, 20) / Worksheets("Report").Cells(X, 6)

        gradient = True
        ratio_scatter = True
        If Len(teststring) > Len(Replace(teststring, "GG5", "")) Then gradient = False
        If Len(teststring) > Len(Replace(teststring, "GGG5", "")) Then gradient = False
        If Len(teststring) > Len(Replace(teststring, "FF5", "")) Then ratio_scatter = False
        If Len(teststring) > Len(Replace(teststring, "FFF5", "")) Then ratio_scatter = False



        pb_fit_call
        If a = 0 Then Sheets("Signal").Activate
        If a = 1 Then Sheets("Fit_Results").Activate
```

```
    answer = add_text_box_and_standardize_scale(atompercent, F_stat_value, gradient, ratio_scatter, totalcounts, minor_to_U235_ratio, U235_atompercent, total_error_forcast)
    counter = counter + 1


'''PPT handler
ActiveChart.ChartArea.Select
ActiveChart.CopyPicture
Set PPSlide = PPPres.Slides.Add(counter, ppLayoutText)
PPApp.Visible = True
PPApp.ActiveWindow.View.GotoSlide Index:=counter
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Left = 0#
    .Top = 0#
    .Height = 442.12
    .Width = 719.75
End With
'''PPT handler

Application.DisplayAlerts = False
Windows("Transient_uncertainties_" & ID & ".xls").Close (False)
Application.DisplayAlerts = True
    If a = 0 And gradient = False Or ratio_scatter = False Then
    'include fstat-graphs
        'include fstat-graphs
    answer = include_fstat_graphs(Specimen, Defaultpath, ID)
ActiveSheet.ChartObjects("Chart 3").Activate
ActiveChart.ChartArea.Select



    ActiveChart.CopyPicture
    PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Left = 0#
    .Top = 417.12
    .Height = 123#
```

```
    .Width = 200.38
End With
    ActiveSheet.ChartObjects("Chart 2").Activate
ActiveChart.ChartArea.Select



    ActiveChart.CopyPicture
    PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Height = 123#
    .Width = 200.12
    .Left = 519.62
    .Top = 416.75
End With



ActiveWindow.Close (True)


End If
End If
Next X

With PPPres
If a = 0 Then
    .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Signal"
Else
    .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Pb-Fit"
End If
End With
End If



'_____________________________________________________________________________________________________________end U235

'_____________________________________________________________________________________________________________Start U236


If Worksheets("Toolbox").CheckBox9 And U236_present Then
```

```
    Specimen = 236
           '''PPT handler

    Set PPApp = CreateObject("Powerpoint.Application")
    PPApp.Visible = True
    Set PPPres = PPApp.Presentations.Add
    PPApp.ActiveWindow.ViewType = ppViewSlide
        '''PPT handler


    counter = 0
    For X = 21 To end_MP
    ID = Worksheets("Report").Cells(X, 3)
    Worksheets("Report").Cells(X, 8).Activate

    If Worksheets("Report").Cells(X, 8) <> "" Then
       atompercent = Worksheets("Report").Cells(X, 8)
       F_stat_value = 0
       teststring = Worksheets("Report").Cells(X, 27)
       totalcounts = Worksheets("Report").Cells(X, 37)
       minor_to_U235_ratio = Worksheets("Report").Cells(X, 6) / Worksheets("Report").Cells(X, 8)
       U235_atompercent = Worksheets("Report").Cells(X, 6)
       total_error_forcast = Worksheets("Report").Cells(X, 21) / Worksheets("Report").Cells(X, 8)

       gradient = True
       ratio_scatter = True
       If Len(teststring) > Len(Replace(teststring, "GG6", "")) Then gradient = False
       If Len(teststring) > Len(Replace(teststring, "GGG6", "")) Then gradient = False
       If Len(teststring) > Len(Replace(teststring, "FF6", "")) Then ratio_scatter = False
       If Len(teststring) > Len(Replace(teststring, "FFF6", "")) Then ratio_scatter = False



       pb_fit_call
       If a = 0 Then Sheets("Signal").Activate
       If a = 1 Then Sheets("Fit_Results").Activate
       answer = add_text_box_and_standardize_scale(atompercent, F_stat_value, gradient, ratio_scatter, totalcounts, minor_to_U235_ratio,
U235_atompercent, total_error_forcast)
       counter = counter + 1


    '''PPT handler
    ActiveChart.ChartArea.Select
    ActiveChart.CopyPicture
```

```
Set PPSlide = PPPres.Slides.Add(counter, ppLayoutText)
PPApp.Visible = True
PPApp.ActiveWindow.View.GotoSlide Index:=counter
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPApp.ActiveWindow.Selection.SlideRange.Shapes(1).Delete
PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Left = 0#
    .Top = 0#
    .Height = 442.12
    .Width = 719.75
End With
'''PPT handler

Application.DisplayAlerts = False
Windows("Transient_uncertainties_" & ID & ".xls").Close (False)
Application.DisplayAlerts = True
Worksheets("Report").Cells(X, 8).Activate
    If a = 0 And gradient = False Or ratio_scatter = False Then
          'include fstat-graphs
    answer = include_fstat_graphs(Specimen, Defaultpath, ID)
ActiveSheet.ChartObjects("Chart 3").Activate
ActiveChart.ChartArea.Select



    ActiveChart.CopyPicture
    PPSlide.Shapes.Paste.Select
Set sr = PPApp.ActiveWindow.Selection.ShapeRange
With sr
    .Fill.Transparency = 0#
    .Left = 0#
    .Top = 417.12
    .Height = 123#
    .Width = 200.38
End With
    ActiveSheet.ChartObjects("Chart 2").Activate
ActiveChart.ChartArea.Select



    ActiveChart.CopyPicture
```

```
        PPSlide.Shapes.Paste.Select
    Set sr = PPApp.ActiveWindow.Selection.ShapeRange
    With sr
        .Fill.Transparency = 0#
        .Height = 123#
        .Width = 200.12
        .Left = 519.62
        .Top = 416.75
    End With
        ActiveWindow.Close (True)




    End If
    End If
    Next X

    With PPPres
    If a = 0 Then
        .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Signal"
    Else
        .SaveAs savepath & "\Transient_Data_Overview  U" & Specimen & "_Pb-Fit"
    End If
    End With
    End If


End Sub
```

Diagramm in besserer Auflösung transferieren nach Powerpoint

Moin moin,

ich will paar Diagramme aus der .xls in eine .ppt schmeißen.
Dank diesem Forum weiß ich das .CopyPicture da bei mir am besten geeignet ist.

Da mir die Bildqualität zu schlecht ist habe ich noch bischen gesucht. Rein theoretisch wäre das hier das richtige für mich:

Code:

```
.CopyPicture xlPrinter, xlBitmap
```

Da kommt aber:

Zitat:

> Die CopyPicture-Methode des ChartObjektes konnte nicht ausgeführt werden.

Wenn ich xlPrinter durch xlScreen, bzw. xlBitmap durch xlPicture ersetzte, funktioniert es.
Warum mag er nur die eine Variante nicht???

Gruß Schokomonster

ANTWORT 1

hi,
nein, ich habe nicht wirklich eine Lösung.
Halte die Shift-Taste gedrückt und gehe auf Bearbeiten/Bild kopieren (oder so ähnlich, habe grad nur XL07 verfügbar. Der Befehl wird aber nur mit Shift zusammen verfügbar).
Dort geht deine Kombi auch nicht. Ich habe noch gefunden, dass xlPrinter in Pixel umwandelt
http://www.internetcomputerforum.com/forum/microsoft-excel-forum/73701-excel-copypicture-powerpoint.html
aber nicht, warum xlBitmap dann nicht geht, sorry

Gruß
stefan

ANTWORT 2

Hi,

die beiden Varianten, die vernunftig funktionieren, sind

.CopyPicture xlScreen, xlBitmap

Da der Bildschirminhalt eben mal nur sinnvoll als BitMap im Speicher abgelegt werden kann, um dann als Grafik (.bmp) irgendwo eingefügt zu werden.

.CopyPicture xlPrinter, xlPicture

Da der Drucker mit einem BitMap als Ausgabe erstmal nichts anfangen kann, der benötigt ein Picture-Object (meist .pst).

REAKTION DES FRAGENSTELLERS

Moin moin ihr beiden,

habt riesen DANK !!!
Dann geh ich mal .pst suchen.

Später schrieb er noch:

Versuch mal ob bei dir
chtObj.CopyPicture xlPrinter, xlBitmap
funktioniert. Wenn ja, dann sollte der weiße Rand weg sein.
Wenn du kein Problem mit einer etwas schlechteren Bildqualität hast, dann versuch es mal mit "xlScreen" statt "xlBitmap".

Diagramme positioniert einfügen und Größe festlegen

```
Sheets("Dia.Cust&Seg").ChartObjects("C-Plant").CopyPicture xlScreen
    i = i + 1
    Set pptSlide = pptPres.Slides(i + 3) '(i+3), Diagramm ab Seite 4
    Set shp = pptSlide.Shapes.Paste
    shp.Top = 100 'Ausrichtung: 100 pixel von oben
    shp.Left = 25 'Ausrichtung: 25 pixel von links
    shp.Height = 330
    shp.ScaleWidth 1.1, msoFalse, msoScaleFromTopLeft 'Skalierung von 110%
```

Diagramm in PowerPoint abgeschnitten

```
Als ich ein Exceldiagramm für IAEA in Powerpoint einfügte, fehlte immer ein Teil von der Legende
Mit Makrorecorder und Recli Bild Bearbeiten, bot PPT an das Diagramm in ein Bild umzuwandeln, was folgende VBA-Zeile
```

erzeugte und das Diagramm korrekt anzeigte:

```
    ActiveWindow.Selection.ShapeRange.Ungroup.Select
```

(In Word gab es dazu den Befehl ConvertToShape, um ein frei verschiebbares Element zu machen.)

Folien erzeugen in PowerPoint für jede Zeile in Excel

Lösung 0

Es gibt ja irgendwo den Befehl mit dem Erzeugen (ADD) der neuen Folie - zB

```
        Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutTitleOnly)
```

oder

```
            Set PPSlide = PPPres.Slides.Add(counter, ppLayoutText)
```

Man sieht in diesem Beispiel schon, dass man einen Parameter mitübergeben kann - nicht nur die Seitennummer, sondern immer auch eine Slide-Layout-Eigenschaft. Hier eine Liste der Möglichkeiten

Ich nehme meist **ppLayoutBlank - denn dann bekommt man eine leere Folie.**

Slide.Layout Property

Office 2007

Returns or sets a **PpSlideLayout** constant that represents the slide layout. Read/write.

Syntax

***expression*.Layout**

expression A variable that represents a **Slide** object.

Remarks

The value of the **Layout** property can be one of these **PpSlideLayout** constants.

ppLayoutBlank
ppLayoutChart
ppLayoutChartAndText

ppLayoutClipartAndText
ppLayoutClipArtAndVerticalText
ppLayoutFourObjects
ppLayoutLargeObject
ppLayoutMediaClipAndText
ppLayoutMixed
ppLayoutObject
ppLayoutObjectAndText
ppLayoutObjectOverText
ppLayoutOrgchart
ppLayoutTable
ppLayoutText
ppLayoutTextAndChart
ppLayoutTextAndClipart
ppLayoutTextAndMediaClip
ppLayoutTextAndObject
ppLayoutTextAndTwoObjects
ppLayoutTextOverObject
ppLayoutTitle
ppLayoutTitleOnly
ppLayoutTwoColumnText
ppLayoutTwoObjectsAndText
ppLayoutTwoObjectsOverText
ppLayoutVerticalText
ppLayoutVerticalTitleAndText
ppLayoutVerticalTitleAndTextOverChart

Example

This example changes the layout of slide one in the active presentation to include a title and subtitle if it initially has only a title.

Visual Basic for Applications

```
With ActivePresentation.Slides(1)
    If .Layout = ppLayoutTitleOnly Then
        .Layout = ppLayoutTitle
    End If
```

```
End With
```

Lösung 1

```
Sub ZeilenNachPPT()
Dim ppt As Object, i
Set ppt = CreateObject("powerpoint.application")
ppt.presentations.Add
ppt.Visible = True
For i = 1 To ActiveSheet.UsedRange.Rows.Count
ppt.activepresentation.slides.Add i, 2
ppt.activepresentation.slides(i).Shapes(1).TextFrame.TextRange.Text = Range("A" & i).Value
Next
End Sub
```

Lösung 2

Create PowerPoint Slide for Each Row in Excel Workbook

BY CRAIG | PUBLISHED JUNE 8, 2011

This may seem like a really weird thing to want to do. Imagine this though: You want a presentation to show off the names of a lot of students on a constant loop at a kiosk, and you don't want to have to retype the names. VBA to the rescue.

This little snippet of code will do the following:

1. Open a given Excel Document
2. For Each used row in column A of sheet 1, create a copy the first slide in the presentation.
3. Change the text of the first text box to the content of that cell

Easy. Here it is:

```
Sub CreateSlides()
'Open the Excel workbook. Change the filename here.
Dim OWB As New Excel.Workbook
Set OWB = Excel.Application.Workbooks.Open("C:\list.xlsx")
'Grab the first Worksheet in the Workbook
Dim WS As Excel.Worksheet
Set WS = OWB.Worksheets(1)
'Loop through each used row in Column A
```

```
For i = 1 To WS.Range("A65536").End(xlUp).Row
    'Copy the first slide and paste at the end of the presentation
    ActivePresentation.Slides(1).Copy
    ActivePresentation.Slides.Paste (ActivePresentation.Slides.Count + 1)

    'Change the text of the first text box on the slide.
    ActivePresentation.Slides(ActivePresentation.Slides.Count).Shapes(1).TextFrame.TextRange.Text = WS.Cells(i, 1).Value
Next
End Sub
```

Paste this into a new module inside your PowerPoint presentation. You will need to add in a reference to Microsoft Excel Objects (Tools -> References). Change the name and location of the Excel file that you want to use.

If you set up the first slide exactly as you want it before running the macro, then the same formats and layout will be copied too. Alternatively use Slide Masters to set up all the slides after they have been created.

Folien in PowerPoint löschen

Folien löschen

```
ppt.activepresentation.slides(1).Delete
```

Gesamter Code

```
Sub ZeilenNachPPT()
Dim ppt As Object, i
Set ppt = CreateObject("powerpoint.application")
ppt.Visible = True
ppt.presentations.Open "D:\test\pres1.ppt"
For i = 1 To ActiveSheet.UsedRange.Rows.Count
ppt.activepresentation.slides(1).Copy
ppt.activepresentation.slides.Paste (ppt.activepresentation.slides.Count + 1)
ppt.activepresentation.slides(i + 1).Shapes(1).TextFrame.TextRange.Text = Range("A" & i).Value
Next
'ppt.activepresentation.slides(1).Delete
End Sub
```

Folienanzahl in PowerPoint ermitteln

ppt.activepresentation.slides.Count

Gehe zu bestimmter Folie

Das macht man am besten mit dem " ActiveWindow.View.GotoSlide 1"-Befehl (in diesem BSP springen wir zur ersten Folie).

```
Dim pptApp As PowerPoint.Application ' the Powerpoint-Application
Dim pptPres As PowerPoint.Presentation ' the actual open Powerpoint-Presentation
Dim pptSlide As PowerPoint.Slide ' the actual Powerpoint-Slide

Set pptApp = CreateObject("Powerpoint.Application")
pptApp.ActiveWindow.View.GotoSlide 1
```

Grafiken exportieren von Excel nach PowerPoint

Kopiert alle Grafiken einer Exceltabelle in eine neue PowerPoint-Datei. Für jede Grafik wird einen neue Folie angelegt.

Hinweis:
Damit die Prozedur funktioniert, müssen Sie im Editor unter **Extras/Verweise** einen Verweis setzen:

- Microsoft PowerPoint x.x Object Library

```
Sub jede_Grafik_nach_PowerPoint()
'Extras - Verweise: Microsoft PowerPoint x.x Object Library
Dim Grafik As Shape
Dim PP As PowerPoint.Application
Dim PP_Datei As PowerPoint.Presentation
Dim PP_Folie As PowerPoint.Slide

On Error GoTo Debug

Set PP = CreateObject("Powerpoint.Application")
With PP
  .Visible = True
  .Presentations.Add
End With

Set PP_Datei = PP.ActivePresentation
```

```
For Each Grafik In ActiveSheet.Shapes
 'neue Folie einfügen
  PP.ActivePresentation.Slides.Add 1, ppLayoutBlank
  Set PP_Folie = PP_Datei.Slides(1)
 'kopieren
  Grafik.CopyPicture
 'einfügen
  PP_Folie.Shapes.Paste
Next

Set PP_Folie = Nothing
Set PP_Datei = Nothing
Set PP = Nothing

Exit Sub

Debug:
Set PP_Folie = Nothing
Set PP_Datei = Nothing
Set PP = Nothing
    MsgBox "FehlerNr.: " & Err.Number & vbNewLine & vbNewLine _
    & "Beschreibung: " & Err.Description _
    , vbCritical, "Fehler"
End Sub
```

Powerpoint in den Vordergrund holen

```
pptApp.Visible = True
pptApp.Activate

pptApp.WindowState = wdWindowStateMinimize
pptApp.WindowState = wdWindowStateMaximize
```

Powerpoint beenden

hier wird zb. das appl. beendet wenn ...

Code:

```
If Not objPPApp Is Nothing Then objPPApp.Quit
```

weiter vorher wird das objPPApp sicherlich definiert.
meine vermutung wäre diese version:

Code:

```
powerpoint.application.quit
```

Powerpoint-Vorlage öffnen, speichern und beenden

```
Dim MSppt As Object
Set MSppt = CreateObject("PowerPoint.Application")
' Laden der Datei
With MSppt
.Visible = msoTrue
.Activate
.Presentations.Open "I:\V\VEE\Mitarbeiter\Vorlage\weiterentwicklung\vorlage.ppt"
End With

kundenname = Worksheets("Rohdaten").Cells(1, 5)
name_pp_datei = "I:\V\VEE\Mitarbeiter\Vorlage\weiterentwicklung\" & kundenname & ".ppt"

'Umspeichern der PP-Vorlage
With MSppt
.ActivePresentation.SaveAs name_pp_datei
.ActivePresentation.Close
End With
```

Shapes in Powerpoint ansprechen

Powerpoint Shapes mit Excel-VBA (um-)benennen

Hat jemand Ahnung, wie ich Shapes (Graphiken etc.) mit VBA anspreche. Folgendes Problem:
In Excel habe ich Daten, die ausgewertet sind. In Powerpoint habe ich parallel eine Musterpreäsentation gebastelt mit leeren Graphiken, die vorformattiert sind. Ich möchte nun die Daten AUTOMATISCH per Makro ind iese Graphik reinkopieren lassen, damkt ich dies auch für weitere Teilnehmer ausführen kann. Damit würde ich mir einen großen Aufwand ersparen. Ich schaffe es leider nicht, diese Graphik in Powerpoint anzusprechen. Hat jedes Shape in PPT eine exakten Namen oder kann ich diesen vergeben?
Bin froh um jede Hilfe.

HAllo Ben

Die Shapes haben einen Namen und einen Index.
Den Namen kannst ändern. Beim Index bin ich mir nicht sicher.
Ansprechen kannst du sie über den Namen und (oder) über den Index.
Schau es dir mal an:

Code:

```
Option Explicit


Public Sub test()
Dim I As Integer
With ActivePresentation.Slides(1) 'Anpassen.
'MsgBox .Name
  For I = 1 To .Shapes.Count
    MsgBox .Shapes(I).Name 'alter Name
    .Shapes(I).Name = "Bild" & I 'Umbenennen
    MsgBox .Shapes(I).Name 'neuer Name
  Next
End With
End Sub
```

Wie bekomme ich den Namen von einem Shape raus. Danke schon einmal für deine Antwort,finde ich klasse.....

HAllo Ben

Markier mal eins von deinen shapes und schubs mal diesen Code an:

Code:

```
Public Sub test()
MsgBox ActiveWindow.Selection.ShapeRange.Name
End Sub
```

Oder Schreib den Namen rein, dann brauchst du nicht soviel Clicken ;-)

Code:

```
Sub machs()
Dim sh As Shape
For Each sh In ActivePresentation.Slides(1).Shapes
   sh.TextFrame.TextRange.Text = sh.Name
Next
End Sub
```

ransi

HAllo Ben

Sorry,
Du sagtest die shapes sind vorformatiert.
Dann pack die Namen besser in einen Kommentar damit du dir die Formate und Texte nicht zerschießt.

Code:

```
Sub machs()
```

```
Dim sh As Shape
Dim C As Comment
For Each sh In ActivePresentation.Slides(1).Shapes
 Set C = ActivePresentation.Slides(1).Comments.Add(sh.Left, sh.Top, "", "", sh.Name)
Next
End Sub
```

ransi

Hat sich erledigt, echt ein super Makro. Vielen lieben Dank. Wünsche dir einen sonnigen Tag!

Tabelle in Powerpoint - Bündigkeit einstellen

Hallo zusammen!

Ich brauch mal einen Profi Tipp!
Ich exportiere mittels Excel Makro Daten in eine durch das Makro neu erstellten Tabelle ins Powerpoint.
Die Schleife befüllt mir die Tabelle zeilenweise.
Das klappt alles einwandfrei.
Ich würde aber gerne zwei der drei Spalten links- und die letzte Spalte rechtsbündig formatieren.
Ich als Hobby Excel Marko tüftler komme mit dem Objektkatalog von Powerpoint nicht zu recht.

Wenn ich das richtig verstanden habe, dann sind im Powerpoint Tabellenzellen als einzelne shapes an zu "Sprechen"?

Die If Funktion verwende ich, da ich die letzte Zeile so formatiert lassen will wie sie ist. (center)

In der If Funktion bekomme ich folgenden Fehler:
"Objekt unterstützt diese Aktion nicht"
konkret bei: PP_Folie.Shapes("Table 1").Table.Cell(I, H).Shape.TextEffect.Alignment = msoTextEffectAlignmentLeft

Hier mein Code:

```
Set PP_Folie = PP_Datei.Slides(Folie)
    Set PP_Table = PP_Folie.Shapes.AddTable(Row, Col, Left, Top, Width, Hight)

    For I = 1 To Row
        For H = 1 To Col
            With PP_Table.Table.Cell(I, H).Shape
                .TextFrame.TextRange.Font.Size = 14
                .TextFrame.TextRange.Font.Name = "Arial"
                '   .BackColor.SchemeColor = ppShadow
                '   .TextRange.Font.Color.SchemeColor = ppForeground
                .TextFrame.TextRange.Font.Bold = msoTrue
                .TextFrame.HorizontalAnchor = msoAnchorCenter
                .TextFrame.VerticalAnchor = 3
                .TextFrame.MarginBottom = 0
                .TextFrame.MarginLeft = 0
                .TextFrame.MarginRight = 0
                .TextFrame.MarginTop = 0
                '.TextEffect.Alignment = msoTextEffectAlignmentLeft
            End With
            PP_Table.Table.Cell(I, H).Shape.TextFrame.TextRange.Text = Sheets(Folienname).Cells(I, H).Value
            If I <> Row Then
                If H = 1 Or H = 2 Then PP_Folie.Shapes("Table 1").Table.Cell(I, H).Shape.TextEffect.Alignment =
msoTextEffectAlignmentLeft

            End If
        Next H
    Next I
```

Antwort:

Das ist doch garnet sooo schwer. Objektstrukturen sind hierachisch aufgebaut.
Bezüglich Text anzuwendente Eigenschaften und Methoden müssen dem zufolge an Textframe (die den gesammten Textblock betreffen) und weiter an Textrange, also Textbestandteile bis runter zu einem Buchstaben hängen. Und Absatzeinstellungen sollten dann der Logik zufolge unter

...TextFrame.TextRange.ParagraphFormat.Alignment = ppAlignCenter

gefunden werden. Wenn die Aufzählung Paragraphs was hergibt, dann sollten die auch getrennt ansteuerbar sein...

..TextFrame.TextRange.Paragraphs(3).ParagraphFormat.Alignment = ppAlignCenter

Reaktion und finale Lösung des Fragestellers

Hi,

Danke für die schnelle Antwort.

Nein so schwer ist es nicht da geb ich dir recht. Ich hab mich nur nicht mehr bei der Fülle an Objekt Möglichkeiten hinaus gesehen. Der Anzeige Struktur des Objektkatalogs / Objektbibliothek verwirrt mich etwas.
Daher danke für deinen Hinweis.

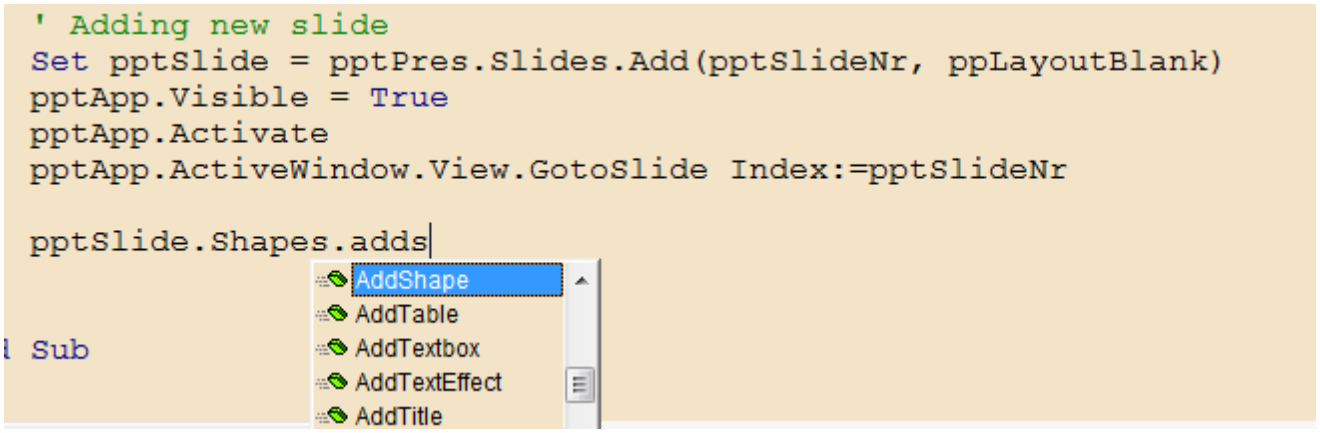
Nachdem ich den Code nochmals durchgesehen habe, ist mir aufgefallen, dass mit den zwei folgenden code Zeilen die Pos. des Textes in der Tabellen-Zelle definiert wird:

```
With PP_Table.Table.Cell(I, H).Shape
.TextFrame.TextRange.Font.Size = 14
.TextFrame.TextRange.Font.Name = "Arial"
' .BackColor.SchemeColor = ppShadow
' .TextRange.Font.Color.SchemeColor = ppForeground
.TextFrame.TextRange.Font.Bold = msoTrue
.TextFrame.HorizontalAnchor = msoAnchorCenter (die Zahlen von 1-3 = links mitte rechts können auch verwendet werden)
.TextFrame.VerticalAnchor = 3
.TextFrame.MarginBottom = 0
.TextFrame.MarginLeft = 0
.TextFrame.MarginRight = 0
.TextFrame.MarginTop = 0
'.TextEffect.Alignment = msoTextEffectAlignmentLeft
End With
```

Textfelder in Powerpoint befüllen

Was sehr nett ist, direkt in Powerpoint sich die möglichen Objekte anzeigen lassen:

```
Public pptPres As PowerPoint.Presentation ' the actual open Powerpoint-Presentation
Public pptSlide As PowerPoint.Slide ' the actual Powerpoint-Slide
```



Wir können sowohl ein Shape einfügen und dessen TextFrame einstellen, oder eine Textbox, oder direkt mit einem Title arbeiten.

Meine Lösung mit Textframe

```
' ****************************************
' ----------------------------------------
' defining public varibales
' ----------------------------------------
' ****************************************

    ' Filevariables

'    Public File_PsWin As String ' Filename of PsWin

    ' Variables for / in MS Powerpoint

'    Public PPTtemplate As String ' Path and filename of the Word-template
    Public pptApp As PowerPoint.Application ' the Powerpoint-Application
    Public pptPres As PowerPoint.Presentation ' the actual open Powerpoint-Presentation
    Public pptSlide As PowerPoint.Slide ' the actual Powerpoint-Slide
    Public pptSlideNr As Integer ' the Number of the actual Powerpoint-Slide
    Public pptFile As String ' the filename of the actual Powerpoint-Presentation
    Public pptShapeRange  ' the shape of the chart in Powerpoint

Sub Create_PPT_Textframe()

' -------------------------
' opening Powerpoint
```

```
' -------------------------

    Set pptApp = CreateObject("Powerpoint.Application") ' starts Powerpoint
    pptApp.Visible = True
    Set pptPres = pptApp.Presentations.Add
    pptApp.ActiveWindow.ViewType = ppViewSlide
    ' bring it to front
    pptApp.WindowState = wdWindowStateMinimize
    pptApp.WindowState = wdWindowStateMaximize

    ' activating Powerpoint
    pptApp.Visible = msoTrue
    Set pptPres = pptApp.ActivePresentation
    pptApp.ActiveWindow.ViewType = ppViewSlide
    pptSlideNr = 0


    ' increasing the Slide-Nr.:
    pptSlideNr = pptSlideNr + 1

    ' Adding new slide
    Set pptSlide = pptPres.Slides.Add(pptSlideNr, ppLayoutBlank)
    pptApp.Visible = True
    pptApp.Activate
    pptApp.ActiveWindow.View.GotoSlide Index:=pptSlideNr

    With pptSlide.Shapes.AddShape(msoShapeRectangle, 10, 10, 690, 30).TextFrame
        .TextRange.Text = "Here is some test text"
        .TextRange.Font.Size = 12
        .MarginBottom = 10
        .MarginLeft = 10
        .MarginRight = 10
        .MarginTop = 10
        .TextRange.ParagraphFormat.Alignment = ppAlignLeft
    End With


End Sub
```

Lösung 1

```
Set myDocument = ActivePresentation.Slides(1)
With myDocument.Shapes _
        .AddShape(msoShapeRectangle, 0, 0, 250, 140).TextFrame
    .TextRange.Text = "Here is some test text"
    .TextRange.Font.Size = 12
    .MarginBottom = 10
    .MarginLeft = 10
    .MarginRight = 10
    .MarginTop = 10
End With
```

Lösung 2

ich möchte ein Textfeld in Powerpoint per VBA befüllen. Folgenden Code habe ich:

'Füge den Inhalt in Folie 1 in einem Textfeld ein
Dim iSlide0 As Object
Set iSlide0 = pp.ActivePresentation.Slides(1)
Sheets("Tabelle1").Select
Wert = Range("b3").Value
With iSlide0.Shapes _
.AddShape(msoShapeRectangle, 222, 110, 350, 95).TextFrame
.TextRange.Text = "Präsentation" & vbNewLine & Wert
.TextRange.Font.Size = 26
.TextRange.Font.Color = RGB(255, 255, 255)
'.TextRange.Backstyle = fmBackStyleTransparent
'.textfeld.Backstyle = 0
.MarginBottom = 0
.MarginLeft = 10
.MarginRight = 10
.MarginTop = 10
End With

Das funktioniert auch soweit. Leider bekomme ich das Textfeld nicht transparent. Falls dies mit Backstyle funktioniert, habe ich dies wohl bisher nicht korrekt angewendet...Weiß jemand Rat? Nutze Office 2010.

Danke für die Hilfe!

Grüßt Euch,

hab es mit dem Powerpoint-Textfeld nicht hinbekommen, stattdessen habe ich in Excel ein Textfeld erzeugt, welches dann sehr einfach zu konfigurieren war und dann in Powerpoint rüberkopiert.

```
Dim textkasten As Object

Sheets("tabelle1").Unprotect

Set textkasten = ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, 369.75, 184.5, _
500.75, 135.75)
textkasten.TextFrame2.TextRange.Characters.Text = _
"Hallo Welt" & vbNewLine & "2012"
With textkasten
.TextFrame2.TextRange.Font.Size = 30
.TextFrame2.TextRange.Font.NameComplexScript = "Arial"
.TextFrame2.TextRange.Font.NameFarEast = "Arial"
.TextFrame2.TextRange.Font.Name = "Arial"
.TextFrame2.TextRange.ParagraphFormat.Alignment = msoAlignRight
.Fill.Visible = msoFalse
.Line.Visible = msoFalse
.TextFrame2.TextRange.Font.Fill.ForeColor.TintAndShade = 1
.TextFrame2.TextRange.Font.Fill.ForeColor.Brightness = 1
.TextFrame2.TextRange.Font.Fill.Transparency = 0

End With

textkasten.Cut
With iSlide0.Shapes.Paste
.Top = 110
.Left = 80
End With
```

Lösung 3

Most often we will come across a scenario where powerpoint slides need to be created automatically.

Here is a sample & simple code to do that. This code is created using VBA (Excel 2000)

```
Sub Create_PowerPoint_Slides()

On Error GoTo Err_PPT

Dim oPA As PowerPoint.Application
Dim oPP As PowerPoint.Presentation
Dim oPS As PowerPoint.Slide
Dim oShape As PowerPoint.Shape
Dim sPath As String
Dim sFile As String
Dim i1 As Integer


sPath = "C:\"
sFile = "MyfileName"

Set oPA = New PowerPoint.Application
oPA.Visible = msoTrue

Set oPP = oPA.Presentations.Add(msoTrue)

For i1 = 1 To 10
oPP.Slides.Add 1, ppLayoutBlank
Next i1

Set oPS = oPP.Slides(1)
Set oShape = oPS.Shapes.AddTextbox(msoTextOrientationHorizontal, 140#, 246#, 400#, 36#)
oShape.TextFrame.WordWrap = msoTrue

oShape.TextFrame.TextRange.Text = "Comments For File : " & sFile
With oShape
```

```
.Fill.Visible = msoTrue
.Fill.Solid
.Fill.ForeColor.RGB = RGB(204, 255, 255)
.Line.Weight = 3#
.Line.Visible = msoTrue
.Line.ForeColor.SchemeColor = ppForeground
.Line.BackColor.RGB = RGB(255, 255, 255)
End With


oPP.SaveAs sPath & sFile & ".ppt"
oPP.Close
oPA.Quit

If Not oPS Is Nothing Then Set oPS = Nothing
If Not oPP Is Nothing Then Set oPP = Nothing
If Not oPA Is Nothing Then Set oPA = Nothing



Err_PPT:
If Err <> 0 Then
MsgBox Err.Description
Err.Clear
Resume Next
End If

End Sub
```

Lösung 4

Hello,

I am a beginier VBA and have 1 post in the powerpoint help thread.

Wanted to post in here and try and get some direction.

The back story:

I have been able to figure out how to copy and past from excel cells to powerpoint by naming and calling out each text box.

Here is an example of my code that is working

```
Private Sub CommandButton1_Click()
    Dim oXL As Object 'Excel.Application
    Dim oWB As Object 'Excel.Workbook
    Dim oSld As Slide
    Set oXL = CreateObject("Excel.Application")
    Set oWB =
oXL.Workbooks.Open(FileName:="C:\Users\smaurer\Documents\AutoFillpptTextBox\textbox_auto_copy_template.xlsx")
    Set oSld = ActivePresentation.Slides(5)
     ' e.g. oSld.Shapes("Textbox 10").TextFrame.TextRange.Text = oWB.Sheets(1).Range("A1").Value
     ' Copy cell contents from the 1st sheet in Excel to the textboxes in PowerPoint.

    oSld.Shapes("Name1").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B5").Value
    oSld.Shapes("Name2").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B6").Value
    oSld.Shapes("Name3").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B7").Value
    oSld.Shapes("Name4").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B8").Value
    oSld.Shapes("Name5").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B9").Value
    oSld.Shapes("Name6").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B10").Value
    oSld.Shapes("Name7").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B11").Value
    oSld.Shapes("Name8").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B12").Value
    oSld.Shapes("Name9").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B13").Value
    oSld.Shapes("Name10").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B14").Value
    oSld.Shapes("Name11").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B15").Value
    oSld.Shapes("Name12").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B16").Value
    oSld.Shapes("Name13").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B17").Value
    oSld.Shapes("Name14").TextFrame.TextRange.Text = oWB.Sheets(1).Range("B18").Value

    oWB.Close
    oXL.Quit
    Set oWB = Nothing
    Set oXL = Nothing
End Sub
```

I am trying to shorten this with a loop. And keep getting a type mismatch error.

```
Sub Auto_Import_Excel()

    Dim oXL As Object 'Excel.Application
    Dim oWB As Object 'Excel.Workbook
    Dim oSld As Slide
    Set oXL = CreateObject("Excel.Application")
    Set oWB =
oXL.Workbooks.Open(FileName:="C:\Users\smaurer\Documents\AutoFillpptTextBox\textbox auto copy template VB2.xlsx ")
    Set oSld = ActivePresentation.Slides(3)

    Dim cellNumber As Integer
    Dim j As Integer
    For j = 1 To 14
        cellNumber = j + 4

        oSld.Shapes("Name" + j).TextFrame.TextRange.Text = oWB.Sheets(1).Range("B" + cellNumber).Value

    Next j

    oWB.Close
    oXL.Quit
    Set oWB = Nothing
    Set oXL = Nothing
End Sub
```

I do not know why I cannot loop this???

Thanks!!

Lösung1:

Try using ampersand instead.

```
oSld.Shapes("Name" & j).TextFrame.TextRange.Text = oWB.Sheets(1).Range("B" & cellNumber).Value
```

Lösung 2:

```
Private Sub CommandButton1_Click()

sn=getobject("C:\Users\smaurer\Documents\AutoFillpptTextBox\textbox_auto_copy_template.xlsx").sheets(1).range("B5:B18")

    For j=1 To UBound(sn)
        ActivePresentation.Slides(5).Shapes("Name" & j").TextFrame.TextRange.Text = sn(j,1)
    Next
End Sub
```

Lösung 5 - Textbox-Schriftgröße

ActivePresentation.Slides(5).Shapes("textbox4").TextFrame.TextRange.Font.Size = 28

Lösung 6

Texteinfügen geht ganz leicht, man muss nur draufkommen :

TextBox1.Text = "Text im Textfeld"

ActiveX-Textfeld

ActivePresentation.Slides(40).Shapes("Textfeld").Name = "asdfjklö"

Textframe

ActivePresentation.Slides.Item(1).Shapes.Item(2).TextFrame.TextRange.Text = "Der Wert beträgt " & sh & " was weiss ich"

Lösung 7

http://*www.office-loesung.de/ftopic66688_0_0_asc.php

Hallo,

kann mir bitte jemand nen tipp geben, wie ich den Zellinhalt einer beliebigen Tabelle in das Textfeld 1 einer Powerpointpräsentation schreibe? Danke.

Gruß Martin

```
Sub ExcelWertInTexfeld()
'Code in Powerpoint in ein Modul einfügen
Dim EX As Object
Dim Wert As Variant
Set EX = CreateObject("Excel.Application")
EX.Workbooks.Open FileName:="J:\Temp\Mappe1.xlsx", ReadOnly:=True
Wert = EX.Workbooks("Mappe1.xlsx").Sheets(1).Cells(1, 1)
ActivePresentation.Slides.Item(1).Shapes.Item(2).TextFrame.TextRange.Text = "Der Termin ist am " & Wert
EX.Quit
End Sub
```

Lösung 8

Copying from Excel to Powerpoint

Instructions

1.
   - *1*

Launch Excel, type "Alan" in "A1," "Daniel" in "A2," "Kitzia" in "A3," "Oscar" in "A4" and "Yarexli" in "A5." Press "CTRL" and "S" to save the workbook in "C:\" as "ExcelFile.xlsx." Close Excel.

- ## 2

Launch PowerPoint, click the "Developer" tab and click "Macros" to launch the Macro dialog window. Type "copyFromExcel" below Macro Name and click the "Create" button. Click the "Tools" menu and click "References" to launch the References dialog window. Scroll down and check the box next to "Microsoft Excel <version number> Object Library" and click "OK."

-

- ## 3

Copy and paste the following to create the variables you will use to copy the data from Excel:

```
Dim sourceXL As Excel.Application

Dim sourceBook As Excel.Workbook

Dim sourceSheet As Excel.Worksheet

Dim dataReadArray(10) As String

Dim myPress As Presentation

Dim newSlide As Slide
```

- ## 4

Set values to the object variables:

```
Set sourceXL = Excel.Application
```

```
Set sourceBook = sourceXL.Workbooks.Open("G:\ExcelFile.xlsx")
Set sourceSheet = sourceBook.Sheets(1)
Set myPres = ActivePresentation
Set newSlide = myPres.Slides.Add(Index:=myPres.Slides.Count + 1, Layout:=ppLayoutText)
```

- ## 5

Read the data in the Excel file and store it in a String array:

```
sourceSheet.Range("A1").Select
dataReadArray(0) = sourceSheet.Range("A1").Value
sourceSheet.Range("A2").Select
dataReadArray(1) = sourceSheet.Range("A2").Value
sourceSheet.Range("A3").Select
dataReadArray(2) = sourceSheet.Range("A3").Value
sourceSheet.Range("A4").Select
dataReadArray(3) = sourceSheet.Range("A4").Value
sourceSheet.Range("A5").Select
dataReadArray(4) = sourceSheet.Range("A5").Value
```

- ## 6

Add the data from the String array to a new slide in your current presentation:

```
newSlide.Shapes(1).TextFrame.TextRange = "Data copied from Excel"
```

```
newSlide.Shapes(2).TextFrame.TextRange = dataReadArray(0) & vbNewLine & _
dataReadArray(1) & vbNewLine & _
dataReadArray(2) & vbNewLine & _
dataReadArray(3) & vbNewLine & _
dataReadArray(4) & vbNewLine
```

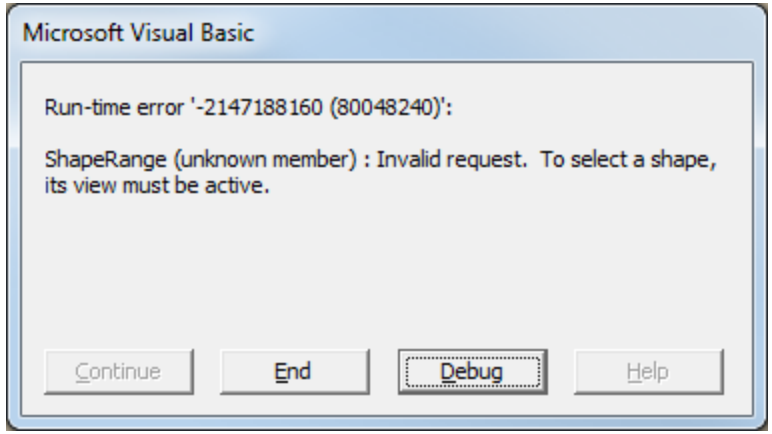
- 7

Close the workbook:

```
sourceBook.Close
```

Read more: How to Automatically Copy From Excel to PowerPoint Using a VBA Macro | eHow.com http://www.ehow.com/how_5551671_automatically-powerpoint-using-vba-macro.html#ixzz2C0nvUnDT

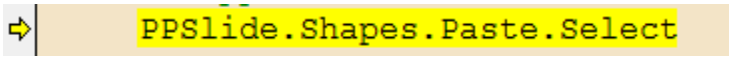
The „To select a shape, its view must be active”-error



Usually the right focus on the correct Powerpoint-slide is not needed – except for one important command: the shape.select-command.

This command is needed to change the inserted shape – f.e. an Excel chart – in its position and size.

The above shown error is always caused by command-lines as the following:



```
PPSlide.Shapes.Paste.Select
```

Here the content of the clipboard (an Excel chart-graphic) is pasted into Powerpoint and selected at the same time.

Two things must be fulfilled for the select-command to work properly:

The focus must be on the active presentation (nevertheless its window can be minimized) and the focus/view must also be set on the active slide:

a.) Setting the focus on the active presentation:

```
    Dim PPApp As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide

' Create / start instance of PowerPoint
Set PPApp = CreateObject("Powerpoint.Application")

' PowerPoint must be visible
PPApp.Visible = True

' Create a presentation
Set PPPres = PPApp.Presentations.Add

' Secureing Powerpoints visibility
' the following line is not needed and replaced by the next command and therefore commented out)
' PPApp.Visible = True

' added: securing that the Window of the newly created presentation is activated
PPPres.Windows.Application.Activate

' Some PowerPoint-commands work best in normal slide view
PPApp.ActiveWindow.ViewType = ppViewSlide

' Maximizing Powerpoint (I'd recommend the wdWindowStateMaximize-command instead)
PPApp.WindowState = ppWindowMaximized

If Hide_During_printing Then ' if a certain checkbox 'Hide_during_printing' is set
```

```
    ' original code to minimize
    PPApp.WindowState = ppWindowMinimized
    ' added code to minimize
    PPApp.WindowState = wdWindowStateMaximize
    PPApp.WindowState = wdWindowStateMinimize

End If
```

The above code is a usual code to start Powerpoint and add a new presentation. The original version was without the yellow marked lines. When running the procedure the first time, no error occurred.

When starting it a second time (and having the before created Powerpoint-presentation of the first run-through left open), the above error occurred.

With the addition of the yellow lines, the code runs errorless and correct.

The WindowState-value ppWindowMinimized obviously looses the focus on the Powerpoint-window of the active Powerpoint-presentation. With the addition of (or replacement with) the wdWindowStateMinimize-command, the code runs error-free.

b.) Setting the focus on the active slide:

The following code adds new slides and because there are several new slides created using a loop, the variable counter is increased every run-through to correctly address the next slide

```
counter = counter + 1

' activate and selecting the chart in Excel that is to be copied
Charts(sheet_to_print).Activate
ActiveChart.ChartArea.Select

' the variable copy_type sets the way the chart is copied in Excel:
' as a full chart with all data-series (value 1), or just as a picture (value 2)

if copy_type = 1 then
      ActiveChart.ChartArea.Copy ' copying the whole chart
End if
if copy_type = 2 then
      ActiveChart.CopyPicture ' copying only the picture
End if
```

```
' adding a new slide (using the variable 'counter' for the correct number of the slide)
Set PPSlide = PPPres.Slides.Add(counter, ppLayoutText)

' setting the focus on the newly added slide
PPApp.ActiveWindow.View.GotoSlide Index:=counter

' pasting the Excelchart (from clipboard) in Powerpoint and selecting it
PPSlide.Shapes.Paste.Select
```

The yellow marked line secures that the focus for pasting and selecting the chart is on the right, newly added slide.

Known issue still left: when out of Excel new Powerpoint-presentations are generated and are left opened, and then Excel is closed and opened again, and than new Powerpoint-presentations are generated again – the new presentations of the new Excel-application have a conflict with the old presentations of the former Excel-application and the "To select a shape, its view must be active"-error" appears again.

But as long Excel is started first and stays open, several new Powerpoint-presentations can be created without any conflict.

Verknüpfte ObjeKte in Powerpoint neu verlinken

Aktualisiert Links von eingebetten Objection in einer Powerpoint Präsentation
Die Anforderung entstand aus dem Auftrag, dass eine Tabelle in welcher die Unternehmensdaten monatlich hinzugefügt und erweitert werden, mit einer PowerPoint-Präsentation verknüpft werden.
Diese EXCEL Tabelle wird aber monatlich unter einem neuen Namen gespeichert, woraus das Problem resultiert, dass die Präsentation jedesmal neu erstellt werden muss weil die Object Links immer noch auf die alte EXCEL Tabelle verweisen. Mit diesem Makro werden die bisherigen Object-Links in der Präsentation auf die, unter einem anderen Namen gespeicherte VATER-Tabelle, umgeleitet .

```
Option Explicit

Const ppPresName As String ="\Pfad\PowerPointPres.ppt"
'Normalweise der gleiche Pfad wie die Präsentation
Const LinkIni As String = "\Pfad\ppLink.ini"


Sub PP_Presentation_Start_and_Update_ObjectLinks()
'(C) Ramses
'Eine Powerpoint Präsentation enthält verschiedene verknüpfte Objecte auf eine EXCEL Tabelle
'Der Namen dieser Tabelle ändert sich jedoch immer wieder
'Um die manuellen Anpassungen zu umgehen werden diese automatisch upgedatet
'
'Dazu wird eine INI Datei angelegt in welcher die alte Verknüpfungsdatei gespeichert wird
'Existiert noch keine, wird eine INI Datei angelegt mit dem Bezug auf die AKTUELLE ARBEITSMAPPE
'Die Objecte in der Präsentation MÜSSEN beim erstmaligen Start also auf
'die aktuell geöffnete Mappe mit diesem Makro verweisen
'
'Beim starten wird dann gefragt ob auf eine neue Mappe Bezug genommen werden soll,
'der Benutzer wählt eine neue Datei aus ( Diese Datei MUSS Identisch sein mit der Originaldatei )
'd.h. die ursprünglich erstellte Datei darf nur unter einem anderen Namen gespeichert werden !!!
'Die Objectnamen dürfen NICHT geändert werden !!!
'
'Die Master-Tabelle kann kaskadieren nach dem Vater - Sohn Prinzip
'Aus der Sohn-Tabelle können weitere Objecte in die Präsentation kopiert werden
'wenn die Objecte mit den neuen Werten auf diese Sohn Tabelle verweisen
'werden diese auch aktualisiert.
'Bei einem Verweis auf die VATERTABELLE werden die Objecte in der Präsentation
'die in der Sohn-Tabelle vorhanden sind, nicht mehr dargestellt!!
'*************************
'Integer Delaration
Dim i As Integer, Qe As Integer
'Object Deklaration
Dim ppApp As Object, ppPres As Object, sh As Object
'String Deklaration
Dim ppFile As String, iniFile As String
Dim LinkFile As String, oldLinkfile As String, NewLinkfile As String
Dim tmpLink As String, onlyOldFileName As String, onlyNewFileName As String, cBSl As Integer
'Variablen füllen
NewLinkfile = ""
'Prüfen ob PP-Datei vorhanden
```

```
ppFile = ThisWorkbook.Path & ppPresName
If Dir(ppFile) = "" Then
    Beep
    Qe = MsgBox("Die Datei " & ppFile & " existiert nicht!", vbCritical + vbOKOnly, "Datei Fehler")
    Exit Sub
End If
'Zwischenspeichern des Namens für die Quelldatei
iniFile = ThisWorkbook.Path & LinkIni
'Prüfen ob INI Datei vorhanden
If Dir(iniFile) = "" Then
  Qe = MsgBox("Die Datei " & Chr$(13) & iniFile & Chr$(13) & "wurde noch nicht definiert," & Chr$(13) & _
    "Es wird eine neue " & Chr$(13) & LinkIni & Chr$(13) & "erstellt mit der Quelle zu " & Chr$(13) & _
    ThisWorkbook.FullName, vbInformation + vbOKCancel, "Source Fehler")
  If Qe = vbCancel Then
    Qe = MsgBox("Das Erstellen der Datei " & Chr$(13) & _
      LinkIni & Chr$(13) & _
      " wurde abgebrochen," & Chr$(13) & _
      "Das Makro zum Starten der Präsentation wird " & Chr(13) & _
      "gestoppt und die PP Links nicht upgedatet !", vbInformation + vbOKOnly, "Source Fehler")
      Exit Sub
  End If
  'Erstellen einer neuen Link.ini
  Open iniFile For Output As #1
    'Schreiben der aktuell geöffneten Datei als Verknüpfung
    Print #1, ThisWorkbook.FullName
  Close #1
End If
'Schliessen einer eventuell geöffneten INI-Datei
Close #1
'Der Speicherort der INI Datei wird in der Const LiniIni definiert
Open iniFile For Input As #1
Do While Not EOF(1)
  'Einlesen der SourceQuelle für die Präsentation
  Input #1, oldLinkfile
Loop
'Schliessen der Datei
Close #1

'Abfrage ob neue Verknüpfungdatei definiert werden soll
Qe = MsgBox("Die aktuelle Verknüpfungdatei von " & Chr$(13) & ppPresName & Chr$(13) & _
  "ist derzeit die Datei " & Chr$(13) & _
```

```
    oldLinkfile & "." & Chr$(13) & _
    "Soll die Verknüpfungsdatei geändert werden ?", vbQuestion + vbYesNo, "Source Definition")
If Qe = vbYes Then
    'Wenn ja
    NewLinkfile = Application.GetOpenFilename("XLS Dateien (*.xls),", True, "Neue Verknüpfungsdatei auswählen", "Übernehmen", False)
    'sicherheitsabfrage
    Qe = MsgBox("Soll die Datei " & Chr$(13) & NewLinkfile & Chr$(13) & "als neue Verknüpfung definiert werden?", _
        vbQuestion + vbOKCancel, "Source Definition")
    If Qe = vbNo Then
        'Verwendung der bisherigen Datei um Update der Präsentation
        Qe = MsgBox("Die Definition der Datei" & Chr$(13) & _
            NewLinkfile & Chr$(13) & _
            " wurde abgebrochen," & Chr$(13) & _
            "Es wird die alte Datei " & oldLinkfile & "verwendet !", vbInformation + vbOKOnly, "Source Definition")
    Else
        'Neue Linkdatei wird geschrieben und zum Update verwendet
        Open iniFile For Output As #1
        'Schreiben der aktuell geöffneten Datei als Verknüpfung
            Write #1, NewLinkfile
        Close #1
    End If
End If
Set ppApp = CreateObject("PowerPoint.Application")
ppApp.Visible = msoTrue
Set ppPres = ppApp.Presentations.Open(ppFile)
'---------------------
'Verknüpfungen updaten
'Wenn die Verknüpfungdatei nicht geändert wurde
'werden nur die Werte aktualisiert
If NewLinkfile = "" Then
    For i = 1 To ppPres.Slides.Count
        For Each sh In ppPres.Slides(i).Shapes
            If sh.Type = msoLinkedOLEObject Then
                With sh.LinkFormat
                    .Update
                End With
            End If
        Next
    Next i
Else
    'Die Verknüpfungsdatei wurde geändert
```

```
    'Dazu muss der Filenamen extrahiert werden
    'den die directen Object Bezüge müssen ebenfalls angepasst werden
    'Variante für alle Excel Versionen
    cBSl = 0
    For i = Len(oldLinkfile) To 1 Step -1
       If Mid(oldLinkfile, i, 1) = "\" Then
          onlyOldFileName = Right(oldLinkfile, Len(oldLinkfile) - i)
          Exit For
       End If
    Next i
    cBSl = 0
    For i = Len(NewLinkfile) To 1 Step -1
       If Mid(NewLinkfile, i, 1) = "\" Then
          onlyNewFileName = Right(NewLinkfile, Len(NewLinkfile) - i)
          Exit For
       End If
    Next i
    For i = 1 To ppPres.Slides.Count
       For Each sh In ppPres.Slides(i).Shapes
          If sh.Type = msoLinkedOLEObject Then
             With sh.LinkFormat
                'Externen Filebezug updaten
                tmpLink = Replace(.SourceFullName, oldLinkfile, NewLinkfile, 1)
                'Updaten der direkten Object Bezüge
                tmpLink = Replace(tmpLink, onlyOldFileName, onlyNewFileName, 1)
                .SourceFullName = tmpLink
                .Update
             End With
          End If
       Next
    Next i
End If
'---------------------
ppPres.SlideShowSettings.Run
'ppApp.Quit
<FONT COLOR=#
```

--- WORD ---

Navigieren, Markieren, Texteinfügen, Maximieren von Word

```
' jumping inside Word

WordApp.Selection.Goto what:=wdGoToPage, which:=wdGoToNext ' jump to next page
WordApp.Selection.Goto what:=wdGoToPage, which:=wdGoToPrevious ' jump to previous page

    Selection.MoveDown Unit:=wdScreen, Count:=1
    Selection.MoveUp Unit:=wdLine, Count:=1
    Selection.MoveUp Unit:=wdScreen, Count:=1
    Selection.MoveDown Unit:=wdLine, Count:=1


WordApp.Selection.EndKey Unit:=wdLine ' jump to end of first line

' marking

    WordApp.Selection.MoveLeft Unit:=wdWord, Count:=1, Extend:=wdExtend ' mark full word before (left to the) actual cursor position

' jump to begin of word-document

    WordApp.Selection.HomeKey Unit:=wdStory

' transfering text to Word

    WordApp.Selection.TypeText (TEXT_TRANSFER)

' bringing Word to front

    ' --- 1. possibility ---
      Application.WindowState = xlMinimized ' Excel minimize

    ' --- 2. possibility ---
      AppActivate (ActiveDocument)

    ' --- 3. possibility ---
      If WordApp.Application.WindowState <> wdWindowStateNormal Then
          WordApp.Application.WindowState = wdWindowStateMinimize
          WordApp.Application.WindowState = wdWindowStateMaximize
        Else
          WordApp.Application.WindowState = wdWindowStateMinimize
          WordApp.Application.WindowState = wdWindowStateMaximize
```

End If

Textfeld einfügen

Hier kommt die neue Variante für Seibersdorf

Dim MyShape as Word.Shape

```
' ------------------------------------------------------
' Here comes the replacement
                Set MyShape = WordApp.ActiveDocument.Shapes.AddTextbox(msoTextOrientationHorizontal, 70, 40, 700, 40)
                MyShape.TextFrame.TextRange.Text = Pic_File
                MyShape.TextFrame.TextRange.ParagraphFormat.Alignment = wdAlignParagraphCenter
                MyShape.Fill.Visible = msoFalse
                MyShape.LINE.Visible = msoFalse
' End of the new code
' ------------------------------------------------------
```

Hier der Code eines anderen Users, der so direkt nur in Word läuft:

```
Sub TextfelderAufErsterUndZweiterSeiteEinfuegen()
Dim wdDoc As Word.Document
Dim wdRange As Word.Range
Dim wdShape As Word.Shape
Dim lngEinfuegenAufSeite As Long
Dim rngSeite As Word.Range

Set wdDoc = Documents.Add(Template:="Normal", NewTemplate:=False, DocumentType:=0)

Set rngSeite = wdDoc.Range.Goto(What:=wdGoToPage, Which:=wdGoToNext, Name:=1)
Set wdShape = wdDoc.Shapes.AddTextbox(msoTextOrientationHorizontal, 50, 0, 225, 75, Anchor:=rngSeite)

wdShape.TextFrame.TextRange.Text = "Test"

Set wdRange = wdDoc.Range(wdDoc.Range.End - 1, wdDoc.Range.End)

wdRange.InsertBreak Type:=wdPageBreak

lngEinfuegenAufSeite = 2 'nicht größer als Anzahl Seiten
```

```
Set rngSeite = wdDoc.Range.Goto(What:=wdGoToPage, Which:=wdGoToNext, Name:=lngEinfuegenAufSeite)

Set wdShape = wdDoc.Shapes.AddTextbox(msoTextOrientationHorizontal, 50, 0, 225, 75, Anchor:=rngSeite)
wdShape.TextFrame.TextRange.Text = "Text auf der 2. Seite"

wdDoc.Saved = True

End Sub
```

Worddatei öffnen - mit Passwort speichern - vermailen

```
' Variablen für Datei
    Public KOPIE As Workbook
    Public DATEINAME As String ' nur der Dateiname - zB Test.xlsx
    Public DATEIORIG As String ' der Dateiname im Tempordner inkl. Pfad - zB C:\Users\Me\AppData\Local\Temp\Test.xlsx
    Public DATEIKOPIE As String ' gleich wie DATEIORIG aber mit dem Interims-Unterstrich - zB C:\Users\Me\AppData\Local\Temp\Test_.xlsx
    Public EMAIL As String ' Die Emailadresse des Empfängers
    Public PASSWORT As String ' Das Passwort für die Exceldatei
    Public PAUSENMODUS ' 1=Pausenmodus =Bearbeitenmodus ist an - 0 oder leer: Standardmodus für Emailversand


' Allgemeine Funktion zum Finden der letzten Zelle einer Tabelle
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)
```

```
  MERKER = Application.ScreenUpdating
  Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
   Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

  Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

  ' letzte Zeile mit Daten herausfinden
  LastRowWithData = ExcelLastCell.Row
  Row = ExcelLastCell.Row
  Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
    Row = Row - 1
  Loop
  LastRowWithData = Row

  ' letzte Spalte mit Daten herausfinden
  LastColWithData = ExcelLastCell.Column
  Col = ExcelLastCell.Column
  Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
    Col = Col - 1
  Loop
  LastColWithData = Col

  Set LETZTEZELLE = TheSheet.Cells(Row, Col)
  Application.ScreenUpdating = MERKER

End Function

Sub Mail_I()

' Dies ist der Code, wenn man auf den MAILEN-Button klickt, um die aktuelle Mappe per Mail zu versenden
' Der Code macht den ersten Teil des Emailversandes: er speichert die aktuelle Datei als temporäre Kopie in den TEMP-Ordner
' Dann übergibt er an das Fenster mit den Kunden und den Emailadressen und dem Passwort der Klienten

' Fehlerüberprüfung (wenn man zB auf das Mail-Icon klickt, aber gar keine Arbeitsmappe offen ist)
   On Error GoTo FEHLER

' Festlegen des reinen Dateinamens
   DATEINAME = ActiveWorkbook.Name

' Festlegen der originalen Datei im Temp-Ordner
   DATEIORIG = Environ("TEMP") & "\" & ActiveWorkbook.Name
```

```
' Prüfen, ob Datei schon gespeichert worden ist als Exceldatei
   If InStr(DATEIORIG, ".xl") = 0 Then DATEIORIG = DATEIORIG & ".xlsx" ' falls nicht, mach sie zu einer Standard-Exceldatei

' Festlegen der temporären Datei (sie muss einen Unterstrich haben, da man nicht zwei Dateien mit gleichem Namen gleichzeitig öffnen kann)
   DATEIKOPIE = Replace(DATEIORIG, ".xl", "_.xl")

' Löschen eventueller alten Dateien im TEMP-Ordner von früheren Emailversandvorgängen
   If Len(Dir(DATEIORIG)) > 0 Then Kill DATEIORIG
   If Len(Dir(DATEIKOPIE)) > 0 Then Kill DATEIKOPIE

' Speichern der Datei als temporäre Kopie
   ActiveWorkbook.SaveCopyAs Filename:=DATEIKOPIE

' DSGVO-Mailer-Fenster einblenden
   Windows("Excel-DSGVO-Mailer.xlsm").Visible = True
   Workbooks("Excel-DSGVO-Mailer.xlsm").Activate

   Exit Sub

FEHLER:
   MsgBox "Der Befehl kann nicht ausgeführt werden - vermutlich gibt es keine Arbeitsmappe"

End Sub

Sub MAIL_II()

' Dies ist der zweite Teil zum Vermailen
' er wird automatisch dann ausgeführt, wenn man in der Liste der Kunden eine Zeile mit einem Kunden angeklickt hat
' Achtung: Der Code muss unterscheiden, ob die Mandantenauswahl in Excel für eine Exceldatei erfolgt, oder ob der Aufruf auf Word erfolgte

' Variablen für WORD-Datei
   Dim WORDDATEIKOPIE As String ' Pfad und Name der Word-Dateikopie
   Dim WORDDATEIORIG As String 'Pfad und Name der originalen Word-Datei
   Dim WORDDATEINAME As String ' Der reine Dateiname für das Emailbetreff
   Dim MYWORD As Object ' Word-Application

' Variablen für Email
   Dim App, Itm

' Auslesen der Daten des gewählten Klienten

   EMAIL = Cells(ActiveCell.Row, 6)
   PASSWORT = Cells(ActiveCell.Row, 7)
```

```
' Prüfen, ob der Aufruf von Word aus erfolgte => dann zum ganz unten befindlichen Code für Word verzweigen
   If GetSetting(appname:="DSGVO", section:="Excel", Key:="Word") = 1 Then GoTo WORDAUFRUF

' Bildschirmaktualisierung aufheben
   Application.ScreenUpdating = False

' Fehler abfangen
   On Error GoTo FEHLER

' Öffnen der temporären Kopie
   Set KOPIE = Workbooks.Open(DATEIKOPIE, False, False) ' Verknüpfungen nicht aktualisieren (false) und Datei nicht nur lesend öffnen (false)

' Speichern der Kopie mit Passwort
   Application.DisplayAlerts = False ' Nicht warnen beim Überschreiben
   KOPIE.SaveAs Filename:=DATEIKOPIE, Password:=PASSWORT, WriteResPassword:="", ReadOnlyRecommended:=False, CreateBackup:=False
   KOPIE.Close
   Application.DisplayAlerts = True ' Wieder normal warnen beim Überschreiben

' Umbenennen der Kopie auf Namen ohne Unterstrich
   Name DATEIKOPIE As DATEIORIG

' E-Mailprogramm ansteuern
   On Error GoTo KEINOUTLOOK
   Set App = CreateObject("Outlook.Application")
   Set Itm = App.CreateItem(0)
   On Error GoTo 0

      With Itm
         .Subject = DATEINAME
         .to = EMAIL
         '.Cc =
         .Attachments.Add (DATEIORIG)
         .display
      End With

   Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunden warten
   Itm.display

   Set App = Nothing
   Set Itm = Nothing

' Ausblenden des DSGVO-Mailers
```

```
    Windows("Excel-DSGVO-Mailer.xlsm").Visible = False
    Application.ScreenUpdating = True

    Exit Sub

FEHLER:
    Application.ScreenUpdating = True
    Exit Sub

KEINOUTLOOK:
    MsgBox "FÜGEN SIE DIE DATEI BITTE MANUELL IN EINE E-MAIL EIN."
    Application.ScreenUpdating = True
    Exit Sub


' -------------------
' --- WORD-AUFRUF ---
' -------------------

' Dies ist der Code, der abgearbeitet wird, wenn die Excelpasswortliste von Word aufgerufen wurde

WORDAUFRUF:

' Zurücksetzen des Aufrufparameters in der Registrierung
    SaveSetting "DSGVO", "Excel", "Word", "0"

' Auslesen der Dateipfade
    WORDDATEIKOPIE = GetSetting(appname:="DSGVO", section:="Excel", Key:="Worddatei")
    WORDDATEIORIG = Replace(WORDDATEIKOPIE, "_.docx", ".docx")
    WORDDATEINAME = Mid(WORDDATEIORIG, InStrRev(WORDDATEIORIG, "\") + 1, 99)

' Öffnen der Kopie der Worddatei
    Set MYWORD = CreateObject("Word.Application") ' starte MS Word
    With MYWORD
      .Visible = True
      .Documents.Open Filename:=WORDDATEIKOPIE
      .Activedocument.SaveAs2 Filename:=WORDDATEIORIG, Password:=PASSWORT
      .Activedocument.Close
      .Application.Quit
    End With

' Umbenennen der Worddateikopie in den originalen Dateinamen
'    Name WORDDATEIKOPIE As WORDDATEIORIG
```

```
' E-Mailprogramm ansteuern
   On Error GoTo KEINOUTLOOK
   Set App = CreateObject("Outlook.Application")
   Set Itm = App.CreateItem(0)
   On Error GoTo 0

      With Itm
         .Subject = WORDDATEINAME
         .to = EMAIL
         '.Cc =
         .Attachments.Add (WORDDATEIORIG)
         .display
      End With

   Application.Wait Now + TimeSerial(0, 0, 1) ' 1 Sekunde warten
   Itm.display

   Set App = Nothing
   Set Itm = Nothing


' Schließen des DSGVO-Mailers
   Application.WindowState = -4137
   Application.Quit

End Sub
```

Word aktivieren

um mittels SENDKEY zB eine andere Anwendung fernsteuern zu können, muss man zuerst den Focus auf die andere Anwendung setzen - und danach meist wieder zurück zu Excel mit:

AppActivate "Microsoft Excel"

`Ein reines Word ohne Vorlage kann man aktivieren mit`

AppActivate "Microsoft Word"

Ist dort jedoch ein Dokument offen, geht es so:

```
AppActivate "Dokument2 - Microsoft Word"
```

Lösung für IAEA von mir

```
' ------------------------------------------------------------
' bringing Excel to front
' ------------------------------------------------------------

       WordApp.Selection.HomeKey Unit:=wdStory
       Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second
       'WordApp.Application.WindowState = wdWindowStateMinimize
       AppActivate (File_PsWin)
       Application.WindowState = xlMinimized
       Application.WindowState = xlMaximized
       Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second

       Workbooks(File_PsWin).Sheets("Wordreport").Activate

' ------------------------------------------------------------
' bringing Word to front
' ------------------------------------------------------------

'    ' --- 1. possibility --- minimizing Excel (secure)
'    '   Application.WindowState = xlMinimized
'
'    ' --- 2. possibility --- activating Word (insecure)
'    '       AppActivate (ActiveDocument)
'
'    ' --- 3. possibility --- minimize and maximize (or normalize) Word (secure)
'       If WordApp.Application.WindowState <> wdWindowStateNormal Then
'            WordApp.Application.WindowState = wdWindowStateMinimize
'            WordApp.Application.WindowState = wdWindowStateMaximize
'         Else
'            WordApp.Application.WindowState = wdWindowStateMinimize
'            WordApp.Application.WindowState = wdWindowStateMaximize
'       End If
```

Word-Dokument von Excel aus steuern

```
Sub Word_Dokument_von_Excel_aus_steuern()
Dim myWord As Object
'Fehlerroutine für die Objectabfrage aktivieren
On Error Resume Next
'Abfragen einer besthenden WORD-Instanz um wiederholtes starten zu verhindern
'9 = Word 2000, 10 = Word XP
Set myWord = GetObject("Word.Application.10")
If Err.Number <> 0 Then
    'Fehlervariable leeren wenn Instanz noch nicht besteht
    Err.Clear
    'Zuweisung der Instanz
    Set myWord = CreateObject("Word.Application.10")
    'Instanz öffnen
    'Um das ganze etwas im Hintergrund laufen zu lassen
    'kann man den Status "wdWindowStateMinimize" verwenden
    myWord.Visible = True: objWW.WindowState = wdWindowStateMaximize
Else
    'Instanz besteht bereits
    myWord.Activate
    'Instanz in der Vordergrund bringen oder
    'mit "wdWindowStateMinimize" im Hintergrund ausführen
    myWord.Visible = True: objWW.WindowState = wdWindowStateMaximize
End If
'Hier muss der der Dateiname stehen der verwendet werden soll
'Es sollte aber eine Dokumentvorlage verwendet werden
'um keine Änderungen an den Textmarken beim einfügen zu verursachen
myWord.Application.Documents.Open "C:\Test.doc"
'Die Textmarken "a1, a2, a3" müssen im Dokument bereits bestehen
'Dann werden nach dem öffnen des Dokuments die Werte von Tabelle1
'A1, B1 und C1 in die jeweiligen Textmarken geschrieben
myWord.ActiveDocument.Bookmarks("a1").Range.Text = Worksheets("Tabelle1").Range("A1")
myWord.ActiveDocument.Bookmarks("a2").Range.Text = Worksheets("Tabelle1").Range("B1")
myWord.ActiveDocument.Bookmarks("a3").Range.Text = Worksheets("Tabelle1").Range("C1")
'Das aktive WordDokument drucken
myWord.ActiveDocument.PrintOut
'Dokument schliessen ohne speichern
myWord.ActiveDocument.Close savechanges:=False
'---
'Speichern mit fixem Namen
'myWord.ActiveDocument.SaveAs Filename:="DokumentName", FileFormat:=wdFormatDocument
'Speichern mit Variable
'myWord.ActiveDocument.SaveAs Filename:=Variable, FileFormat:=wdFormatDocument
```

```
'---
'WORD-Instanz schliessen
myWord.Application.Quit (True)
'Variable leeren
Set myWord = Nothing
End Sub
```

Diagramme von Excel nach Word kopieren + Textfelder in Word + Kopf und Fußzeile + Bilder einfügen

Dies ist der Code, den ich für Seibersdorf geschrieben habe

' The code in this module was first programmed by Stefan Wenninger / www.simplesoft.at from July - September 2011.
' Contrary to revised program-code by Stefan Wenninger this code is not commented with his usual commenting lines like: ' NEW: 2012_07_02_008 Wenninger ....

```
' ***************************************
' ---------------------------------------
' defining public varibales
' ---------------------------------------
' ***************************************

    ' Filevariables

    Public File_PsWin As String ' Filename of PsWin

'    Public File_CO As String ' Countrates_Overview-Sourcefile with charts - is not needed any longer
    Public File_APM As String ' APM_Investigator-Sourcefile with charts
    Public File_APM_MP As String ' APM_vs_MP_Data-Sourcefile with charts
    Public File_Hydride As String ' Hydride_Correction-Sourcefile with charts
    Public File_Error As String ' Error_Budgets-Sourcefile with charts
    Public File_Mixing As String ' Mixing Particles-Sourcefile with charts

    ' Variables for / in MS Word

    Public Wordtemplate As String ' Path and filename of the Word-template
    Public WordApp As Word.Application ' for remote control of MS Word
    Public WordDoc As Word.Document ' for remote control of a certain Word document
```

```
    Public MyShape As Word.Shape
    Public TextTransfer As String ' for transferring text like sample-number to Word-Report
    Public Y_Axis_Maximum As Double ' for setting maximum of y-axis in charts - especially to reset it to original value after removing the backup-data

' ----------------------------------------------------

' ****************************************
' ----------------------------------------
        Sub Word_Report()
' ----------------------------------------
' ****************************************

' This is the procedure that is called-up by pressing the WORD-REPORT-Button in Sheet WORDREPORT


' --------------------
' Setting variables
' --------------------

    File_PsWin = ActiveWorkbook.Name
    TEXT_TRANSFER = Worksheets("Report").Range("C6") ' sample-number

' defining the variable with full path and filename for the Word-Template

    Wordtemplate = Sheets("Wordreport").Range("C3") & "\" & Sheets("Wordreport").Range("L3") ' f.e.: C:\Analysis IAEA\Reports\Cover letter
template.doc

' -------------------------
' preparing MS Word template
' -------------------------

' open the MS Word template

    Set WordApp = CreateObject("Word.Application") ' starts MS Word
    With WordApp
        .Visible = True
        Set WordDoc = .Documents.Open(Wordtemplate) ' declares and opens the desired Word document
    End With
```

```
' jump to begin of word-document (Word usually starts there anyway but some users use the "jump back to my last edited
position"/WordHistory.GoBack-function when opening a new document)

    WordApp.Selection.HomeKey Unit:=wdStory

' transfering sample-number to MS Word

    ' 1.) in the title of the first page

    WordApp.Selection.EndKey Unit:=wdLine ' jump to end of first line
    WordApp.Selection.MoveLeft Unit:=wdWord, Count:=1, Extend:=wdExtend ' mark final word (= the default "XXXXXXXXX")
    WordApp.Selection.TypeText (TEXT_TRANSFER) ' inserting text with sample-nr.

    ' 2.) in the header for all pages
    WordApp.ActiveWindow.ActivePane.View.SeekView = wdSeekCurrentPageHeader ' change to header-view
    WordApp.Selection.EndKey Unit:=wdLine ' jump to the end of the header
    WordApp.Selection.MoveLeft Unit:=wdWord, Count:=1, Extend:=wdExtend ' mark the 'xxx'-dummy at the end
    WordApp.Selection.TypeText (TEXT_TRANSFER) ' inserting text with sample-nr.
    WordApp.ActiveWindow.ActivePane.View.SeekView = wdSeekMainDocument ' close header-view and return to normal view


' ---------------------------------------------------------------
' generating Excel-Reports and transfer their charts to MS Word
' ---------------------------------------------------------------

    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A11").Value = True Then ' when CHECKBOX "ACTIVE" in CELL B10 is checked
        If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating Countrates Overview Report"
        Report_Countrates_Overview
    End If
    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A41").Value = True Then
        If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating APM Investigator Report"
        Report_APM_Investigator
    End If
    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A71").Value = True Then
        If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating APM VS MP Report"
        Report_APM_VS_MP
    End If
```

```
    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A101").Value = True Then
       If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating Hydride Correction Report"
       Report_Hydrid_Correction
    End If
    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A131").Value = True Then
       If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating Error Budgets Report"
       Report_Error_Budgets
    End If
    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A161").Value = True Then
       If Workbooks(File_PsWin).Sheets("Wordreport").Range("J5").Value = True Then MsgBox "Creating Mixing Particles Report"
       Report_Mixing_Particles
    End If

' ------------------------------------------------------------
' Inserting pictures in MS Word
' ------------------------------------------------------------

    If Workbooks(File_PsWin).Sheets("Wordreport").Range("A4").Value = True Then Insert_Pictures

' ------------------------------------------------------------
' bringing Excel to front
' ------------------------------------------------------------

        WordApp.Selection.HomeKey Unit:=wdStory
        Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second
        'WordApp.Application.WindowState = wdWindowStateMinimize
        AppActivate (File_PsWin)
        Application.WindowState = xlMinimized
        Application.WindowState = xlMaximized
        Application.Wait Now + TimeSerial(0, 0, 1) ' wait 1 second

        Workbooks(File_PsWin).Sheets("Wordreport").Activate

' ------------------------------------------------------------
' bringing Word to front
' ------------------------------------------------------------

'    ' --- 1. possibility --- minimizing Excel (secure)
```

```
'    '   Application.WindowState = xlMinimized
'
'    ' --- 2. possibility --- activating Word (insecure)
'    '       AppActivate (ActiveDocument)
'
'    ' --- 3. possibility --- minimize and maximize (or normalize) Word (secure)
'       If WordApp.Application.WindowState <> wdWindowStateNormal Then
'            WordApp.Application.WindowState = wdWindowStateMinimize
'            WordApp.Application.WindowState = wdWindowStateMaximize
'          Else
'            WordApp.Application.WindowState = wdWindowStateMinimize
'            WordApp.Application.WindowState = wdWindowStateMaximize
'       End If




End Sub

' ****************************************
' ----------------------------------------
Sub Report_Countrates_Overview()
' ----------------------------------------
' ****************************************

' this is a sub-procedure to the procedure 'Sub WORD_Report()' and contains the code for
' generating the countrates-overview-report and transferring its charts to MS Word

' as two charts are transferred to a different part of the Word document, the Countrates-Overview-Report is genereated twice

    Application.DisplayAlerts = False ' deactivating the "negative or zero value"-warning during the VBA code

' switching to the Sheet "Report" with the "Countrates Overview"-button CommandButton27_Click

    Workbooks(File_PsWin).Sheets("Report").Activate

' ----------------------------------------------------------------------
```

```
' generating 'countrates-overview'-Excelfile for the first two charts
' ------------------------------------------------------------------


' --- Code of CommandButton27_Click (the procedure of this button cannot be called directly as it is not in a module)

    Dim ID As Variant
    Dim series As Variant
    Dim answer As Variant

    ' If Worksheets("ToolBox").Ignore_Errors = True Then On Error Resume Next
    ID = ""
    series = Workbooks(File_PsWin).Worksheets("Report").Range("C6")
    Countrates_overview_new_1280
    answer = Assign_Chart_Titles(ID, series)

' --- End of Code of CommandButton27_Click

' saving filename of the countrate-overview-file (in case it is not Countrates_Overview.xls but books2 or books3 ...)

' File_CO = ActiveWorkbook.Name ' is not needed any longer

' --- Checking if a chart is activated in Sheet WORDREPORT / column c but there is no name in column D

For t = 11 To 34
    'MsgBox Workbooks(File_PsWin).Sheets("Wordreport").Cells(t, 2).Value & Workbooks(File_PsWin).Sheets("Wordreport").Cells(t, 4).Value
    If Workbooks(File_PsWin).Sheets("Wordreport").Cells(t, 3).Value = True And Workbooks(File_PsWin).Sheets("Wordreport").Cells(t, 4).Value = ""
Then
        MsgBox "In Sheet 'WORDREPORT' in cell 'D" & t & "' is no report-sheetname. Please insert a report-sheetname or uncheck checkbox in cell 'C"
& t & """
        End
    End If
Next t

' ----------------------------------------
' transferring CO-1.Chart
' ----------------------------------------
```

```
If Workbooks(File_PsWin).Sheets("Wordreport").Range("C11").Value = True Then ' if checkbox of chart 1 is activated

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D11").Value).Activate ' activate sheet with 1. chart

    Call Chart_Optimize(11) ' removing backup-data / trend-line / switching to log/lin-View / resetting y-scale - variable: line-number in sheet "Wordreport" with variables for this chart

    WordApp.ActiveDocument.Bookmarks("OC_Chart1_2").Select ' jumping to the desired position in the Word document (that has to be set in Word-cocument in menu INSERT with its function BOOKMARK)

    Call Chart_copy_paste(380, 40, 315)  'copies and pastes the chart - variables: 1st: width of the chart - 2nd: left position - 3rd: top position in the MS Word document

End If

' ----------------------------------------------

' ----------------------------------------
' transferring CO-2.Chart          (for additional comments see also chart 1)
' ----------------------------------------

If Workbooks(File_PsWin).Sheets("Wordreport").Range("C12").Value = True Then

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D12").Value).Activate ' activate sheet with chart
    Call Chart_Optimize(12)
    WordApp.ActiveDocument.Bookmarks("OC_Chart1_2").Select
    Call Chart_copy_paste(380, 420, 315)

End If

' ----------------------------------------------
' closing excel-report-file
' ----------------------------------------------

    ActiveWorkbook.Close savechanges:=False

' ------------------------------------------------------------------------
' generating 'countrates-overview'-Excelfile again for the rest of the charts
```

```
' ------------------------------------------------------------------------

' --- Code of CommandButton27_Click (the procedure of this button cannot be called directly as it is not in a module)

    ID = ""
    series = Workbooks(File_PsWin).Worksheets("Report").Range("C6")
    Countrates_overview_new_1280
    answer = Assign_Chart_Titles(ID, series)

' --- End of Code of CommandButton27_Click

' ----------------------------------------------

' choosing the next bookmark in Word for the OC-charts 3-24

    WordApp.ActiveDocument.Bookmarks("OC_Chart3").Select


' ---------------------------------------
' transferring CO-3.Chart
' ---------------------------------------

If Workbooks(File_PsWin).Sheets("Wordreport").Range("C13").Value = True Then

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D13").Value).Activate
    Call Chart_Optimize(13)
    Call Chart_copy_paste(780, 30, 67)

    ' preparing a new page for next OC-chart if needed

    If Workbooks(File_PsWin).Sheets("Wordreport").Range("C14").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C15").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C16").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C17").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C18").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C19").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C20").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C21").Value = True Or _
```

```
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C22").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C23").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C24").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C25").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C26").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C27").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C28").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C29").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C30").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C31").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C32").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C33").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C34").Value = True Then ' if any other chart of this area in the word-document is
activated...
          WordApp.Selection.InsertBreak Type:=wdPageBreak ' ... create a new page for it
    End If
End If

' ----------------------------------------

' ---------------------------------------
' transferring CO-4.Chart
' ---------------------------------------

If Workbooks(File_PsWin).Sheets("Wordreport").Range("C14").Value = True Then

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D14").Value).Activate
    Call Chart_Optimize(14)
    Call Chart_copy_paste(780, 30, 67)

    ' preparing a new page for next OC-chart if needed

    If Workbooks(File_PsWin).Sheets("Wordreport").Range("C15").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C16").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C17").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C18").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C19").Value = True Or _
       Workbooks(File_PsWin).Sheets("Wordreport").Range("C20").Value = True Or _
```

```
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C21").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C22").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C23").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C24").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C25").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C26").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C27").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C28").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C29").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C30").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C31").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C32").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C33").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C34").Value = True Then ' if any other chart of this area in the word-document is
activated...
            WordApp.Selection.InsertBreak Type:=wdPageBreak ' ... create a new page for it
    End If

End If

' ----------------------------------------------

' ---------------------------------------
' transferring CO-5.Chart
' ---------------------------------------

If Workbooks(File_PsWin).Sheets("Wordreport").Range("C15").Value = True Then

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D15").Value).Activate
    Call Chart_Optimize(15)
    Call Chart_copy_paste(780, 30, 67)

    ' preparing a new page for next OC-chart if needed

    If Workbooks(File_PsWin).Sheets("Wordreport").Range("C16").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C17").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C18").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C19").Value = True Or _
```

```
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C20").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C21").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C22").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C23").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C24").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C25").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C26").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C27").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C28").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C29").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C30").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C31").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C32").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C33").Value = True Or _
        Workbooks(File_PsWin).Sheets("Wordreport").Range("C34").Value = True Then ' if any other chart of this area in the word-document is
activated...
            WordApp.Selection.InsertBreak Type:=wdPageBreak ' ... create a new page for it
    End If

End If

' ..... und so gehts immer weiter

' ----------------------------------------
' transferring CO-24.Chart
' ----------------------------------------

If Workbooks(File_PsWin).Sheets("Wordreport").Range("C34").Value = True Then

    Sheets(Workbooks(File_PsWin).Sheets("Wordreport").Range("D34").Value).Activate
    Call Chart_Optimize(34)
    Call Chart_copy_paste(780, 30, 67)


End If

' ----------------------------------------------
```

```
' closing excel-report-file if selected

If Workbooks(File_PsWin).Sheets("Wordreport").Range("A10").Value = True Then
   ActiveWorkbook.Close savechanges:=False
End If

End Sub

' ****************************************
' ----------------------------------------
   Sub Insert_Pictures()
' ----------------------------------------
' ****************************************

' This procedures insertes all pictures within a preset folder

   Dim Pic_File
   Dim Pic_Folder

' Correcting Folderpaths (remove a final \)
   If Right(Workbooks(File_PsWin).Sheets("Wordreport").Range("C4"), 1) = "\" Then Workbooks(File_PsWin).Sheets("Wordreport").Range("C4") =
left(Workbooks(File_PsWin).Sheets("Wordreport").Range("C4"), Len(Workbooks(File_PsWin).Sheets("Wordreport").Range("C4")) - 1)
   If Right(Workbooks(File_PsWin).Sheets("Wordreport").Range("L4"), 1) = "\" Then Workbooks(File_PsWin).Sheets("Wordreport").Range("L4") =
left(Workbooks(File_PsWin).Sheets("Wordreport").Range("L4"), Len(Workbooks(File_PsWin).Sheets("Wordreport").Range("L4")) - 1)

   WordApp.ActiveDocument.Bookmarks("Pictures").Select
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.InsertBreak Type:=wdPageBreak ' make a page-break

   ' make 5 line-breaks
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.TypeParagraph ' line-break
   WordApp.Selection.MoveUp Unit:=wdLine, Count:=5 ' move back to first line on page
```

```
    Pic_Folder = Workbooks(File_PsWin).Sheets("Wordreport").Range("C4") & "\" & Sheets("Wordreport").Range("L4") & "\"

    On Error GoTo PATH_NOT_FOUND
    ChDir Pic_Folder

    ' activate Word-Template-Window (why: as deselecting the textbox with the filename cannot be done directly, we have to send the pressed ESC-Button to the Word-Window and before it must me activated)
    AppActivate (left(Workbooks(File_PsWin).Sheets("Wordreport").Range("L3"), 5)) ' The 5 letters of the Coverletter-template-filename are enough to activate its window

    ' inserting all the pictures
    With Application.FileSearch
        .NewSearch
        .LookIn = Pic_Folder
        .FileName = "*.jpg" ' oder png, bmp etc
        .SearchSubFolders = True

        If .Execute() > 0 Then
            For Each Pic_File In .FoundFiles
                WordApp.Selection.InlineShapes.AddPicture FileName:=Pic_File, LinkToFile:=False, SaveWithDocument:=True ' insert the picture
                WordApp.Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend ' select the newly inserted picture

                With WordApp.Selection.InlineShapes(1).ConvertToShape ' convert picture to free moveable shape

                    ' --- resizing ---
                     .LockAspectRatio = msoTrue ' locking heigth and width to each other
                     If .width > 700 Then .width = 700
                     If .Height > 430 Then heigth = 400

                    ' --- positioning ---
                    ' Option 1.) moving a shape can be done to an absolute position
                     .left = 420 - .width / 2 ' absolute left position
                     .top = 70 ' absolute top position

                    ' Option 2.) moving a shape can be done relatively
                    ' .IncrementLeft -35  ' move shape to the left
                    ' .IncrementTop -39.75 ' move shape to the top
                    ' .IncrementLeft 10 ' move shape to the right
```

```
            ' .IncrementTop 220 ' move shape downwards
        End With

' ------------------------------------------------------
' This was our first attempt to insert a Textbox into Word - as its creation-code "AddTextbox..." SELECTS the Textbox and
we have great troubles to unselect it, we use a new code-variant
                'WordApp.ActiveDocument.Shapes.AddTextbox(msoTextOrientationHorizontal, 70, 40, 700, 40).Select ' add a
textbox for file-name
                'WordApp.Selection.TypeText Text:=Pic_File  'Right(Pic_File, Len(Pic_File) - Len(Pic_Folder)) ' inserting
filename
                'WordApp.Selection.ShapeRange.Fill.Visible = msoFalse ' no white background for the textbox
                'WordApp.Selection.ShapeRange.LINE.Visible = msoFalse ' no frame-line for the textbox
                'WordApp.Selection.ShapeRange.TextFrame.TextRange.ParagraphFormat.Alignment = wdAlignParagraphCenter '
center filename
                'AppActivate (Left(Workbooks(File_PsWin).Sheets("Wordreport").Range("L3"), 5))
                'SendKeys "{ESC}" ' First ESC unselects the textarea
                'SendKeys "{ESC}" ' Second ESC unselect the whole textbox
                'SendKeys "{ESC}" ' Third ESC should be unnecessary but on slow machines it can replace a missing 1st or
2nd Esc
                'Application.Wait Now + TimeSerial(0, 0, Workbooks(File_PsWin).Sheets("Wordreport").Range("Q3") / 1000) '
wait 1 second for Word to really unselect the textbox
' ------------------------------------------------------

' ------------------------------------------------------
' Here comes the replacement
                Set MyShape = WordApp.ActiveDocument.Shapes.AddTextbox(msoTextOrientationHorizontal, 70, 40, 700, 40)
                MyShape.TextFrame.TextRange.Text = Pic_File
                MyShape.TextFrame.TextRange.ParagraphFormat.Alignment = wdAlignParagraphCenter
                MyShape.Fill.Visible = msoFalse
                MyShape.LINE.Visible = msoFalse
' End of the new code
' ------------------------------------------------------


        WordApp.Selection.MoveDown Unit:=wdLine, Count:=4 ' move 4 line down

        WordApp.Selection.InsertBreak Type:=wdPageBreak ' make a page-break

        ' make 5 line-breaks
        WordApp.Selection.TypeParagraph ' line-break
        WordApp.Selection.TypeParagraph ' line-break
```

```
            WordApp.Selection.TypeParagraph ' line-break
            WordApp.Selection.TypeParagraph ' line-break
            WordApp.Selection.TypeParagraph ' line-break
            WordApp.Selection.MoveUp Unit:=wdLine, Count:=5 ' move back to first line on page


          Next Pic_File
        End If

    End With

    Exit Sub

PATH_NOT_FOUND:

    MsgBox "The defined picture-folder " & Pic_Folder & " cannot be found." & vbCrLf & vbCrLf & _
    "Please set the right folder in cells 'C4' and 'L4'."

End Sub

' ********************************************
' ******* HILFSROUTINEN ***************
' ********************************************

Sub Chart_Optimize(LINE As Integer)

    ' help-routine for Modul Word__Report, that optimizes Excel-Charts before transfering them to MS Word

    ' preventing an error-message:
    ' 1.) in case there are no backup-data and trendline and user has requested to remove them
    ' 2.) and in case of a requested change to logarythmic axis-display

    On Error Resume Next

    ' saving actual y-axis-maximum
    Y_Axis_Maximum = ActiveChart.Axes(xlValue).MaximumScale

    ' deleting backup data if requested
```

```
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("G" & LINE)) = "YES" Then
        ActiveChart.SeriesCollection(2).Select
        Application.CutCopyMode = False
        Selection.Delete
    End If

    ' deleting trendline if requested
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("H" & LINE)) = "YES" Then
        ActiveChart.SeriesCollection(1).Trendlines(1).Select
        Selection.Delete
    End If

    ' resetting original Y-axis-maximum (when Y-axis-maximum is set to 'automatic' and backup-data have been removed by request, the original y-
axis-maximum can be reset)
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("I" & LINE)) = "YES" Then
        ActiveChart.Axes(xlValue).MaximumScale = Y_Axis_Maximum '  setting y-axis maximum-scale
    End If

    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("E11")) = "LIN" Then ' changing x-axis-displaymode to linear
        ActiveChart.Axes(xlCategory).ScaleType = xlLinear       ' set x-axis to linear-mode
    End If
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("E" & LINE)) = "LOG" Then ' changing x-axis-displaymode to linear
        ActiveChart.Axes(xlCategory).ScaleType = xlLogarithmic       ' set x-axis to logarythmic-mode
    End If
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("F" & LINE)) = "LIN" Then ' changing y-axis-displaymode to linear
        ActiveChart.Axes(xlValue).ScaleType = xlLinear       ' set y-axis to linear-mode
    End If
    If UCase(Workbooks(File_PsWin).Sheets("Wordreport").Range("F" & LINE)) = "LOG" Then ' changing y-axis-displaymode to linear
        ActiveChart.Axes(xlValue).MinimumScaleIsAuto = True       ' setting auto-minimum to allow zero-values for the logarythmic-scale-command in
next line
        ActiveChart.Axes(xlValue).ScaleType = xlLogarithmic       ' set y-axis to logarythmic-mode
    End If

    ' resetting the error management
    On Error GoTo 0

End Sub
Sub Chart_copy_paste(Chart_Width As Single, Chart_Left As Single, Chart_Top As Single)
```

```
' help-routine for Modul Word__Report, that copys the actual Excel-Chart and pastes it into MS Word

' --- copying chart ---

' version 1 - copying the whole chartobject (with all its database in all Excel-worksheets => large clipboard and large word-file)
'    ActiveSheet.ChartArea.Copy ' copy the chart

' version 2 - copying only the chart-graphic
ActiveChart.ChartArea.Select
ActiveChart.CopyPicture

' pasting it to Word
WordApp.Selection.Paste

' marking the inserted chart

WordApp.Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend

' formatting and positioning of the chart

' Version 1: working directly with the inserted chart being an inlineshape
'    WordApp.Selection.InlineShapes(1).Width = WordApp.Selection.InlineShapes(1).Width * Zoom_Factor ' new width
'    WordApp.Selection.InlineShapes(1).Height = WordApp.Selection.InlineShapes(1).Height * Zoom_Factor ' new height

' Version 2(better): first converting InlineShape to a normal shape, which can be positioned freely and indepentendly from any text
With WordApp.Selection.InlineShapes(1).ConvertToShape

    ' --- resizing ---
     .LockAspectRatio = msoTrue ' locking heigth and width to each other
     .width = Chart_Width

    ' --- positioning ---

    ' Option 1.) moving a shape can be done to an absolute position
     .left = Chart_Left ' absolute left position
     .top = Chart_Top ' absolute top position
```

```
    ' Option 2.) moving a shape can be done relatively
    ' .IncrementLeft -35  ' move shape to the left
    ' .IncrementTop -39.75 ' move shape to the top
    ' .IncrementLeft 10 ' move shape to the right
    ' .IncrementTop 220 ' move shape downwards
  End With

  ' unselect shape
  WordApp.Selection.Collapse
  WordApp.Selection.EndKey Unit:=wdLine ' jump to end of actual line

End Sub
```

Word-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallWord
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einem Word-Dokument aufrufen
- Ablaufbeschreibung:
    - Variablendeklaration
    - Name des Worddokumentes an String-Variable übergeben
    - Wenn die Datei nicht existiert...
    - Negativmeldung
    - Sonst...
    - Word-Instanz bilden
    - Word-Dokument öffnen
    - Word-Makro aufrufen

- Word-Instanz schließen
- Objektvariable zurücksetzen

- Code:

```
Sub CallWord()
   Dim wdApp As Object
   Dim sFile As String
   sFile = ThisWorkbook.Path & "\vb07_WordTest.doc"
   If Dir$(sFile) = "" Then
      MsgBox "Test-Word-Dokument " & sFile & " wurde nicht gefunden!"
   Else
      With CreateObject("Word.Application")
          .documents.Open sFile
          .Run "Project.Modul1.WdMeldung"
          .Quit
      End With
   End If
End Sub
```

Word Rechtschreibprüfung verwenden

```
'  Title: Use MS Word's Spell Checker in Visual Basic
'


' ------------------------------------------------------------------------

' ----------
' Function
' ----------

Public Function SpellCheck(ByVal IncorrectText$) As String
       Dim Word As Object, retText$
```

```
        On Error Resume Next

        ' Create the Object and open Word
        Set Word = CreateObject("Word.Basic")

        ' Change the active window to Word,
        'and insert the text from Text1 into Word.
        Word.AppShow
        Word.FileNew

        Word.Insert IncorrectText

        ' Runs the Speller Corrector
        Word.ToolsSpelling
        Word.EditSelectAll

        ' Trim the trailing character from the returned text.
        retText = Word.Selection$()
        SpellCheck = Left$(retText, Len(retText) - 1)

        ' Close the Document and return to Visual Basic.
        Word.FileClose 2
        Show

        ' Set the word object to nothing to liberate the'occupied memory
        Set Word = Nothing
End Function
```

Telefonnummer wählen mittels TAPI

```
' ------------------------------------------------------------------------
'
'  Title: Use TAPI32 to dial a phone number
'


' ------------------------------------------------------------------------

' ----------------------------
' Constants & API Declarations
```

```
' ----------------------------

Private Declare Function tapiRequestMakeCall& Lib "TAPI32.DLL" (ByVal DestAddress$, ByVal AppName$, ByVal CalledParty$,
ByVal Comment$)

Private Const TAPIERR_NOREQUESTRECIPIENT = -2&
Private Const TAPIERR_REQUESTQUEUEFULL = -3&
Private Const TAPIERR_INVALDESTADDRESS = -4&


' ---------
' Function
' ---------

public Sub DialNumber(strNumber As String, strLocation As String)
    Dim strBuff As String
    Dim lngResult As Long

    lngResult = tapiRequestMakeCall&(strNumber, CStr(Caption), strLocation, "")

    If lngResult <> 0 Then
        strBuff = "Error dialing number : "

        Select Case lngResult
            Case TAPIERR_NOREQUESTRECIPIENT
                strBuff = strBuff & "No Windows Telephony dialing application is running and none could be started."
            Case TAPIERR_REQUESTQUEUEFULL
                strBuff = strBuff & "The queue of pending Windows Telephony dialing requests is full."
            Case TAPIERR_INVALDESTADDRESS
                strBuff = strBuff & "The phone number is not valid."
            Case Else
                strBuff = strBuff & "Unknown error."
        End Select

        MsgBox strBuff
    End If
End Sub
```

ZELLEN - SPALTEN - ZEILEN

-→ Grundlagen RANGE und SELECTION

Range(Cells(1, 1), Cells(10, 5)).Copy ' Der Bereich A1 bis E5 wird in Zwischenablage kopiert

Colums(1).select ' markiert die Spalte A

Range(Columns(1), Columns(5)).Select ' markiert die Spalten A-E

Range(Columns(1), Columns(1).Offset(0, 10)).Select ' markiert die Spalte A und die nächsten 10

```
' den gesamten genutzten Tabellenbereich ab A1 leeren
ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
```

Range is one of the most widely used objects in Excel VBA, as it allows the manipulation of a row, column, cell or a range of cells in a spreadsheet.

When recording absolute macros, a selection of methods and properties use this object:

```
Range("A1").Select

Range("A1").FormulaR1C1 = 10
```

A generic global object known as **Selection** can be used to determine the current selection of a single or range cells.

When recording relative macros, a selection of methods and properties use this object:

```
Selection.Clear

Selection.Font.Bold = True
```

There are many properties and methods that are shared between **Range** and **Selection** objects and below are some illustrations (my choice of commonly used identifiers):

ADDRESS Property

Returns or sets the reference of a selection.

```
Sub AddressExample()
    MsgBox Selection.Address '$A$1 (default) - absolute
    MsgBox Selection.Address(False, True) '$A1 - column absolute
    MsgBox Selection.Address(True, False) 'A$1 - row absolute
    MsgBox Selection.Address(False, False) 'A1 - relative

End Sub
```

AREAS Property

Use this property to detect how many ranges (*non-adjacent*) are selected.

```
'Selects three non-adjacent ranges
Sub AreaExample()
    Range("A1:B2", E4, G10:J25").Select
    MsgBox Selection.Area.Count 'Number '3' - ranges returned
End Sub
```

The **Count** method returns the number selected as the **Areas** is a property only.

```
'Check for multiple range selection
Sub AreaExample2()
   If Selection.Areas.Count > 1 Then
        MsgBox "Cannot continue, only one range must be selected."
        Exit Sub
    End If

    [Code continues here...]
End Sub
```

Use the **Areas** property to check the state of a spreadsheet. If the system detects multiple ranges, a prompt will appear.

CELLS Property

This property can be used as an alternative to the absolute range property and is generally more flexible to work with, as variables are easier to pass into it.

There are two optional arguments:

```
Cells([row] [,column])
```

Leaving the arguments empty (*no brackets*), it will detect the current selection as the active range.

Adding an argument to either row or column with a number will refer to the co-ordination of the number passed.

Adding both arguments will explicitly locate the single cell's co-ordinate.

```
'Examples of the Cells property
Sub CellsExample()
    Cells.Clear 'clears active selection
    Cells(1).Value = "This is A1 - row 1"
    Cells(, 1).Value = "This is A1 - col 1"
    Cells(1, 1).Value = "This is A1 - explicit"
    Cells(3, 3).Value = "This is C3"
    Cells(5, 3).Font.Bold = True
End Sub
```

Variables can be passed into the **Cells** property and then nested into the **Range** object as in the following example:

```
'Two InputBoxes for rows and columns
Sub CellsExample2()
    On Error GoTo handler
    Dim intRows As Integer
    Dim intCols As Integer
    intRows = CInt(InputBox("How many rows to populate?"))
    intCols = CInt(InputBox("How many columns to populate?"))
    'starts at cell A1 to the number of rows and columns passed
    Range(Cells(1, 1), Cells(intRows, intCols)).Value = "X"
    Exit Sub
handler:
    'Error code is handled here...
End Sub
```

By wrapping a range property around two cell properties, the flexibility of passing variables becomes apparent.

```
Range(Cells(1, 1), Cells(intRows, intCols))
```

Error handlers and InputBox functions are covered later in this guide.

Column(s) and Row(s) Properties

Four properties that return the column or row number for the focused range.

The singular (***Column*** *or* ***Row***) returns the active cell's co-ordinate and the plural (***Columns*** *or* ***Rows***) can be used to count the current selections configuration.

```
Sub ColRowExample()
    MsgBox "Row " & ActiveCell.Row & _
        " : Column " & ActiveCell.Column

End Sub
```

```
Sub ColsRowsCountExample()
    MsgBox Selection.Rows.Count & " rows by " _
        & Selection.Columns.Count & " columns selected"

End Sub
```

CURRENTREGION Property

Selects from the active cell's position all cells that are adjacent (*known as a region*) until a blank row and blank column breaks the region.

Use this statement to select a region.

```
Selection.CurrentRegion.Select
```

Make sure you have some data to work with.

To select a region of data and exclude the top row for a data list:

	A	B	C
1	Customer ID	Company Name	Contact Name
2	ALWAO	Always Open Quick Mart	Melissa Adams
3	ANDRC	Andre's Continental Food Market	Heeneth Ghandi
4	ANTHB	Anthony's Beer and Ale	Mary Throneberry
5	AROUT	Around the Horn	Thomas Hardy
6	BABUJ	Babu Ji's Exports	G.K.Chattergee
7	BERGS	Bergstad's Scandinavian Grocery	Tammy Wong

Run this piece of code:

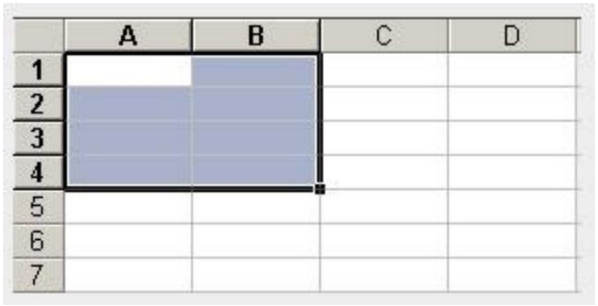
```
Sub RegionSelection()
    ActiveCell.CurrentRegion.Offset(1, 0).Resize( _
        ActiveCell.CurrentRegion.Rows.Count - 1, _
            ActiveCell.CurrentRegion.Columns.Count).Select

End Sub
```

Make sure the active cell is in the region of data you wish to capture before running the above procedure.

RESIZE Property

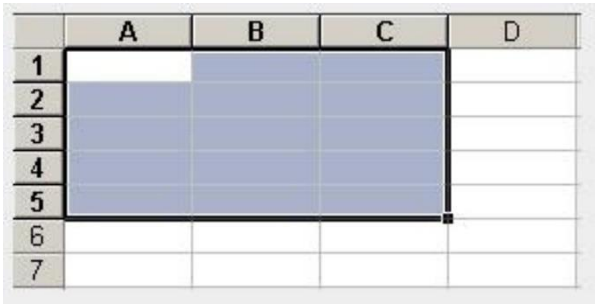
This property is useful for extending or re-defining a new size range.

To extend this range



	A	B	C	D
1				
2				
3				
4				
5				
6				
7				

by one row and one column to



use the code snippet below:

```
Sub ResizeRange()
   Dim rows As Integer
    Dim cols As Integer
    cols = Selection.Columns.Count
    rows = Selection.rows.Count
    Selection.Resize(rows + 1, cols + 1).Select

End Sub
```

Resizing a range can be increased, decreased or change the configuration (*shape*) by combining positive and negative values inside the **Resize** property's arguments.

OFFSET Property

This property is used in many procedures as it controls references to other cells and navigation.

Two arguments are passed into this property that is then compounded with either another property or a method.

```
Selection.OffSet(1, 2).Select

ActiveCell.OffSet(0, -1).Value = "X"
```

Consider referring to an offset position rather than physically navigating to it – this will speed up the execution of code particularly while iterating.

For example:

```
Sub OffSetExample1()
    Dim intCount As Integer
    Do Until intCount = 10
        ActiveCell.Offset(intCount, 0).Value = "X"
        intCount = intCount + 1
    Loop

End Sub
```

is quicker to execute than:

```
Sub OffSetExample2()
    Dim intCount As Integer
    Do Until intCount = 10
        ActiveCell.Value = "X"
        ActiveCell.Offset(1, 0).Select
        intCount = intCount + 1
    Loop

End Sub
```

Do...Loops (iterations) are covered later in this guide

The above two examples produce the same result but instead of telling Excel to move to the active cell and then enter a value, it is more efficient to refer (*or point*) to the resulting cell and remain in the same position.

- A positive value for the row argument refers to a row downwards.
- A positive value for the column argument refers to a column to its right.
- A negative value for the row argument refers to a row upwards.
- A negative value for the column argument refers to a column to its left.

Be careful to introduce error-handling procedures when working with the *'Offset'* property as if you navigate or refer to a position outside the scope of the fixed dimensions of a worksheet, this will certainly cause a run time error (See *Error Handling & Debugging*).

ACTIVATE Method

This method should not be confused with the **Select** method as commonly used in VBA.

The **Select** method means go and navigate to it.

```
Range("A1").Select.

Range("A1:C10").Select
```

The **Activate** method selects a cell within a selection.

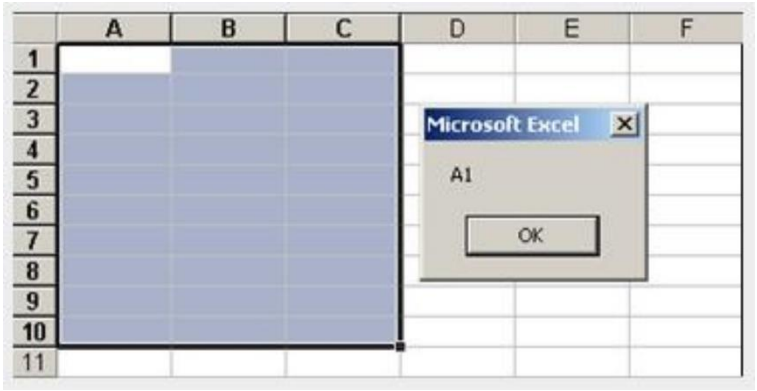
By default, in a selection of cells, the first (*top left position*) is the active cell *(white cell in a block*).

Example:

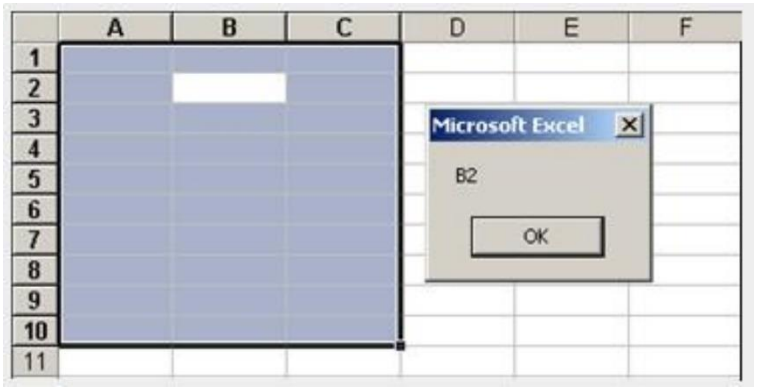
```
Sub ActivateMethodExample()
    'select a fixed range
    Range("A1:C10").Select
    MsgBox ActiveCell.Address(False, False)
    Range("B2").Activate
    MsgBox ActiveCell.Address(False, False)
End Sub
```

The above procedure selects a fixed range of cells with a message box confirming the address of the active cell. Then, using the **Activate** method, move the active cell to address B2.

From



to



CLEAR Methods

There are six variations of this method:

1. **Clear** – all attributes are cleared and reset to default
2. **ClearComments** – clear comments only
3. **ClearContents** – clear contents only (delete key command)
4. **ClearFormats** – clear formats only (revert to general format)
5. **ClearNotes** – clear comments and sound notes only
6. **ClearOutline** – clear on outlines implemented

Simply locate the object and use of the above methods:

```
'Different ways to refer to a selection
```

```
Sub ClearMethodsExamples()
    Range("A1:C10").Clear
    Selection.ClearComments
    Selection.CurrentRegion.ClearContents
    ActiveCell.ClearFormats
    Range(Cells(1, 1), Cells(5, 3)).ClearNotes
    Columns("A:E").ClearOutline
End Sub
```

CUT, COPY and PASTESPECIAL Methods

These methods simulate the windows clipboard cut, copy and paste commands.

There are a few different types of these methods where most arguments are optional and by changing the argument settings, will change the behaviour of the method.

Some examples:

```
'Simple Copy and Paste
Sub CopyPasteData1()
    Range("A1").Copy
    Range("B1").PasteSpecial xlPasteAll
End Sub
```

```
'Copy and Paste Values only (no format)
Sub CopyPasteData2()
    Range("A1").Copy
    Range("B1").PasteSpecial xlPasteValues
End Sub
```

```
'Simple Cut and Paste
Sub CutPasteData()
    Range("A1").Cut Range("B1")
End Sub
```

If the copy and cut methods omit the argument **Destination**, the item is copied to the windows clipboard.

INSERT and DELETE Methods

These methods can add or remove cells, rows or columns and is best used with other properties to help establish which action to execute.

Some examples:

```
'Inserts an entire row at the active cell
Sub InsertRow()
```

```
    ActiveCell.EntireRow.Insert 'or EntireColumn
End Sub
```

```
'Deletes an entire row at the active cell
Sub DeleteRow()
    ActiveCell.EntireRow.Delete 'or EntireColumn
End Sub
```

```
'Inserts an entire row at row 4
Sub InsertAtRow4()
    ActiveSheet.rows(4).Insert
End Sub
```

```
'Insert columns and move data to the right
Sub InsertColumns()
    Range("A1:C5").Insert Shift:=xlShiftToRight
End Sub
```

Using the SET Keyword Command

Users can create and set a range object instead and like all other object declarations, use the **Set** command (which is used for object variable declarations).

```
'Alternative way of referring to a range
Sub RangeObject()
    Dim rng As Range
    Set rng = Range("A1:B2")
    With rng
        .Value = "X"
        .Font.Bold = True
        .Borders.LineStyle = xlDouble
        'any other properties.........
    End With
    Set rng = Nothing
End Sub
```

This is an alternative way of working with the range object and is sometimes preferred as it exposes more members (properties and methods).

For example, using a **For...Loop** (*see* For...Loop *section about this control flow*), iterating in a collection is carried out by declaring and setting a variable as a **Range** object:

```
'Loops through the elements of the Range object
Sub IterateRangeObject()
    Dim r1 As Range
    Dim c As Object
    Set r1 = Range("A1:C10")
    For Each c In r1
        If c.Value = Empty Then
            c.Value = "0"
        End If
    Next c
End Sub
```

The above procedure checks each cell in a fixed range (*A1 to C10*) and determines its value, placing a 0 (*zero*) if the cell is empty.

EXAMPLES RANGE AND SHEET

Specific Range references

`Range("A1")`	Cell A1
`Range("A1:E10")`	Range A1 to E10
`[A1]`	Cell A1
`[A1:E10]`	Range A1 to E10
`ActiveCell.Range("A2")`	The cell below the active cell
`Cell(1)`	Cell A1
`Range(Cells(1,1),Cells(10,5))`	Range A1 to E10
`Range("A:A")`	Column A
`[A:A]`	Column A

`Range("5:5")`	Row 5
`[5:5]`	Row 5
`Sheets("Sheet1")`	Sheet called Sheet1
`Worksheets("Sheet1")`	Worksheets called Sheet1
`Sheets(2)`	Second worksheet in workbook
`Worksheets(3)`	Third worksheet in workbook
`Worksheets("Sheet1").Range("A1")`	Cell A1 in Sheet1
`[Sheet1].[A1]`	Cell A1 in Sheet1
`ActiveSheet.Next`	The sheet after the active sheet
`Workbook("Test")`	Workbook file called Test.xls

Alle Namen in einer Mappe entfernen

' Im Namensmanager kann man die Namen immer nur einzeln löschen - folgender Code entfernt alle:

```
Sub Namen_weg()
Dim n As Name
For Each n In ThisWorkbook.Names
  n.Delete
Next n
End Sub
```

Alle Zellen in einer Tabelle mit For Each

```
Dim ZELLE As Range

For Each ZELLE In ActiveSheet.UsedRange.Cells

   If IsDate(ZELLE) = True And ZELLE.Column <> 1 Then ZELLE = ZELLE + 1462 ' Repariert in allen Zellen außer in Spalte A das Datum nach Umstellung 1.1.1904=>31.12.1899

Next ZELLE
```

Alle Zellen in einer Spalte mit For Each

Im Gegensatz zur Usedrange-Funktion, die nicht auf das Ende der aktuellen Spalte eingeht, sondern das Ende ALLER Spalten nimmt, geht End(xlUp) zur letzten Zelle in der aktuellen Spalte mit Zell-INHALT.

```
Dim Zelle As Range
Dim Bereich As Range

Range("A2:A" & Range("A65536").End(xlUp).Row).Select  ' alle verwendeten Zellen in einer Spalte (hier A und ab Zeile 2), die einen Inhalt haben

For Each Zelle In Selection
   Zelle = Zelle * 10
Next Zelle
```

Alle Zellen in einer Zeile mit For Each

```
Dim Zelle As Range
Dim Bereich As Range

Range(Cells(2, 1), Cells(2, Range("IV2").End(xlToLeft).Column)).Select  ' alle verwendeten Zellen in Zeile 2 mit Zellinhalt (hier ab Spalte A)

For Each Zelle In Selection
   Zelle = Zelle * 10
Next Zelle
```

Alle Zellen in einem Bereich mit For Each

LÖSUNG 1:

```
Dim ZELLE As Range

Range("A2:C10").Select

For Each ZELLE In Selection
   ZELLE = ZELLE * 10
Next ZELLE
```

LÖSUNG 2:

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each ZELLE In BEREICH
   ZELLE = ZELLE * 10
Next ZELLE
```

Alle Spalten in einem Bereich mit For Each

```
Dim SPALTE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each SPALTE In BEREICH.Columns
   MsgBox SPALTE.Column
Next SPALTE
```

Alle Zeilen in einem Bereich mit For Each

```
Dim ZEILE As Range
Dim BEREICH As Range

Set BEREICH = Range("A2:C10")

For Each ZEILE In BEREICH.Rows
   MsgBox ZEILE.Row
Next ZEILE
```

Alle Bereiche in einer Markierung mit For Each

Da man mit gehaltener STRG-Taste mehrere unabhängige Zellbereiche gleichzeitig markieren kann, braucht man folgenden Code, um durch alle markierten Bereiche zu wandern

VARIANTE 1 (besser als Variante 2, weil sie alle Zellen einzeln als Schleife durchläuft und in der Schleife eine IF-Funktion für jede Zelle einzeln gecheckt werden kann)

```
Dim ZELLE As Range

For Each ZELLE In Selection
   If Cells(5, ZELLE.Column) = "" And Cells(9, ZELLE.Column) <> "Sa" And Cells(9, ZELLE.Column) <> "So" Then ' wenn kein Feiertag,Sa oder So
      ZELLE.Interior.ColorIndex = 30
   End If
Next ZELLE
```

VARIANTE 2 (schlechter, weil sie nicht schleifenartig durch jede markierte Zelle durchgeht und eine IF-Frage nicht funktioniert)

```
Dim MARKIERUNG As Range
Dim BEREICH As Range

Set MARKIERUNG = Application.Selection

For Each BEREICH In MARKIERUNG.Areas
   MsgBox BEREICH.Address
Next BEREICH
```

Alle Zellen mit Formeln in einem Bereich mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("B1:C99")
'  Set BEREICH = Application.Selection ' Alternative, die die aktuelle Auswahl nimmt

'  Arbeite nur mit Zellen, die eine Formel enthalten
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("B1:C99")
'  Set BEREICH = Application.Selection ' Alternative, die die aktuelle Auswahl nimmt

'  Arbeite nur mit Zellen, die eine Formel enthalten
For Each ZELLE In BEREICH.SpecialCells(xlFormulas, 23)
   MsgBox ZELLE.Address
Next ZELLE
```

PRÜFEN OB EINE ZELLE EINE FORMEL ENTHÄLT - FALLS JA WEITERSPRINGEN

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
        If Target.HasFormula = True Then
```

```
            Target.Offset(0, 1).Select
        End If
End Sub
```

Alle Zellen mit Formel-Fehlerwerten in einem Bereich mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("B1:C99").SpecialCells(xlCellTypeFormulas, xlErrors)

For Each ZELLE In BEREICH.Cells
   MsgBox ZELLE.Address
Next ZELLE
```

Alle Zellen mit bestimmten Formel-Ergebnissen (Wert, Text,) mit For Each

Formeln, die Text ergeben:

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A1:C99")

'  Arbeite nur mit Zellen, die eine Formel enthalten und die als Ergebnis einen Text haben
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A1:C99")

'  Arbeite nur mit Zellen, die eine Formel enthalten und die als Ergebnis einen Text haben
   For Each ZELLE In BEREICH.SpecialCells(xlCellTypeFormulas, xlTextValues)
      MsgBox ZELLE.Address
   Next ZELLE
```

Formeln, die Zahlen ergeben:

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A1:C99")

'  Arbeite nur mit Zellen, die eine Formel enthalten und die als Ergebnis einen Text haben
   For Each ZELLE In BEREICH.SpecialCells(xlCellTypeFormulas, xlNumbers)
      MsgBox ZELLE.Address
   Next ZELLE
```

Alle Zellen, die leer sind mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("A1:C99")

' Arbeite nur mit Zellen, die eine Formel enthalten und die leer sind
For Each ZELLE In BEREICH.SpecialCells(xlCellTypeBlanks)
   MsgBox ZELLE.Address
Next ZELLE
```

Alle Zellen mit einem Kommentar mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("D1:D99")

' Arbeite nur mit Zellen, die einen Kommentar enthalten
For Each ZELLE In BEREICH.SpecialCells(xlCellTypeComments, 23)
   MsgBox ZELLE.Address
Next ZELLE
```

Alle sichtbaren Zellen mit For Each

```
Dim ZELLE As Range
Dim BEREICH As Range

Set BEREICH = Range("D1:D99")

' Arbeite nur mit Zellen, die sichtbar und nicht ausgeblendet sind
For Each ZELLE In BEREICH.SpecialCells(xlCellTypeVisible, 23)
   MsgBox ZELLE.Address
Next ZELLE
```

Alle Zeilen und Spalten einblenden

```
Cells.EntireColumn.Hidden = False ' Alle Spalten einblenden
Cells.EntireRow.Hidden = False ' Alle Zeilen einblenden
```

Alle verbundenen Zellen den Zellverbund wieder aufheben

```
Cells.UnMerge ' alle verbundenen Zellen aufheben
```

Allgemeines zu Spalten und Zeilen

```
Rows(3).Delete                  Die dritte Zeile wird gelöscht
```
Columns(5) ist gleich wie Columns("E")
```
Range(Columns(1), Columns(5)).Select ' markiert die Spalten A-E
Range(Columns(1), Columns(1).Offset(0, 10)).Select ' markiert die Spalte A und die nächsten 10

Columns("I:I").Copy             Die Spalte I wird kopiert

Range(Columns(1), Columns(5)).Select ' markiert die Spalten
Range(Columns(1), Columns(1).Offset(0, 10)).Select ' markiert die Spalte A und die nächsten 10
```

`Range(Cells(1, 1), Cells(10, 5)).Copy ' Der Bereich A1 bis E5 wird in Zwischenablage kopiert`

```
Range("A2").Resize(2, 10).Select ' markiert Zelle A2 und eine Zelle tiefer und 9 Zeilen weiter nach rechts

Columns("D:H").ColumnWidth = 6  Die Spaltenbreite von D-H wird eingestellt
Rows("1:1").Select              Zeile 1 wird ausgewählt
=Selection.Columns.Count        = die Anzahl Spalten der aktuellen Auswahl
=Selection.Rows.Count           = Anzahl der Zeilen in aktueller Auswahl

Dieses Beispiel setzt den Wert aller Zellen in der ersten Spalte im Bereich "A1:D10" auf 0
Range("A1:D10").Columns(1).Value = 0
```

Allgemeines zu Zelladressen und Zellmerkmalen

```
MyCellAddress = ActiveCell.Address  ' Address of current active cell in the form ($I$3) -- absolute reference
MyCellAddress = ActiveCell.Address(RowAbsolute, ColumnAbsolute)  'Address of current cell in (I3)-form
MyCellAddress = ActiveCell.Address(ReferenceStyle:=xlR1C1) ' Address of current cell in form (R5C3)
MyCellColumnLetter = Mid(ActiveCell.Address, 2, (InStr(2, ActiveCell.Address, "$")) - 2)    ' Give letter name of column of active cell
MyCellRow = Range(ActiveCell.Address).Row   ' NOTE: ActiveCell.Address can be substituted for "A1" etc.
MyCellColumnNumber = Range("A1").Column         ' returns the column number of the specified cell
MyCellValue = Range("A1").Value             ' returns the value of the specified cell
range.Text                                  ' formatted string with content of the cell
range.Characters(start, number)             ' individual characters of text
MyCellFormula = Range("A1").Formula         ' obtain formula at A1
MyCellFormula = Range("A1").HasFormula   ' True or False response (depending whether formula present)
range.NoteText(text, start, end)            ' reasons or changes up to 255 characters of a cell note
range.Font                                  ' refers to a font object
range.VerticalAlignment                     ' vertical alignment (left/right/center/justified)
range.HorizontalAlignment                   ' horizontal alignment (upper/lower/middle)
range.Orientation                           ' text orientation (horizontal/vertical)
MyCellWrapText = Range("A1").WrapText       ' returns True or False depending on Wrap property
MyCellColumnWidth = Range("A1").ColumnWidth    ' returns width of cell specified
Selection.ColumnWidth = 10                  ' sets the column width to a particular value (in this case 10)
range.RowHeight                             ' height of an entire row
range.NumberFormat                          ' string with number format
```

MyCellStyle = Range("A1").Style ' returns style name of specified cell

range.BordersAround style, weight ' sets the entire border

range.Borders ' reference to border object

MyCellRow = Range("A1").Row ' returns row number of the specified range

ACTIVATE und SELECT

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Druckversion&action=edit&section=8

Selektieren und Aktivieren

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-1

Selection, muss das sein?

Die nachfolgende Abhandlung mag manchem in der Entschiedenheit übertrieben erscheinen, dennoch hält der Autor eine klare Position in diesem Thema für angebracht, da das Selektieren und Aktivieren von Trainern und Dozenten auch nach einigen Jahren VBA weiter unterstützt wird und sie in der Regel selbst zu eifrigen Selektierern gehören. Ein kleiner Teil hebt sich wohltuend von der Mehrheit ab. Auch in der Literatur wird aus der Angst heraus, sich Laien gegenüber nicht verständlich machen zu können, das Thema falsch behandelt.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-2

Worum geht es hier?

Es gibt in MS Office wie auch im wirklichen Office mehrere Möglichkeiten, ein Objekt (MS Office) oder einen Mitarbeiter (Office) anzusprechen oder ihm Anweisungen zu erteilen. Um einem Mitarbeiter in einer Abteilung eines anderen Werkes die freudige Mitteilung einer Gehaltserhöhung - über die sich sein danebenstehender Kollege gelb ärgert - zu übermitteln, kann man ihm das entweder über die Hauspost mitteilen lassen oder ihn in dem anderen Werk besuchen.

In VBA wäre die erste Vorgehensweise Referenzieren und die zweite Selektieren. Als Code sieht die erste Variante so aus:

```
Sub Referenzieren()
   With Workbooks("Factory.xls").Worksheets("Abteilung").Range("A1")
      .Value = "Gehaltserhöhung"
      .Interior.ColorIndex = 3
      .Font.Bold = True
      With .Offset(1, 0)
```

```
         .Interior.ColorIndex = 6
         .Font.Bold = False
      End With
   End With
End Sub
```

Der Selektierer hat, um zum gleichen Ergebnis zu kommen, schon etwas mehr Arbeit:

```
Sub Hingehen()
   Dim wkb As Workbook
   Application.ScreenUpdating = False
   Set wkb = ActiveWorkbook
   Workbooks("Factory.xls").Activate
   Worksheets("Abteilung").Select
   Range("A1").Select
   With Selection
      .Value = "Gehaltserhöhung"
      .Interior.ColorIndex = 3
      .Font.Bold = True
   End With
   Range("A2").Select
   With Selection
      .Interior.ColorIndex = 6
      .Font.Bold = False
   End With
   wkb.Activate
   Application.ScreenUpdating = True
End Sub
```

Im Bürobeispiel bekommt er für seine Mehrleistung den Zusatznutzen, die Freude des Gehaltserhöhten und den Neid dessen Kollegen live mitzuerleben, bei VBA bleibt es bei der Mehrarbeit.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-3

Wieso ist das Selektieren so verbreitet?

Dass man kaum Code ohne Selektiererei sieht - hiervon sind viele Code-Beispiele aus dem Hause Microsoft nicht ausgeschlossen - ist vor allem in folgenden Dingen begründet:

- Fast jeder in MS Excel mit VBA Programmierende hat seine ersten VBA-Schritte mit dem Makrorecorder gemacht. Der Recorder ist der Meister des Selektierens und des überflüssigen Codes. Es sei ihm gestattet; er hat keine andere Chance.
- Es erleichtert die Flucht vor abstraktem Denken, indem in die Objekte eine Begrifflichkeit gelegt wird, die nur fiktiv ist.

- Es wird von denen, die VBA vermitteln sollen, eingesetzt, um den Lernenden einen Bezug zu den Objekten zu vermitteln. Dies erleichtert zugegebenermaßen die ersten Schritte in diese Programmiersprache, wirkt sich jedoch später eher als Fluch aus.
- In wesentlich stärkerem Maße als bei anderen Programmiersprachen kommen die Programmierenden aus dem Anwenderbereich und/oder dem der autodidaktisch Lernenden und besitzen in der Regel keine umfassende Ausbildung in den Grundlagen der Programmierung.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-4

Selektieren und Referenzieren aufgrund unterschiedlichen Denkens?

Der typischer Gedankengang eines Selektierers:

Wenn ich jetzt in das Arbeitsblatt Tabelle1 der Arbeitsmappe Test1 und dort in Zelle F10 gehe, den dortigen Zellinhalt kopiere, ihn dann in Arbeitsblatt Tabelle2 von Arbeitsmappe Test2 trage und in Zelle B5 ablade, habe ich das Ergebnis, was ich haben möchte. Jetzt kann ich wieder in die Arbeitsmappe zurückgehen, von der aus ich losgegangen bin.

Diese Überlegung schlägt sich bei ihm in folgendem Code nieder:

```
Sub SelektiertKopieren()
   Dim wkb As Workbook
   Set wkb = ActiveWorkbook
   Workbooks("Test1.xls").Activate
   Worksheets("Tabelle1").Select
   Range("F10").Select
   Selection.Copy
   Workbooks("Test2.xls").Activate
   Worksheets("Tabelle2").Select
   ActiveSheet.Range("B5").Select
   ActiveSheet.Paste Destination:=ActiveCell
   wkb.Activate
   Application.CutCopyMode = False
End Sub
```

Wäre er kein Selektierer, würde er sich sagen, ich kopiere aus Arbeitsmappe Test1, Tabelle1, Zelle F10 nach Arbeitsmappe Test2, Tabelle2, Zelle B5.

So sähe dann sein Code auch aus:

```
Sub ReferenziertKopieren()
   Workbooks("Test1").Worksheets("Tabelle1").Range("F10").Copy _
      Workbooks("Test2").Worksheets("Tabelle2").Range("B5")
   Application.CutCopyMode = False
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-5

Warum soll nicht selektiert werden?

Neben der bekannten Tatsache, dass es sich beim Cursor um keinen Auslauf benötigenden Dackel handelt, eher um einen ausgesprochen faulen Hund, der nichts mehr als seine Ruhe liebt, spricht noch folgendes gegen das Selektieren:

- Selektieren macht den Code unübersichtlich. Da an jeder Ecke von Selection gesprochen wird, verliert man leicht den Überblick, was denn nun gerade selektiert ist. Besonders gravierend fällt dies bei der VBA-Bearbeitung von Diagrammen auf.
- Werden Programme von Dritten weiterbearbeitet, sollte man den nachfolgend damit Beschäftigten die Herumirrerei im Selection-Dschungel ersparen.
- Es wird erheblich mehr Code benötigt. Jede zusätzliche Codezeile ist eine zusätzliche potentielle Fehlerquelle und wirkt sich negativ auf die Performance aus. Die Dateigröße verändert sich nicht entscheidend.
- Der Programmablauf wird unruhig und flackernd. Dies kann nicht in jedem Fall durch Setzen des ScreenUpdating-Modus auf False verhindert werden.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-6

In welchen Fällen sollte selektiert werden?

Es gibt einige Situationen, in denen Selectieren entweder notwendig oder sinnvoll ist. Verlangt wird es von Excel nur in einer verschwindend geringen Anzahl von Fällen. Um einen zu nennen: Das Fenster ist nur zu fixieren, wenn die Tabelle, für die die Fixierung gelten soll, aktiviert ist. Sinnvoll kann es sein, wenn umfangreicher Code mit Arbeiten an und mit Objekten in zwei Arbeitsblättern befasst ist - beispielsweise einem Quell- und einem Zielblatt, zum Programmstart aber ein drittes das Aktive ist. Um den Code übersichtlich und die Schreibarbeit in Grenzen zu halten, kann man jetzt eines der beiden Blätter aktivieren und das andere in einen With-Rahmen einbinden. Man erspart sich dadurch die beidseitige Referenzierung.

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Selektieren&action=edit&section=T-7

Wie kann ich das Selektieren verhindern?

Die Selektiererei lässt sich verhindern durch eine exakte Variablendeklaration und -dimensionierung sowie einer darauf aufbauenden genauen Referenzierung der Objekte.

Im Nachfolgenden einige Beispiele:

Kopieren eines Zellbereiches von einer zur anderen Arbeitsmappe, aufgerufen aus einer dritten

```
Sub Kopieren()
   Dim rngSource As Range, rngTarget As Range
   Set rngSource = Workbooks("Test1.xls").Worksheets(1).Range("A1:F14")
   Set rngTarget = Workbooks("Test2.xls").Worksheets(2).Range("C16")
```

```
   rngSource.Copy rngTarget
End Sub
```

Einfügen einer Grafik in eine zweite Arbeitsmappe

```
Sub BildEinfuegenPositionieren()
   Dim wks As Worksheet
   Dim pct As Picture
   Set wks = Workbooks("Test1.xls").Worksheets(1)
   Set pct = wks.Pictures.Insert("c:\excel\zelle.gif")
   pct.Left = 120
   pct.Top = 150
End Sub
```

In Arbeitsblättern 3 bis 12 je einer Serie von 8 Diagrammen in jedem 2. Diagramm den ersten drei SeriesCollections Trendlinien hinzufügen

```
Sub Aufruf()
   Dim wks As Worksheet
   Dim intCounter As Integer
   For intCounter = 3 To 12
      Call Trendlinie(wks)
   Next intCounter
End Sub

Private Sub Trendlinie(wksTarget As Worksheet)
   Dim trdLine As Trendline
   Dim intChart As Integer, intCll As Integer
   For intChart = 1 To 7 Step 2
      With wksTarget.ChartObjects(intChart).Chart
         For intCll = 1 To 3
            Set trdLine = .SeriesCollection(intCll).Trendlines.Add(Type:=xlLinear)
            With trdLine.Border
               Select Case intCll
                  Case 1
                     .ColorIndex = 5
                     .LineStyle = xlDot
                     .Weight = xlThin
                  Case 2
                     .ColorIndex = 7
                     .LineStyle = xlDot
                     .Weight = xlThin
                  Case 3
                     .ColorIndex = 6
                     .LineStyle = xlDot
                     .Weight = xlThin
               End Select
            End With
         Next intCll
```

```
      End With
   Next intChart
End Sub
```

Bereich im aktiven Blatt filtern und die gefilterten Daten in eine neue Arbeitsmappe kopieren. Am Ende wird die aktive Zelle selektiert, um die Filterauswahl aufzuheben.

```
Sub FilternKopieren()
   Dim wkb As Workbook
   Set wkb = ActiveWorkbook
   Application.ScreenUpdating = False
   Range("A1").AutoFilter field:=3, Criteria1:="*2*"
   Range("A1").CurrentRegion.SpecialCells(xlCellTypeVisible).Copy
   Workbooks.Add
   ActiveSheet.Paste Destination:=Range("A1")
   Columns.AutoFit
   wkb.Activate
   ActiveSheet.AutoFilterMode = False
   Application.CutCopyMode = False
   ActiveCell.Select
End Sub
```

Verwenden Sie die Select-Methode, um eine Zelle oder einen Zellbereich zu markieren. Verwenden Sie die Activate-Methode, um eine einzelne Zelle zur aktiven Zelle zu machen. Aktiviert das Objekt entsprechend der folgenden Tabelle:

Objekt	Beschreibung
Worksheet	Macht dieses Blatt zum aktiven Blatt. Entspricht dem Klicken auf das Register des Tabellenblatts.
Range	Aktiviert eine einzelne Zelle, die sich innerhalb der aktuellen Markierung befinden muss. Verwenden Sie die **Select**-Methode, um einen Zellbereich zu markieren.
Workbook	Aktiviert das erste mit der Arbeitsmappe verbundene Fenster. Durch die Aktivierung wird kein der Arbeitsmappe zugeordnetes Auto_aktivieren- oder Auto_deaktivieren-Makro ausgeführt (verwenden Sie die **RunAutoMacros**-Methode, um diese Makros auszuführen).

Beispiel zur Activate-Methode

In diesem Beispiel wird Sheet1 aktiviert:

Worksheets("Sheet1").Activate

In diesem Beispiel wird der Bereich A1:C3 in Sheet1 markiert und dann B2 zur aktiven Zelle gemacht.

```
Worksheets("Sheet1").Activate
Range("A1:C3").Select
Range("B2").Activate
```

In diesem Beispiel wird die Arbeitsmappe MAPPE4.XLS aktiviert. Falls MAPPE4.XLS aus mehreren Fenstern besteht, wird das erste Fenster (MAPPE4.XLS:1) aktiviert.

```
Workbooks("BOOK4.XLS").Activate
```

Aktuelle Zelle in anderem Tabellenblatt auslesen

Das geht leider nicht, ohne die andere Tabelle zu aktivieren.

Daher am besten kurz Application.screenupdating abdrehen, andere Tabelle aktivieren, aktive Zelle auslesen, dann zurückspringen

Aktuelle Zelle speichern für zurückspringen

Unter Excel 2010 gab es Instabilitäten beim Zurückspringen – Abhilfe: arbeite mit Variante 0

Variante 0

```
z=activecell.row
s=activecell.column
…
cells(z,s).select
```

Variante 1

```
Dim Zelleaktuell As Range

Set Zelleaktuell = ActiveCell  ' Damit hat Zelleaktuell alles, was die aktuelle zelle hat
                               ' es gibt Zelleaktuell.Value  Zelleaktuell.row

Zelleaktuell.Select ' zurückspringen zur vorigen Position
```

Möchte man die Adresse der aktuellen Zelle selber haben:

```
Dim Zelladresse as String
Zelladresse = ActiveCell.Address (0,0)   '  das (0, 0) führt zur Schreibweise ohne "$"
Msgbox Zelladresse ' ergibt zb "A1"
```

Aktuell markierter Bereich siehe Markierter Bereich

Autofilter zurücksetzen

siehe Filter wieder zurücksetzen

Bestimmte Zellen finden

Hier arbeitet man vorwiegend mit der SpecialCells-Methode

Nachfolgender Code etwa entfernt bei allen Zellen das Zellenschutz-Merkmal und setzt es anschließend nur bei Zellen mit Formeln

```
Sub TeileDerTabelleSchützen()
 Sheets("DB").Activate
 Cells.Select
 Selection.Locked = False
 Selection.SpecialCells(xlCellTypeFormulas).Select
 Selection.Locked = True
 Sheets("DB").Protect Password:="hero", _
 DrawingObjects:=True, Contents:=True, Scenarios:=True
End Sub
```

Anstatt Zellen mit Formeln, kann man auch nach folgenden besonderen Zellen suchen:

xlCellTypeAllFormatConditions	Zellen mit beliebigem Format
xlCellTypeAllValidation	Zellen mit Gültigkeitskriterien
xlCellTypeBlanks	Leerzellen
xlCellTypeComments	Zellen mit Anmerkungen
xlCellTypeConstants	Zellen mit Konstanten
xlCellTypeFormulas	Zellen mit Formeln

xlCellTypeLastCell	Die letzte Zelle im verwendeten Bereich
xlCellTypeSameFormatConditions	Zellen mit gleichem Format
xlCellTypeSameValidation	Zellen mit gleichen Gültigkeitskriterien
xlCellTypeVisible	Alle sichtbaren Zellen

Arbeitet man mit **xlCellTypeConstants** oder **xlCellTypeFormulas** kann man optional noch einschränken auf besondere Werte:

xlErrors
xlLogical
xlNumbers
xlTextValues

Der gesamte Ausdruck um alle Zellen mit Formeln, die einen Fehler ergeben zu finden lautet dann zB: **SpecialCells**(**xlCellTypeFormulas**, xlErrors)

Bestimmte Zellen SCHNELL FINDEN für Index-Referenzierung

SCHNELLES FINDEN DER RICHTIGEN INDEX-ZEILE

Wenn ich Daten aus einer Tabelle (zB dvo-Import) in eine Zieltabelle kopiere, suche ich meist Zeile für Zeile in der Zieltabelle das richtige Konto bzw den richtigen Index, damit ich die Werte der Importdatei in die Zieltabelle beim richtigen Index bzw beim richtigen Konto hinaddieren kann.

Hat mein Importfile aber 2500 Konten (was bei KD+LF-Saldenlisten sein kann) und meine Zieltabelle auch 3000 Konten => 7. Mio. Zeilen zu durchlaufen dauert eine Ewigkeit.

Daher : besser ist es

Tipp: Die Frage ob gefunden geht mit: If Not FUNDZELLE is Nothing Then

VERSION 1 - SUCHE IN BESTIMMTER SPALTE / BESTIMMTER ZEILE

CODE FÜR NORMALE MODULE (nicht direkt im VBA-Bereich einer Tabelle)

```
   Dim SUCHKRITERIUM       ' Danach wird gesucht
   Dim FUNDZELLE As Range          ' hier wurde es gefunden

        W.Activate ' Wechseln in Zieltabelle
        Range("B1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung - gewählte Zelle darf NICHT
verbunden sein !!!
        Set FUNDZELLE = Columns("B").Find(What:= SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) '
rows würde in Zeile suchen, LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
        If FUNDZELLE Is Nothing Then
            SP.Activate ' Rückkehr zur Quelltabelle
            GoTo KURSENDE
        End If
        ZZ = FUNDZELLE.Row
        KURS = Cells(FUNDZELLE.Row, 3)
        SP.Activate ' Rückkehr zur Quelltabelle
```

ACHTUNG: wenn der Suchcode im VBA-Bereich einer Tabelle stattfindet - z.B. im Bereich Worksheet.Change und die eigentliche Suche in einer anderen Tabelle stattfindet, dann muss der untenstehende Code der Variante 1 jeweils um die Nennung der Tabelle ergänzt werden, wie sie in den nachfolgenden Zeilen rot gefärbt sind bei den betroffenen Zeilen. Hinweis: die zu durchsuchende Tabelle hat den VBA-Namen W in unserem Beispiel

```
   Dim SUCHKRITERIUM       ' Danach wird gesucht
   Dim FUNDZELLE As Range          ' hier wurde es gefunden
```

```
      W.Activate ' Wechseln in Zieltabelle
      W.Range("B1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung - gewählte Zelle darf NICHT verbunden sein !!!
      Set FUNDZELLE = W.Columns("B").Find(What:= SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) ' rows würde in Zeile suchen,
LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
      If FUNDZELLE Is Nothing Then
         SP.Activate ' Rückkehr zur Quelltabelle
         GoTo KURSENDE
      End If
      ZZ = FUNDZELLE.Row
      KURS = W.Cells(FUNDZELLE.Row, 3)
      SP.Activate ' Rückkehr zur Quelltabelle
```

VARIANTE 1: **Wenn Suche erfolglos, dann soll Schleife dennoch weiterarbeiten** - wichtig: wenn das Suchkriterium in einer anderen Tabelle ist, muss auch der Code für das NICHT-FINDEN zur Quelltabelle zurückspringen (hier T2.Activate)

```
  Dim SUCHKRITERIUM      ' Danach wird gesucht
  Dim FUNDZELLE As Range       ' hier wurde es gefunden
```

```
        ' Suchen der Kontonummer in der Zieltabelle
        T4.Activate ' Wechseln in Zieltabelle
        Range("D1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung - gewählte Zelle darf NICHT
verbunden sein !!!
        Set FUNDZELLE = Columns("D").Find(What:=KONTO, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) ' rows würde
in Zeile suchen, LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
        If FUNDZELLE Is Nothing Then
            MsgBox "Das Konto " & KONTO & " wurde in der Reportingtabelle leider nicht gefunden - bitte dort anlegen und
das Reporting wiederholen. Das Programm fährt dennoch fort."
            T2.Activate ' Rückkehr zur Quelltabelle
            GoTo SCHLEIFENENDE
        End If
        ZZ = FUNDZELLE.Row
        T2.Activate ' Rückkehr zur Quelltabelle
```

VARIANTE 2: Wenn Suche erfolglos, dann soll Schleife abbrechen

```
  Dim SUCHKRITERIUM       ' Danach wird gesucht
  Dim FUNDZELLE As Range         ' hier wurde es gefunden
  SUCHKRITERIUM = "2333"   ' Suchen nach Wert 2333
  Range("B1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
  Set FUNDZELLE = Columns("B").Find(What:=SUCHKRITERIUM, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) ' Rows(5) würde in Zeile 5 suchen
(WICHTIG: unbedingt zuvor in die Zeile springen mit Range("A5").activate!!!) , LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden
  If FUNDZELLE Is Nothing Then
   MsgBox "Das Konto wurde leider nicht gefunden"
   Exit Sub ' nix gefunden
  End If
  MsgBox FUNDZELLE.Row
```

VERSION 2 - SUCHE IM GANZEN BLATT

Wichtig: Typ der Suchvariable (Konto bzw strstring) muss gleich sein dem Datentyp, des durchsuchten Bereiches - daher am Besten mit Variant-Typ bei der Suchvariable arbeiten

```
Sub SUCHE_WERT_IN_GANZEM_BLATT()

Dim FUNDZELLE As Range

        ' Zellmarkierung in eine Zelle geben, die keinen Fund enthalten kann
        Cells(1, 1).Select
        KONTO = 2334
        ' Suchen
        Set FUNDZELLE = Cells.Find(What:=KONTO, After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, SearchOrder:=xlByColumns,
SearchDirection:=xlNext, MatchCase:=False, SearchFormat:=False) ' LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden

        ' Wenn die Fundzelle einen echten Wert enthält: Fund anspringen
        If Not FUNDZELLE Is Nothing Then FUNDZELLE.Activate
        ' Vergleichen ob nun in anderer Zeile
        If ActiveCell.Row > 1 Then
            MsgBox "das Konto wurde gefunden in der Zeile " & ActiveCell.Row
          Else
            MsgBox "Das Konto " & KONTO & " wurde nicht in der Saldenliste gefunden. Bitte nachtragen und erneut starten."
            End
        End If

End Sub
```

VERSION 3 - SUCHE IN BESTIMMTEN BEREICH / BESTIMMTER SPALTE / BESTIMMTER ZEILE

```
Sub SUCHE_WERT_IN_BEREICH()
  Dim strSuche        ' Danach wird gesucht
  Dim rngFound As Range         ' hier wurde es gefunden
  strSuche = "2333"   ' Suchen nach Wert 2333
  Range("B1").Activate ' in den Suchbereich hineinspringen - sonst kommt Fehlermeldung
  Set rngFound = Range("B1:B5").Find(What:=strSuche, After:=ActiveCell, LookIn:=xlValues, LookAt:=xlWhole) ' LookAt:=xlPart würde auch irgendwo in
Zelle den Wert finden

  If rngFound Is Nothing Then
   MsgBox "Das Konto wurde leider nicht gefunden"
   Exit Sub ' nix gefunden
  End If
  MsgBox rngFound.Row
End Sub
VERSION 3
```

1.) eine Zelle oberhalb der ersten Zelle zu springen, ab der das betreffende Konto bzw der Index gesucht werden soll. Angenommen ab der Zeile 10 gibt es in Spalte C die Konten :

```
Cells(9,3).Select ' ich springe eine Zelle oberhalb der ersten Kontenzelle
```

2.) nun springt man mittels Such-Befehl zu der Zeile, wo der Zellinhalt genau dem gesuchten Konto entspringt: da es zu einem Fehler führt, wenn es keinen Fund gibt kann man vorher dies abfangen (siehe aber auch alternativen Programmcode im Anschluss, der kein ON ERROR BRAUCHT)

```
        on error resume next
            Cells.Find(What:=KONTO, After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, SearchOrder:=xlByColumns, SearchDirection:=xlNext,
MatchCase:=False, SearchFormat:=False).Activate
        on error goto 0
```

möchte man nur nach Werten suchen, klappt es wohl besser mit LookIn:=xlValues

3.) Nun vergleiche ich, ob sich die Zeile der aktuellen Zelle verändert hat:

```
If ActiveCell.Row > 9 Then
        msgbox "das Konto wurde gefunden in der Zeile " & ActiveCell.Row
End if
```

Denn wenn das Konto nicht gefunden wurde, dann bleibt die Zellmarkierung auf der Zelle oberhalb der ersten Kontozelle.

Bestimmte Zelle anspringen und anzeigen

Siehe Eintrag

Springe bestimmte Zelle an

Bestimmte Zellen nicht anwählen können

1.) Oft wird man mit Blattschutz arbeiten und nur die veränderbaren Zellen bleiben ungeschützt

2.) Man kann mit Selection.Change arbeiten und analysieren wo der User hinklickt - Nachteil: Bilder kann er anklicken, ohne dass Selection.Change ausgelöst wird. Man kann die natürlich mit einem Makro belegen, das sie schützt. Oder man setzt den Tabellenblattschutz, dann werden Grafiken auch geschützt

3.) ActiveSheet.Scrollarea ="$D$1:$E$10": man kann nur den erlaubten Zellbereich anwählen. Empfiehlt sich nur für überschaubare Tabellen. Der User kann in Tabellen, die nicht am Bildschirm vollständig sichtbar sind, nur durch den erlaubten Bereich scrollen und nicht erlaubte Bereiche sind durch Scrollen nicht sichtbar machenbar

4.) Worksheet.EnableSelection (greift nur bei Tabellenblattschutz)

Worksheet.EnableSelection-Eigenschaft

Gibt die Elemente zurück, die auf dem Blatt ausgewählt werden können, oder legt diese fest. **XlEnableSelection**-Wert mit Lese-/Schreibzugriff.

Syntax

Ausdruck.**EnableSelection**

Ausdruck Eine Variable, die ein **Worksheet**-Objekt darstellt.

Anmerkungen

Diese Eigenschaft ist nur wirksam, wenn das Arbeitsblatt geschützt ist: **xlNoSelection**xlNoSelection verhindert jegliche Markierung auf dem Blatt, **xlUnlockedCells**xlUnlockedCells gestattet nur die Markierung der Zellen, deren **Locked**-Eigenschaft **False**False ist, und **xlNoRestrictions**xlNoRestrictions erlaubt die Markierung einer beliebigen Zelle.

Beispiel

In diesem Beispiel wird festgelegt, dass im ersten Arbeitsblatt keine Elemente markiert werden können.

Visual Basic für Applikationen

```
With Worksheets(1)
    .EnableSelection = xlNoSelection
    .Protect Contents:=True, UserInterfaceOnly:=True
```

```
End With
```

Druckbereich einer Tabelle automatisch auf letzte Zeile einstellen

siehe gleichnamigen Eintrag im Bereich DRUCKEN + PDF

Doppelte Werte / Dubletten in einer Spalte finden

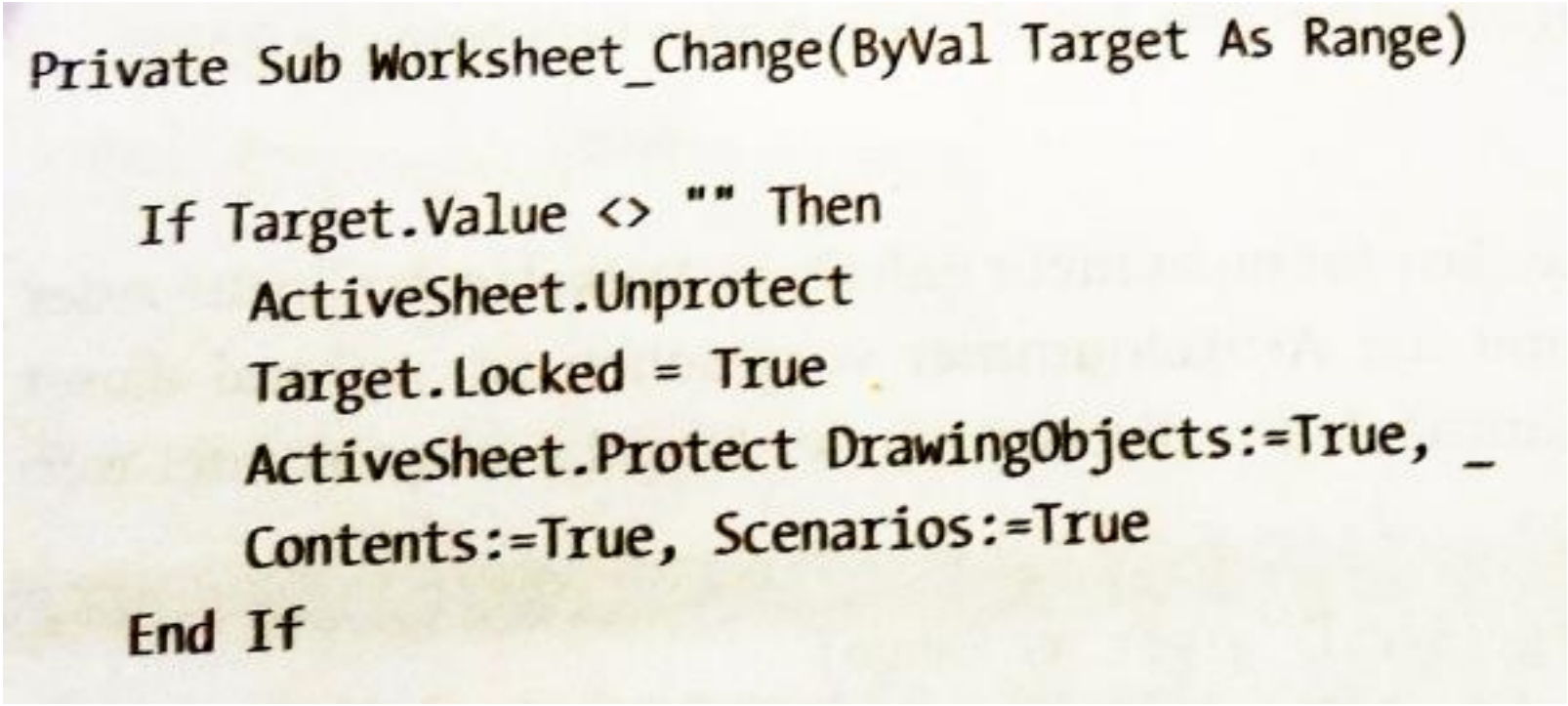
```
Sub DUBLETTEN_CHECK()

' Diese Routine sucht nach doppelten Werten in einer Spalte

   Dim SUCHE             ' Danach wird gesucht
   Dim FUND As Range     ' hier wurde es gefunden
   Dim Z As Single       ' Schleifenvariable
   Dim LZ                ' Letztezeile in der aktuellen Tabelle mit Werten
   Dim ANZAHLFUNDE As Single ' Merkt sich die Anzahl der Dupletten

   LZ = LETZTEZELLE(ActiveSheet.Name).Row ' ermitteln der letzten Zeile mit Werten mit Hilfsfunktion LETZTEZELLE

   For Z = 10 To LZ-1 ' Schleife durch alle Zeilen mit Werten (die letzte Zelle muss nicht mit sich selbst untersucht
werden, es ergäbe einen irrtümlichen Fund)

        SUCHE = Cells(Z, 4) ' Wir prüfen die Werte in der 4. Spalte D
        If Len(SUCHE) > 0 Then ' Prüfung nur, wenn die Zelle nicht leer ist
            Range("D" & Z + 1).Activate ' wir springen in den Suchbereich hinein (der eine Zeile tiefer beginnt) - sonst
kommt Fehlermeldung
            Set FUND = Range("D" & Z + 1 & ":D" & LZ).Find(What:=SUCHE, After:=ActiveCell, LookIn:=xlValues,
LookAt:=xlWhole) ' LookAt:=xlPart würde auch irgendwo in Zelle den Wert finden

            If FUND Is Nothing Then
                ' wenn nichts gefunden wurde ist nichts zu tun
              Else
                MsgBox "Der Wert " & SUCHE & " ist auch in der Zeile " & FUND.Row & " vorhanden. " & vbCrLf & vbCrLf & _
                "Bitte notieren Sie sich die Zeile und korrigieren danach den Wert. " & vbCrLf & vbCrLf & "Das Programm
fährt nun fort."
```

```
                ANZAHLFUNDE = ANZAHLFUNDE + 1
            End If
        End If

    Next Z

    If ANZAHLFUNDE = 0 Then MsgBox "Es wurden keine Dubletten gefunden"
    If ANZAHLFUNDE = 1 Then MsgBox "Es wurde 1 Dublette gefunden"
    If ANZAHLFUNDE > 1 Then MsgBox "Es wurden " & ANZAHLFUNDE & " Dubletten gefunden"

End Sub
```

Einmaliges Befüllen von Zellen

Manchmal möchte man, dass ein User nur einmal die Möglichkeit hat eine Zelle zu befüllen und danach nicht mehr.

Dies realisiert man mit einem Blattschutz, der die betreffenden Zellen anfangs noch ungeschützt hat und bei jeder Änderung einer solchen Zelle wird sie geschützt.



```
Private Sub Worksheet_Change(ByVal Target As Range)

    If Target.Value <> "" Then
        ActiveSheet.Unprotect
        Target.Locked = True
        ActiveSheet.Protect DrawingObjects:=True, _
        Contents:=True, Scenarios:=True
    End If
```

Einzelnes Wort in Zelle färben

```
Sub EinzelnesWortEinfaerben()
    'Sucht in einer Zelle ein Wort und färtb dieses Rot ein
    'Hinweis : Das funktioniert nur mit festen Texten, nicht mit
    'Ergebnissen aus Funktionen !
    '25.08.2010, NoNet - www.excelei.de

    Dim strWort As String, lngZ As Long, rngZelle As Range

    Set rngZelle = [A30] 'In Zelle A30 soll das Wort gesucht werden
    strWort = "Versuch" 'Das Wort, das Rot eingefärtb werden soll

    lngZ = InStr(rngZelle, strWort)

    If lngZ Then
```

```
        rngZelle.Characters(Start:=lngZ, Length:=Len(strWort)).Font.Color = vbRed
    End If
End Sub
```

Einzelne Wörter in Zelle färben

Mein Code:

In der Spalte 23 (W) befindet sich ein Text und ich suche darin meinen Suchtext (Suchtext-Array Suchtext(S)) und färbe darin den Text – festgelegt durch die Länge – rot, zudem wird der Text auch Fett und Kursiv dargestellt

```
' Färben des Textes
With Cells(Z, 23).Characters(Start:=InStr(ZEILENTEXT, UCase(SUCHTEXT(S))), Length:=Len(SUCHTEXT(S))).Font
    .Color = RGB(200, 0, 0)
    .FontStyle = "Bold Italic"
End With
```

Wichtig: beim nächsten Lauf müssen alle Zellen in Spalte W wieder schwarz und nicht fett / nicht kursiv gesetzt werden:

```
With Cells(Z, 23).Characters.Font
        .Color = RGB(0, 0, 0)
        .FontStyle = "Standard"
End With
```

Variante 2

Hi - wenn nur die einzelnen Wörter gefärbt werden sollen und nicht der ganze Text, dann ist es nur nur mit VBA möglich.
ich denke mal, daß du es so haben willst: *die Ausl ist geprüft, Import starten*

hier ist der Code dazu:

Code:

```
Sub test()
Dim Zelle As Range
Dim Begriffe(3) As String
Dim Farbe(3) As Long
```

```
Dim Pos As Long
Dim i As Long

Begriffe(0) = "Ausl"
Begriffe(1) = "Prob."
Begriffe(2) = "geprüft"
Begriffe(3) = "Import"

Farbe(0) = 255
Farbe(1) = 255
Farbe(2) = 65280
Farbe(3) = 49407

For Each Zelle In Columns(3).SpecialCells(xlCellTypeConstants, 2)
   For i = 0 To UBound(Begriffe)
      Pos = InStr(Zelle.Value, Begriffe(i))
      If Pos <> 0 Then
         Zelle.Characters(Start:=Pos, Length:=Len(Begriffe(i))).Font.Color = Farbe(i)
      End If
   Next
Next

End Sub
```

die Farbnummern kannst du leicht selber rausfinden, indem du eine Zelle mit der entsprechenden Hintergrundfarbe versiehst und dann im Direktfenster: *?Selection.Interior.Color* eintippst. Dann bekommst du die Farbnummer der Hintergrundfarbe der selektierten Zelle angezeigt.

VARIANTE 3

Code:

```
Sub WoerterBedingtFaerben()
```

```
Dim iavarDaten    As Long
Dim avarDaten     As Variant
Dim iavarWort     As Variant
Dim avarWort      As Variant
Dim intFarbe      As Integer
Dim rngBereich    As Range
Dim intWortStart  As Integer
Dim intWortLaenge As Integer

Dim colWortFarbe As Collection

With Tabelle1.Cells(1).CurrentRegion.Columns(3)
  Set rngBereich = .Cells
  avarDaten = .Value2
End With

Set colWortFarbe = New Collection

colWortFarbe.Add 3, CStr("Ausl.")   ' 3 = rot
colWortFarbe.Add 3, CStr("Prob.")   ' 3 = rot
colWortFarbe.Add 4, CStr("geprüft") ' 4 = grün
colWortFarbe.Add 45, CStr("Import") ' 45 = orange

rngBereich.Font.ColorIndex = xlColorIndexAutomatic

On Error Resume Next
For iavarDaten = LBound(avarDaten) To UBound(avarDaten)
  If avarDaten(iavarDaten, 1) <> Empty Then
    avarWort = Split(avarDaten(iavarDaten, 1), " ")
    For iavarWort = LBound(avarWort) To UBound(avarWort)
      intFarbe = colWortFarbe(avarWort(iavarWort))
      If intFarbe > 0 Then
        intWortStart = InStr(avarDaten(iavarDaten, 1), avarWort(iavarWort))
        intWortLaenge = Len(avarWort(iavarWort))
```

```
            rngBereich.Cells(iavarDaten, 1).Characters(intWortStart, intWortLaenge).Font.ColorIndex = intFarbe
            intFarbe = 0
          Else
            Err.Clear
          End If
        Next
      End If
    Next
    On Error GoTo 0
  End Sub
```

Einfügen von einer Zeile

Rows(10).insert - fügt in Zeile 10 eine neue Zeile ein und verschiebt den Rest automatisch nach unten

Erste leere Zelle finden

```
Sub Finde()
Selection.SpecialCells(xlBlanks).Areas(1).Cells(1).Select
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Kombinationen&action=edit&section=T-1

Erste leere Zelle ermitteln

Es wird zuerst geprüft, ob Zelle A1 einen Wert besitzt. Wenn nein, wird die Prozedur verlassen. Danach wird der Zeilenzähler initialisiert. Es folgt eine Schleife über alle Zellen in Spalte A, bis die erste leere Zelle erreicht wird. Die Adresse der ersten leeren Zelle wird in einer MsgBox ausgegeben.

```
Sub GeheBisLeer()
   Dim intRow As Integer
   If IsEmpty(Range("A1")) Then Exit Sub
   intRow = 1
   Do Until IsEmpty(Cells(intRow, 1))
      intRow = intRow + 1
   Loop
```

```
   MsgBox "Letzte Zelle mit Wert: " & _
      Cells(intRow - 1, 1).Address(False, False)
End Sub
```

Erste leere Zelle einer Spalte finden

```
Sub Finde()
    Columns(MyColumnNumber).SpecialCells(xlCellTypeBlanks).Cells(1)
End Sub
Sub Finde()
    Cells(Application.WorksheetFunction.CountA(Columns(MyColumnNumber)) + 1,
    MyColumnNumber)
End Sub
```

Fehler-Prüfung (ob Zelle einen Fehler enthält)

If IsError(Cells(QZ,1)) = True Then

Filter setzen und wieder zurücksetzen

Mit nachfolgendem Wechselschaltercode dreht man - in Zeile 10 - den Autofilter ein und wieder aus

```
Sub Filter_AUS_EIN()
'
' Filterein Makro
'

'
   Rows("10:10").Select
   Selection.AutoFilter
   Range("A10").Select
End Sub
```

Filter wieder zurücksetzen

Hat man einen Autofilter gesetzt, vergisst man bisweilen einzelne Auswahlfilter wieder zurückzusetzen, um alles zu sehen

Manuell geht man ins Menü DATEN _ FILTER _ ALLE ANZEIGEN (ist aber ab einer gewissen Excelversion leider eingestellt worden)

Mit VBA geht’s aber weiterhin in allen neuen Versionen

If ActiveSheet.FilterMode = True Then ActiveSheet.ShowAllData ' oder Sheets("1").ShowAllData

Achtung: man bekommt beim ShowAllData-Setzen eine Fehlermeldung, wenn kein Autofilter gesetzt war. Darum frägt der obige Befehl ja auch ab, ob der Autofilter aktiv ist (FilterMode=True)

MsgBox ActiveSheet.AutoFilterMode ' Abfragen ob ein Autofilter eingerichtet ist (unabhängig davon ob er aktiv ist oder nicht)

MsgBox ActiveSheet.FilterMode ' Abfragen ob ein eingerichteter Autofilter auch aktiv ist

Wie richtet man einen Autofilter ein ? (also dass überhaupt ein Autofilter danach zur Verfügung steht - quasi der VBA-Befehl dazu, dass man im Menüdropdownfeld FILTERN UND SORTIEREN auf FILTERN klickt)

ActiveSheet.`UsedRange.AutoFilter`

Wie aktiviert man einen Autofilter

	A	B	C	D	E	F	G
1							Wareneingang
2							01.01.1994
3							01.01.2000
4	Pers-Nr	Nachname	Vorname	Abteilung	Eintritt		
5	001	Meier	Paul	Wareneingang	01.01.1990		
6	002	Schmitz	Toni	Wareneingang	01.02.1995		
7	003	Müller	Hugo	Qualitätskontrolle	01.05.1995		
8	004	Meier	Ernst	Qualitätskontrolle	01.01.2000		
9	005	Schmitz	Erwin	Personalabteilung	01.01.2002		
10							
11							

Code1)
Setzt und entfernt den Autofilter im Bereich A4 bis E4 im aktiven Tabellenblatt.

```
Public Sub Autofilter1()
  Range("A4:E4").Autofilter
End Sub
```

Code2)
Filtert die 2te Spalte im Bereich A4 bis E4, also Spalte B nach dem Suchkriterium "Meier".

```
Public Sub Autofilter2()
  Range("A4:E4").Autofilter Field:=2, Criteria1:="Meier"
End Sub
```

Code3)
Filtert die 4te Spalte im Bereich A4 bis E4, also Spalte D nach dem Suchkriterium "Wareneingang".

```
Public Sub Autofilter3()
  Range("A4:E4").Autofilter Field:=4, Criteria1:="Wareneingang"
End Sub
```

Code4)
Filtert die 4te Spalte im Bereich A4 bis E4, also Spalte D nach dem Suchkriterium in Zelle G1. Hier "Wareneingang".

```
Public Sub Autofilter4()
  Range("A4:E4").Autofilter Field:=4, Criteria1:=Range("G1")
End Sub
```

Code5)
Filtert die 3te Spalte im Bereich A4 bis E4, also Spalte C nach den Kriterien "Toni" oder "Hugo". Es werden beide Treffer angezeigt.

```
Public Sub Autofilter5()
  Range("A4:E4").Autofilter Field:=3, Criteria1:="Toni", Operator:=xlOr, Criteria2:="Hugo"
End Sub
```

Code6)

Filtert die 5te Spalte im Bereich A4 bis E4, also Spalte E nach dem Kriterium: Größer als Datumseintrag in Zelle G2.
CDbl wandelt explizit den Eintrag in eine Dezimalzahl. Excel arbeitet bei Datumsformaten mit Ganz- bzw Nachkommazahlen.

```
Public Sub Autofilter6()
  Range("A4:E4").Autofilter Field:=5, Criteria1:=">" & CDbl(Range("G2"))
End Sub
```

Code7)
Filtert die 5te Spalte im Bereich A4 bis E4, also Spalte E nach dem Kriterium: Größer oder gleich dem Datumseintrag in Zelle G2 und kleiner oder gleich dem Datumseintrag in Zelle G3.
CDbl wandelt explizit den Eintrag in eine Dezimalzahl. Excel arbeitet bei Datumsformaten mit Ganz- bzw Nachkommazahlen.

```
Public Sub Autofilter7()
  Range("A4:E4").Autofilter Field:=5, Criteria1:=">=" & CDbl(Range("G2")), Operator:=xlAnd, Criteria2:="<=" &
CDbl(Range("G3"))
End Sub
```

Code8)
Öffnet alle gesetzten Filter im aktiven Tabellenblatt. Aber ACHTUNG - Code läuft in einen Fehler, wenn keine Spalte gefiltert ist.

```
Public Sub Autofilter8()
  ActiveSheet.ShowAllData
End Sub
```

Code9)
Code prüft, ob im aktiven Tabellenblatt ein Autofilter existiert, und ob mindestens eine Spalte gefiltert ist. Trifft dies zu, so werden alle Filter geöffnet. Somit wird eine etwaige Fehlermeldung wie bei Code Nr8 umgangen.

```
Public Sub Autofilter9()
  With ActiveSheet
    If .AutoFilterMode Then
      If .FilterMode Then .ShowAllData
    End If
  End With
End Sub
```

Code10)
Filtert die 4te Spalte im Bereich A4 bis E4, also Spalte D nach allen Textwerten.
(Es werden in diesem Beispiel natürlich alle Daten gelistet, da es nur Texte gibt)

```
Public Sub Autofilter10()
  Range("A4:E4").Autofilter Field:=4, Criteria1:="=***"
End Sub
```

Code11)
Filtert die 4te Spalte im Bereich A4 bis E4, also Spalte D nach allen Zahlenwerten.
(Es werden in diesem Beipiel natürlich keine Daten angezeigt, da es keine Zahlenwerte gibt)

```
Public Sub Autofilter11()
  Range("A4:E4").Autofilter Field:=4, Criteria1:="<>***"
End Sub
```

Format einer Zelle kopieren auf Zielbereich

Siehe Eintrag Zellformat übertragen

Formeln in Werte umwandeln (ganze Tabelle oder Bereich)

Möchte man eine ganze Tabelle, einen Tabellenbereich alle Formeln in Werte umwandeln geht dies ganz einfach mit

Range("A1:AF100").Value= Range("A1:AF100").Value

Formel auf ganzen Bereich ausrollen

Um eine Formel nicht nur auf eine Zelle zu übernehmen, sondern zB auf eine ganze Spalte, muss man dazu nur den ganzen Bereich ansprechen. In unserem Fall ermitteln wir zuerst für die Variable LZ (LetzteZeile) die Zeilennummer der letzten Zeile und übertragen damit dann für die ganze Spalte die gewünschte Formel:

```
' Formeln eintragen
LZ = LETZTEZELLE(ActiveSheet.Name).Row
Range("AJ11:AJ" & LZ).FormulaR1C1 = "=IFERROR(VLOOKUP(RC[-34],Tausch!C[-34]:C[-30],2,FALSE),RC[-34])"
```

Formeln einer Zelle kopieren in andere Zelle

Mit normalem Cells(1,2)=cells(1,1) werden nur Zellwerte übertragen, aber nicht Formeln.

Zum Kopieren von Formeln gibt es mehrere Wege:

```
' Version 1: Die Formel des Bereichs A1 wird auf den Bereich A1:A20 kopiert

Sub test2()
Range(Cells(1, 1), Cells(1, 1)).AutoFill Destination:=Range(Cells(1, 1), Cells(20, 1)), Type:=xlFillDefault
End Sub

' Version 2

Sub test3()
For T = 2 To 20
    Cells(T - 1, 1).Select
    Selection.Copy
    Cells(T, 1).Select
    ActiveSheet.Paste
Next T
Application.CutCopyMode = False
End Sub

' Version 3: hier wird die Formel 1:1 übertragen und nicht verändert (wie man es meistens will) - wenn also in A1 die Formel =B1+10 steht, dann steht auch in allen befüllten anderen Zellen nur genau die gleiche Formel =B1+10 und nicht C1+10, D1+10 …

Sub Test4()
Dim Formel As String
Formel = Cells(1, 1).Formula
For T = 2 To 20
    Cells(T, 1).Formula = Formel
Next T

End Sub
```

Getrennte Zellbereiche siehe Zellbereiche die getrennt sind

Kommentar einer Zelle auslesen

Range("A1").Comment.Text

Zeilenumbrüche macht man mit & Chr(10)

Auslesen ob Kommentar enthalten ist

```
If ActiveCell.Comment Is Nothing Then
        MsgBox "kein Kommentar"
   else
        MsgBox "Kommentar"
End if
```

Alle Zellen mit Kommentar durchlaufen

```
   Dim ZELLE As Range
   Dim BEREICH As Range

   Set BEREICH = Range("D1:E99")

'  Arbeite nur mit Zellen, die einen Kommentar enthalten
   For Each ZELLE In BEREICH.SpecialCells(xlCellTypeComments, 23)
      MsgBox ZELLE.Address
   Next ZELLE
```

Kopieren direkt bei gleich großem Quell- u. Ziel-Bereich

Wenn der Quellbereich gleich groß ist wie der Zielbereich, kann man direkt kopieren - wichtig ist hier allerdings die Übergabe von .Value !

```
Sheets("SK").Range("A10:L1000").Value = Sheets("SK-SIMPLE").Range("A10:L1000").Value
```

Kopieren+Einfügen mit PASTE / PasteSpecial und INSERT

1. Paste inkl. aller Zellformate

```
Selection.Copy
Range("B2").Select
ActiveSheet.Paste
```

2. PasteSpecial nur der Inhalte

Sheets("1").Range("A10:H29").Copy
Sheets("2").Range("A10").PasteSpecial

3. Insert

Unterschied PASTE und INSERT: Paste überschreibt - Insert fügt ein und verschiebt zuvor den Zellinhalt nach rechts oder unten (xldown)

****
* FEHLER bei INSERT *
****
- bei autom. Skontoausbuchung in der Simplefibu kam es bei jedem dritten SKONTO zu einem Excelinternen Laufzeitfehler in der Hilfsprozedur "Zeileanfügen" und dort beim Befehl der eine neue Zeile einfügt

der nun Fehlerfreie Code lautet:

```
Range("A" & AKTIVEZEILE + 1).Select
Selection.EntireRow.Insert Shift:=xlDown
```

Code Zuvor: es wurde die ganze Zeile markiert und dann eingefügt, was zu Fehler führte:

```
Rows(AKTIVEZEILE + 1 & ":" & AKTIVEZEILE + 1).Select
Selection.Insert Shift:=xlDown
```

Kopieren-Modus beenden, damit Kopiermarkierung weg

```
Application.CutCopyMode = False
```

Leeren Tabellenblatt nur verwendeter Bereich

```
' den gesamten genutzten Tabellenbereich ab A1 leeren
' dies geht nur, wenn das Tabellenblatt auch aktiviert ist

ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
```

Leerzellen in einem Zellbereich auslesen

Dazu gibt es in Excel die Formel ANZAHLLEERZELLEN. Diese kann auch in VBA aufgerufen warden wie soviele andere Excelformeln auch.

```
MsgBox Application.WorksheetFunction.CountBlank(ActiveSheet.Range("C5:C11"))
```

Damit zählt man die Anzahl der Leerzellen im Bereich C5 bis C11

Letzte benutzte Zelle einer Seite

0.) Letzte Zelle mit Inhalt in einer Spalte

Range("A65536").End(xlUp).Row

Sehr gerne arbeite ich mit Range("A65536").End(xlUp).Row - allerdings hat es einen Nachteil - es wird nur die letzte Zelle mit Inhalt in Spalte A gefunden und nicht der gesamten Tabelle.

oder:

Du kannst auch in eine Zelle folgende Funktion einfügen und ihr Ergebnis als jene Zellenzeilennummer nehmen der letzten Zelle, wo - trotz Formel - das Ergebnis der Zelle leer ist bzw 0

=VERGLEICH(0;A:A;-1)

Möchte ich in der Zelle A1 kurz die letzte Zellen-Zeilnummer der Spalte D reinschreiben, die nicht leer ist, dann nehme ich diesen Code

```
Range("A1").FormulaR1C1 = "=MATCH(0,C[3],-1)" ' Ermitteln der letzten Zelle in Spalte D mit Inhalt
LZ = Range("A1").Value
Range("A1").ClearContents

For Z = 11 To LZ
   …
Next Z
```

1.) LETZTEZELLE-Funktion (nimmt letzte Zelle, die formatiert ist oder Formel hat, auch wenn KEIN Wert enthalten !)

Vorsicht: geht NICHT bei gesetztem Tabellenblattschutz

Neue Version von LETZTEZELLE-Funktion

sie nimmt die letzte Zelle einer Seite, die Inhalt hat oder irgendwie formatiert wurde (Rahmenlinie, Farbe …). Die Range("A65536").End(xlup).Select-Funktion nimmt ja nur die letzte Zelle mit INHALT in einer Spalte!

JEDOCH ACHTUNG: wenn man mit Autofilter arbeitet, wird die letzte angezeigte Zelle ausgegeben. Wenn die letzt tatsächliche Zelle mit Daten darunter ist, aber beim aktuellen Fllter nicht angezeigt wird, wird sie nicht berücksichtigt.

Daher: für ein sicheres Funktionieren sollte immer der Code für das Zurücksetzen des Autofilters aktiv bleiben

Aber wichtig: wenn in eine Zelle mittels Cells(x,y)="" ein "" reingeschrieben wurde, dann wird diese Zelle als mit Inhalt (wenn auch mit leerem) gefunden. Daher besser immer mit Cells(X,y).clearcontents leer machen

Wichtig: Die Funktion braucht Zeit – daher bitte nicht in Schleifen verwenden.
Statt For Z=1 to LETZTEZELLE(ActiveSheet.Name).Row nimm LETZTEZEILE = LETZTEZELLE(ActiveSheet.Name).Row FOR Z=1 to LETZTEZEILE

```
Sub TEST()
   MsgBox LETZTEZELLE(ActiveSheet.name).Row ' letzte Zeile für die aktuelle Tabelle
   MsgBox LETZTEZELLE("TABELLENBLATT XY").Column ' letzte Spalte für die Tabelle "Tabellenblatt XY"
End Sub

Public Function LETZTEZELLE(TABELLENBLATT As String) As Range

' ermittelt die letzte Zelle einer Tabelle mit Inhalt oder Zellformat

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
```

```
  Row = ExcelLastCell.Row
  Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
    Row = Row - 1
  Loop
  LastRowWithData = Row

  ' letzte Spalte mit Daten herausfinden
  LastColWithData = ExcelLastCell.Column
  Col = ExcelLastCell.Column
  Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
    Col = Col - 1
  Loop
  LastColWithData = Col

  Set LETZTEZELLE = TheSheet.Cells(Row, Col)
  Application.ScreenUpdating = MERKER

End Function
```

Unter Excel 2007 hatte ich immer wieder bei der Simplefibu "Laufzeitfehler 16 - Ausdruck zu komplex" - und zwar nur bei den drei Tabellen SK, LF und KD. Immer in der Zeile, wo ich die letzte Zeile der Seite feststellen wollte - egal ob ich mit USEDRANGE oder end(Xlup) oder LETZTEZELLE hier arbeitete. Der wirkliche Fehler lag an unvermuteter Stelle:

Es ging um eine Schleife die von Zeile 10 bis zur letzten Zeile gehen sollte:

For T=10 to Letztezelle("SK").row führte zur Fehlermeldung genauso wie mit usedrange oder end(xlup) - ja sogar wenn ich die letzte Zelle in eine Zelle schrieb und der Code so war:
For T=10 to sheets("SK").range("B1")

Das wirkliche Problem war gar nicht diese letzte Zelle, sondern die Variablendefinition von T. Da ich die 32.000-Schranke von INTEGER aufheben wollte, sollten mal ganz viele Buchungen zu verarbeiten sein, hatte ich irrtümlich auf SINGLE umgestellt - und das ging gut unter Office 2003 - aber nicht unter Office 2007. Daher nicht die Fließkommazahl Single nehmen, sondern LONG.

2.) alte Version von LETZTEZELLE-Funktion

Ich arbeite am Liebsten mit der Funktion LETZTEZELLE - hier mit der Prozedur TEST aufgerufen

```
Sub TEST()

  MsgBox LETZTEZELLE(Worksheets(ActiveSheet.name)).Row
  MsgBox LETZTEZELLE(Worksheets(ActiveSheet.name)).Column

End Sub


Function LETZTEZELLE(TheSheet As Worksheet) As Range
```

```
  Dim ExcelLastCell As Range
  Dim Row As Long
  Dim Col As Long
  Dim LastRowWithData As Long
  Dim LastColWithData As Long

  Application.ScreenUpdating = False

  Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

  ' letzte Zeile mit Daten herausfinden
  LastRowWithData = ExcelLastCell.Row
  Row = ExcelLastCell.Row
  Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
    Row = Row - 1
  Loop
  LastRowWithData = Row

  ' letzte Spalte mit Daten herausfinden
  LastColWithData = ExcelLastCell.Column
  Col = ExcelLastCell.Column
  Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
    Col = Col - 1
  Loop
  LastColWithData = Col

  Set LETZTEZELLE = TheSheet.Cells(Row, Col)

End Function
```

3.) ***Range("A65536").End(xlUp).Row***

Sehr gerne arbeite ich mit Range("A65536").End(xlUp).Row - allerdings hat es einen Nachteil - es wird nur die letzte Zelle mit Inhalt in Spalte A gefunden und nicht der gesamten Tabelle.

Wenn nun andere Spalten länger gehen als die Spalte A, möchte man die letzte tatsächliche Zeile mit Inhalt - egal in welcher Spalte - finden.

4.) Die UsedRange-Funktion

ist weniger gut, weil sie auch leere Zellen findet, wenn sie eine Rahmenlinie haben oder das Zellenformat irgendwann mal eingestellt wurde.

In diesem Beispiel wird der verwendete Bereich in Sheet1 ausgewählt.

```
Worksheets("Sheet1").Activate
ActiveSheet.UsedRange.Select
```

Hier wird die letzte Zeile ausgegeben, die zumindest irgendwie formatiert ist (also auch wenn leer):

ActiveSheet.UsedRange.Rows.Count

5.) Die letzte Zeile finden, in der ein Inhalt ist

AKTUELLEENDZEILE = Cells.Find(What:="", After:=[a1], SearchOrder:=xlByRows, SearchDirection:=xlPrevious).Row*

6.) SpecialCells(xlLastCell)-Funktion

Auch die SpecialCells(xlLastCell)-Funktion hat Fehler, denn wenn eine Zelle in der aktuellen Sitzung einmal Inhalt hatte, aber sie mittlerweile gelöscht wurde, wird sie dennoch von der SpecialCells(xlLastCell)-Funktion berücksichtigt. (Erst ein Speichern würde den Fehler beheben).

7.) Letzte Zelle mit Inhalt vor einer leeren Zelle in einer Spalte

```
Sub LastCellBeforeBlankInColumn()
  Range("A1").End(xldown).Select
End Sub
```

8.) Letzte Zelle in einer Spalte auch wenn dazwischen mal leere Zellen sind

```
Sub LastCellInColumn()
  Range("A65536").End(xlup).Select
End Sub
```

9.) Letzte Zelle mit Inhalt vor einer leeren Zelle in einer Zeile

```
Sub LastCellBeforeBlankInRow()
  Range("A1").End(xlToRight).Select
End Sub
```

10.) Letzte Zelle in einer Zeile auch wenn dazwischen mal leere Zellen sind

```
Sub LastCellInRow()
  Range("IV1").End(xlToLeft).Select
End Sub
```

http://de.wikibooks.org/w/index.php?title=VBA_in_Excel/_Kombinationen&action=edit&section=T-1

Erste leere Zelle ermitteln

Es wird zuerst geprüft, ob Zelle A1 einen Wert besitzt. Wenn nein, wird die Prozedur verlassen. Danach wird der Zeilenzähler initialisiert. Es folgt eine Schleife über alle Zellen in Spalte A, bis die erste leere Zelle erreicht wird. Die Adresse der ersten leeren Zelle wird in einer MsgBox ausgegeben.

```
Sub GeheBisLeer()
```

```
   Dim intRow As Integer
   If IsEmpty(Range("A1")) Then Exit Sub
   intRow = 1
   Do Until IsEmpty(Cells(intRow, 1))
      intRow = intRow + 1
   Loop
   MsgBox "Letzte Zelle mit Wert: " & _
      Cells(intRow - 1, 1).Address(False, False)
End Sub
```

Letzte leere Zeile in Zeilenbereich (leere Zeile erkennen)

Meine Autokontierungstabelle mit den Regeln hat in den ersten Spalten A-AZ die Regelbedingungen und dann rechts davon - BA bis BZ - die zu übernehmenden Werte. Nun möchte ich leere Regeln gar nicht erst wirksam werden lassen. Da die Schleife durch alle Zeilen mit Regeln aber auch leere Regelzeilen enthalten kann, möchte ich bei solchen den betreffenden Regelcode gar nicht erst ausführen. Ich überprüfe einfach in der betreffenden Regelzeile (die Schleife durch die Regelzeilen arbeitet mit der Variable RZ / Regelzeile), ob, wenn ich in der aktuellen Regelzeile von der Spalte AZ weg nach links suchend die erste Leerzeile nehme und wenn deren Spaltennummer die 1 ist (daher: die erste Leerzelle von der Spalte AZ nach Links suchend ist ganz vorne in der Spalte A), dann sind quasi zwischen der Spalte A und der Spalte AZ ALLE Zellen leer und die ganze Zeile enthält keine Regelwerte.

If Range("AZ" & RZ).End(xlToLeft).Column = 1 Then GoTo WEITER ' wenn die erste leere Zelle vor AZ die Spalte 1 ist, dann ist diese Regel leer und wir springen weiter

Löschen - Leeren - Entformatieren

Rows(T).delete	löscht die Zeile die die Variable T enthält
Rows("1:8").Delete	löscht die ersten 8 Zeilen
Rows(T & ":" & T+1).Delete	löscht die Zeile der Variable T und die Zeile danach
Columns("1:8").Delete	löscht die ersten 8 Spalten
ActiveCell.EntireRow.Select	die aktuelle Zeile markieren
Selection.ClearContents	leert die aktuelle Markierung - **ACHTUNG: wenn in der aktuellen Markierung Zellen verbunden sind, kann man nicht direkt schreiben: range("A1:H15").clearcontents - sondern muss schreiben: range("A1:H15").select und anschließend Selection.Clearcontents !**

.Delete	löscht ganz - auch ein Objekt
.Clear	löscht Inhalt und Formatierung
.ClearContent	löscht nur den Inhalt aber nicht Formatierung
.ClearFormats	löscht nur die Formate
.ClearComments	löscht die Kommentare

.EntireRow.Delete	Löscht absolut alles - inkl der Zeile / Zeilenhöhe etc

EINE / MEHRERE SPALTEN LÖSCHEN

```
Worksheets("Tabelle1").Columns("B:D").Delete Shift:=xlToLeft
```

Markierter Bereich

' Selection.Count übergibt die Anzahl der markierten Zellen

```
If Selection.Count = 1 Then
```

' Selection.Rows.Count übergibt Anzahl der markierten Zeilen

```
  MsgBox Selection.Rows.Count ' Gibt Anzahl der markierten Zeilen aus
  MsgBox Selection.Columns.Count ' Gibt Anzahl der markierten Spalten aus
```

Makro für Bereichübergabe

```
Sub Bereich_Export_CSV(Speicherpfad As String)

' www.simplesoft.at
' Dieses Makro exportiert den aktuell markierten Bereich in den übergebenen Speicherpfad
' Dieser Speicherpfad muss auch den CSV-Speicherdateinamen enthalten am Ende

Dim Bereich As Range, Zeile As Object, Zelle As Object
Dim strTrennzeichen As String
Dim blnAnfuehrungszeichen As Boolean

' Parameter setzen

   strTrennzeichen = ";" ' Trennzeichen

   blnAnfuehrungszeichen = False ' o. True - bestimmt, ob beim Export " verwendet werden sollen
```

```
' Auslesen markierter Bereich

   Set Bereich = Selection

' Warnungen übergehen (ev. Überschreiben einer existierende Datei)

   Application.DisplayAlerts = False

' Öffnen CSV-Datei

   Open Speicherpfad For Output As #1

' Export

   For Each Zeile In Bereich.Rows
      For Each Zelle In Zeile.Cells
         If blnAnfuehrungszeichen = True Then
            strTemp = strTemp & """" & CStr(Zelle.Text) & """" & strTrennzeichen
         Else
            strTemp = strTemp & CStr(Zelle.Text) & strTrennzeichen
         End If
      Next
      If Right(strTemp, 1) = strTrennzeichen Then strTemp = Left(strTemp, Len(strTemp) - 1)
      Print #1, strTemp
      strTemp = ""
   Next

  Close #1
  Set Bereich = Nothing


' Warnungen wieder aktivieren

   Application.DisplayAlerts = False

End Sub
```

Seine Adresse:

```
Dim MarkierterBereich As String
  MarkierterBereich = ActiveCell.CurrentRegion.Address
```

Sein Bereich Version2

```
Sub test()

    Dim rngMarkiert As Range
    Set rngMarkiert = ActiveSheet.Range(Selection.Address)

    Sheets("Tabelle1").Select
    Range("B2").Select
    Selection.Copy

'und das hier klappt jetzt eben nicht:
   ActiveSheet.Range(rngMarkiert.Address).Select

    Selection.FormatConditions.Delete
    Selection.PasteSpecial Paste:=xlPasteAllExceptBorders, Operation:=xlNone, _
        SkipBlanks:=False, Transpose:=False
End Sub
```

Markierung - Anzahl Spalten / Zeilen - Mehrfachmarkierung

' Selection.Count übergibt die Anzahl der markierten Zellen

```
If Selection.Count = 1 Then
```

' Selection.Rows.Count übergibt Anzahl der markierten Zeilen

MsgBox Selection.Rows.Count ' Gibt Anzahl der markierten Zeilen aus
MsgBox Selection.Columns.Count ' Gibt Anzahl der markierten Spalten aus

Im folgenden Beispiel wird die Anzahl der markierten Spalten angezeigt.
Es wird auch auf eine Mehrfachmarkierung geprüft. Trifft dies zu, werden alle Teile der Mehrfachmarkierung in einer Schleife bearbeitet.

```
areaCount = Selection.Areas.Count ' Anzahl der markierten Bereiche
If areaCount <= 1 Then ' wenn nur 1 Bereich markiert ist
    MsgBox "Die Markierung enthält " & Selection.Columns.Count & " Spalten."
Else
    For i = 1 To areaCount ' Schleife durch alle markierten Bereiche
        MsgBox "Bereich " & i & " der Auswahl enthält " &  Selection.Areas(i).Columns.Count & " Spalten."
    Next i
End If
```

Im nachfolgenden Code werden die Zellwerte (IDs) in Spalte C (3), die markiert sind in ein Array marked_ids eingelesen

```
Sub TEST()
Dim marked_ids(10000)
Dim c As Range
t1 = Selection.Count
For Each c In Selection
   T = T + 1
   row_id = Range(c.Address).Row
   marked_ids(T) = Cells(row_id, 3) ' lies den Wert der Spalte C aus bei den markierten Zeilen / Zellen-Zeilen
Next
For T = 0 To 99
   MsgBox marked_ids(T)
Next T
End Sub
```

```
Dim marked_ids(10000)
Dim c As Range

t1 = Selection.Count
 For Each c In Selection
 t = t + 1
 row_id = Range(c.Address).Row
 marked_ids(t) = Cells(row_id, 3)
 Next
```

Mehrfach Attribute einer Zelle zuweisen

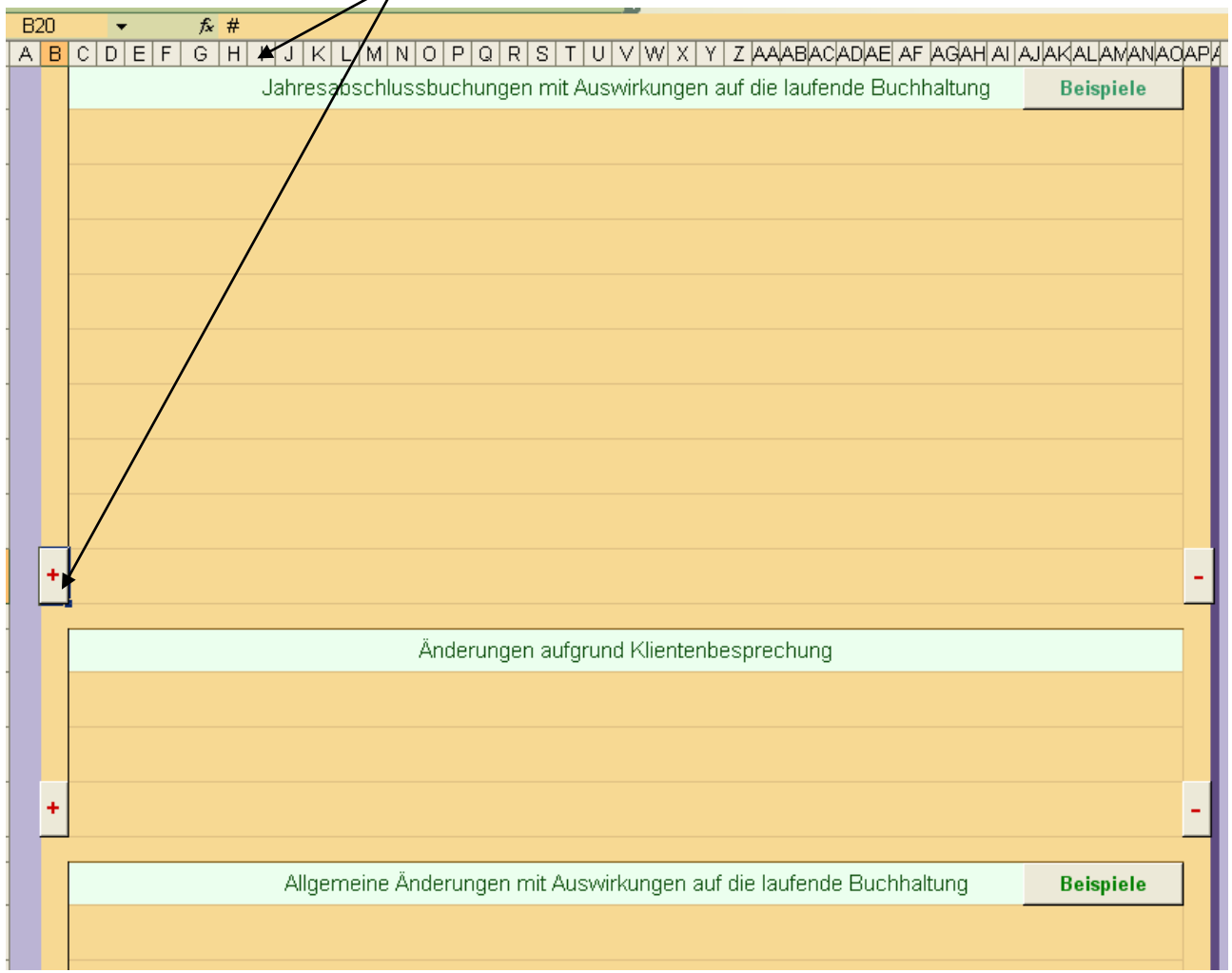
```
For I=1 to 10
   With Cells(I, 2)
      .Value = "Hallo"
      .NumberFormat = "General":  ' Entfernen des Zellenzahlenformats
   End With
Next I

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

Neue Leerzeile für User mit Button einfügen oder löschen

Anwendung: siehe Vorlage INFOBLATT JA FÜR BH (Hübner), die ich für Business Service Team erstellt habe

Es geht darum dem User zu helfen rasch in einem Block von auszufüllenden Zeilen neue Zeilen einzufügen oder nicht benötigte zu entfernen:

Unter der ersten +Taste ist ein # unter der 2. +Taste sind 2 # - unter der ersten -Taste ist ein *, unter der zweiten sind 2 *




```
Sub ZeileneuTextblock1()
```

```
' ************************
' *                      *
' ****     AUFRUF     ****
' *                      *
' Marko wird ausgeführt, wenn die oberste PLUS-Funktionstaste in der Spalte A gedrückt wird
' *                      *
' *                      *
' ****    FUNKTION    ****
' *                      *
' durch das Drücken dieser Plus-Taste wird der aktuelle Textblock um eine neue Leerzeile erweitert
' *                      *
' ****    VORGANG     ****
' *                      *
' es wird eine neue, leere Zeile oberhalb der letzten Zeile in diesem Textblock erzeugt
' die letzte Zeile ist immer die, wo die PLUS-Taste ist
' in der neu erzeugten, vorletzten Zeile wird ein eventueller Text der letzten Zeile verschoben
' und das Format der letzten Zeile auf diese neue vorletzte Zeile übertragen
' *                      *
' ************************

Dim ZEILE As Integer ' Zeilennummer, in der die PLUS-Funktionstaste ist, die dieses Makro hier aufruft
Dim T As Integer ' Laufvariable durch die Zeilen bei der Suche

' Suche die Zeile, in der die PLUS-Taste ist
' direkt unter der Plus-Taste wurde in Spalte A ein #-Zeichen eingefügt zum Finden der aktuellen Zeilennummer
' in der sich die PLUS-Taste befindet - diese verschiebt sich ja bei jedem Ausführen

ZEILE = 0
For T = 1 To 999
   If Cells(T, 2) = "#" Then ZEILE = T
Next T

' prüfen, ob diese Zeile gefunden wurde
If ZEILE = 0 Then
   MsgBox "Es wurde das #-Zeichen gelöscht, das direkt unter der + Taste sein sollte," & vbCrLf & _
   "die Sie eben angeklickt haben." & vbCrLf & vbCrLf & _
   "Bitte gehen Sie in der Spalte  A  direkt in die Zelle, in der sich die + Taste befindet" & vbCrLf & _
   "und tragen dort ein # Zeichen ein, (auf der Tastatur rechts vom Ä)." & vbCrLf & vbCrLf & _
   "Anschließend führen Sie die + Taste hier erneut aus."
   End
End If

' Neu einfügen einer leeren Zeile davor
```

```
    Rows(ZEILE & ":" & ZEILE).Select
    Selection.Insert Shift:=xlDown
    Rows(ZEILE + 1 & ":" & ZEILE + 1).Select ' Format der vormals letzten Zeile, die nun eins tiefer gewandert ist
    Selection.Copy
    Rows(ZEILE & ":" & ZEILE).Select
    Selection.PasteSpecial Paste:=xlFormats, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
    Application.CutCopyMode = False

' Verschieben des Inhaltes der vormals letzten Zeile in die nun vorletzte Zeile
    Cells(ZEILE, 3) = Cells(ZEILE + 1, 3)
    Cells(ZEILE + 1, 3) = ""
    Cells(ZEILE + 1, 3).Select
    ActiveWindow.SmallScroll down:=1.1

End Sub

Sub ZeileloeschenTextblock1()
' *************************
' *                       *
' ****     AUFRUF     ****
' *                       *
' Marko wird ausgeführt, wenn die oberste MINUS-Funktionstaste in der Spalte A gedrückt wird
' *                       *
' ****    FUNKTION    ****
' *                       *
' durch das Drücken dieser Minus-Taste wird beim aktuellen Textblock die letzte Zeile gelöscht
' *                       *
' ****    VORGANG     ****
' *                       *
' es wird die letzte Zeile gelöscht - eine Warnung kommt, wenn diese noch Text enthält
' (genaugenommen wird die vorletzte Zeile gelöscht und ihr Inhalt in die letzte Zeile verschoben,
'  da die letzte nicht gelöscht werden darf wegen den Makro-Funktionstasten, die in dieser Zeile sind.)
' *                       *
' *************************

Dim ZEILE As Integer ' Zeilennummer, in der die Minus-Funktionstaste ist, die dieses Makro hier aufruft
Dim T As Integer ' Laufvariable durch die Zeilen bei der Suche
Dim Wahl ' enthält Antwort auf Abfrage, ob User fortfahren möchte mit Löschen, wenn letzte Zeile etwas enthält

' Suche die Zeile, in der die MINUS-Taste ist
' direkt unter der Minus-Taste wurde in Spalte AO ein *-Zeichen eingefügt zum Finden der aktuellen Zeilennummer
' in der sich die MINUS-Taste befindet - diese verschiebt sich ja bei jedem Ausführen

ZEILE = 0
```

```
For T = 1 To 999
   If Cells(T, 42) = "*" Then ZEILE = T
Next T

' prüfen, ob diese Zeile gefunden wurde
If ZEILE = 0 Then
   MsgBox "Es wurde das Stern-Zeichen (*) gelöscht, das direkt unter der - Taste sein sollte," & vbCrLf & _
   "die Sie eben angeklickt haben." & vbCrLf & vbCrLf & _
   "Bitte gehen Sie in der Spalte  AO  direkt in die Zelle, in der sich die - Taste befindet" & vbCrLf & _
   "und tragen dort ein Stern-Zeichen (*) ein, (auf der Tastatur rechts vom Ü)." & vbCrLf & vbCrLf & _
   "Anschließend führen Sie die - Taste hier erneut aus."
   End
End If

' Prüfen, ob es nicht die letzte weiße Zeile des Blocks ist => daher: die Hintegrundfarbe der Zeile
' davor muss weiß oder ohne Farbe sein
   Cells(ZEILE - 1, 3).Select
   If Selection.Interior.ColorIndex <> 2 And Selection.Interior.ColorIndex <> -4142 Then
      Cells(ZEILE, 3).Select
      MsgBox "Die letzte leere Zeile kann nicht gelöscht werden."
      End
   End If
      Cells(ZEILE, 3).Select

' Prüfen, ob letzte Zeile leer ist
If Cells(ZEILE, 3) <> "" Then
   Wahl = MsgBox("Die letzte Zeile enthält Text oder zumindest Leerzeichen !" & vbCrLf & vbCrLf & _
   "Möchten Sie diese Zeile dennoch löschen ?", vbOKCancel)
   If Wahl = vbCancel Then End
End If

' verschieben Inhalt der vorletzten Zeile und löschen der vorletzten Zeile
   Cells(ZEILE, 3) = Cells(ZEILE - 1, 3)
   Rows(ZEILE - 1 & ":" & ZEILE - 1).Select
   Selection.Delete Shift:=xlUp
   Application.CutCopyMode = False
   Cells(ZEILE - 1, 3).Select
   ActiveWindow.SmallScroll up:=1.1
End Sub
```

Nummer der aktuellen Zeile

Dim Aktivezeile As Integer: ' Nummer der Zeile, in der Cursor gerade ist
Aktivezeile = ActiveCell.Row

Nummer und Buchstabe der aktuellen Spalte

MyColumnNumber = ActiveCell.Column
MyColumnLetter = Mid(ActiveCell.Address, 2, (InStr(2, ActiveCell.Address, "$")) - 2)

Oberste sichtbare Zelle anscrollen

Nach oben scrollen in erste sichtbare Zeile:

Das mache ich gerne mit dem Select-Befehl einer Zelle in der betreffenden Zeile. Das Verschieben der Zellmarkierung activecell.offset(1,0).select eine Zelle weiter nach rechts ist in der Regel gar nicht notwendig. Wichtig aber ist, dass die betreffende Zeile eingeblendet ist, die select-iert wird.

Wenn aber die Zeile ausgeblendet ist, scrollt der Befehl Cells(x,y).select die Tabelle nicht zu dieser Zeile. Abhilfe: eine Schleife, die von oben weg durch alle Zeilen geht und die betreffende Zelle selektiert und damit aufhört, sobald man in einer eingeblendeten Zeile ist

```
Private Sub Worksheet_Activate()

For Z = 2 To 64000
   Cells(Z, 1).Select
   If Rows(Z).Hidden = False Then Exit For
Next Z

End Sub
```

OFFSET - relative Zellbezüge

msgbox ActiveCell.Offset(0, -1).Value ' ergibt den Zellenwert der Zelle links von aktueller Zelle

ActiveCell.Offset(1,2).Select verschiebt die Markierung 1 tiefer und 2 nach rechts
ActiveCell.Offset(-2,0).Select verschiebt die Markierung 2 höher

```
Range(Columns(1), Columns(1).Offset(0, 10)).Select ' markiert die Spalte A und die nächsten 10


Range("A2").Resize(2, 10).Select ' markiert Zelle A2 und eine Zelle tiefer und 9 Zeilen weiter nach rechts
```

Alternative:

RELATIVES SELEKTIEREN mit Selection

Range("B2").Select Zuerst Absolutes Selektieren
Selection.Range("B2").Select Danach relatives Selektieren durch vorangestelltes Selection

Nach den beiden Befehlen ist die Markierung auf C3 !

Eleganter geht es aber mit Offset !

Prüfen ob mehr als eine Zelle ausgewählt / selektiert ist

' Selection.Count übergibt die Anzahl der markierten Zellen

```
If Selection.Count = 1 Then
```

' Selection.Rows.Count übergibt Anzahl der markierten Zeilen

```
  MsgBox Selection.Rows.Count ' Gibt Anzahl der markierten Zeilen aus
  MsgBox Selection.Columns.Count ' Gibt Anzahl der markierten Spalten aus
```

Prüfen ob eine Zelle eine Formel enthält

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
        If Target.HasFormula = True Then
           Target.Offset(0, 1).Select
        End If
End Sub
```

Prüfen ob mehr als eine Zelle markiert ist

```
        If Target.Cells.Count > 1 Then Msgbox "Mehr als eine Zelle markiert"
```

Prüfen ob Zelle einen Fehler enthält

```
If IsError(Cells(QZ,1)) = True Then
```

Rangebereiche mit einer Variablen ansprechen

```
Dim Pos as Integer
Pos=5

Range ("B" & Pos & ":B" & Pos + 10).Value = "Hallo"
```

Das &-Zeichen verbindet in VBA immer Textteile - hier das B und den Wert 5 von Pos , sowie das ":B" und den errechneten Wert 15 (Pos+10) zu "B5:B15"

Im folgenden wird der Inhalt der Spalte A bis D in jener Zeile gelöscht, die eins tiefer als die aktuelle Zelle ist:

```
Dim Aktivezeile as Integer
Aktivezeile=Activecell.Row
Range("A" & Aktivezeile + 1 & ":D" & Aktivezeile + 1).ClearContents
```

Reparieren kaputtgemachter Bezüge

Durch Löschen oder Einfügen neuer Zeilen können Bezüge auf diese Zeilen verloren gehen: zB die Formel in E18 greift auf eine Zeile höher zu (E17) - und nach löschen der Zeile E17, hat die Formel in E18 ihren Bezug verloren.

Um eine Formel in einer Zelle wiederherzustellen nimmt man einfach mit dem Makrorekorder einmal diese Formel auf:

1.) man Beginnt nun mit der Aufnahme
2.) man doppelclickt die Zelle an und drückt ENTER (damit gilt die Formel als neu eingegeben)
3.) Makro beenden

Man erhält einen Befehl der für alle Zellen passt, die die gleiche Formel mit relativen Bezügen hat:
zB wird in der Zelle E18 aus =WENN(B18<>"";E17+C18-D18;"")

```
ActiveCell.FormulaR1C1 = "=IF(RC[-3]<>"""",R[-1]C+RC[-2]-RC[-1],"""")"
```

(Übersetzt: Wenn die Zelle in gleicher Zeile aber 3 Zellen weiter links nicht "" ist, dann nimm den Wert der Zelle einer Zeile höher plus ...)

Weist man zb der Zelle in Zeile 18 diese Formel via VBA zu, dann greift sie ebenso auf die Zelle eine Zeile höher (Zeile 17) zu, als wenn man dies einer Zelle in Zeile 30 zuweist, die auf die Zelle eine Zeile höher (Zeile 29) zugreift.

Schleife durch alle Zellen einer Markierung

NEUE VARIANTE

Da man mit gehaltener STRG-Taste mehrere unabhängige Zellbereiche gleichzeitig markieren kann, braucht man folgenden Code, um durch alle markierten Bereiche zu wandern

```
Dim ZELLE As Range

For Each ZELLE In Selection
   If Cells(5, ZELLE.Column) = "" And Cells(9, ZELLE.Column) <> "Sa" And Cells(9, ZELLE.Column) <> "So" Then ' wenn kein Feiertag,Sa oder So
      ZELLE.Interior.ColorIndex = 30
   End If
Next ZELLE
```

ALTE VARIANTE

```
' Schleife durch die markierten Zeilen einer Spalte

Sub ZELLEINZUG_MEHR()
' Dies ist die Funktion zum Erhöhen des Zelleinzuges

' check, dass nicht mehr als eine Spalte markiert ist
If Selection.Columns.Count > 1 Then
   MsgBox Tabelle3.Range("E718")
   Exit Sub
End If

For ZEILE = ActiveCell.Row To ActiveCell.Row + Selection.Rows.Count - 1
   ' check, dass Zelle nicht geschützt ist
   If Cells(ZEILE, ActiveCell.Column).Locked = True Then
      MsgBox Tabelle3.Range("E719")
      Exit Sub
```

```
    End If
    ' erhöhe den Einzug
    Cells(ZEILE, ActiveCell.Column).InsertIndent 1

Next ZEILE

End Sub

Sub ZELLEINZUG_WENIGER()
' Dies ist die Funktion zum Verringern des Zelleinzuges

' check, dass nicht mehr als eine Spalte markiert ist
If Selection.Columns.Count > 1 Then
    MsgBox Tabelle3.Range("E718")
    Exit Sub
End If

For ZEILE = ActiveCell.Row To ActiveCell.Row + Selection.Rows.Count - 1
    ' check, dass Zelle nicht geschützt ist
    If Cells(ZEILE, ActiveCell.Column).Locked = True Then
        MsgBox Tabelle3.Range("E719")
        Exit Sub
    End If
    ' erhöhe den Einzug
    Cells(ZEILE, ActiveCell.Column).InsertIndent -1

Next ZEILE

End Sub
```

Schleife durch alle Zellen einer Spalte (Powertools-Schleife auch durch Markierung)

```
Sub FORMAT_ZAHL()
' Beschreibung: wenn nur eine Zelle markiert ist, dann wird ihr Format und das aller Zellen darunter umgewandelt ins Zahlenformat
'               wenn mehr als eine Zelle markiert ist, dann wird der entsprechende Bereich umgewandelt
'
' Anwendung: wenn eingespielte Daten im Textformat sind und man möchte sie ins Zahlenformat umwandeln
' (manchmal übernimmt Excel das Zahlenformat auch nach normalen Umformatieren nicht - sondern erst, wenn
' man anschließend, nach dem Umformatieren die betreffende Zelle (zB mit F2) aktiviert und dann mit ENTER
' abschließt - auch dies wird hier berücksichtigt
'

Dim ZEILE, SPALTE
```

```
Dim STARTZEILE As Integer, STARTSPALTENNUMMER As Integer ' Zeile und Spalte der ersten Zelle, die formatiert wird
Dim STARTSPALTENTEXT As String ' Spaltenbuchstaben der ersten Zelle, die formatiert wird
Dim ENDZEILE As Integer, ENDSPALTENNUMMER As Integer ' Zeile und Spalte der letzten Zelle, die formatiert wird
Dim ENDSPALTENTEXT As String ' Spaltenbuchstabend der letzten Zelle, die formatiert wird

Dim I As Integer, T As Integer ' Laufvariablen
Dim WAHL

Dim BEREICH As Range ' der zu formatierende Bereich

' Abfrage, ob nur eine Zelle markiert ist => dann wird alles unterhalb von ihr mit formatiert
'     oder ob ein Bereich markiert ist    => dann wird der gesamte Bereich formatiert

If Selection.Rows.Count = 1 And Selection.Columns.Count = 1 Then

' Nur aktive Zelle und darunter wird formatierr

    STARTZEILE = ActiveCell.Row

    'Spaltenbuchstaben ermitteln
    STARTSPALTENNUMMER = ActiveCell.Column
    If STARTSPALTENNUMMER > 26 Then
      STARTSPALTENTEXT = Chr(STARTSPALTENNUMMER \ 26 + 64) & Chr(STARTSPALTENNUMMER Mod 26 + 64)
    Else
      STARTSPALTENTEXT = Chr(STARTSPALTENNUMMER + 64)
    End If

    ENDSPALTENTEXT = STARTSPALTENTEXT
    ENDSPALTENNUMMER = STARTSPALTENNUMMER
    ENDZEILE = Range(STARTSPALTENTEXT & "65536").End(xlUp).Row

    'WAHL = MsgBox("ES WERDEN NUN IN SPALTE  '" & STARTSPALTENTEXT & "' AB ZEILE " & STARTZEILE & vbCrLf & _
    "." & vbCrLf & vbCrLf & _
    "(ANWENDUNG: )" & vbCrLf & vbCrLf & _
    "MÖCHTEN SIE DIESEN VORGANG AUSFÜHREN LASSEN", vbYesNo)
    If WAHL = vbNo Then End

    'MsgBox "Startspaltennummer" & STARTSPALTENNUMMER & "- Startspaltentext" & STARTSPALTENTEXT & "- Startzeile" & STARTZEILE & vbCrLf & vbCrLf &
_
    "Endspaltennummer" & ENDSPALTENNUMMER & "- Endspaltentext" & ENDSPALTENTEXT & "- Endzeile" & ENDZEILE

Else ' wenn ein Bereich markiert ist

    STARTZEILE = Selection.Row ' (ergibt immer die Zeile der obersten Zelle)
```

```
    'Spaltenbuchstaben ermitteln von Startspalte
    STARTSPALTENNUMMER = Selection.Column ' Spalte der linkesten Zelle
    If SPALTE > 26 Then
      STARTSPALTENTEXT = Chr(STARTSPALTENNUMMER \ 26 + 64) & Chr(STARTSPALTENNUMMER Mod 26 + 64)
    Else
      STARTSPALTENTEXT = Chr(STARTSPALTENNUMMER + 64)
    End If

    ' Spaltenbuchstabe ermitteln von Endspalte
    ENDSPALTENNUMMER = Selection.Column + Selection.Columns.Count - 1
    If SPALTE > 26 Then
      ENDSPALTENTEXT = Chr(ENDSPALTENNUMMER \ 26 + 64) & Chr(ENDSPALTENNUMMER Mod 26 + 64)
    Else
      ENDSPALTENTEXT = Chr(ENDSPALTENNUMMER + 64)
    End If

    ENDZEILE = Selection.Row + Selection.Rows.Count - 1

    'WAHL = MsgBox("ES WERDEN NUN IM ZELLBEREICH " & STARTSPALTENTEXT & STARTZEILE & ":" & ENDSPALTENTEXT & ENDZEILE & vbCrLf & _
    "." & vbCrLf & vbCrLf & _
    "(ANWENDUNG: )" & vbCrLf & vbCrLf & _
    "MÖCHTEN SIE DIESEN VORGANG AUSFÜHREN LASSEN", vbYesNo)
    If WAHL = vbNo Then End


End If


'On Error Resume Next

Application.ScreenUpdating = False

' die eigentliche Schleife durch den gewünschten Bereich (eine Spalte unter aktueller Zelle - oder markierter Bereich)
' Wenn durch den Code eine Zeile gelöscht werden soll, muss man die Schleifenparameter durch die Zeilen anpassen, da
' sich durch das Löschen der zu durchlaufende Schleifenbereich verkürzt
' ZEILE=ZEILE-1 ' da durch Löschen die unteren Zeilen um eins hochrutschen, muss die aktuelle Zeile erneut durchlaufen werden
' ENDZEILE=ENDEZEILE-1 ' dafür verkürzt sich aber das Ende der Schleife


For SPALTE = STARTSPALTENNUMMER To ENDSPALTENNUMMER
    For ZEILE = STARTZEILE To ENDZEILE

          ' normaler code :
```

```
        Cells(ZEILE, SPALTE).NumberFormat = "#,##0.00"

        ' oder  ein Zeilenlöschcode
        ' nachfolgende 6 Zeilen werden nur gebraucht, wenn durch den Code in dieser Schleife Tabellenzeilen gelöscht werden
        ' If....
        '       Rows(ZEILE).Delete Shift:=xlUp
        '       ZEILE=ZEILE-1
        '       ENDZEILE=ENDEZEILE-1
        ' End if
        'If ZEILE >= ENDZEILE Then GoTo SCHLEIFENENDE ' diese Zeile muss AUSSERHALB des If - Then Lösche-Block stehen



    Next ZEILE
Next SPALTE

Range(STARTSPALTENTEXT & STARTZEILE & ":" & ENDSPALTENTEXT & ENDZEILE).Select

For Each BEREICH In Selection
    SendKeys "{F2}", True
    SendKeys "{ENTER}", True
Next BEREICH


Application.ScreenUpdating = True
Cells(STARTZEILE, STARTSPALTENNUMMER).Activate


End Sub
```

Schleife durch alle Zellen sobald Wert sich ändert in einer Spalte und bei Wechsel des Wertes etwas tun

Ich brauche regelmäßig einen Code, der eine Spalte von oben nach unten durchgeht und wenn sich darin der Wert ändert, soll VBA etwas tun. Allerdings muss die erste und die letzte Zeile eines jeweiligen Blocks ermittelt werden, damit man zB Summenzeilen bilden kann über den Block

VERSION 1 - Nur Blockbereich ermitteln

```
Dim NEU As String ' neuer Organamen für Summenzeilen
Dim ALT As String ' alter Organamen für Summenzeilen

' Erzeugen der 3 Summenzeilenvarianten
```

```
' 1. Summe über alle Orgaeinheiten mit gleichem Namen

LZ = LETZTEZELLE(ActiveSheet.Name).Row

For Z = 2 To LZ

   NEU = Cells(Z, 1) ' Speichern der neuen Orga-einheit
   If NEU = ALT Then ' Wenn neue Zeile gleich wie letzte Zeile
      EZ = Z ' Endblockzeile erhöhen
      If NEU <> Cells(Z + 1, 1) Then ' wenn Zelle in nächster Zeile anders ist
         ' hier endet aktueller Block
         MsgBox AZ & "-" & EZ
         AZ = Z + 1
         ALT = Cells(Z + 1, 1)
      End If

    Else ' wenn Wert in aktueller Zeile anders ist als in voriger Zeile
         AZ = Z
         ALT = NEU

   End If



Next Z
```

VERSION 2: Daten zusammenfassen - mehrere Zeilen mit gleichem Wert in einer Spalte, sollen zusammengefasst werden in eine gemeinsame Summenzeile

```
' Zusammenfassen der Datensätze gleicher Bed.Kat-Dienstnehmer einer Organisationseinheit

   LZ = LETZTEZELLE(TD2.Name).Row

   For Z = 2 To LZ

      ' Speichern des Wertes der aktuellen, neuen Zeile
      LETZTWERT = Cells(Z, 1) & Cells(Z, 4)
      ' Schauen ob Wert der aktuellen Zeile gleich ist wie Wert der vorigen Zeile
      If LETZTWERT = ERSTWERT Then
         EZ = Z
         ' Schauen ob Wert in der nächsten Zeile anders ist wie der Wert in dieser Zeile
         If LETZTWERT <> Cells(Z + 1, 1) & Cells(Z + 1, 4) Then
            If AZ <> EZ Then ' wenn der Block mehr als eine Zeile lang ist
```

```
                ' Zusammenfassen der Werte der Spalten G-W
                For S = 7 To 23 ' G-W
                   WERT = 0 ' Reset
                   For ZZ = AZ To EZ ' Schleife durch alle Zeilen
                      WERT = WERT + Cells(ZZ, S) ' Aufsummieren der Werte in dieser Spalte
                   Next ZZ
                   Cells(AZ, S) = WERT ' Eintragen der Summe in oberste Zeile der betreffenden Spalte
                Next S
                ' Zusammenfassen der Werte der Spalten AA-AG
                For S = 27 To 33 ' AA-AG
                   WERT = 0 ' Reset
                   For ZZ = AZ To EZ ' Schleife durch alle Zeilen
                      WERT = WERT + Cells(ZZ, S) ' Aufsummieren der Werte in dieser Spalte
                   Next ZZ
                   Cells(AZ, S) = WERT / (EZ - AZ + 1) ' Eintragen der Summe in oberste Zeile der betreffenden Spalte, jedoch durchschnittlich (daher: dividiert
durch Anzahl der Personen)
                Next S
                ' Speichern der Anzahl der Personen
                Cells(AZ, 6) = EZ - AZ + 1
                ' Löschen der nicht mehr benötigten Zeilen
                For ZZ = EZ To AZ + 1 Step -1 ' wir löschen von unten nach oben, damit die gelöschte Zeile nicht die nächsten zu löschenden Zeilen beeinflusst
                   Rows(ZZ).Delete
                   LZ = LZ - 1 ' Pro Löschvorgang wandert die Letztezeile eins höher
                Next ZZ
             End If
          End If

        Else ' wenn wert in aktueller Zeile anders ist
          AZ = Z
          ERSTWERT = LETZTWERT
       End If

    Next Z
```

VERSION 3 Summenzeile bilden

```
Dim NEU As String ' neuer Organamen für Summenzeilen
Dim ALT As String ' alter Organamen für Summenzeilen

' Erzeugen der Summenzeile

' 1. Summe über alle Orgaeinheiten mit gleichem Namen
```

```
LZ = LETZTEZELLE(ActiveSheet.Name).Row

For Z = 2 To LZ

   NEU = Cells(Z, 1) ' Speichern der neuen Orga-einheit
   If NEU = ALT Then ' Wenn neue Zeile gleich wie letzte Zeile
      EZ = Z ' Endblockzeile erhöhen
      If NEU <> Cells(Z + 1, 1) Then ' wenn Zelle in nächster Zeile anders ist
         ' hier endet aktueller Block
               ' MsgBox AZ & "-" & EZ ' nur zur Kontrolle
         ' Erzeugen der Summenzeile
         Rows(Z + 1).Select ' Folgezeile markieren
         Selection.Insert Shift:=xlDown ' eine neue Zeile einfügen
         ' Beschriften und Formatieren
         Cells(Z + 1, 2) = "SUMME " & NEU
               Cells(Z + 1, 2).Select
               With Selection.Borders(xlEdgeTop)
                  .LineStyle = xlContinuous
                  .ColorIndex = xlAutomatic
                  .Weight = xlThin
               End With
               With Selection.Borders(xlEdgeBottom)
                  .LineStyle = xlContinuous
                  .ColorIndex = xlAutomatic
                  .Weight = xlMedium
               End With
               With Selection.Interior
                  .Pattern = xlSolid
                  .PatternColorIndex = xlAutomatic
                  .Color = 16316664
               End With

         ' Summen bilden
         For S = 7 To 23 ' Schleife durch alle Spalten mit Werten
            WERT = 0 ' Reset für neue Spalte
            For ZZ = AZ To EZ ' Schleife durch den Block aller Zeilen
               WERT = WERT + Cells(ZZ, S)
            Next ZZ
            ' Übernehmen der Summe
            Cells(Z + 1, S) = WERT
            ' Formatieren
            Cells(Z + 1, S).Select
               With Selection.Borders(xlEdgeTop)
```

```
                .LineStyle = xlContinuous
                .ColorIndex = xlAutomatic
                .Weight = xlThin
            End With
            With Selection.Borders(xlEdgeBottom)
                .LineStyle = xlContinuous
                .ColorIndex = xlAutomatic
                .Weight = xlMedium
            End With
            With Selection.Interior
                .Pattern = xlSolid
                .PatternColorIndex = xlAutomatic
                .Color = 16316664
            End With

        Next S
        ' Merken, dass eine weitere Zeile entstand
        Z = Z + 1
        LZ = LZ + 1
        ' Speichern, dass ab nächster Zeile ein neuer Block beginnt
        AZ = Z + 1
        ALT = Cells(Z + 1, 1)
    End If

  Else ' wenn Wert in aktueller Zeile anders ist als in voriger Zeile

        AZ = Z
        ALT = NEU

 End If


Next Z
```

Scrollen in eine bestimmte Zeile

1.) nur hinscrollen

möchte man eine bestimmt Zelle anscrollen - z.B. die aktuelle Zelle, die mit SELECT ausgewählt wurde, aber nicht im aktuell angezeigten Fensterbereich ist:

```
ActiveWindow.ScrollRow = ActiveCell.Row
```

2.) hinscrollen und in die obere linke Ecke stellen

Während mit der Select-Methode eine Zelle zwar angesprungen werden kann, aber der Bildschirm nicht automatisch zu dieser Zelle springt, gibt es den Goto-Befehl, der genau diese macht: er springt auch zur gewählten Zelle / Bereich. Dabei gibt es die Scroll-Möglichkeit: mit dem Wert TRUE wird die Zelle so angescrollt, dass sie am oberen, linken Rand ist - ohne True bleibt sie am unteren, rechten Rand.

Kurzform:

```
Application.Goto Range("CF8") ' Default-Scrollwert false= nicht scrollen
Application.Goto Range("CF8"), True ' True=scrollen
```

Langform:
mit ActiveSheet kommt man gut zurecht, wenn man in VBA-Code eines Tabellenblattes in einem anderen Tabellenblatt als jenem, in dem der VBA-Code ist, arbeiten möchte, so man dieses andere Blatt zuvor aktiviert hat

```
Application.Goto Reference:=ActiveSheet.Range("CF8")
Application.Goto Reference:=ActiveSheet.Range("CF8"), scroll:=True
```

es geht auch die direkte Übergabe eines Tabellenblattes

```
Application.Goto Reference:=Worksheets("Apr").Range("$L$72:$O$76")
```

Natürlich kann man auch eine ganze Mappe mitübergeben

```
Application.Goto Reference:=Workbooks("Mappe1.xlsx").Worksheets("Apr").Range("$L$72:$O$76")
```

Alternative zur SCROLL-TRUE Variante

```
Range("CF8").Select
ActiveWindow.ScrollColumn = ActiveCell.Column
ActiveWindow.ScrollRow = ActiveCell.Row
```

Siehe auch den Eintrag

Springe bestimmte Zelle an

Selektieren einer Zelle und gleichzeitig anspringen

Siehe den Eintrag: **Springe bestimmte Zelle an**

Selektieren einer Zelle

Cells(1,1).Select	zuerst Zeilennummer, dann Spaltennummer eingeben
Cells(1).Select	nimmt die erste Zelle in erster Zeile
Cells(257).Select	nimmt die erste Zelle in zweiter Zeile (da es 256 Zellen pro Zeile gibt und Cells mit nur einem Wert durchzählt)
Cells.Font.Size =8	Da gar nichts angegeben ist, gilt der Befehl für ALLE Zellen
Range ("B2").Select	eine einzelne Zelle markieren
Range ("B2,C3,D4").Select	mehrere Zellen markieren, die NICHT nebeneinander sind
(=) ActiveCell.Value (=)	den Wert der aktuellen Zelle auslesen / ändern

ActiveCell.EntireRow.Cells(1, 3).Value ' springt in der aktuellen Zeile auf die Zelle 3 (die eins für Zeile bedeutet, dass in die erste Zeile gesprungen wird - zB eines markierten Bereiches - hier ohnedies nur eine Zeile markiert)

Selektieren eines Zellbereiches

Range("B2:D4").Select	markiert den Bereich von B2 bis D4
Range("B2", "D4").Select	markiert den Bereich von B2 bis D4 ebenfalls

Range(Cells(Zeileoben,Spaltelinks),Cells(Zeileunten,Spalterechts)).Select

BsP: *Range(Cells(1,1),Cells(2,2)).Select* markiert die ersten 4 Zellen oben links

Range("E:E,K:K,L:L").Select	Die Spalten E, K und L werden markiert
Range("F1,M1,N1").Select	Die Zellen F1, M1 und N1 werden selektiert

In diesem Beispiel wird der Wert 5 für die zweite Zelle der Zeile gesetzt, die die aktive Zelle enthält:
ActiveCell.EntireRow.Cells(1, 2).Value = 5

Dasselbe geht auch für eine Spalte:
ActiveCell.EntireColumn.Cells(1, 1).Value = 5

ActiveCell.EntireRow.Select ' die aktuelle Zeile markieren
Rows(1).Select ' Die Zeile 3 markieren

Sichtbare Zellen kopieren oder selektiveren (bei aktivem Autofilter)

```
' Kopieren der sichtbaren Zeilen inkl. Überschrift
   Range("A10:Z" & Cells(Rows.Count, 1).End(xlUp).Row).SpecialCells(xlCellTypeVisible).Copy
```

Sortieren eines Zellbereiches

Tipp: möchte man nach mehr als 3-Spalten sortieren, dann macht man die Sortierung mehrfach und beginnt mit den Spalten, nach denen zuletzt sortiert werden soll und dann nach denen, die zuerst sortiert werden sollen. Das kann man für unendlich viele Sortierspalten machen.

Version für nur eine Sortierspalte

```
' 1.) Den zu sortierenden Bereich markieren

    Range("A2:Z64000").Select ' A ist die erste Spalte, 2 die erste Zeile, Z die letzte Spalte, 64000 die letzte Zeile

' 2.) die eigentliche Sortierung Key1 enthält die oberste Zelle in der Spalte, nach der sortiert werden soll, zB B2
'  Option Order1: xlAscending = aufsteigend - xlDescending = absteigend

    Selection.Sort Key1:=Range("B2"), Order1:=xlAscending, Header:=xlGuess, OrderCustom:=1, MatchCase:=False, _
    Orientation:=xlTopToBottom, DataOption1:=xlSortTextAsNumbers ' wenn Sortierung falsch entferne DataOption1 zur Gänze

' 3.) Aufhebung der Markierung
    Range("A1").Select
```

Version für nur drei Sortierspalten

```
' 1.) Den zu sortierenden Bereich markieren

    Range("A2:Z64000").Select ' A ist die erste Spalte, 2 die erste Zeile, Z die letzte Spalte, 64000 die letzte Zeile

' 2.) die Sortierung Key1 enthält die oberste Zelle in der Spalte, nach der zuerst sortiert werden soll, zB B2
'  Option Order1: xlAscending = aufsteigend - xlDescending = absteigend

    Selection.Sort Key1:=Range("B2"), Order1:=xlAscending, Key2:=Range("C2"), Order2:=xlAscending, Key3:=Range("D2"),
Order3:=xlAscending, _
    Header:=xlGuess, OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom, DataOption1:=xlSortTextAsNumbers

' 3.) Aufhebung der Markierung
    Range("A1").Select
```

Version die auch unter Excel 2003 geht

```
Sub VERSE_SORTIEREN()

' Diese Prozedur sortiert die Bibelverse in der aktuellen Tabelle (AT oder NT)
```

```
' Variablendefinition für den zu sortierenden Bereich

Dim ERSTEZEILE As Long
Dim LETZTEZEILE As Long
Dim ERSTESPALTE As String
Dim LETZTESPALTE As String

' Restliche Variablendefinitionen für die Sortierung

Dim SORTIERSPALTE1 As String ' nach den Werten welcher 1.Spalte soll sortiert werden
Dim OBERSTESORTIERZELLE1 As String ' Der VBA-Befehl SELECTION.SORT braucht die oberste Zelle der zu sortierenden Spalte
Dim SORTIERSPALTE2 As String ' nach den Werten welcher 2.Spalte soll sortiert werden
Dim OBERSTESORTIERZELLE2 As String
Dim SORTIERSPALTE3 As String ' nach den Werten welcher 2.Spalte soll sortiert werden
Dim OBERSTESORTIERZELLE3 As String

' Variablen setzen

    ERSTEZEILE = 100
    LETZTEZEILE = 200
    ERSTESPALTE = "F"
    LETZTESPALTE = "Z"

    SORTIERSPALTE1 = "W"
    OBERSTESORTIERZELLE1 = SORTIERSPALTE1 & ERSTEZEILE
    SORTIERSPALTE2 = "B"
    OBERSTESORTIERZELLE2 = SORTIERSPALTE2 & ERSTEZEILE
    SORTIERSPALTE3 = "A"
    OBERSTESORTIERZELLE3 = SORTIERSPALTE3 & ERSTEZEILE

' 1.) Den zu sortierenden Bereich markieren

        Range(ERSTESPALTE & ERSTEZEILE & ":" & LETZTESPALTE & LETZTEZEILE).Select

' 2.) die eigentliche Sortierung (xlAscending = aufsteigend - xlDescending = absteigend)

    Selection.Sort Key1:=Range(OBERSTESORTIERZELLE1), Order1:=xlAscending, _
                   Key2:=Range(OBERSTESORTIERZELLE2), Order2:=xlAscending, Key3:=Range(OBERSTESORTIERZELLE3),
Order3:=xlAscending, _
                   Header:=xlGuess, OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom,
DataOption1:=xlSortTextAsNumbers
```

```
' 3.) Aufhebung der Markierung
    Range(OBERSTESORTIERZELLE1).Select

End Sub
```

Version, die auch ab 2007 geht

```
  Range("A2:J8819").Select
  ActiveWorkbook.Worksheets("Excel-Export (3)").Sort.SortFields.Clear
  ActiveWorkbook.Worksheets("Excel-Export (3)").Sort.SortFields.Add Key:=Range( _
     "E2:E467"), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:= _
     xlSortNormal
  ActiveWorkbook.Worksheets("Excel-Export (3)").Sort.SortFields.Add Key:=Range( _
     "G2:G467"), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:= _
     xlSortNormal
  With ActiveWorkbook.Worksheets("Excel-Export (3)").Sort
     .SetRange Range("A1:J467")
     .Header = xlYes
     .MatchCase = False
     .Orientation = xlTopToBottom
     .SortMethod = xlPinYin
     .Apply
  End With
```

Spalten aus und einblenden

Siehe Zeilen + Spalten ausblenden / einblenden

Spalten ohne Buchstaben ansprechen

Colums(1).select ' markiert die Spalte A

Range(Columns(1), Columns(5)).Select ' markiert die Spalten A-E

Range(Columns(1), Columns(1).Offset(0, 10)).Select ' markiert die Spalte A und die nächsten 10

Spaltenbreite einstellen

```
Sub Spaltenbreite_kleiner()

Dim Breite
Breite = Columns("L").ColumnWidth
Breite = Breite - 0.1
If Breite < 0.1 Then Exit Sub

Columns("L:ASA").ColumnWidth = Breite

End Sub
Sub Spaltenbreite_breiter()

Dim Breite
Breite = Columns("L").ColumnWidth
Breite = Breite + 0.1

If Breite > 2 Then Exit Sub

Columns("L:ASA").ColumnWidth = Breite

End Sub
```

Spaltenbuchstabe umwandeln in Zahl

Achtung: Ist der Spaltenbuchstabe in einer Zelle, muss diese mit .Value ausgelesen werden !

FALSCH:
Msgbox Columns(Sheets("Parameter").Range("K4")).Column

RICHTIG:

```
Msgbox Columns(Sheets("Parameter").Range("K4").Value).Column

   SPALTENZEICHEN = "E"
   SPALTENNUMMER = Columns(SPALTENZEICHEN).Column
```

Spaltennummer umwandeln in Spaltenbuchstaben

Version 1

Neues Makro von mir - klappt bei ALLEN Spalten

```
Public Function SPALTENTEXT(SPALTENNUMMER As Integer) As String
   If SPALTENNUMMER > 26 Then
     SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 2)
   Else
     SPALTENTEXT = Mid(Columns(SPALTENNUMMER).Address, 2, 1)
   End If
End Function

Sub testing()
MsgBox SPALTENTEXT(256)
End Sub
```

Version 2

Zusammenfassung
In verschiedenen Situationen ist Ihnen die Spaltennummer bekannt, möchten aber den Namen (z.B. A oder GT) der Spalte wissen, um diesen beispielsweise dem Benutzer anzuzeigen. Die hier vorgestellte Funktion GetColumnName ermittelt den Spaltennamen einer beliebigen Spaltennummer, wobei die Funktion in Excel und in VBA verwendet werden kann. Der Funktion wird eine Spaltennummer zwischen 1 und 256 als Argument übergeben. Wird eine ungültige Spaltennummer angegeben, gibt die Funktion die Fehlerwert "#WERT!" zurück.

VBA-Code

```
Public Function GetColumnName(ByVal intColumnNumber As Integer) As String
  If intColumnNumber <= 0 Or intColumnNumber > Columns.Count Then
    GetColumnName = "#WERT!"
  Else
    GetColumnName = Left$(Cells(1, intColumnNumber).Address(False, False), _
      Len(Cells(1, intColumnNumber).Address(False, False)) - 1)
  End If
End Function
```

Syntax
Result = GetColumnName(ColumnIndex)
Result: Zeichenfolge (String)
ColumnIndex: Ganzzahl von 1 bis 256 (Integer)

Funktionsaufruf in einer Zelle
=GetColumnName(167)

Funktionsaufruf in VBA
strSpalte = GetColumnName(167)

Version 3 (mangelhaft da nur bis AY)

folgendes Makro klappt nur bis zur Spalte AY

wenn ich die Zelle "AY16" markiere und dann das folgende Makro starte, so gibt mir die MsgBox die Zelladresse korrekt aus:

```
Spalte=3
if Spalte > 26 then
        Spaltenbuchstabe = Chr(Spalte \ 26 + 64) & Chr(Spalte Mod 26 + 64)
  else
        Spaltenbuchstabe = Chr(Spalte + 64)
End if
Msgbox Spaltenbuchstabe
```

oder für aktuelle Zelle

```
Sub Spalte_als_Buchstabe()
Dim Spalte As Long, Zeile As Long
Dim Text As String
   Spalte = ActiveCell.Column
   Zeile = ActiveCell.Row
   If Spalte > 26 Then
     Text = Chr(Spalte \ 26 + 64) & Chr(Spalte Mod 26 + 64)
   Else
     Text = Chr(Spalte + 64)
   End If
   MsgBox Text & Zeile
End Sub
```

SpecialCells-Limit-problem

Ron de Bruin (last update 15-July-2007)
Go back to the Excel tips page

Using SpecialCells in VBA is very useful for a lot of things in Excel.

XlCellType options are:
• xlCellTypeAllFormatConditions
• xlCellTypeAllValidation
• xlCellTypeBlanks
• xlCellTypeComments
• xlCellTypeConstants
• xlCellTypeFormulas
• xlCellTypeSameFormatConditions
• xlCellTypeSameValidation
• xlCellTypeVisible

The only problem is that there is a limit of 8192 areas that it can handle.
http://support.microsoft.com/default.aspx?scid=kb;en-us;832293

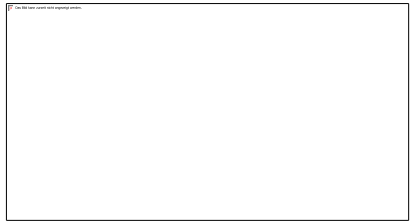
Example for xlCellTypeBlanks

Note: All xlCellType options have this problem

Example: Manual select all Blanks in a column to delete the rows

1) Select the column
2) F5>Special
3) Select Blanks
4) OK

If there are more then 8192 areas you will see this MsgBox



But if we do the same with VBA, It works great if there are no more then 8192 areas.
However, if there are more than 8192 areas, then all the data on your sheet will be deleted without any warning. (And that's not funny!)

If we use this macro below and there are more then 8192 areas we have a problem.

```
Sub DeleteBlankRows_1()
'This macro delete all rows with a blank cell in column A
    On Error Resume Next  'In case there are no blank cells
    Columns("A").SpecialCells(xlCellTypeBlanks).EntireRow.Delete
    On Error GoTo 0
End Sub
```

With the code below, we can test if we have more the 8192 areas so we won't have the problem

that it delete all date on the worksheet.

```
Sub DeleteBlankRows_2()
'This macro delete all rows with a blank cell in column A
'If there are no blanks or there are too many areas you see a MsgBox
    Dim CCount As Long
    On Error Resume Next

    With Columns("A")  ' You can also use a range like this Range("A1:A8000")

        CCount = .SpecialCells(xlCellTypeBlanks).Areas(1).Cells.Count

        If CCount = 0 Then
            MsgBox "There are no blank cells"
        ElseIf CCount = .Cells.Count Then
            MsgBox "There are more then 8192 areas"
        Else
            .SpecialCells(xlCellTypeBlanks).EntireRow.Delete
        End If

    End With

    On Error GoTo 0
End Sub
```

WORKAROUND

To work around this behavior, you may want to create a looping structure in your VBA macro that handles blocks of 16384 cells (not possible to have more then 8192 different areas in 16384 cells).

See also this page from David McRitchie
http://www.mvps.org/dmcritchie/excel/delempty.htm#failure

Another way is to sort you data column before you apply the filter.

There is error checking in the add-in below for the 8192 limit if you use Tables in Excel 2007 :
http://www.rondebruin.nl/table.htm

Springe bestimmte Zelle an

Während mit der Select-Methode eine Zelle zwar angesprungen werden kann aber der Bildschirm nicht automatisch zu dieser Zelle springt, gibt es den Goto-Befehl, der genau diese macht: er springt auch zur gewählten Zelle / Bereich. Dabei gibt es die Scroll-Möglichkeit: mit dem Wert TRUE wird die Zelle so angescrollt, dass sie am oberen, linken Rand ist - ohne True bleibt sie am unteren, rechten Rand.

Kurzform:

```
Application.Goto Range("CF8") ' Default-Scrollwert false= nicht scrollen
Application.Goto Range("CF8"), True ' True=scrollen
```

Langform:
mit ActiveSheet kommt man gut zurecht, wenn man in VBA-Code eines Tabellenblattes in einem anderen Tabellenblatt als jenem, in dem der VBA-Code ist, arbeiten möchte, so man dieses andere Blatt zuvor aktiviert hat

```
Application.Goto Reference:=ActiveSheet.Range("CF8")
Application.Goto Reference:=ActiveSheet.Range("CF8"), scroll:=True
```

es geht auch die direkte Übergabe eines Tabellenblattes

```
Application.Goto Reference:=Worksheets("Apr").Range("$L$72:$O$76")
```

Natürlich kann man auch eine ganze Mappe mitübergeben

```
Application.Goto Reference:=Workbooks("Mappe1.xlsx").Worksheets("Apr").Range("$L$72:$O$76")
```

Alternative zur SCROLL-TRUE Variante

```
Range("CF8").Select
ActiveWindow.ScrollColumn = ActiveCell.Column
ActiveWindow.ScrollRow = ActiveCell.Row
```

Suche bestimmten Zellwert

Siehe Eintrag BESTIMMTE ZELLEN SCHNELL FINDEN

Summenzeile einfügen sobald Wert in Spalte sich ändert

Siehe Schleife durch alle Zellen sobald Wert sich ändert

Überprüfen ob Zelle eine Formel enthält

If Cells(ZEILE, SPALTE).HasFormula = True Then ISTFORMEL = 1 Else ISTFORMEL = 0

PRÜFEN OB EINE ZELLE EINE FORMEL ENTHÄLT - FALLS JA WEITERSPRINGEN

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
        If Target.HasFormula = True Then
            Target.Offset(0, 1).Select
        End If
End Sub
```

Überprüfen ob Zelle ein Datum, Fehler, Zahl/Betrag, Nichts enthält

If IsDate(Cells(ZEILE, SPALTE)) = True Then

Arbeite mit
IsEmpty
IsNull
IsDate
IsError
IsNumeric

Verbundene Zellen wieder aufheben

```
Cells.EntireColumn.Hidden = False ' Alle Spalten einblenden
Cells.EntireRow.Hidden = False ' Alle Zeilen einblenden
Cells.UnMerge ' alle verbundenen Zellen aufheben
```

Verbundene Zellen prüfen

Dieser Code untersucht, ob die Zelle B12 Teil eines Zellverbundes ist - und wenn ja, wieviele Zeilen und Spalten der Verbund hat

Es ist dabei unerheblich, ob B12 am Beginn oder irgendwo mitten im Zellverbund ist, wir erfahren immer die gleiche Anzahl von Zeilen und Spalten, die der gesamte Verbund hat mit dem rows.count und columns.count-Befehl. Erst der MergeArea.Address-Befehl zeigt uns, wo die Zelle im aktuellen Verbund ist, weil er uns die gesamte Adresse des Verbunds anzeigt.

```
MsgBox Range("B12").MergeCells
MsgBox Range("B12").MergeArea.Rows.Count
MsgBox Range("B12").MergeArea.Columns.Count
MsgBox Range("D12").MergeArea.Address
```

Werte einer Zelle - Leerzelen-Werte

ActiveCell.Value = " " *o. VARIABLE = ActiveCell.Value*
Cells(1, 3).Value = " " *o. VARIABLE = Cells(1, 3).Value* (1=Zeile/3=Spalte)
ActiveSheet.Range("J1").Value

ActiveCell.Value = ActiveCell.Value+1 erhöht den Wert der Zelle

Range("D" & ActiveCell.Row).Value ' liest den Wert in Spalte D in der aktuellen Zeile aus

Der Wert einer leeren Zelle ist nicht Null sondern "FALSCH":
Msgbox ActiveCell.Value ' ergibt FALSCH bei leerer Zelle
' Abhilfe:
Msgbox Val(ActiveCell.Value) ' ergibt 0 bei leeren Zellen

Werte einfügen + Zwischenablage leeren

Kopieren nur von Werten geht so:

```
Range("A1:BB86").Copy
Range("A1").Select
Application.DisplayAlerts = False ' keine Warnung zum Überschreiben der früheren Daten
Selection.PasteSpecial Paste:=xlPasteValues ' Einfügen nur noch der Werte - geht aber bei verbundenen Zellen nicht
Application.CutCopyMode = False ' Zwischenspeicher löschen
Application.DisplayAlerts = True ' Warnungen wieder aufdrehen
```

Werte einfügen Problem 'Für diese Aktion müssen alle verbundenen Zellen dieselbe Größe haben'

Normales Kopieren geht so:

Range("A1:BB86").Copy
Range("A1").Select
Application.DisplayAlerts = False ' keine Warnung zum Überschreiben der früheren Daten
ActiveSheet.Paste ' Einfügen der Daten inkl. Formatierung
Application.DisplayAlerts = True ' Warnungen wieder aufdrehen

Kopieren nur von Werten geht so:

Range("A1:BB86").Copy
Range("A1").Select
Application.DisplayAlerts = False ' keine Warnung zum Überschreiben der früheren Daten
Selection.PasteSpecial Paste:=xlPasteValues ' Einfügen nur noch der Werte - geht aber bei verbundenen Zellen nicht
Application.DisplayAlerts = True ' Warnungen wieder aufdrehen

Problem: sobald die Quelle verbundene Zellen hat, funktioniert das Einfügen nur der Werte nicht mehr.
Auch nicht, wenn man die verbundenen Zellen alle schon mal 1:1 in der Zieltabelle so eingefügt hat und vor dem Einfügen der Werte genau denselben Bereich markiert wie in der Quelltabelle. Es kommt immer die Fehlermeldung: Für diese Aktion müssen alle verbundenen Zellen dieselbe Größe haben

Die Internetforen sind voll zu dieser Fehlermeldung und es gibt immer dieselbe Empfehlung: in der Quelle nicht mit verbundenen Zellen arbeiten.

Einziger Workaround: die Werte der Zellen einzeln ins Ziel schreiben. Ich arbeite dabei gerne mit einer Doppelschleife (siehe Code unten), die durch alle Zeilen geht und dabei immer die erste Zelle markiert in jeder Zeile und deren Wert überträgt und dann mit OFFSET springe ich zur nächsten Zelle und übertrage sie. In der inneren Schleife, die durch die verbundenen Zellen einer Zeile springt kann man keine fixe Anzahl von Spalten eingeben, weil man ja nicht weiß, wie viele Zellen so eine Zeile hat, wenn mit verbundenen Zellen gearbeitet wird. Daher: man kann immer nur auslesen in welcher Spalte man beim nächsten OFFSET(1,0)-Sprung ist - und wenn man dabei schon raus ist aus dem gewünschten Bereich, kann man die innere Schleife verlassen.
Wichtig: dies geht nur, wenn die verbundenen Zellen innerhalb einer Zeile sind - wenn die verbundenen Zellen auch über Zeilengrenzen hinausgehen, dann muss man den Code vermutlich noch anpassen.

Meine Lösung Werte einzeln schaufeln

```
Sub Zellen_Uebertragen()

Dim Z As Integer ' Zeilennummer
Dim S As Integer ' Spaltennummer

Sheets("Ziel").Activate

For Z = 1 To 49 ' Schleife durch alle Zeilen

    S = 1 ' Zurücksetzen auf erste Spalte A

    Do ' Schleife durch alle Spalten
```

```
        Cells(Z, S).Activate ' Erste Zelle in aktueller Zeile markieren
        Cells(Z, S).Value = Sheets("Quelle").Cells(Z, S).Value ' ihren Wert aus der Quelle übertragen
        Cells(Z, S).Offset(0, 1).Select ' in der Zieltabelle zur nächsten Zelle nach rechts springen
        S = ActiveCell.Column ' Auslesen der aktuellen Spaltennummer

    Loop Until S > 26 ' Schleife abbrechen, sobald jenseits der Spalte Z

Next Z

End Sub
```

Lösung Thomas Ramel

der bekannte VBA-Fuchs hat für dieses Problem noch eine andere Schleifenlösung, die das Problem auch lösen soll - auch indem alle Zellen einzeln übertragen werden.

```
Public Sub Copy_Column()
Dim lngRow        As Long
Dim ws1           As Worksheet
Dim ws2           As Worksheet

   Set ws1 = Worksheets("Tabelle1")
   Set ws2 = Worksheets("Tabelle2")

   lngRow = 1
   Do
      ws1.Range("A" & (2 * lngRow - 1)).Value = ws2.Range("A" & lngRow).Value
      lngRow = lngRow + 1
   Loop Until IsEmpty(ws2.Range("A" & lngRow).Value)
End Sub
```

Wert suchen in Bereich oder Tabellenblatt

mein schnelles Finden eines Wertes - siehe hier im Bereich " Bestimmte Zellen für Index-Referenzierung finden "

Find value in Range, Sheet or Sheets with VBA

Ron de Bruin (last update 9-Sept-2007)
Go back to the Excel tips page
Find is a very powerful option in Excel and is very useful.
Together with the Offset function you can also change cells around the found cell.
Below are a few basic examples that you can use to make your own code.

Use find to select a cell
Mark cells with the same value in column A in the B column
Color cells with the same value in a Range, worksheet or all worksheets
Copy cells to another sheet with Find
More Information

Use Find to select a cell

The examples below will search in column A of a sheet named "Sheet1"
for the inputbox value. Change the sheet name or range in the code to your sheet/range.

Tip: You can replace the inputbox with a string or a reference to a cell like this
FindString = "SearchWord"
Or
FindString = Sheets("Sheet1").Range("D1").Value

This will select the first cell in the range with the InputBox value.

```
Sub Find_First()
   Dim FindString As String
   Dim Rng As Range
   FindString = InputBox("Enter a Search value")
   If Trim(FindString) <> "" Then
      With Sheets("Sheet1").Range("A:A")
         Set Rng = .Find(What:=FindString, _
                   After:=.Cells(.Cells.Count), _
```

```
                    LookIn:=xlValues, _
                    LookAt:=xlWhole, _
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlNext, _
                    MatchCase:=False)
        If Not Rng Is Nothing Then
            Application.Goto Rng, True
        Else
            MsgBox "Nothing found"
        End If
    End With
  End If
End Sub
```

If you have more then one occurrence of the value this will select the last occurrence.

```
Sub Find_Last()
  Dim FindString As String
  Dim Rng As Range
  FindString = InputBox("Enter a Search value")
  If Trim(FindString) <> "" Then
    With Sheets("Sheet1").Range("A:A")
      Set Rng = .Find(What:=FindString, _
                    After:=.Cells(1), _
                    LookIn:=xlValues, _
                    LookAt:=xlWhole, _
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlPrevious, _
                    MatchCase:=False)
      If Not Rng Is Nothing Then
        Application.Goto Rng, True
      Else
        MsgBox "Nothing found"
      End If
    End With
  End If
End Sub
```

If you have date's in column A then this example will select the cell with today's date.

```
Sub Find_Todays_Date()
  Dim FindString As Date
  Dim Rng As Range
  FindString = CLng(Date)
```

```
    With Sheets("Sheet1").Range("A:A")
        Set Rng = .Find(What:=FindString, _
                    After:=.Cells(.Cells.Count), _
                    LookIn:=xlFormulas, _
                    LookAt:=xlWhole, _
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlNext, _
                    MatchCase:=False)
        If Not Rng Is Nothing Then
            Application.Goto Rng, True
        Else
            MsgBox "Nothing found"
        End If
    End With
End Sub
```

Mark cells with the same value in column A in the B column

This example search in Sheets("Sheet1") in column A for every cell with "ron" and use Offset to mark the cell in the column to the right. Note: you can add more values to the array MyArr.

```
Sub Mark_cells_in_column()
    Dim FirstAddress As String
    Dim MyArr As Variant
    Dim Rng As Range
    Dim I As Long

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Search for a Value Or Values in a range
    'You can also use more values like this Array("ron", "dave")
    MyArr = Array("ron")

    'Search Column or range
    With Sheets("Sheet1").Range("A:A")

        'clear the cells in the column to the right
```

```
        .Offset(0, 1).ClearContents

        For I = LBound(MyArr) To UBound(MyArr)

            'If you want to find a part of the rng.value then use xlPart
            'if you use LookIn:=xlValues it will also work with a
            'formula cell that evaluates to "ron"

            Set Rng = .Find(What:=MyArr(I), _
                        After:=.Cells(.Cells.Count), _
                        LookIn:=xlFormulas, _
                        LookAt:=xlWhole, _
                        SearchOrder:=xlByRows, _
                        SearchDirection:=xlNext, _
                        MatchCase:=False)

            If Not Rng Is Nothing Then
                FirstAddress = Rng.Address
                Do
                    'mark the cell in the column to the right if "Ron" is found
                    Rng.Offset(0, 1).Value = "X"
                    Set Rng = .FindNext(Rng)
                Loop While Not Rng Is Nothing And Rng.Address <> FirstAddress
            End If
        Next I
    End With

    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

Color cells with the same value in a Range, worksheet or all worksheets

This example color all cells in the range Sheets("Sheet1").Range("B1:D100") with "ron".
See the comments in the code if you want to use all cells on the worksheet.
I use the color index in this example to give all cells with "ron" the color 3 (normal this is red)

See this site for all the 56 index numbers

http://www.mvps.org/dmcritchie/excel/colors.htm

Tip: For changing the Font color see the example lines below the macros.

```
Sub Color_cells_In_Range_Or_Sheet()
    Dim FirstAddress As String
    Dim MySearch As Variant
    Dim myColor As Variant
    Dim Rng As Range
    Dim I As Long

    'Fill in the search Value and color Index
    MySearch = Array("ron")
    myColor = Array("3")

    'You can also use more values in the Array
    'MySearch = Array("ron", "jelle", "judith")
    'myColor = Array("3", "6", "10")


    'Fill in the Search range, for the whole sheet use
    'you can use Sheets("Sheet1").Cells
    With Sheets("Sheet1").Range("B1:D100")

        'Change the fill color to "no fill" in all cells
        .Interior.ColorIndex = xlColorIndexNone

        For I = LBound(MySearch) To UBound(MySearch)

            'If you want to find a part of the rng.value then use xlPart
            'if you use LookIn:=xlValues it will also work with a
            'formula cell that evaluates to MySearch(I)
            Set Rng = .Find(What:=MySearch(I), _
                      After:=.Cells(.Cells.Count), _
                      LookIn:=xlFormulas, _
                      LookAt:=xlWhole, _
                      SearchOrder:=xlByRows, _
                      SearchDirection:=xlNext, _
                      MatchCase:=False)

            If Not Rng Is Nothing Then
                FirstAddress = Rng.Address
                Do
```

```
            Rng.Interior.ColorIndex = myColor(I)
            Set Rng = .FindNext(Rng)
        Loop While Not Rng Is Nothing And Rng.Address <> FirstAddress
      End If
    Next I
  End With
End Sub
```

Example for all worksheets in the workbook

```
Sub Color_cells_In_All_Sheets()
  Dim FirstAddress As String
  Dim MySearch As Variant
  Dim myColor As Variant
  Dim Rng As Range
  Dim I As Long
  Dim sh As Worksheet

  'Fill in the search Value and color Index
  MySearch = Array("ron")
  myColor = Array("3")

  'You can also use more values in the Array
  'MySearch = Array("ron", "jelle", "judith")
  'myColor = Array("3", "6", "10")

  For Each sh In ActiveWorkbook.Worksheets

    'Fill in the Search range, for a range on each sheet
    'you can use sh.Range("B1:D100")
    With sh.Cells

      'Change the fill color to "no fill" in all cells
      .Interior.ColorIndex = xlColorIndexNone

      For I = LBound(MySearch) To UBound(MySearch)

        'If you want to find a part of the rng.value then use xlPart
        'if you use LookIn:=xlValues it will also work with a
        'formula cell that evaluates to MySearch(I)
        Set Rng = .Find(What:=MySearch(I), _
                  After:=.Cells(.Cells.Count), _
                  LookIn:=xlFormulas, _
                  LookAt:=xlWhole, _
```

```
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlNext, _
                    MatchCase:=False)

            If Not Rng Is Nothing Then
               FirstAddress = Rng.Address
               Do
                  Rng.Interior.ColorIndex = myColor(I)
                  Set Rng = .FindNext(Rng)
               Loop While Not Rng Is Nothing And Rng.Address <> FirstAddress
            End If
         Next I
      End With
   Next sh
End Sub
```

Change the Font color instead of the Interior color

Replace:

'Change the fill color to "no fill" in all cells
.Interior.ColorIndex = xlColorIndexNone

With

'Change the font in the column to Automatic
.Font.ColorIndex = 0

And Replace:

Rng.Interior.ColorIndex = myColor(I)
With
Rng.Font.ColorIndex = myColor(I)

Copy cells to another sheet with Find

The example below will copy all cells with a E-Mail Address in the range Sheets("Sheet1").Range("A1:E100") to a new worksheet in your workbook.
Note: I use xlPart in the code instead of xlWhole to find each cell with a @ character.

```
Sub Copy_To_Another_Sheet_1()
    Dim FirstAddress As String
    Dim MyArr As Variant
    Dim Rng As Range
    Dim Rcount As Long
    Dim I As Long
    Dim NewSh As Worksheet

    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    'Fill in the search Value
    MyArr = Array("@")

    'You can also use more values in the Array
    'myArr = Array("@", "www")

    'Add new worksheet to your workbook to copy to
    'You can also use a existing sheet like this
    'Set NewSh = Sheets("Sheet2")
    Set NewSh = Worksheets.Add

    With Sheets("Sheet1").Range("A1:Z100")

        Rcount = 0

        For I = LBound(MyArr) To UBound(MyArr)

            'If you use LookIn:=xlValues it will also work with a
            'formula cell that evaluates to "@"
            'Note : I use xlPart in this example and not xlWhole
            Set Rng = .Find(What:=MyArr(I), _
                      After:=.Cells(.Cells.Count), _
                      LookIn:=xlFormulas, _
```

```
                LookAt:=xlPart, _
                SearchOrder:=xlByRows, _
                SearchDirection:=xlNext, _
                MatchCase:=False)
        If Not Rng Is Nothing Then
            FirstAddress = Rng.Address
            Do
                Rcount = Rcount + 1

                Rng.Copy NewSh.Range("A" & Rcount)

                ' Use this if you only want to copy the value
                ' NewSh.Range("A" & Rcount).Value = Rng.Value

                Set Rng = .FindNext(Rng)
            Loop While Not Rng Is Nothing And Rng.Address <> FirstAddress
        End If
    Next I
  End With

  With Application
    .ScreenUpdating = True
    .EnableEvents = True
  End With
End Sub
```

More Information

If you only want to replace values in your worksheet then you can use Replace manual (Ctrl+h) or use Replace in VBA. The code below replace ron for dave in the whole worksheet. Change xlPart to xlWhole if you only want to replace cells with only ron.

```
  ActiveSheet.Cells.Replace What:="ron", Replacement:="dave", LookAt:=xlPart, _
                SearchOrder:=xlByRows, MatchCase:=False, _
                SearchFormat:=False, ReplaceFormat:=False
```

Wort (einzelnes) in Zelle färben

Siehe Einzelnes Wort in Zelle färben

Vorlagen-Spalten ein/ausblenden

Bei der Projektmamagement-Software haben wir immer wieder Vorlagen-Spalten ausgeblendet - etwa, wenn wir dem User nur 5 per default eingeblendet sein lassen, aber er kann mit +/- Buttons sich z.B. weitere 10 Spalten ein oder auch wieder ausblenden

```
Sub Datum_Ausblenden()

For S = 0 To 10
   If Columns(18 - S).Hidden = False Then
      Columns(18 - S).Hidden = True
      End
   End If
Next S

End Sub
Sub Datum_Einblenden()

For S = 0 To 10
   If Columns(9 + S).Hidden = True Then
      Columns(9 + S).Hidden = False
      End
   End If
Next S

End Sub
```

Vorlagen-Zeile duplizieren / löschen

Dies ist die einfache Variante, wo nur Zeilen eingefügt und wieder gelöscht werden. Im Gegensatz zum nächsten Beispiel " Zeilen-Vorlage duplizieren / Block ein-ausblenden" müssen keine Infoblöcke auf einmal ausgeblendet werden können:

Beschreibung der Vor- und Nachprojektphase
Projektname / Titel 1

		Z
8	**Ergebnisse der Vorprojektphase**	
9	**Wesentliche Stakeholder-Beziehungen der Vorprojektphase**	t
10	Website-Entwickler	1
11	Lieferanten von Offlinemarketing-Material	
12	**Getroffene Entscheidungen der Vorprojektphase**	t
13	Umzug des RGC Office mit Office Warming im Sept 2013	1
14	Bedarf nach neuem Online-Auftritt	
15	**Wesentliche in der Vorprojektphase erstellte Dokumente**	t
16	PHB RGC Umzug	1
17	Drafts von Briefing-Dokumenten für Optimierung des Online-Auftritts (tlw. auch CD)	
18	Angebote zur Optimierung des Online-Auftritts (tlw. auch CD)	
		x
20	**Erwartungen bezüglich der Nachprojektphase**	x
21	**Erwartete Weiterentwicklung der Stakeholder-Beziehungen**	t
22	Kooperationsbeziehung zu grafikfabrik für Offline-/Onlinelösungen	1
23	Neue Lieferantenbeziehungen für Offlinemarketing-Material	
24	**Maßnahmen in der Nachprojektphase**	t
25	Controlling der Einhaltung der CD Richtlinien	1
26		
27		x

```
' Theorie zum Einfügen und Löschen von Zeilen
' wir machen in der danach ausgeblendeten Spalte Z:
'   in Titelzeilen ein t - klickt man hier auf Zeile einfügen, springt der Code eins tiefer in die erste Datenzeile und kopiert diese
'   in der jeweils nicht löschbaren ersten Datenzeile eine 1
'   und in Leerzeilen, in ev. übergeordneten Haupttitelzeilen und in der Schlusszeile ein x - hier funktioniert das Einfügen von Zeilen nicht

Sub Zeile_Neu_VN()

' es darf nur eine Zelle / Zeile markiert sein
    If Selection.Rows.Count > 1 Then
        MsgBox Tabelle3.Range("E720")
        Exit Sub
    End If

' Es darf nicht vor Zeile 8 eingefügt werden
    If ActiveCell.Row < 8 Then
        MsgBox Tabelle3.Range("E722")
```

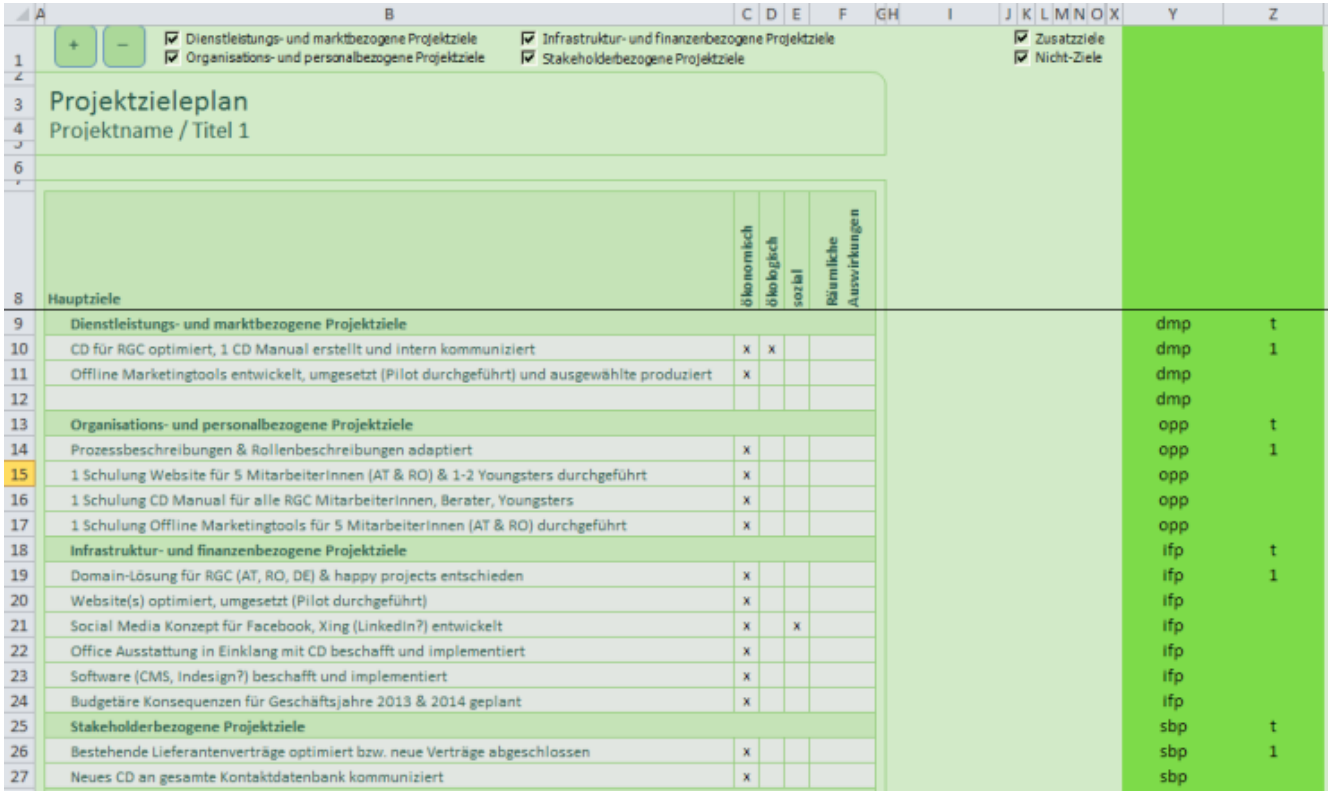
```
        Exit Sub
    End If

' Es darf nicht bei Leerzeilen / Hauptüberschriftzeilen und der Schlusszeile (mit x in Spalte Z) eingefügt werden
    If UCase(Cells(ActiveCell.Row, 26)) = "X" Then
        MsgBox Tabelle3.Range("E722")
        Exit Sub
    End If

' Kopieren und Einfügen

    ' in Titelzeilen springe eins tiefer in die erste Datenzeile
    If UCase(Cells(ActiveCell.Row, 26)) = "T" Then Cells(ActiveCell.Row + 1, 2).Select

    ' Kopieren und einfügen
    Rows(ActiveCell.Row).Copy
    Rows(ActiveCell.Row + 1).Insert Shift:=xlDown
    Application.CutCopyMode = False
    Cells(ActiveCell.Row + 1, 26) = "" ' ein eventuelles "1" der ersten Datenzeile löschen
    Cells(ActiveCell.Row + 1, 2).Select

End Sub


Sub Zeile_Entfernen_VN()

' es darf nur eine Zelle / Zeile markiert sein
If Selection.Rows.Count > 1 Then
    MsgBox Tabelle3.Range("E720")
    Exit Sub
End If

' Es darf nicht vor Zeile 10 gelöscht werden
If ActiveCell.Row < 10 Then
    MsgBox Tabelle3.Range("E721")
    Exit Sub
End If

' Es darf keine Titelzeile (t), keine Schlusszeile (x), Haupttitelzeile (x), Leerzeile (x) und keine erste Datenzeile mit 1 in Spalte Z gelöscht werden
If Cells(ActiveCell.Row, 26) <> "" Then
    MsgBox Tabelle3.Range("E721")
    Exit Sub
End If
```

```
' löschen
    Rows(ActiveCell.Row).Delete
    Cells(ActiveCell.Row, 2).Select

End Sub
```

Vorlagen-Zeile duplizieren / Block ein-ausblenden

Im Gegensatz zum nächsten Eintrag "Zeilen-Vorlage ein/ausblenden" wo nur fix angelegte Vorlagenzeilen ein- oder ausgebelendet werden, macht dieser Code mehr:

Mit der PLUS/MINUS-Taste können neue Zeilen eingefügt werden (als Duplikat der ersten Vorlagen-Datenzeile unendlich einfügbar).
Mit der Checkbox kann immer ein ganzer Bereich auf einmal ausgeblendet/eingeblendet werden



Hauptziele	ökonomisch	ökologisch	sozial	Räumliche Auswirkungen			
Dienstleistungs- und marktbezogene Projektziele						dmp	t
CD für RGC optimiert, 1 CD Manual erstellt und intern kommuniziert	x	x				dmp	1
Offline Marketingtools entwickelt, umgesetzt (Pilot durchgeführt) und ausgewählte produziert	x					dmp	
						dmp	
Organisations- und personalbezogene Projektziele						opp	t
Prozessbeschreibungen & Rollenbeschreibungen adaptiert	x					opp	1
1 Schulung Website für 5 MitarbeiterInnen (AT & RO) & 1-2 Youngsters durchgeführt	x					opp	
1 Schulung CD Manual für alle RGC MitarbeiterInnen, Berater, Youngsters	x					opp	
1 Schulung Offline Marketingtools für 5 MitarbeiterInnen (AT & RO) durchgeführt	x					opp	
Infrastruktur- und finanzenbezogene Projektziele						ifp	t
Domain-Lösung für RGC (AT, RO, DE) & happy projects entschieden	x					ifp	1
Website(s) optimiert, umgesetzt (Pilot durchgeführt)	x					ifp	
Social Media Konzept für Facebook, Xing (LinkedIn?) entwickelt	x		x			ifp	
Office Ausstattung in Einklang mit CD beschafft und implementiert	x					ifp	
Software (CMS, Indesign?) beschafft und implementiert	x					ifp	
Budgetäre Konsequenzen für Geschäftsjahre 2013 & 2014 geplant	x					ifp	
Stakeholderbezogene Projektziele						sbp	t
Bestehende Lieferantenverträge optimiert bzw. neue Verträge abgeschlossen	x					sbp	1
Neues CD an gesamte Kontaktdatenbank kommuniziert	x					sbp	

```
' THEORIE
' Es sollen neue Zeilen eingefügt werden können und auch wieder gelöscht werden
' und es sollen ganze Bereiche ausgeblendet werden können
' in Spalte Z ist ein x bei der Schlusszeile, damit sie weder gelöscht noch vervielfältigt werden kann
' in Spalte Z ist ein t bei Titelzeilen der Blöcke und eine 1 bei deren erster Zeile - weil diese nicht gelöscht werden sollen
' in Spalte Y ist fürs Ausblenden bei den zusammengehörigen Zeilen jeweils ein Kürzel - zB nizi für die Nichtziele-Zeilen
```

```
Sub Zeile_Neu_Ziele()

' es darf nur eine Zelle / Zeile markiert sein
If Selection.Rows.Count > 1 Then
    MsgBox Tabelle3.Range("E720")
    Exit Sub
End If

' Es darf nicht vor Zeile 9 eingefügt werden
If ActiveCell.Row < 9 Then
    MsgBox Tabelle3.Range("E722")
    Exit Sub
End If

' Es darf nicht bei der Schlusszeile (mit x in Spalte Z) eingefügt werden
If Cells(ActiveCell.Row, 26) = "x" Then
    MsgBox Tabelle3.Range("E722")
    Exit Sub
End If

' wenn dieser Code läuft sind wir entweder in einer Titelzeile (t) oder in der ersten Zeile (1) oder in einer normalen Datenzeile (leer)
    If UCase(Cells(ActiveCell.Row, 26)) = "T" Then Cells(ActiveCell.Row + 1, 5).Select ' in Titelzeilen: gehe eins tiefer
    Rows(ActiveCell.Row).Copy
    Rows(ActiveCell.Row + 1).Insert Shift:=xlDown
    Application.CutCopyMode = False
    Cells(ActiveCell.Row + 1, 26) = "" ' ein eventuelles "1" der ersten Zeile löschen
    Cells(ActiveCell.Row + 1, 2).Select

End Sub


Sub Zeile_Entfernen_Ziele()

' es darf nur eine Zelle / Zeile markiert sein
If Selection.Rows.Count > 1 Then
    MsgBox Tabelle3.Range("E720")
    Exit Sub
End If

' Es darf nicht vor Zeile 10 gelöscht werden
If ActiveCell.Row < 10 Then
    MsgBox Tabelle3.Range("E721")
    Exit Sub
```

```
End If

' Es darf die Schlusszeile mit x in Spalte Z, eine Titelzeile mit t und keine erste Zeile mit 1 in Spalte Z gelöscht werden
If Cells(ActiveCell.Row, 26) <> "" Then
   MsgBox Tabelle3.Range("E721")
   Exit Sub
End If

' löschen
   Rows(ActiveCell.Row).Delete
   Cells(ActiveCell.Row, 2).Select

End Sub
Sub Blende_DMP()
' dies ist der Code für 1.Checkbox zum Ein/Ausblenden der Dienstleistungs und Marktbezogenen Ziele

For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "DMP" Then
      If Tabelle8.CheckBoxes("DMP").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
      End If
   End If
Next Z
End Sub
Sub Blende_OPP()
For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "OPP" Then
      If Tabelle8.CheckBoxes("OPP").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
      End If
   End If
Next Z
End Sub
Sub Blende_IFP()
For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "IFP" Then
      If Tabelle8.CheckBoxes("IFP").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
```

```
      End If
   End If
Next Z
End Sub
Sub Blende_SBP()
For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "SBP" Then
      If Tabelle8.CheckBoxes("SBP").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
      End If
   End If
Next Z
End Sub
Sub Blende_ZUZI()
For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "ZUZI" Then
      If Tabelle8.CheckBoxes("Zuzi").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
      End If
   End If
Next Z
End Sub
Sub Blende_NIZI()
For Z = 9 To LETZTEZELLE(ActiveSheet.Name).Row
   If UCase(Cells(Z, 25)) = "NIZI" Then
      If Tabelle8.CheckBoxes("Nizi").Value = 1 Then
         Rows(Z).Hidden = False
       Else
         Rows(Z).Hidden = True
      End If
   End If
Next Z
End Sub
```

Vorlagen-Zeile ein/ausblenden

Siehe auch den vorigen Eintrag Zeilen-Vorlage duplizieren / Block ein-ausblenden

Bei der Projektmamagement-Software haben wir immer wieder Vorlagen-Zeilen ausgeblendet - etwa, wenn wir dem User nur 5 per default eingeblendet sein lassen, aber er kann mit +/- Buttons sich z.B. weitere 45 Zeilen ein oder auch wieder ausblenden

```
Sub Status_Blende_Ein_IPK()

For Z = 0 To 44
   If Rows(36 + Z).Hidden = True Then ' 36 ist die oberste und letzte Zeile, die ausgeblendet wird
      Rows(36 + Z).Hidden = False
      End
   End If
Next Z

End Sub


Sub Status_Blende_Aus_IPK()

For Z = 0 To 44
   If Rows(80 - Z).Hidden = False Then ' 80 ist die unterste und erste Zeile, die ausgeblendet wird
      Rows(80 - Z).Hidden = True
      End
   End If
Next Z

End Sub
```

Zeilen + Spalten ausblenden / einblenden

siehe auch den Eintrag hier "Vorlagen-Zeile ein/ausblenden"

Wichtig: Befindet sich VBA-Code direkt in einem Tabellenblatt, bezieht sich Range("A1") IMMER auf die Zelle A1 in diesem Tabellenblatt, SELBST WENN DER VBA-CODE auf eine andere Tabelle umgeschalten hat und diese aktiv wäre. Nur in Modulen greifen Range und Cells-Befehle OHNE Tabellenbezug auf die aktuelle Tabelle.

Wechselschalter für Zeile aus-/ein-blenden:

```
Rows(3).Hidden = Not Rows(3).Hidden
Rows(4).Hidden = Not Rows(4).Hidden
Rows(5).Hidden = Not Rows(5).Hidden
Rows(6).Hidden = Not Rows(6).Hidden
Rows(7).Hidden = Not Rows(7).Hidden
I.Rows(9).Hidden = Not I.Rows(9).Hidden ' Die Zeile 9 klappt aus unerfindlichen Grund nur, wenn man den Tabellennamen (I) mitübergibt
```

Ausblenden einer Zeile:

```
ActiveSheet.Rows("2:10").Hidden = True
ActiveSheet.Rows(2).Hidden = True

Rows("153:153").EntireRow.Hidden = True
Rows(z).EntireRow.Hidden = True ' Wenn die Zeilennummer in der Variable Z gespeichert ist
```

Einblenden mehrerer Zeilen:

```
Rows("153:155").EntireRow.Hidden = False
```

Ausblenden von Spalten

```
Columns("E:G").EntireColumn.Hidden = True
```

Will man Spalten mit Ziffern ausblenden muss das Sheet-Objekt mit übergeben werden

ActiveSheet.Columns(5).Hidden = True
Tabelle3.Columns(1).Hidden = False

Ist in einer Parametertabelle in der Zelle D21 ein Spaltenbuchstabe hinterlegt, dann übergibt man diesen z.B. so:

Tabelle3.Columns(Columns(Range("D21").Value).Column).Hidden = True

Und möchte man Spalten voll automatisch durch eine Definitionszeile aus- und einblenden lassen in deren Worksheet_Change-Code:


```
If Target.Row = 20 Then '
    On Error Resume Next
    Application.ScreenUpdating = False
    If Target.Value = "JA" Then
       If Cells(21, Target.Column) <> "" Then
          Tabelle3.Columns(Columns(Cells(21, Target.Column).Value).Column).Hidden = False
       End If
     Else
       If Cells(21, Target.Column) <> "" Then
          Tabelle3.Columns(Columns(Cells(21, Target.Column).Value).Column).Hidden = True
       End If
    End If
    On Error GoTo 0
    Application.ScreenUpdating = True
```

```
End If
```

ALLE ZEILEN EINBLENDEN

```
Tabelle1.Columns.Hidden = False
```

Achtung: ich bekam immer wieder mal einen 400-er Fehler, wenn der Code für das Ausblenden / Einblenden auf einer Tabelle im Tabellen-Code einer anderen Tabelle vorkam – auch wenn ich dort ordnungsgemäß die gewünschte Tabelle aktiviere und mich in ihr befand. Lösung: der betreffende Code muss einfach in ein allgemeines Modul.

Zeile einfügen und löschen

```
Rows("1").Insert Shift:=xlDown ' einfügen vor Zeile 1

Rows("1:3").delete ' die Zeilen 1-3 löschen
```

Zeile kopieren + einfügen + löschen

```
' Eine Zeile Kopierem
   Cells(1,1).Copy
' Zielzelle markieren
   Range("B2").Select
' Werte einfügen
   Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks :=False, Transpose:=False
```

Code von Simplefibu, der aktuelle Zeile kopiert und unten wieder einfügt

Sub Zeile_Neu()

```
' es darf nur eine Zelle / Zeile markiert sein
If Selection.Rows.Count > 1 Then
   MsgBox Tabelle3.Range("E720")
```

```
    Exit Sub
End If

' Es darf nicht vor Zeile 7 eingefügt werden
If ActiveCell.Row < 7 Then
    MsgBox Tabelle3.Range("E722")
    Exit Sub
End If

' Es darf nicht bei der Schlusszeile (mit x in Spalte A) eingefügt werden
If Cells(ActiveCell.Row, 1) <> "" Then
    MsgBox Tabelle3.Range("E722")
    Exit Sub
End If

' wenn dieser Code läuft sollten wir eigentlich in einer Zeile sein, die wir kopieren und unten einfügen können
    Rows(ActiveCell.Row).Copy
    Rows(ActiveCell.Row + 1).Insert Shift:=xlDown
    Application.CutCopyMode = False
    Cells(ActiveCell.Row + 1, 2).Select
```

End Sub

Sub Zeile_Entfernen()

```
' es darf nur eine Zelle / Zeile markiert sein
If Selection.Rows.Count > 1 Then
    MsgBox Tabelle3.Range("E720")
    Exit Sub
End If

' Es darf nicht vor Zeile 8 gelöscht werden
If ActiveCell.Row < 8 Then
    MsgBox Tabelle3.Range("E721")
    Exit Sub
End If

' Es darf die Schlusszeile mit x in Spalte A nicht gelöscht werden
If Cells(ActiveCell.Row, 1) <> "" Then
    MsgBox Tabelle3.Range("E721")
    Exit Sub
End If

' löschen
    Rows(ActiveCell.Row).Delete
    Cells(ActiveCell.Row, 2).Select
```

End Sub

Zeilenformat kopieren + mehrfach einfügen (Vorlagenzeile)

Für variable lange Auswertungen lege ich gerne eine Default Vorlagenzeile an (in der Regel oberhalb des variabel langen Auswertungsblockes) und lasse sie immer ausgeblendet.

Für die eigentliche Auswertung mache ich eine Schleife durch alle Auswertungs-Resultatzeilen und ich lege die benötigten Resultatzeilen nur durch Kopieren der ganzen Zeile und einfügen nur des Formates ein. Es werden also keine echten, neuen Zeilen eingefügt, sondern die Schleife geht einfach nur die benötigten Auswertungszeilen durch nach unten und überträgt von der Vorlagenzeile die Formate (inkl. Zeilenhöhe) in die jeweilige Auswertungszeile

Wichtig: Formatübertragen geht nur, wenn die Vorlagenzeile eingeblendet ist - sondern wird ihr "HIDDEN"-Status auch gleich noch mit übertragen :o)

Nachfolgendes Beispiel aus der Projektmanagement-Software von mir überträgt gleich einen Auswertungszeilenblock (5 Zeilen hoch)

```
' ----------------------------
' --- Übertragen der Daten ---
' ----------------------------

' Layouten des ausgeblendeten Vorlagendatensatzes

    ' Einblenden (wichtig für Übertragen des Formats)
    Rows("12:16").Hidden = False

    ' Z: das ist die Zeilenvariable hier für die Auswertungstabelle
    Z = 20 ' Defaultzeile für oberste Zeile des ersten Datensatzes hier in der Auswertungs-Tabelle

' Schleife durch alle Zeilen mit Projektdaten in der Datentabelle (die Variable DZ geht durch die ganzen Datenzeilen dieser Tabelle)

For DZ = 11 To 100



SCHLEIFENBEGINN:

    ' Kopieren der Vorlagenzeilen

    Rows("12:16").Copy
    Rows(Z & ":" & Z + 4).Select
    Selection.PasteSpecial Paste:=xlPasteFormats ', Operation:=xlNone, SkipBlanks:=False, Transpose:=False

    '  Weiterer Code zum Befüllen des Auswertungsblockes
.
.
.
```

.
.

```
    ' Weiterspringen zur nächsten Anfangszeile
    Z = Z + 6

Next DZ

' Ausblenden der Vorlagenzeilen
    Rows("12:16").Hidden = True
    Range("D20").Select

Application.ScreenUpdating = True
```

Zeilenhöhe automatisch optimieren

nur bei unformatierten Zellen (wenn nicht verbunden) kann man - indem man im Zellformat den Zellumbruch aktiviert - damt einstellen, dass beim Ausfüllen von einer Zelle die Zeilenhöhe automatisch angepasst wird, sobald die Eingabe über die Zellenbreite hinausgeht.

Meist geht das auch - aber wenn die Zellen schon irgendwie formatiert sind, dann greift der Automatismus irgendwie nicht mehr. Vor allem, wenn manuell schon mal die Zeilenbreite vom Excelstandard abweichend hinterlegt wurde. Am besten die betreffenden Zeilen markieren und mit DOPPELClick auf einen Zeilennummer-Zellrand die AUTO-Zeilenhöhe wieder aktivieren.

Alternativ hilft VBA: Etwa die betreffenden Zeilen Autofitten: Rows("10:20").Autofit

Wichtig: Vorher muss ein ev. gesetzter Tabellenblattschutz aufgehoben sein, sonst kommt die Fehlermeldung: "Die AutoFit-Methode des Range-Objektes konnte nicht ausgeführt werden"

Um Zeilenhöhen bei Zelleingaben generell anzupassen, kann man auf VBA zurückgreifen.

Hier meine Lösung vom RGC-Tool

Die betreffende Zelle ist ein Zellverbund aus mehreren Zellen (z.B. A1:C1) und darum greift das Häkchen bei ZEILENUMBRUCH im Zellformat nicht. Man muss es aber für unsere Lösung dennoch setzen in diesem Bereich A1:C1 !

Ich mache eine Spalte (z.B. AA) ganz weit rechts außerhalb des Arbeitsbereiches genauso breit wie die verbundenen Zellen. Diese Zelle AA1 muss dieselbe Schriftgröße haben wie der verbundene Bereich. Und in diese "Dummy"-Zelle AA1 verlinke ich mittels Formel =A1 direkt in meine verbundene Zelle, sodass die Dummy-Zelle AA1 den Inhalt genau erbt. Wenn ich nun in dieser Dummyzelle AA1 den Zeilenumbruch aktiviere, wird automatisch beim Ausfüllen von der Zelle A1:C1 die Zeilenhöhe angepasst.

Im betreffenden Arbeitsblatt fügt man noch folgenden Code ein:

```
Private Sub Worksheet_Change(ByVal Target As Range)

        ' Zeilenhöhe autom. anpassen
        Rows(Target.Row).EntireRow.AutoFit

End Sub
```

TIPP: Die Dummyspalte AA kann ausgeblendet werden ! Sie muss nur dieselbe Breite haben, wie die Spalten A bis C.

Eventuell wenn Schreibschutz aktiv ist, diesen kurz öffnen

```
' Zeilenhöhe autom. anpassen
   If Target.Row > 20 Then
      MERKER = ActiveSheet.ProtectContents
      If MERKER = True Then AUF
         Rows(Target.Row).EntireRow.AutoFit
      If MERKER = True Then ZU
   End If
```

Variante 1: USED RANGE

```
With ActiveSheet.UsedRange
        .Columns.AutoFit
        .Rows.AutoFit
End With
```

Damit werden alle Zellen in der aktuellen Tabelle, sowohl hinsichtlich Zeilenhöhe, als auch Spaltenbreite optimiert.

Variante 2: für alle Zellen bei ihrer Eingabe:

Gut geht es auch direkt im allgemeinen Code des Tabellenblattes, dass man jede Zelle, die befüllt wird, anschließend automatisch gleich optimiert:

Dies geht mit folgendem Code nur bei NICHT verbundenen Zellen:

```
Private Sub Worksheet_Change(ByVal Target As Range)

   Rows(Target.Row & ":" & Target.Row).Select
   Selection.Rows.AutoFit
   Target.Cells.Select

End Sub
```

Variante 3: fixer Bereich

ActiveSheet.Range("A1:A10").Rows.EntireRow.AutoFit

Damit werden alle Zellen im Bereich A1:A10 hinsichtlich der Zeilenhöhe optimiert

Variante 4: automatische Überwachung aller Zellen

Der Code ist im betreffenden Tabellenblatt direkt im Bereich WORKSHEET einzufügen - oder, wenn man es für die gesamte Arbeitsmappe möchte, direkt im Tabellenblatt DieseArbeitsmappe (jedoch dort mit Private Sub **Workbook_SheetChange**)

```
Private Sub Worksheet_Change(ByVal Target As Range)

' Eingaben sollen nur in den Zeilen 10-100 und dort nur in den Spalten D-F zur Anpassung führen
   If ActiveCell.Row > 9 And ActiveCell.Row < 101 And ActiveCell.Column > 3 And ActiveCell.Column < 7 Then
      Rows(Target.Row & ":" & Target.Row).Select
      Selection.Rows.AutoFit
      Target.Cells.Select
   End If

End Sub
```

Hier werden alle Zellen im gewünschten Bereich der betreffenden Tabelle überwacht und sobald man in einer Zelle was eingibt, wird die Zeile der Zelle optimiert.

Variante 5 - für Verbundene Zellen:

1. Listing:

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim CurrentRowHeight As Single, MergedCellRgWidth As Single
   Dim CurrCell As Range
   Dim ActiveCellWidth As Single, PossNewRowHeight As Single
   If ActiveCell.Row > 0 And ActiveCell.Row < 101 Then
   If ActiveCell.MergeCells Then
     With ActiveCell.MergeArea
          If .Rows.Count = 1 And .WrapText = True Then
             Application.ScreenUpdating = False
             CurrentRowHeight = .RowHeight
             ActiveCellWidth = ActiveCell.ColumnWidth
             For Each CurrCell In Selection
                MergedCellRgWidth = CurrCell.ColumnWidth + MergedCellRgWidth
             Next
             .MergeCells = False
             .Cells(1).ColumnWidth = MergedCellRgWidth
```

```
            .EntireRow.AutoFit
            PossNewRowHeight = .RowHeight
            .Cells(1).ColumnWidth = ActiveCellWidth
            .MergeCells = True
            .RowHeight = IIf(CurrentRowHeight > PossNewRowHeight, _
              CurrentRowHeight, PossNewRowHeight)
          End If
        End With
    End If
    End If
Application.ScreenUpdating = True
End Sub
```

2.Listing

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim CurrentRowHeight As Single, MergedCellRgWidth As Single
   Dim CurrCell As Range
   Dim ActiveCellWidth As Single, PossNewRowHeight As Single
   If ActiveCell.MergeCells Then
     With ActiveCell.MergeArea
        If .Rows.Count = 1 And .WrapText = True Then
           Application.ScreenUpdating = False
           CurrentRowHeight = .RowHeight
           ActiveCellWidth = ActiveCell.ColumnWidth
           For Each CurrCell In Selection
              MergedCellRgWidth = CurrCell.ColumnWidth + MergedCellRgWidth
           Next
           .MergeCells = False
           .Cells(1).ColumnWidth = MergedCellRgWidth
           .EntireRow.AutoFit
           PossNewRowHeight = .RowHeight
           .Cells(1).ColumnWidth = ActiveCellWidth
           .MergeCells = True
           .RowHeight = IIf(CurrentRowHeight > PossNewRowHeight, _
             CurrentRowHeight, PossNewRowHeight)
        End If
      End With
   End If
Application.ScreenUpdating = True
End Sub
```

3.Listing:

```
Sub autofitXLMergedCells()
```

```
'Passt Zeilenhöhe an den Text innerhalb von verbundenen Zellen im selektierten Bereich an
'(Die Autofit-Methode des Excel-Range-Objektes funktioniert für verbundene Zellen nicht)
On Error GoTo Err_autofitXLMergedCells

Dim lo_CurRange As Excel.Range

Dim lsg_SumCellWidths As Single
Dim lsg_OriginalWidthFirstCol As Single
Dim lsg_NewRowHeight As Single
Dim li_MergedCellsCount As Integer


With Selection
   If .MergeCells Then
      If .Rows.Count = 1 And .WrapText = True Then
         lsg_OriginalWidthFirstCol = .Cells(, 1).ColumnWidth

         'Einzelzellbreiten und Breiten der Gitterlinien summieren
         For Each lo_CurRange In Selection
            lsg_SumCellWidths = lo_CurRange.ColumnWidth + lsg_SumCellWidths
            li_MergedCellsCount = li_MergedCellsCount + 1
         Next

         lsg_SumCellWidths = lsg_SumCellWidths + (li_MergedCellsCount - 1) * 0.71

         'Verbindung der Zellen aufheben, erste (datentragende) Zelle auf Gesamtbreite ausdehnen und
         'Höhe anpassen über Standardmethode
         .MergeCells = False
         .Cells(1).ColumnWidth = lsg_SumCellWidths
         .EntireRow.AutoFit

         'Resultierende Zeilenhöhe merken, erste Zelle zurücksetzen, Verbindung wiederherstellen, Höhe anpassen
         lsg_NewRowHeight = .RowHeight
         .Cells(1).ColumnWidth = lsg_OriginalWidthFirstCol
         .MergeCells = True
         .RowHeight = lsg_NewRowHeight
      End If
   End If
End With

Exit Sub
'Nur für's Debuggen
Resume
```

```
Err_autofitXLMergedCells:
MsgBox Err.Number & ": " & Err.Description

End Sub
```

Zellbereich mit Maustaste auswählen und Markierung anzeigen

```
Dim Bereich As Range
Set Bereich = Application.InputBox("Wählen Sie die Zelle in die der Eintrag erfolgen soll", Type:=8)
```

Zellbereiche die getrennt sind

GETRENNTE TABELLENBEREICHE MARKIEREN

```
Dim GESAMTBEREICH As Range
Set GESAMTBEREICH = Range("A1") ’ alternativ: Range(Cells(r + 21, 3).Address)
Set GESAMTBEREICH = Union(GESAMTBEREICH, Range("A5")) ’ alternativ: Range(Cells(r + 21, 3).Address)
GESAMTBEREICH.Select
```

In der Regel wird in irgendeiner Schleife der Gesamtbereich immer mehr erweitert durch den UNION-Befehl.

Vor dem ersten Union-Befehl muss aber der Gesamtbereich bereits mittels zB SET GESAMTBEREICH = RANGE ("A1") einmal befüllt werden, sonst kann er nicht mittels UNION mit neuem Bereich fusioniert werden.

Meine Empfehlung:

```
Dim BEREICH As Range

Set BEREICH = Range("IV65000") ' defaultwert jenseits von Gut und Böse :o)
.
.
.
If … ' Bedingung für Markierungserweiterung

        If BEREICH.Address = "$IV$65000" Then ' wenn noch Defaultwert
            Set BEREICH = Range(FORMATSPALTENANFANG & ZEILE & ":" & FORMATSPALTENENDE & ZEILE)
          Else
```

```
            Set BEREICH = Union(BEREICH, Range(FORMATSPALTENANFANG & ZEILE & ":" & FORMATSPALTENENDE & ZEILE))
        End If

End if
```

SCHLEIFE DURCH ALLE MARKIERTEN ZELLEN auch wenn getrennt

```
Sub Zeilen()
   For Each a In Selection.Areas
         MsgBox a.Rows.Row
   Next a
End Sub
```

MARKIERTE ZELLEN MERKEN UND AUSWÄHLEN AUCH WENN GETRENNT

```
Dim MARKIERUNG

Set MARKIERUNG = Selection
' nun kann man was anderes markieren
....
' und hier markiert man wieder die ursprüngliche Markierung
MARKIERUNG.Select
```

Zellbereich anwählen verhindern

Siehe auch nachfolgenden Eintrag ZELLBEREICH BESCHRÄNKEN DER ANWÄHLBAREN SCROLLAREA

```
ActiveSheet.EnableSelection = xlNoSelection ' nichts anwählen können
ActiveSheet.EnableSelection = xlUnlockedCells ' die ungeschützten anwählen können
ActiveSheet.EnableSelection = xlNoRestrictions ' alle Zellen anwählen können
```

Excel-Entwicklerreferenz

Worksheet.EnableSelection-Eigenschaft

Gibt die Elemente zurück, die auf dem Blatt ausgewählt werden können, oder legt diese fest. **XlEnableSelection**-Wert mit Lese-/Schreibzugriff.

Syntax

Ausdruck.**EnableSelection**

Ausdruck Eine Variable, die ein **Worksheet**-Objekt darstellt.

Anmerkungen

Diese Eigenschaft ist nur wirksam, wenn das Arbeitsblatt geschützt ist: **xlNoSelection**xlNoSelection verhindert jegliche Markierung auf dem Blatt, **xlUnlockedCells**xlUnlockedCells gestattet nur die Markierung der Zellen, deren **Locked**-Eigenschaft **False**False ist, und **xlNoRestrictions**xlNoRestrictions erlaubt die Markierung einer beliebigen Zelle.

Beispiel

In diesem Beispiel wird festgelegt, dass im ersten Arbeitsblatt keine Elemente markiert werden können.

Visual Basic für Applikationen

```
With Worksheets(1)
    .EnableSelection = xlNoSelection
    .Protect Contents:=True, UserInterfaceOnly:=True
End With
```

Zellbereich beschränken der anwählbaren ScrollArea

Als ScrollArea wird ein definierter Bereich bezeichnet, innerhalb dessen in Excel gescrollt werden kann. Die Einstellung der Eigenschaft finden Sie im VBA-Editor (Aufruf mit [*Alt*] + [*F11*]). Wählen Sie dazu das gewünschte Tabellenblatt aus und tragen Sie bei der Eigenschaft ScrollArea die gewünschte Angabe, beispielsweise $A$1:$Z$100 ein. Anschließend kann nur noch auf diesen Bereich zugegriffen werden. Bereiche außerhalb der angegebenen Range können nicht mehr angesprochen und verwendet werden.

Allerdings funktioniert diese Einstellung nur so lange, bis Sie die Datei geschlossen haben. Nach dem erneuten Aufrufen der Datei, auch wenn diese gespeichert wurde, ist die eingestellte ScrollArea nicht mehr vorhanden und es kann wieder auf den gesamten Zellbereich der Tabelle zugegriffen werden.

Um die **ScrollArea dauerhaft** einzustellen, müssen Sie diese bei jedem Öffnen der Datei erneut eintragen. Gehen Sie dazu wie folgt vor:

1. Öffnen Sie über die Tastenkombination [*Alt*] + [*F11*] den VBA-Editor.
2. Öffnen Sie den Code-Container "Diese Arbeitsmappe" und erfassen Sie folgenden Code:

```
1.Private Sub Workbook_Open()
```

```
2.ThisWorkbook.Sheets(1).ScrollArea = "$A$1:$Z$100"

3.End Sub
```

3. Wenn Sie nun die Datei schließen und erneut öffnen, wird der gewünschte Scroll-Bereich eingetragen und steht dauerhaft zur Verfügung.

Wenn Sie die ScrollArea für unterschiedliche Tabellenblätter eintragen möchten, können Sie den VBA-Code entsprechend erweitern. So zeigt der folgende VBA-Code die Einstellung der ScrollArea für die Tabellenblätter 1, 3 und 4.

```
Private Sub Workbook_Open()
ThisWorkbook.Sheets(1).ScrollArea = "$A$1:$Z$100"
ThisWorkbook.Sheets(3).ScrollArea = "$A$10:$B$50"
ThisWorkbook.Sheets(4).ScrollArea = "$A$1:$H$500"
End Sub
```

Wie lautet die Anweisung, um meherer Bereiche z.B. A1:A10, C1:C10 und F1:H20 beim Scrollen gleichzeitig einzuschränken?

bei mehreren Bereichen mußt Du immer die letzte Zelle des Bereiches aktivieren, damit der neue Gültig wid.

```
scrollAreas definieren, wenn du Lust hast:
Private Sub Worksheet_Activate()
ActiveSheet.ScrollArea = "$A$1:$C$10"
End Sub
,hier wird zwischen zweien gewechselt
,du kannst aber noch mehrere definieren
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
If Target.Address = "$C$10" Then
    ActiveSheet.ScrollArea = "$D$11:$E$20"
ElseIf Target.Address = "$E$20" Then
    ActiveSheet.ScrollArea = "$A$1:$C$10"
End If
End Sub
```

Zelleinzug erhöhen / verringern

```
Sub ZELLEINZUG_MEHR()
' Dies ist die Funktion zum Erhöhen des Zelleinzuges

' check, dass nicht mehr als eine Spalte markiert ist
If Selection.Columns.Count > 1 Then
   MsgBox Tabelle3.Range("E718")
   Exit Sub
End If

For ZEILE = ActiveCell.Row To ActiveCell.Row + Selection.Rows.Count - 1
   ' check, dass Zelle nicht geschützt ist
   If Cells(ZEILE, ActiveCell.Column).Locked = True Then
      MsgBox Tabelle3.Range("E719")
      Exit Sub
   End If
   ' erhöhe den Einzug
   Cells(ZEILE, ActiveCell.Column).InsertIndent 1

Next ZEILE

End Sub

Sub ZELLEINZUG_WENIGER()
' Dies ist die Funktion zum Verringern des Zelleinzuges

' check, dass nicht mehr als eine Spalte markiert ist
If Selection.Columns.Count > 1 Then
   MsgBox Tabelle3.Range("E718")
   Exit Sub
End If

For ZEILE = ActiveCell.Row To ActiveCell.Row + Selection.Rows.Count - 1
   ' check, dass Zelle nicht geschützt ist
   If Cells(ZEILE, ActiveCell.Column).Locked = True Then
      MsgBox Tabelle3.Range("E719")
      Exit Sub
   End If
   ' erhöhe den Einzug
   Cells(ZEILE, ActiveCell.Column).InsertIndent -1
```

Next ZEILE

End Sub

Zellen kopieren ohne Zwischenablage

```
Sub Kopieren()
    Dim aBereich As Range, bBereich As Range
    Set aBereich = Range("A1:B2")
    Set bBereich = Range("F1:G2")
''  Werte übertragen
    bBereich.Value = aBereich.Value
''  Zahlenformate übertragen
    bBereich.NumberFormat = aBereich.NumberFormat
End Sub
```

Zellen mit Formeln schützen

Am einfachsten schützt man Zellen mit Formeln durch einen Blattschutz. Wenn man dabei das Anwählen von geschützten Zellen NICHT erlaubt, hat man den Vorteil, dass das Programm automatisch über diese Zellen drüberspringt bis zur nächsten Zelle, wo wieder kein Schutz ist. Das führt also zu einer automatischen Ausfüllung.

Wenn man jedoch keinen Blattschutz setzen möchte und Zellen mit Formeln dennoch schützen möchte, kann man folgenden Code direkt im Tabellenblatt-Code verwenden, der automatisch eine Zeile höher springt mit der Markierung:

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

   If ActiveCell.HasFormula = True Then ActiveCell.Offset(-1, 0).Select

End Sub

Zelle in anderer Arbeitsmappe auslesen

```
' Zellen aus einer Tabelle in einer Arbeitsmappe auslesen
Workbooks("MeineDaten.xls").Sheets("Tabelle1").Range("A1:C1").Copy
Workbooks("DeineDaten.xls").Sheets(1).Cells(3, 5).PasteSpecial xlpastevalues

' Wenn man die Tabelle mit ihrem VBA-Namen (hier: nur P) ansprechen möchte
Workbooks("DeineDaten.xls").Sheets(P.Name).Cells(3, 5).Copy
```

Zelle suchen in Bereich oder Tabellenblatt

mein schnelles Finden eines Wertes - siehe hier im Bereich " Bestimmte Zellen für Index-Referenzierung finden "

Zellformat übertragen

Möchte man nicht den Inhalt des Ziels überschreiben, sondern nur das Zellformat übertragen, geht dies am einfachsten so:

```
Range("A1:B1").Copy
Range("C1").PasteSpecial xlFormats ' alternativ: xlPasteFormats
Application.CutCopyMode = False
```

Der größere Bereich der Quelle wird automatisch auf den Zielbereich ausgeweitet, sodass dort auch D1 formatiert wird

Zellformat auslesen (ob Datum, Zahlenformat)

```
MsgBox IsDate(ActiveCell.Value)

MsgBox ActiveCell.NumberFormat
```

Wichtig: bei aktivierter 1904-Option dürfen Zellen mit Datum nur mit VALUE übertragen werden und nicht ohne, weil sonst Excel die 4 Jahre Differenz übergibt.
Also nicht: cells(1,1)=cells(2,1), sondern cells(1,1).Value=cells(2,1).Value

Zellformat (Zahlenformat) einstellen

```
Selection.NumberFormat = "General"
```

```
Selection.NumberFormat = "0.00"
```

Zellformat (Farbe, Schriftart, Rahmenlinie)

```
Selection.Interior.ColorIndex = 1          ' ändert die Feldfarbe der Auswahl auf Schwarz
Selection.Interior.ColorIndex = xlNone     ' ändert auf farblose Feldfarbe
```

BSP: Ändern der Schriftart eines Bereiches

```
With Selection.Font
    .Name = "Times New Roman"
    .Size = 10
End With

Selection.Font.ColorIndex = 2  ' Schriftfarbe auf Weiss setzen
```

Rahmenlinien

Setzen der Linie schnell

```
Selection.Borders(xlEdgeTop).LineStyle = xlContinuous ' Linie oben
```

Setzen der Linie genau (daher, auch Linienart und Liniendicke):

```
With Selection.Borders(xlEdgeLeft) ' linke Linie
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = xlAutomatic
End With
```

Löschen von Rahmenlinien

```
Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Borders(xlDiagonalUp).LineStyle = xlNone
Selection.Borders(xlEdgeLeft).LineStyle = xlNone
Selection.Borders(xlEdgeTop).LineStyle = xlNone
Selection.Borders(xlEdgeBottom).LineStyle = xlNone
Selection.Borders(xlEdgeRight).LineStyle = xlNone
Selection.Borders(xlInsideVertical).LineStyle = xlNone
```

```
    Selection.Borders(xlInsideHorizontal).LineStyle = xlNone
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
```

Zellformat - Farbe Spezial

FARBEN FÜR ZELLINHALT UND RAHMENLINIEN EINSTELLEN

```
' Zellinhalt färben
Selection.Interior.ColorIndex = 56
Selection.Interior.ColorIndex = xlNone ' alle Zellfarben löschen

' eine bestimmte Farbe ändern
    ActiveWorkbook.Colors(15) = RGB(234, 234, 234)


' Rahmenlinien löschen
    Selection.Borders(xlEdgeLeft).LineStyle = xlNone
    Selection.Borders(xlEdgeTop).LineStyle = xlNone
    Selection.Borders(xlEdgeBottom).LineStyle = xlNone
    Selection.Borders(xlEdgeRight).LineStyle = xlNone
    Selection.Borders(xlInsideVertical).LineStyle = xlNone
    Selection.Borders(xlInsideHorizontal).LineStyle = xlNone

' Rahmenlinien ziehen und färben
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlThin ' xlMedium or xlThick
        .ColorIndex = 44
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = 44
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = 1
```

```
End With
With Selection.Borders(xlEdgeRight)
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = 1
End With


' 4 Arten eine Farbe zu übergeben an Userform-Element  - hier jedesmal ROT
UserForm2.CommandButton1.BackColor = vbRed
UserForm2.CommandButton1.BackColor = &HFF& ' Langfassung = &H000000FF&
UserForm2.CommandButton1.BackColor = QBColor(12)
UserForm2.CommandButton1.BackColor = ActiveWorkbook.Colors(3)
```

Farbnummer in Excel

Nummern der Tasten in meiner Useform

ft.at

80	79	78	77	76	75	74	73
40	39	38	37	36	35	34	33

EXCELS DEFAULTFARBENWERTE

(1) = RGB(0, 0, 0)
(2) = RGB(255, 255, 255)
(3) = RGB(255, 0, 0)
(4) = RGB(0, 255, 0)
(5) = RGB(0, 0, 255)
(6) = RGB(255, 255, 0)
(7) = RGB(255, 0, 255)
(8) = RGB(0, 255, 255)
(9) = RGB(128, 0, 0)
(10) = RGB(0, 128, 0)
(11) = RGB(0, 0, 128)
(12) = RGB(128, 128, 0)
(13) = RGB(128, 0, 128)
(14) = RGB(0, 128, 128)
(15) = RGB(192, 192, 192)
(16) = RGB(128, 128, 128)
(33) = RGB(0, 204, 255)
(34) = RGB(204, 255, 255)
(35) = RGB(204, 255, 204)
(36) = RGB(255, 255, 153)
(37) = RGB(153, 204, 255)
(38) = RGB(255, 153, 204)
(39) = RGB(200, 153, 255)
(40) = RGB(255, 204, 153)
(41) = RGB(51, 102, 255)
(42) = RGB(51, 204, 204)
(43) = RGB(153, 204, 0)
(44) = RGB(255, 204, 0)
(45) = RGB(255, 153, 0)
(46) = RGB(255, 102, 0)
(47) = RGB(102, 102, 153)

(48) = RGB(150, 150, 150)
(49) = RGB(0, 51, 102)
(50) = RGB(51, 153, 102)
(51) = RGB(0, 51, 0)
(52) = RGB(51, 51, 0)
(53) = RGB(153, 51, 0)
(54) = RGB(153, 51, 102)
(55) = RGB(51, 51, 153)
(56) = RGB(51, 51, 51)

MEINE EXCEL-FARBEN (nur die Abweichungen)

neu Version 2009:
(3) = RGB(218, 0, 0)
(4) = RGB(173, 250, 134)
(5) = RGB(17, 57, 255)
(6) = RGB(255, 255, 155)
(7) = RGB(255, 37, 37)
(8) = RGB(109, 255, 199)
(9) = RGB(104, 0, 0)
(10) = RGB(32, 153, 23)
(11) = RGB(15, 0, 210)
(12) = RGB(255, 168, 51)
(13) = RGB(118, 0, 118)
(14) = RGB(0, 102, 138)
(15) = RGB(232, 232, 232)
(16) = RGB(176, 176, 176)
(33) = RGB(55, 150, 255)
(34) = RGB(200, 255, 255)
(35) = RGB(219, 255, 200)
(36) = RGB(255, 255, 200)
(37) = RGB(183, 210, 255)
(38) = RGB(255, 201, 201)
(39) = RGB(219, 183, 255)
(40) = RGB(255, 233, 200)
(41) = RGB(105, 141, 255)
(42) = RGB(87, 239, 255)
(43) = RGB(255, 226, 51)
(44) = RGB(254, 209, 144)
(45) = RGB(230, 181, 116)
(46) = RGB(218, 137, 70)
(47) = RGB(89, 62, 142)
(48) = RGB(211, 211, 211)

(49) = RGB(0, 12, 102)
(50) = RGB(78, 231, 57)
(51) = RGB(0, 90, 0)
(52) = RGB(164, 82, 0)
(53) = RGB(103, 58, 13)
(54) = RGB(155, 96, 234)
(55) = RGB(49, 43, 133)
(56) = RGB(77, 77, 77)

alt 2008:

(3) = RGB(218, 0, 0)
(4) = RGB(173, 250, 134)
(5) = RGB(17, 57, 255)
(6) = RGB(255, 255, 155)
(7) = RGB(255, 37, 37)
(8) = RGB(109, 255, 199)
(9) = RGB(104, 0, 0)
(10) = RGB(32, 153, 23)
(11) = RGB(15, 0, 210)
(12) = RGB(255, 168, 51)
(13) = RGB(118, 0, 118)
(14) = RGB(0, 102, 138)
(15) = RGB(232, 232, 232)
(16) = RGB(176, 176, 176)
(33) = RGB(55, 150, 255)
(34) = RGB(217, 255, 255)
(35) = RGB(219, 255, 213)
(36) = RGB(255, 255, 217)
(37) = RGB(183, 210, 255)
(38) = RGB(255, 183, 183)
(39) = RGB(219, 183, 255)
(40) = RGB(252, 238, 212)
(41) = RGB(105, 141, 255)
(42) = RGB(87, 239, 255)
(43) = RGB(255, 226, 51)
(44) = RGB(254, 209, 144)
(45) = RGB(230, 181, 116)
(46) = RGB(218, 137, 70)
(47) = RGB(89, 62, 142)
(48) = RGB(211, 211, 211)
(49) = RGB(0, 12, 102)
(50) = RGB(78, 231, 57)
(51) = RGB(0, 72, 0)
(52) = RGB(164, 82, 0)
(53) = RGB(103, 58, 13)
(54) = RGB(155, 96, 234)
(55) = RGB(49, 43, 133)
(56) = RGB(67, 67, 67)

Displaying A Color Picker Dialog

The modColorFunctions module contains a function named ChooseColorDialog that will display a Windows Color Picker dialog and return the RGB Long color value. If the user cancels the dialog, the result is -1. For example,

```
Dim RGBColor As Long
Dim Default As Long
Default = RGB(255, 0, 255) 'default to purple
RGBColor = ChooseColorDialog(DefaultColor:=Default)
If RGBColor < 0 Then
    Debug.Print "*** USER CANCELLED"
Else
    Debug.Print "Choice: " & Hex(RGBColor)
End If
```

Hier mein Code aus Powertools

```
Option Explicit
Option Compare Text


Public Const C_RGB_RED As Long = &HFF&
Public Const C_RGB_GREEN As Long = &HFF00&
Public Const C_RGB_BLUE As Long = &HFF0000
Public Const C_RGB_WHITE As Long = &HFFFFFF
Public Const C_RGB_BLACK As Long = &H0&
Public Const C_MIN_COLOR_INDEX = 1
Public Const C_MAX_COLOR_INDEX = 56
Public Const C_MIN_RGB = C_RGB_BLACK
Public Const C_MAX_RGB = C_RGB_WHITE
Public Const C_SHIFT_ONE_BYTE = &H100&
Public Const C_SHIFT_TWO_BYTES = &H10000
```

```
Private Declare Function GetActiveWindow Lib "user32.dll" () As Long
Private Declare Function ChooseColorDlg Lib "comdlg32.dll" Alias "ChooseColorA" ( _
    pChoosecolor As CHOOSECOLOR) As Long


Private Const CC_RGBINIT = &H1&
Private Const CC_FULLOPEN = &H2&
Private Const CC_PREVENTFULLOPEN = &H4&
Private Const CC_SHOWHELP = &H8&
Private Const CC_ENABLEHOOK = &H10&
Private Const CC_ENABLETEMPLATE = &H20&
Private Const CC_ENABLETEMPLATEHANDLE = &H40&
Private Const CC_SOLIDCOLOR = &H80&
Private Const CC_ANYCOLOR = &H100&


Private Type CHOOSECOLOR
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    rgbResult As Long
    lpCustColors As Long
    flags As Long
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type


Private dwCustClrs(0 To 15) As Long
Private Init As Boolean

Public Function ChooseColorDialog(DefaultColor As Long) As Long
    Dim lpChoosecolor As CHOOSECOLOR
    With lpChoosecolor
        .lStructSize = Len(lpChoosecolor)
        .hwndOwner = GetActiveWindow
        .rgbResult = DefaultColor
        .lpCustColors = VarPtr(dwCustClrs(0))
        .flags = CC_ANYCOLOR Or CC_RGBINIT Or CC_FULLOPEN
    End With
```

```
    If ChooseColorDlg(lpChoosecolor) Then
        ChooseColorDialog = lpChoosecolor.rgbResult
    Else
        ChooseColorDialog = -1
    End If
End Function

Sub teste()
MsgBox ActiveCell.Offset(-1, 0)
End Sub


Sub FORMAT_FARBWAHL()

    Dim RGBColor As Long
    Dim Default As Long
    Default = RGB(255, 0, 255) 'default to purple
    Default = ActiveCell.Offset(1, 0) 'default to purple
    RGBColor = ChooseColorDialog(DefaultColor:=Default)
    If RGBColor < 0 Then
        Debug.Print "*** USER CANCELLED"
    Else
        ActiveCell.Offset(1, 0) = RGBColor
        ActiveWorkbook.Colors(1) = RGBColor
    End If

End Sub


Functions For Color Values
The modColorFunctions module contains a number of functions for working with RGB colors and color index values.
ColorIndexOfRGBLong
This returns the Color Index value of the specified RGB Long color value, if it exists in the current pallet. Otherwise, it returns 0.
IsColorPalletDefault
This returns True if the pallet associated with the specified workbook is the application default pallet. This returns False if the pallet has been modified with
Workbook.Colors.
IsColorIndexDefault
This returns True if the color associated with the specified color index is the same as the application default color index value. This tells you if the color
associated with a color index value has been changed.
RGBComponentsFromRGBLongToVariables
This splits an RGB Long value into the constituent red, green, and blue values, which are returned to the caller in the ByRef variables. The function's result
```

is True if the input value was a valid RGB color or False if the input value was not a valid RGB color. For example,

```
    Dim RGBColor As Long
    Dim Red As Long
    Dim Green As Long
    Dim Blue As Long
    Dim B As Boolean

    RGBColor = ActiveCell.Interior.Color
    B = RGBComponentsFromRGBLongToVariables(RGBColor, Red, Green, Blue)
    If B = True Then
        Debug.Print "Red: " & Red, "Blue: " & Blue, "Green: " & Green
    Else
        Debug.Print "Invalid value in RGBColor"
    End If
```

RGBComponentsFromRGBLong

This splits an RGB Long color value into the red, green, and blue components and returns them as an array of Longs.

```
    Arr(1) = Red
    Arr(2) = Green
    Arr(3) = Blue
```

UND HIER DER CODE FÜR MODECOLORFUNCTIONS:

```
Attribute VB_Name = "modColorFunctions"
Option Explicit
Option Compare Text
Option Base 1
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' modColorFunctions
' By Chip Pearson, chip@cpearson.com
'       www.cpearson.com
```

```
'      This module described at www.cpearson.com/Excel/Colors.aspx
' Date: 24-January-2008.
'
' This module contains procedures for working with colors in
' Excel. The following functions are provided:
'      ChooseColorDialog
'          This display a standard Windows color picker
'          dialog and returns the selected RGB Long.
'      RGBLong
'          This returns the RGB Long Integer from a Red,
'          Blue, and Green color values.
'      ColorIndexOfRGBLong
'          This returns the ColorIndex associated with an
'          RGB color value.
'      RGBLongFromColorIndex
'          This returns the RGB value corresponding to
'          a ColorIndex.
'      ColorIndexOfRange
'          This returns an array of ColorIndex values
'          of a range of cells. The ColorIndex of either
'          the Font property or the Interior property may
'          be used.
'      ColorIndexOfOneCell
'          This returns the ColorIndex of a single cell.
'          The ColorIndex of either the Font property or
'          the Interior property may be used.
'      IsValidColorIndex
'          This indicates whether a value is a valid ColorIndex.
'      RGBComponentsFromRGBLong
'          This returns a 1-base 3-element array contain the
'          component Red, Green, and Blue values of an RGB color.
'      RGBComponentsFromRGBLongToVariables
'          This sets ByRef Long variables to the component
'          Red, Green, and Blue values of an RGB color.
'      IsValidRGBLong
'          This indicates if a Long Integer is a valid RGB color.
'      CountColors
'          This counts the number of cells in a range that have
'          a specified color index.
'          The ColorIndex of either the Font property or
'          the Interior property may be used.
```

```
'     SumColors
'         This sums the cells that have a specified color
'         index value. You can specify different ranges
'         for the color test range and the values to sum
'         range (e.g., sum values in C where colum A is red).
'         The ColorIndex of either the Font property or
'         the Interior property may be used.
'     DefaultColorPallet
'         This returns an array of the DEFAULT color values.
'         The array is either a 1-based array of 56 RGB colors
'         or a 0-based array of 57 elements where
'         element 0 = -1 And elements 1 to 56 are RGB colors,'
'         depending on the Option Base of the module that
'         contains the function. Regardless of Option Base,
'         a ColorIndex may be used as an index into the array.
'     DefaultColorNames
'         This returns an array of the DEFAULT US-English
'         color names. These are the same names that appear
'         as control tips in the color drop down for the
'         fill color and text color command buttons. ColorIndex
'         values 1 to 16 and 26 to 56 have names. ColorIndex
'         values from 17 to 25 do not have names. The array
'         is properly based so that a ColorIndex value may
'         be used as the index into the array.
'     IsColorIndexDefault
'         This returns TRUE if color corresponding to the
'         specified ColorIndex is the default Excel color
'         or FALSE if the color corresponding to the specified
'         ColorIndex is not the system default.
'     IsColorPalletDefault
'         This returns TRUE if the color pallet for the specified
'         workbook is unchanged, or FALSE if the color pallet
'         has been changed.
'     ColorName
'         This returns the US-English name of a specified RGB
'         color. The specified RGB color must be present in
'         the Excel Default Pallet, but it is not required
'         that the RGB color reside the same location in the
'         Excel Default Pallet as the location in the current
'         in-use pallet. If no matching RGB color is found,
'         the result is vbNullString. If the color resides
```

```
'           in the Excel Default Pallet in locations 17 through
'           25 (inclusive), the result is "UNNAMED". There
'           is no way to get a color name if the RGB color
'           is not a member of the Excel Default Pallet.
'       RangeOfColors
'           This returns a Range object containing the individual
'           cells whose Font or Interior have the specified
'           color index.
'
'
' In all functions, ColorIndex is a value between 1 and 56
' (inclusive) or xlColorIndexNone or xlColorIndexAutomatic.
' RGBLong is a Long Integer between &H00000000 and &H00FFFFFF.
' An RGB color is stored in a 8-byte Long Integer, having bytes
' equal to, left to right:
'   ||00|BB|GG|RR||
'   where BB are the Blue bytes, GG are the Green bytes, and
'   RR are the Red bytes.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Const C_RGB_RED As Long = &HFF&
Public Const C_RGB_GREEN As Long = &HFF00&
Public Const C_RGB_BLUE As Long = &HFF0000
Public Const C_RGB_WHITE As Long = &HFFFFFF
Public Const C_RGB_BLACK As Long = &H0&
Public Const C_MIN_COLOR_INDEX = 1
Public Const C_MAX_COLOR_INDEX = 56
Public Const C_MIN_RGB = C_RGB_BLACK
Public Const C_MAX_RGB = C_RGB_WHITE
Public Const C_SHIFT_ONE_BYTE = &H100&
Public Const C_SHIFT_TWO_BYTES = &H10000


Private Declare Function GetActiveWindow Lib "user32.dll" () As Long
Private Declare Function ChooseColorDlg Lib "comdlg32.dll" Alias "ChooseColorA" ( _
    pChoosecolor As CHOOSECOLOR) As Long


Private Const CC_RGBINIT = &H1&
Private Const CC_FULLOPEN = &H2&
```

```
Private Const CC_PREVENTFULLOPEN = &H4&
Private Const CC_SHOWHELP = &H8&
Private Const CC_ENABLEHOOK = &H10&
Private Const CC_ENABLETEMPLATE = &H20&
Private Const CC_ENABLETEMPLATEHANDLE = &H40&
Private Const CC_SOLIDCOLOR = &H80&
Private Const CC_ANYCOLOR = &H100&


Private Type CHOOSECOLOR
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    rgbResult As Long
    lpCustColors As Long
    flags As Long
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type


Private dwCustClrs(0 To 15) As Long
Private Init As Boolean

Public Function ChooseColorDialog(DefaultColor As Long) As Long
    Dim lpChoosecolor As CHOOSECOLOR
    With lpChoosecolor
        .lStructSize = Len(lpChoosecolor)
        .hwndOwner = GetActiveWindow
        .rgbResult = DefaultColor
        .lpCustColors = VarPtr(dwCustClrs(0))
        .flags = CC_ANYCOLOR Or CC_RGBINIT Or CC_FULLOPEN
    End With
    If ChooseColorDlg(lpChoosecolor) Then
        ChooseColorDialog = lpChoosecolor.rgbResult
    Else
        ChooseColorDialog = -1
    End If
End Function
```

```
Function RGBLong(Red As Long, Green As Long, Blue As Long) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RGBLong
' This function returns a Long integer representing the color
' defined by the Red, Green, and Blue values. It returns -1
' if any parameter is not in the range 0 <= V <= 255.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    If (Red < 0) Or (Red > 255) Then
        RGBLong = -1
        Exit Function
    End If
    If (Green < 0) Or (Green > 255) Then
        RGBLong = -1
        Exit Function
    End If
    If (Blue < 0) Or (Blue > 255) Then
        RGBLong = -1
        Exit Function
    End If
    RGBLong = RGB(Red, Green, Blue)
End Function

Function ColorIndexOfRGBLong(RGBLong As Long) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ColorIndexOfRGBLong
' This returns the ColorIndex into the color pallet of the
' color represented by RGBLong. It returns -1 if the RGBLong
' is not found in the pallet or if RGBLong is < C_RGB_BLACK or
' greater than C_RGB_WHITE.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim V As Variant
If (RGBLong < C_RGB_BLACK) Or (RGBLong > C_RGB_WHITE) Then
    ColorIndexOfRGBLong = 0
    Exit Function
End If

V = Application.Match(RGBLong, ThisWorkbook.Colors, 0)
If IsError(V) = True Then
    ColorIndexOfRGBLong = 0
Else
```

```
    ColorIndexOfRGBLong = CLng(V)
End If

End Function

Function RGBLongFromColorIndex(ColorIndex As Long) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RGBLongFromColorIndex
' This returns the RGB Color corresponding to the specified
' ColorIndex. It returns -1 if ColorIndex is not between 1
' and 56.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
If (ColorIndex <= 0) Or (ColorIndex >= 57) Then
    RGBLongFromColorIndex = -1
Else
    RGBLongFromColorIndex = ThisWorkbook.Colors(ColorIndex)
End If
End Function

Function ColorIndexOfRange(InRange As Range, _
    Optional OfText As Boolean = False, _
    Optional DefaultColorIndex As Long = -1) As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ColorIndexFromRange
' This function returns an array of values, each of which is
' the ColorIndex of a cell in InRange. If InRange contains both
' multiple rows and multiple columns, the array is two dimensional,
' number of rows x number of columns. If InRange is either a single
' row or a single column, the array is single dimensional. If
' InRange has multiple rows, the array is transposed before
' returning it. The DefaultColorIndex indicates what color
' index to value to substitute for xlColorIndexNone and
' xlColorIndexAutomatic. If OfText is True, the ColorIndex
' of the cell's Font property is returned. If OfText is False
' or omitted, the ColorIndex of the cell's Interior property
' is returned.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Arr() As Long
Dim NumRows As Long
Dim NumCols As Long
```

```
Dim RowNdx As Long
Dim ColNdx As Long
Dim CI As Long
Dim Trans As Boolean

Application.Volatile True
If InRange Is Nothing Then
    ColorIndexOfRange = CVErr(xlErrRef)
    Exit Function
End If
If InRange.Areas.Count > 1 Then
    ColorIndexOfRange = CVErr(xlErrRef)
    Exit Function
End If
If (DefaultColorIndex < -1) Or (DefaultColorIndex > 56) Then
    ColorIndexOfRange = CVErr(xlErrValue)
    Exit Function
End If

NumRows = InRange.Rows.Count
NumCols = InRange.Columns.Count

If (NumRows > 1) And (NumCols > 1) Then
    ReDim Arr(1 To NumRows, 1 To NumCols)
    For RowNdx = 1 To NumRows
        For ColNdx = 1 To NumCols
            CI = ColorIndexOfOneCell(Cell:=InRange(RowNdx, ColNdx), _
                OfText:=OfText, DefaultColorIndex:=DefaultColorIndex)
            Arr(RowNdx, ColNdx) = CI
        Next ColNdx
    Next RowNdx
    Trans = False
ElseIf NumRows > 1 Then
    ReDim Arr(1 To NumRows)
    For RowNdx = 1 To NumRows
        CI = ColorIndexOfOneCell(Cell:=InRange.Cells(RowNdx, 1), _
            OfText:=OfText, DefaultColorIndex:=DefaultColorIndex)
        Arr(RowNdx) = CI
    Next RowNdx
    Trans = True
Else
```

```
    ReDim Arr(1 To NumCols)
    For ColNdx = 1 To NumCols
        CI = ColorIndexOfOneCell(Cell:=InRange.Cells(1, ColNdx), _
            OfText:=OfText, DefaultColorIndex:=DefaultColorIndex)
        Arr(ColNdx) = CI
    Next ColNdx
    Trans = False
End If

If IsObject(Application.Caller) = False Then
    Trans = False
End If

If Trans = False Then
    ColorIndexOfRange = Arr
Else
    ColorIndexOfRange = Application.Transpose(Arr)
End If

End Function

Function ColorIndexOfOneCell(Cell As Range, OfText As Boolean, _
    DefaultColorIndex As Long) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ColorIndexOfOneCell
' This returns the ColorIndex of the cell referenced by Cell.
' If Cell refers to more than one cell, only Cell(1,1) is
' tested. If OfText True, the ColorIndex of the Font property is
' returned. If OfText is False, the ColorIndex of the Interior
' property is returned. If DefaultColorIndex is >= 0, this
' value is returned if the ColorIndex is either xlColorIndexNone
' or xlColorIndexAutomatic.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim CI As Long

Application.Volatile True
If OfText = True Then
    CI = Cell(1, 1).Font.ColorIndex
Else
    CI = Cell(1, 1).Interior.ColorIndex
End If
```

```
If CI < 0 Then
    If IsValidColorIndex(ColorIndex:=DefaultColorIndex) = True Then
        CI = DefaultColorIndex
    Else
        CI = -1
    End If
End If

ColorIndexOfOneCell = CI

End Function

Private Function IsValidColorIndex(ColorIndex As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsValidColorIndex
' This returns TRUE if ColorIndex is between 1 and 56 or equal
' to either xlColorIndexNone or xlColorIndexAutomatic. It
' returns FALSE otherwise.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Select Case ColorIndex
    Case 1 To 56, xlColorIndexNone, xlColorIndexAutomatic
        IsValidColorIndex = True
    Case Else
        IsValidColorIndex = False
End Select
End Function


Function RGBComponentsFromRGBLong(RGBLong As Long) As Long()
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RGBComponentsFromRGBLong
' This accepts an RGBLong and returns a 1-based array with
' three elements, containing, left-to-right, the Red, Green,
' and Blue components of the RGB Color. If RGBLong is not
' a valid RGB color, all elements of the returned array
' are -1.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Arr(1 To 3) As Long

If IsValidRGBLong(RGBLong:=RGBLong) = False Then
```

```
    Arr(1) = -1
    Arr(2) = -1
    Arr(3) = -1
    Exit Function
End If

Arr(1) = RGBLong And C_RGB_RED
Arr(2) = (RGBLong And C_RGB_GREEN) \ C_SHIFT_ONE_BYTE ' shift right 1 byte
Arr(3) = (RGBLong And C_RGB_BLUE) \ C_SHIFT_TWO_BYTES ' shift right 2 bytes

RGBComponentsFromRGBLong = Arr

End Function

Function RGBComponentsFromRGBLongToVariables(RGBLong As Long, _
    ByRef Red As Long, ByRef Green As Long, ByRef Blue As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RGBComponentsFromRGBLongToVariables
' This set the variables references Red, Green, and Blue to
' the component colors of the RGBLong color. It returns
' True if RGBLong is a valid color (between &H00000000 and
' &H00FFFFFF) or False if RGBLong is not a valid RGB color.
' If RGBLong is invalid, the component variables are set to -1.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Arr As Variant
If IsValidRGBLong(RGBLong) = True Then
    Arr = RGBComponentsFromRGBLong(RGBLong)
    Red = Arr(1)
    Green = Arr(2)
    Blue = Arr(3)
    RGBComponentsFromRGBLongToVariables = True
Else
    Red = -1
    Green = -1
    Blue = -1
    RGBComponentsFromRGBLongToVariables = False
End If

End Function

Function IsValidRGBLong(RGBLong As Long) As Boolean
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsValidRGBLong
' This returns True if RGBLong is between &H00000000 and
' &H00FFFFFF or False otherwise.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
If (RGBLong >= C_MIN_RGB) And (RGBLong <= C_MAX_RGB) Then
    IsValidRGBLong = True
Else
    IsValidRGBLong = False
End If

End Function

Function CountColor(InRange As Range, ColorIndex As Long, _
    Optional OfText As Boolean = False) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' CountColor
' This function counts the cells in InRange whose ColorIndex
' is equal to the ColorIndex parameter. The ColorIndex of the
' Font is tested if OfText is True, or the Interior property
' if OfText is omitted or False. If ColorIndex is not a valid
' ColorIndex (1 -> 56, xlColorIndexNone, xlColorIndexAutomatic)
' 0 is returned. If ColorIndex is 0, then xlColorIndexNone is
' used if OfText is Fasle or xlColorIndexAutomatic if OfText
' is True. This allows the caller to use a value of 0 to indicate
' no color for either the Interior or the Font.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim R As Range
Dim N As Long
Dim CI As Long

If ColorIndex = 0 Then
    If OfText = False Then
        CI = xlColorIndexNone
    Else
        CI = xlColorIndexAutomatic
    End If
Else
    CI = ColorIndex
End If
```

```
Application.Volatile True
Select Case ColorIndex
    Case 0, xlColorIndexNone, xlColorIndexAutomatic
        ' OK
    Case Else
        If IsValidColorIndex(ColorIndex) = False Then
            CountColor = 0
            Exit Function
        End If
End Select

For Each R In InRange.Cells
    If OfText = True Then
        If R.Font.ColorIndex = CI Then
            N = N + 1
        End If
    Else
        If R.Interior.ColorIndex = CI Then
            N = N + 1
        End If
    End If
Next R

CountColor = N

End Function

Function SumColor(TestRange As Range, SumRange As Range, _
    ColorIndex As Long, Optional OfText As Boolean = False) As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' SumColor
' This function returns the sum of the values in SumRange where
' the corresponding cell in TestRange has a ColorIndex (of the
' Font is OfText is True, or of the Interior is OfText is omitted
' or False) equal to the specified ColorIndex. TestRange and
' SumRange may refer to the same range. An xlErrRef (#REF) error
' is returned if either TestRange or SumRange has more than one
' area or if TestRange and SumRange have differing number of
' either rows or columns. An xlErrValue (#VALUE) error is
```

```
' returned if ColorIndex is not a valid ColorIndex value.
' If ColorIndex is 0, xlColorIndexNone is used if OfText is
' False or xlColorIndexAutomatic if OfText is True. This allows
' the caller to specify 0 for no color applied.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim D As Double
Dim N As Long
Dim CI As Long

Application.Volatile True
If (TestRange.Areas.Count > 1) Or _
   (SumRange.Areas.Count > 1) Or _
   (TestRange.Rows.Count <> SumRange.Rows.Count) Or _
   (TestRange.Columns.Count <> SumRange.Columns.Count) Then
   SumColor = CVErr(xlErrRef)
   Exit Function
End If

If ColorIndex = 0 Then
   If OfText = False Then
      CI = xlColorIndexNone
   Else
      CI = xlColorIndexAutomatic
   End If
Else
   CI = ColorIndex
End If

Select Case CI
   Case 0, xlColorIndexAutomatic, xlColorIndexNone
      ' ok
   Case Else
      If IsValidColorIndex(ColorIndex:=ColorIndex) = False Then
         SumColor = CVErr(xlErrValue)
         Exit Function
      End If
End Select

For N = 1 To TestRange.Cells.Count
   With TestRange.Cells(N)
   If OfText = True Then
```

```
        If .Font.ColorIndex = CI Then
            If IsNumeric(.Value) = True Then
                D = D + .Value
            End If
        End If
    Else
        If .Interior.ColorIndex = CI Then
            If IsNumeric(.Value) = True Then
                D = D + .Value
            End If
        End If
    End If
    End With
Next N

SumColor = D

End Function

Function ColorName(RGBLong As Long) As String
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ColorName
' This function returns the US-English color name corresponding
' to the RGBLong color value. If the position of the RGBColor
' in the DEFAULT pallet is between 1 and 16 or 26 and 56, the
' color name is returned. If the location is between 17 and 25
' or the color is not in the DEFAULT pallet, the result is
' vbNullString. It is NOT required that the pallet in use be
' the default pallet, only that the RGBLong is one of the colors
' in the default pallet. The location of RGBLong in the current
' pallet is irrelevant.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim N  As Long
Dim DefaultNdx As Long
Dim DefaultPallet As Variant
Dim ColorNames As Variant

If IsValidRGBLong(RGBLong:=RGBLong) = False Then
    ColorName = vbNullString
    Exit Function
```

```
End If

DefaultPallet = DefaultColorPallet()
DefaultNdx = 0
For N = C_MIN_COLOR_INDEX To C_MAX_COLOR_INDEX
    If RGBLong = DefaultPallet(N) Then
        DefaultNdx = N
        Exit For
    End If
Next N
If DefaultNdx = 0 Then
    ColorName = vbNullString
Else
    ColorNames = DefaultColorNames()
    ColorName = ColorNames(DefaultNdx)
End If

End Function



Function DefaultColorPallet() As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DefaultColorPallet
' This returns an array of the DEFAULT color values for
' Excel 2003. Effects of an Option Base setting for the
' module that contains this function (NOT the module from
' which this function was called):
'   Option Base 0
'       Array is 0-based with 57 elements, element 0
'           is -1. Elements 1 to 56 are RGB colors.
'   Option Base 1
'       Array is 1-based with 56 elements, all of which are
'           RGB colors.
'   Option Base Omitted
'       Same as Option Base 0
' This allows the ColorIndex as the index into the array
' regardless of the module's Option Base setting.
'''''''''''' ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim L(1) As Long
```

```
If LBound(L) = 0 Then
    DefaultColorPallet = Array(-1, _
        &H0&, &HFFFFFF, &HFF&, &HFF00&, &HFF0000, &HFFFF&, &HFF00FF, &HFFFF00, &H80&, &H8000&, _
        &H800000, &H8080&, &H800080, &H808000, &HC0C0C0, &H808080, &HFF9999, &H663399, &HCCFFFF, &HFFFFCC, _
        &H660066, &H8080FF, &HCC6600, &HFFCCCC, &H800000, &HFF00FF, &HFFFF&, &HFFFF00, &H800080, &H80&, _
        &H808000, &HFF0000, &HFFCC00, &HFFFFCC, &HCCFFCC, &H99FFFF, &HFFCC99, &HCC99FF, &HFF99CC, &H99CCFF, _
        &HFF6633, &HCCCC33, &HCC99&, &HCCFF&, &H99FF&, &H66FF&, &H996666, &H969696, &H663300, &H669933, _
        &H3300&, &H3333&, &H3399&, &H663399, &H993333, &H333333)
Else
    DefaultColorPallet = Array( _
        &H0&, &HFFFFFF, &HFF&, &HFF00&, &HFF0000, &HFFFF&, &HFF00FF, &HFFFF00, &H80&, &H8000&, _
        &H800000, &H8080&, &H800080, &H808000, &HC0C0C0, &H808080, &HFF9999, &H663399, &HCCFFFF, &HFFFFCC, _
        &H660066, &H8080FF, &HCC6600, &HFFCCCC, &H800000, &HFF00FF, &HFFFF&, &HFFFF00, &H800080, &H80&, _
        &H808000, &HFF0000, &HFFCC00, &HFFFFCC, &HCCFFCC, &H99FFFF, &HFFCC99, &HCC99FF, &HFF99CC, &H99CCFF, _
        &HFF6633, &HCCCC33, &HCC99&, &HCCFF&, &H99FF&, &H66FF&, &H996666, &H969696, &H663300, &H669933, _
        &H3300&, &H3333&, &H3399&, &H663399, &H993333, &H333333)
End If
End Function


Function DefaultColorNames() As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DefaultColorNames
' This returns an array of the US-English color names associated with the
' DEFAULT color pallet. Effect of the Option Base statement of the module
' that contains this function (NOT the module from which this function is
' called):
'       Option Base 0
'           The returned array is 0-based with 57 elements, with element 0
'           equal to "UNNAMED" and elements 1 to 56 having the color name
'       Option Base 1
'           The returned array is 1-based with 56 elements, each of which
'           is a color name.
'       Option  Base Omitted
'           Same as Option Base 0
'   Regardless of the module's Option Base statement, the ColorIndex is
'   a valid index into the array.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim L(1) As Long
If LBound(L) = 0 Then
```

```
    DefaultColorNames = Array("UNNAMED", _
    "Black", "White", "Red", "Bright Green", "Blue", "Yellow", "Pink", "Turquoise", _
    "Dark Red", "Green", "Dark Blue", "Dark Yellow", "Violet", "Teal", "Gray 25%", "Gray 50%", _
    "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", _
    "Dark Blue", "Pink", "Yellow", "Turquoise", "Violet", "Dark Red", "Teal", "Blue", _
    "Sky Blue", "Light Turquoise", "Light Green", "Light Yellow", "Pale Blue", "Rose", "Lavender", "Tan", _
    "Light Blue", "Aqua", "Lime", "Gold", "Light Orange", "Orange", "Blue Gray", "Gray 40%", _
    "Dark Teal", "Sea Green", "Dark Green", "Olive Green", "Brown", "Plum", "Indigo", "Gray 80%")
Else
    DefaultColorNames = Array( _
    "Black", "White", "Red", "Bright Green", "Blue", "Yellow", "Pink", "Turquoise", _
    "Dark Red", "Green", "Dark Blue", "Dark Yellow", "Violet", "Teal", "Gray 25%", "Gray 50%", _
    "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", "UNNAMED", _
    "Dark Blue", "Pink", "Yellow", "Turquoise", "Violet", "Dark Red", "Teal", "Blue", _
    "Sky Blue", "Light Turquoise", "Light Green", "Light Yellow", "Pale Blue", "Rose", "Lavender", "Tan", _
    "Light Blue", "Aqua", "Lime", "Gold", "Light Orange", "Orange", "Blue Gray", "Gray 40%", _
    "Dark Teal", "Sea Green", "Dark Green", "Olive Green", "Brown", "Plum", "Indigo", "Gray 80%")
End If

End Function

Function IsColorIndexDefault(ColorIndex As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsColorIndexCustom
' This tests whether the RGB color corresponding to ColorIndex is the
' default pallet color or a custom color in the pallet. If the color corresponding
' to the ColorIndex is the same as the default color, the function returns TRUE.
' If the ColorIndex is not the default, the function return FALSE. If ColorIndex
' is not valid, the result is TRUE.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim ColorPallet As Variant
Dim DefaultRGB As Long
Dim PalletRGB As Long
ColorPallet = DefaultColorPallet()
If (ColorIndex < C_MIN_COLOR_INDEX) Or (ColorIndex > C_MAX_COLOR_INDEX) Then
    IsColorIndexDefault = False
    Exit Function
End If
DefaultRGB = ColorPallet(ColorIndex)
PalletRGB = ThisWorkbook.Colors(ColorIndex)
If DefaultRGB = PalletRGB Then
```

```
    IsColorIndexDefault = True
Else
    IsColorIndexDefault = False
End If

End Function

Function IsColorPalletDefault(Optional Workbook As Workbook = Nothing) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsColorPalletDefault
' This returns True or False indicating whether the color pallet associated
' with Workbook is the default pallet or has been modifed. The function returns
' True if all the colors in the pallet are the default colors or False if one
' or more colors have been modified from the default. The Workbook parameter
' specifies which workbook's pallet to test. If this parameter is missing, the
' ActiveWorkbook is tested.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim WB As Workbook
Dim DefaultPallet As Variant
Dim N As Long
If Workbook Is Nothing Then
    If Application.ActiveWorkbook Is Nothing Then
        IsColorPalletDefault = True
    Else
        Set WB = ActiveWorkbook
    End If
Else
    Set WB = Workbook
End If
DefaultPallet = DefaultColorPallet()
For N = 1 To 56
    If WB.Colors(N) <> DefaultPallet(N) Then
        IsColorPalletDefault = False
        Exit Function
    End If
Next N

IsColorPalletDefault = True

End Function
```

```
Function RangeOfColor(TestRange As Range, _
    ColorIndex As Long, Optional OfText As Boolean = False) As Range
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RangeOfColors
' This function returns a Range object containing the individual cells of
' TestRange whose Font (if OfText is True) or Interior (if OfText is False or
' omitted) has the color index specified by ColorIndex. Note that the function
' may return Nothing.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim R As Range
Dim RR As Range
Dim CI As Long

If ColorIndex = 0 Then
    If OfText = False Then
        CI = xlColorIndexNone
    Else
        CI = xlColorIndexAutomatic
    End If
End If

If ColorIndex <> 0 Then
    If IsValidColorIndex(ColorIndex) = False Then
        Exit Function
    End If
End If

For Each R In TestRange.Cells
    If OfText = False Then
        If R.Interior.ColorIndex = CI Then
            If RR Is Nothing Then
                Set RR = R
            Else
                Set RR = Application.Union(RR, R)
            End If
        End If
    Else
        If RR Is Nothing Then
            Set RR = R
        Else
```

```
            Set RR = Application.Union(RR, R)
        End If
    End If
Next R
Set RangeOfColor = RR

End Function

Function ColorNameOfRGB(RGBLong As Long) As String
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ColorNameOfRGB
' Returns the name of the color specified by RGBLong
' if RGBLong is in the application default pallet.
' Otherwise, returns vbNullString.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim N As Long
Dim V As Variant
If IsValidRGBLong(RGBLong) = False Then
    Exit Function
End If
V = Application.Match(RGBLong, ThisWorkbook.Colors, 0)
If IsError(V) = True Then
    Exit Function
End If

V = DefaultColorNames(V)
If V <> vbNullString Then
    ColorNameOfRGB = V
End If

End Function
```

Zellinhalt auftrennen nach Zellumbruch

Siehe FUNKTIONEN TEXT

Zellsprungrichtung

```
Application.MoveAfterReturnDirection = xlDown
```

```
Application.MoveAfterReturnDirection = xlToRight
Application.MoveAfterReturn = False ' kein Sprung
```

Zellverbund prüfen

Dieser Code untersucht, ob die Zelle B12 Teil eines Zellverbundes ist - und wenn ja, wieviele Zeilen und Spalten der Verbund hat

Es ist dabei unerheblich, ob B12 am Beginn oder irgendwo mitten im Zellverbund ist, wir erfahren immer die gleiche Anzahl von Zeilen und Spalten, die der gesamte Verbund hat mit dem rows.count und columns.count-Befehl. Erst der MergeArea.Address-Befehl zeigt uns, wo die Zelle im aktuellen Verbund ist, weil er uns die gesamte Adresse des Verbunds anzeigt.

```
MsgBox Range("B12").MergeCells
MsgBox Range("B12").MergeArea.Rows.Count
MsgBox Range("B12").MergeArea.Columns.Count
MsgBox Range("D12").MergeArea.Address
```

Zellwert ist Fehler-Wert

Manchmal möchte man Zellen, die einen Fehlerwert enthalten (weil nicht gültiger Bezug oder Division durch Null) von einer Bearbeitung durch VBA ausschließen.

Mit folgendem Befehl IsError kann man eine Zelle dahingehend untersuchen

```
' wenn kein Fehlerwert, dann …

If IsError(Workbooks(QUELLDATEI).Sheets(QUELLTABELLE).Cells(ZQ, 1)) = False Then
```

Zellformeln in Werte umwandeln

Möchte man eine ganze Tabelle, einen Tabellenbereich alle Formeln in Werte umwandeln geht dies ganz einfach mit

```
Range("A1:AF100").Value= Range("A1:AF100").Value
```

Zwischenablage gezielt einfügen

Neue VERSION 2015 inkl Hilfsprozedur LETZTEZELLE
Version 1: SPALTENREIHENFOLGE BLEIBT ERHALTEN

```
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range
' Neue Version 2015

   Dim ExcelLastCell As Range
   Dim Row As Long
   Dim Col As Long
   Dim LastRowWithData As Long
   Dim LastColWithData As Long
   Dim TheSheet As Worksheet
   Dim MERKER

   Set TheSheet = Worksheets(TABELLENBLATT)

   MERKER = Application.ScreenUpdating
   Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
    Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

   Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

   ' letzte Zeile mit Daten herausfinden
   LastRowWithData = ExcelLastCell.Row
   Row = ExcelLastCell.Row
   Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
      Row = Row - 1
   Loop
   LastRowWithData = Row

   ' letzte Spalte mit Daten herausfinden
   LastColWithData = ExcelLastCell.Column
   Col = ExcelLastCell.Column
   Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
      Col = Col - 1
   Loop
   LastColWithData = Col

   Set LETZTEZELLE = TheSheet.Cells(Row, Col)
   Application.ScreenUpdating = MERKER

End Function
```

```
Sub Zwischenablage_einfuegen()

' Fügt die Zwischenablage ein - jedoch über eine ausgeblendete Zwischentabelle "Zwischenablage"
' diese Zwischentabelle darf nicht jedes Mal neu erzeugt werden (Sheets.add), da dadurch die Zwischenablage geleert
wird
' daher: sie wird beim ersten Mal angelegt, wenn es sie noch nicht geben sollte - und dann immer ausgeblendet


Dim ERSTE_QUELLZEILE As Integer ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
Dim ERSTE_QUELLSPALTE As Integer ' was ist die erste Spalte von der Zwischenablage
Dim ERSTE_ZIELZEILE As Integer ' was ist die erste Zeile in der empfangenden Zieltabelle, ab der die Daten eingefügt
werden sollen
Dim ERSTE_ZIELSPALTE As Integer ' was ist die erste Spalte in der empfangenden Zieltabelle, ab der die Daten eingefügt
werden sollen
Dim AUFTRENNUNG_TRENNZEICHEN As Integer ' wurde eine TXT-Datei eingefügt, die mit Trennzeichen getrennte Daten enthält
- 1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, 0=keine Trennung

Dim AKTUELLE_TABELLE As String ' Für den Rücksprung in die aktuelle Tabelle



' ************************************************************************************************************
' -----------------
' Variablen setzen
' -----------------

' Hier bitte die Einstellungen für die Prozedur machen

    ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
    ERSTE_QUELLZEILE = 1

    ' was ist die erste Spalte von der Zwischenablage, die kopiert werden soll - in der Regel will man immer alle
Spalten, daher ab Spalte 1
    ERSTE_QUELLSPALTE = 1

    ' was ist die erste Zeile in der empfangenden Zielzeile, ab der die Daten eingefügt werden sollen
    ERSTE_ZIELZEILE = 2
```

```
    ' was ist die erste Spalte in der empfangenden Zielzeile, ab der die Daten eingefügt werden soll - in der Regel ab
Spalte 1
    ERSTE_ZIELSPALTE = 1

    ' sollen TXT-Daten aus der Zwischenablage noch aufgetrennt werden ?
    ' BSP: 0;100002377;20150629;4000 ... soll in 4 eigenständige Spalten getrennt werden
    ' 0= keine Auftrennung - 1=Semikolon - 2=Beistrich - 3=Tabulator - 4=Leerzeichen
    AUFTRENNUNG_TRENNZEICHEN = 1

' ************************************************************************************************************

    Application.ScreenUpdating = False

' merken des Namens der aktuellen Tabelle
    AKTUELLE_TABELLE = ActiveSheet.Name

' Falls nachfolgender Code nicht geht, weil die Tabelle Zwischenablage noch nicht vorhanden ist, soll Code zum
Tabellenanlegecode verzweigen
    On Error GoTo ZWISCHENABLAGE_ANLEGEN

' Zwischenablage_Tabelle einblenden
    Worksheets("Zwischenablage").Visible = True

' Eigentliche Fehlerabfangroutine, wenn Zwischenablage leer ist
    On Error GoTo FEHLER

' Einfügen der Daten aus der Zwischenablage in der Interimstabelle ZWISCHENABLAGE
    Worksheets("Zwischenablage").Activate
    Range("A1").Select ' wir fügen die Daten immer ab der ersten Zeile in Spalte A ein - eine eventuell andere erste
Zelle in der Zieltabelle wird unten eingestellt
    ActiveSheet.Paste

' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten)
    Sheets(AKTUELLE_TABELLE).Activate
    Sheets(AKTUELLE_TABELLE).Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE),
Cells(Sheets(AKTUELLE_TABELLE).UsedRange.Rows.Count, Sheets(AKTUELLE_TABELLE).UsedRange.Columns.Count)).ClearContents

' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
```

```
    Range(Cells(ERSTE_QUELLZEILE, ERSTE_QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row,
LETZTEZELLE(ActiveSheet.Name).Column)).Copy

' alternatives Einfügen, wenn der Code der Hilfsprozedur LETZTEZELLE nicht vorhanden wäre
    ' Range(Cells(1, 1), Cells(ActiveSheet.Name.UsedRange.Rows.Count, ActiveSheet.Name.UsedRange.Columns.Count)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' Falls aus Zwischenablage keine Excelzellen-Daten kommen, sondern aus zB dem Editor die Werte einer TXT-Datei
' die noch aufgetrennt werden müssen - zB 0;100002377;20150629;4000 ...
' Hier gemäß dem Trennzeichen erfolgt die Auftrennung: 1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, (0=keine
Trennung)

    If AUFTRENNUNG_TRENNZEICHEN > 0 Then ' falls Auftrennung gewünscht, markiere die Spalte, in der alle Daten sind
        Range(Cells(ERSTE_ZIELZEILE, ERSTE_ZIELSPALTE), Cells(ActiveSheet.UsedRange.Rows.Count,
ERSTE_ZIELSPALTE)).Select
    End If

    Select Case AUFTRENNUNG_TRENNZEICHEN
        Case 1 ' Semikolon
            Selection.TextToColumns DataType:=xlDelimited, Semicolon:=True
        Case 2 ' Beistrich
            Selection.TextToColumns DataType:=xlDelimited, Comma:=True
        Case 3 ' Tabulator
            Selection.TextToColumns DataType:=xlDelimited, Tab:=True
        Case 4 ' Leerzeichen
            Selection.TextToColumns DataType:=xlDelimited, Space:=True
    End Select

' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
    Worksheets("Zwischenablage").Activate
    ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
    Worksheets("Zwischenablage").Visible = False
```

```
' Rückkehr zur aktuellen Tabelle mit den eingefügten Daten
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_QUELLZEILE, ERSTE_QUELLSPALTE).Select

' Hinweismeldung

    Application.ScreenUpdating = True

    If Range("A2") <> "Firma" Or Range("C2") <> "Kontoklasse" Or Range("D2") <> "KontoNr" Then
            MsgBox "Die Daten wurden eingefügt - aber Sie scheinen nicht im richtigen Format zu sein. Gebraucht wird
eine Saldenliste 101 Jahr/Monat mit Klassensummen"
        Else
            MsgBox "Die Daten wurden eingefügt"
    End If
' ------------------------
' Ende des normalen Codes
' ------------------------

Exit Sub

' ----------------------------------------------------
' Fehlerroutine 1, wenn keine Daten in Zwischenablage
' ----------------------------------------------------

FEHLER:
    Sheets(AKTUELLE_TABELLE).Activate
    Application.ScreenUpdating = True
    MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte erneut die Daten kopieren"
    Exit Sub

' ----------------------------------------------------
' Fehlerroutine 2, wenn Tabelle Zwischenablage fehlt
' ----------------------------------------------------

ZWISCHENABLAGE_ANLEGEN:

    Worksheets.Add.Move After:=Worksheets(Worksheets.Count): ' Neues Tabellenblatt am Ende anlegen
    ActiveSheet.Name = "Zwischenablage": ' Neues Blatt umbenennen
    MsgBox "Die Hilfstabelle 'Zwischenablage' war noch nicht vorhanden und musste erst erzeugt werden." & vbCrLf &
vbCrLf & _
```

```
    "Dadurch wurden leider die Daten in der Zwischenablage geleert. Bitte wiederholen Sie noch einmal das Kopieren der
Daten in die Zwischenablage und klicken Sie dann erneut auf den Button hier für das Einfügen der Zwischenablage -
danke."
    Sheets(AKTUELLE_TABELLE).Activate

End Sub
```

Version 2: SPALTENREIHENFOLGE EINZELN STEUERBAR

```
Public Function LETZTEZELLE(TABELLENBLATT As String) As Range
' Neue Version 2015

  Dim ExcelLastCell As Range
  Dim Row As Long
  Dim Col As Long
  Dim LastRowWithData As Long
  Dim LastColWithData As Long
  Dim TheSheet As Worksheet
  Dim MERKER

  Set TheSheet = Worksheets(TABELLENBLATT)

  MERKER = Application.ScreenUpdating
  Application.ScreenUpdating = False

On Error Resume Next ' Falls kein Autofilter gesetzt, käme nun eine Fehlermeldung in der nächsten Zeile
   Worksheets(TABELLENBLATT).ShowAllData
On Error GoTo 0

  Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

  ' letzte Zeile mit Daten herausfinden
  LastRowWithData = ExcelLastCell.Row
  Row = ExcelLastCell.Row
  Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
    Row = Row - 1
  Loop
  LastRowWithData = Row

  ' letzte Spalte mit Daten herausfinden
```

```
    LastColWithData = ExcelLastCell.Column
    Col = ExcelLastCell.Column
    Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
       Col = Col - 1
    Loop
    LastColWithData = Col

    Set LETZTEZELLE = TheSheet.Cells(Row, Col)
    Application.ScreenUpdating = MERKER

End Function
```

Sub Zwischenablage_einfuegen()

```
' neue Version des Zwischenablageeinfügens 2016
' mit der die Spalten einzeln kopiert werden und in anderer Reihenfolge eingefügt werden kann.


Dim ERSTE_QUELLZEILE As Integer ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
Dim QUELLSPALTE As Integer ' die betreffende Spalte von der Zwischenablage
Dim ERSTE_ZIELZEILE As Integer ' was ist die erste Zeile in der empfangenden Zieltabelle, ab der die Daten eingefügt werden sollen
Dim ZIELSPALTE As Integer ' die betreffende Spalte in der empfangenden Zieltabelle, in der die Daten eingefügt werden sollen
Dim AUFTRENNUNG_TRENNZEICHEN As Integer ' wurde eine TXT-Datei eingefügt, die mit Trennzeichen getrennte Daten enthält - 1=Semikolon, 2=Beistrich, 3=Tabulator,
4=Leerzeichen, 0=keine Trennung

Dim AKTUELLE_TABELLE As String ' Für den Rücksprung in die aktuelle Tabelle



' ************************************************************************************************************
' -----------------
' Variablen setzen
' -----------------

' Hier bitte die Einstellungen für die Prozedur machen

    ' was ist die erste Zeile von der Zwischenablage, die kopiert werden soll
    ERSTE_QUELLZEILE = 7

    ' was ist die erste Zeile in der empfangenden Zielzeile, ab der die Daten eingefügt werden sollen
    ERSTE_ZIELZEILE = 2

    ' sollen TXT-Daten aus der Zwischenablage noch aufgetrennt werden ?
    ' BSP: 0;100002377;20150629;4000 ... soll in 4 eigenständige Spalten getrennt werden
    ' 0= keine Auftrennung - 1=Semikolon - 2=Beistrich - 3=Tabulator - 4=Leerzeichen
    AUFTRENNUNG_TRENNZEICHEN = 1
```

```
' ********************************************************************************************************************

    Application.ScreenUpdating = False

' merken des Namens der aktuellen Tabelle
    AKTUELLE_TABELLE = ActiveSheet.Name

' Falls nachfolgender Code nicht geht, weil die Tabelle Zwischenablage noch nicht vorhanden ist, soll Code zum Tabellenanlegecode verzweigen
    On Error GoTo ZWISCHENABLAGE_ANLEGEN

' Zwischenablage_Tabelle einblenden
    Worksheets("Zwischenablage").Visible = True

' Eigentliche Fehlerabfangroutine, wenn Zwischenablage leer ist
 ' On Error GoTo 0
    On Error GoTo FEHLER

' ---------------------------------------------------------
' Einfügen der Zwischenablage in der Zwischenablage-Tabelle
' ---------------------------------------------------------

' Einfügen der Daten aus der Zwischenablage in der Interimstabelle ZWISCHENABLAGE
    Worksheets("Zwischenablage").Activate
    Range("A1").Select ' wir fügen die Daten immer ab der ersten Zeile in Spalte A ein - eine eventuell andere erste Zelle in der Zieltabelle wird unten eingestellt
    ActiveSheet.Paste

' Falls aus Zwischenablage keine Excelzellen-Daten kommen, sondern aus zB dem Editor die Werte einer TXT-Datei
' die noch aufgetrennt werden müssen - zB 0;100002377;20150629;4000 ...
' Hier gemäß dem Trennzeichen erfolgt die Auftrennung: 1=Semikolon, 2=Beistrich, 3=Tabulator, 4=Leerzeichen, (0=keine Trennung)

    If AUFTRENNUNG_TRENNZEICHEN > 0 Then ' falls Auftrennung gewünscht, markiere die Spalte, in der alle Daten sind
        Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count, 1)).Select
    End If

    Select Case AUFTRENNUNG_TRENNZEICHEN
        Case 1 ' Semikolon
            Selection.TextToColumns DataType:=xlDelimited, Semicolon:=True
        Case 2 ' Beistrich
            Selection.TextToColumns DataType:=xlDelimited, Comma:=True
        Case 3 ' Tabulator
            Selection.TextToColumns DataType:=xlDelimited, Tab:=True
        Case 4 ' Leerzeichen
            Selection.TextToColumns DataType:=xlDelimited, Space:=True
    End Select
```

```
' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten)
    Sheets(AKTUELLE_TABELLE).Activate
    Sheets(AKTUELLE_TABELLE).Range(Cells(ERSTE_ZIELZEILE, 1), Cells(Sheets(AKTUELLE_TABELLE).UsedRange.Rows.Count,
Sheets(AKTUELLE_TABELLE).UsedRange.Columns.Count)).ClearContents


' ------------------------------------------------------------
' Einfügen der ersten Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 1
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 1

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' ------------------------------------------------------------
' Einfügen der 2. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 3
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 3

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
```

```
' ------------------------------------------------------------
' Einfügen der 3. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 4
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 2

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' ------------------------------------------------------------
' Einfügen der 4. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 13
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 4

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' ------------------------------------------------------------
' Einfügen der 5. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 14
```

```
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 6

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' ------------------------------------------------------------
' Einfügen der 6. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 15
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 8

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' ------------------------------------------------------------
' Einfügen der 7. Spalte aus der Zwischenablage-Tabelle
' ------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 16
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 7

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy
```

```
' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' -------------------------------------------------------------
' Einfügen der 8. Spalte aus der Zwischenablage-Tabelle
' -------------------------------------------------------------

    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
    QUELLSPALTE = 12
    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
    ZIELSPALTE = 12

    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
    Worksheets("Zwischenablage").Activate
    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy

' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
    Sheets(AKTUELLE_TABELLE).Activate
    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select

' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

'' -------------------------------------------------------------
'' Einfügen der 9. Spalte aus der Zwischenablage-Tabelle
'' -------------------------------------------------------------
'
'    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
'    QUELLSPALTE = 1
'    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
'    ZIELSPALTE = 1
'
'    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
'    Worksheets("Zwischenablage").Activate
'    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy
'
'' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
'    Sheets(AKTUELLE_TABELLE).Activate
'    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select
'
'' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
```

```
'    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
'
'' ---------------------------------------------------------
'' Einfügen der 10. Spalte aus der Zwischenablage-Tabelle
'' ---------------------------------------------------------
'
'    ' die aktuelle Spalte von der Tabelle Zwischenablage, die kopiert werden soll
'    QUELLSPALTE = 1
'    ' die aktuelle Ziel-Spalte in der empfangenden Zielzeile, in der die Daten eingefügt werden sollen
'    ZIELSPALTE = 1
'
'    ' kopieren der Daten aus der Zwischenablage_Zwischendatei - nur die Daten ab der 2. Zeile (A2)
'    Worksheets("Zwischenablage").Activate
'    Range(Cells(ERSTE_QUELLZEILE, QUELLSPALTE), Cells(LETZTEZELLE(ActiveSheet.Name).Row, QUELLSPALTE)).Copy
'
'' Markieren der Zelle in der aktuellen Tabelle, ab der die Daten eingefügt werden sollen
'    Sheets(AKTUELLE_TABELLE).Activate
'    Cells(ERSTE_ZIELZEILE, ZIELSPALTE).Select
'
'' Wir fügen immer nur Werte ein, damit die Formatierung nicht verloren geht
'    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False




' ---------------------------------------------------------
' Abschluss nach dem letzten Datentransfer
' ---------------------------------------------------------

' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
   Worksheets("Zwischenablage").Activate
   ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count, ActiveSheet.UsedRange.Columns.Count)).ClearContents
   Worksheets("Zwischenablage").Visible = False

' Rückkehr zur aktuellen Tabelle mit den eingefügten Daten
   Sheets(AKTUELLE_TABELLE).Activate
   Cells(ERSTE_QUELLZEILE, 1).Select

' Hinweismeldung


   Application.ScreenUpdating = True
   MsgBox "Die Daten wurden eingefügt"

' ------------------------
' Ende des normalen Codes
```

```
' ------------------------

Exit Sub

' ----------------------------------------------------
' Fehlerroutine 1, wenn keine Daten in Zwischenablage
' ----------------------------------------------------

FEHLER:
    Tabelle2.Activate
    MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte erneut die Daten kopieren mit STRG-A und STRG-C und dann erneut klicken auf
ZWISCHENABLAGE EINFÜGEN."
    Exit Sub

' ----------------------------------------------------
' Fehlerroutine 2, wenn Tabelle Zwischenablage fehlt
' ----------------------------------------------------

ZWISCHENABLAGE_ANLEGEN:

    Worksheets.Add.Move after:=Worksheets(Worksheets.Count): ' Neues Tabellenblatt am Ende anlegen
    ActiveSheet.Name = "Zwischenablage": ' Neues Blatt umbenennen
    Worksheets("Zwischenablage").Visible = False ' gleich wieder umbenennen
    Sheets(AKTUELLE_TABELLE).Activate

    MsgBox "Die Hilfstabelle 'Zwischenablage' war noch nicht vorhanden und musste erst erzeugt werden." & vbCrLf & vbCrLf & _
    "Dadurch wurden leider die Daten in der Zwischenablage geleert. " & vbCrLf & vbCrLf & _
    "Bitte wiederholen Sie noch einmal das Kopieren der Daten in die Zwischenablage und klicken Sie dann erneut auf den Button hier für das Einfügen der Zwischenablage -
danke."


End Sub
```

VERSION ALT 1 (war meine erste Version, die 2014 gut ging und dann ne Zeit lang hakte(

Sub Zwischenablage_einfuegen()

' Fügt die Zwischenablage ein - jedoch über eine Zwischentabelle "Zwischenablage"

```
' diese Zwischentabelle darf nicht neu erzeugt werden (Sheets.add), da dadurch die Zwischenablage geleert wird
' daher immer ausgeblendet und geleert in der Arbeitsmappe mitführen

Dim AKTUELLE_TABELLE As String

Application.ScreenUpdating = False

'On Error GoTo fehler

' merken Name der aktuellen Tabelle
AKTUELLE_TABELLE = ActiveSheet.Name

' Zwischenablage_Tabelle einblenden
Worksheets("Zwischenablage").Visible = True
Worksheets("Zwischenablage").Activate
Range("A1").Select ' wir fügen die Daten immer ab der ersten Zeile in Spalte A ein - eine eventuell andere erste Zelle
in der Zieltabelle wird unten eingestellt
ActiveSheet.Paste

' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten - in der Regel ab C2)
Sheets(AKTUELLE_TABELLE).Activate
Sheets(AKTUELLE_TABELLE).Range(Cells(2, 3), Cells(Sheets(AKTUELLE_TABELLE).UsedRange.Rows.Count,
Sheets(AKTUELLE_TABELLE).UsedRange.Columns.Count)).ClearContents

' kopieren der Daten aus der Zwischenablage_Zwischendatei - in der Regel nur die Daten ab der 1. Zeile (A1)
Worksheets("Zwischenablage").Activate
Range(Cells(1, 1), Cells(LETZTEZELLE(ActiveSheet.Name).Row, LETZTEZELLE(ActiveSheet.Name).Column)).Copy
' alternativ:
' Range(Cells(1, 1), Cells(ActiveSheet.Name.UsedRange.Rows.Count,
ActiveSheet.Name.UsedRange.Columns.Count)).Copy
```

```
' einfügen in aktueller Tabelle ab Zeile 2 - konkret ab C2
Sheets(AKTUELLE_TABELLE).Activate
Range("C2").Select
' nur Werte einfügen
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
Worksheets("Zwischenablage").Activate
ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count,
ActiveSheet.UsedRange.Columns.Count)).ClearContents
Worksheets("Zwischenablage").Visible = False

' Rückkehr zur aktuellen Tabelle mit den eingefügten Daten
Sheets(AKTUELLE_TABELLE).Activate



' Falls aus der Zwischenablage die Werte nur zusammen in Spalte A ankommen (weil es zB eine CSV-Datei ist mit
Semikolon oder Beistrich):
' Daher müssen die Werte, die mit Semikolon/Beistrich in Spalte A sind, noch aufgetrennt werden auf die Spalten
rechts davon

'    Range("A2:A50000").Select
'    wenn SEMIKOKOLON
'    Selection.TextToColumns DataType:=xlDelimited, Semicolon:=True
'    wenn BEISTRICH
'    Selection.TextToColumns DataType:=xlDelimited, Comma:=True

' ------------------------------------------------------------------------

Application.ScreenUpdating = True
```

```
Exit Sub

FEHLER:
    MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte erneut die Daten kopieren"

End Sub
```

VERSION ALT 2

Sub Zwischenablage_einfuegen()

```
' Fügt die Zwischenablage ein - jedoch über eine Zwischentabelle "Zwischenablage"
' diese Zwischentabelle darf nicht neu erzeugt werden (Sheets.add), da dadurch die Zwischenablage geleert wird
' daher immer ausgeblendet und geleert in der Arbeitsmappe mitführen

Dim AKTUELLE_TABELLE As String

Application.ScreenUpdating = False

'On Error GoTo fehler

' merken Name der aktuellen Tabelle
AKTUELLE_TABELLE = ActiveSheet.Name

' Zwischenablage_Tabelle einblenden
Worksheets("Zwischenablage").Visible = True
Worksheets("Zwischenablage").Activate
Range("A1").Select
ActiveSheet.Paste

' aktuelle Tabelle leeren (nur den verwendeten Bereich mit Daten)
Sheets(AKTUELLE_TABELLE).Activate
Sheets(AKTUELLE_TABELLE).Range(Cells(2, 1), Cells(Sheets(AKTUELLE_TABELLE).UsedRange.Rows.Count,
Sheets(AKTUELLE_TABELLE).UsedRange.Columns.Count)).ClearContents

' kopieren der Daten aus der Zwischenablage_Zwischendatei
```

```
Worksheets("Zwischenablage").Activate
Range(Cells(2, 1), Cells(LETZTEZELLE(ActiveSheet.Name).Row, LETZTEZELLE(ActiveSheet.Name).Column)).Copy
' alternativ:
' Range(Cells(1, 1), Cells(ActiveSheet.Name.UsedRange.Rows.Count, ActiveSheet.Name.UsedRange.Columns.Count)).Copy

' einfügen in aktueller Tabelle ab Zeile 2
Sheets(AKTUELLE_TABELLE).Activate
Range("A2").Select
' nur Werte einfügen
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False

' leeren und ausblenden der nicht mehr benötigten Zwischentabelle
Worksheets("Zwischenablage").Activate
ActiveSheet.Range(Cells(1, 1), Cells(ActiveSheet.UsedRange.Rows.Count, ActiveSheet.UsedRange.Columns.Count)).ClearContents
Worksheets("Zwischenablage").Visible = False

Sheets(AKTUELLE_TABELLE).Activate
Application.ScreenUpdating = True

Exit Sub
fehler:
MsgBox "In der Zwischenablage wurden leider keine Daten gefunden. Bitte erneut die Daten kopieren"
End

End Sub
```