Online - Handbuch EXCEL VBA

Zusammengestellt von
Bernd Busko
Hompepage

Version 1.5

31.10.98

Stichwortverzeichnis

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Inhaltsverzeichnis

API Aufrufe

1. Speicherpfad erfragen

```
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
```

2. Dialog Positionieren

für Excel 5 gilt folgender API-Aufruf:

```
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
```

➡ Wells

für Excel 7 folgenden API-Aufruf verwenden:

```
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
```

Bullen

3. Steuerungsmenü des Dialogfeld rahmens entfernen

Folgender API Aufruf entfernt das Steuerungsmenü (Verschieben/Schliessen u.ä.) aus einem Dialogfeld. Das Makro muss dem Dialogfeldrahmen zugewiesen werden. Läuft nur unter Windows 95.

```
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 window style
```

```
    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
```

Bullen

4. Vorhandensein eines Farbdrucker s prüfen

Folgender Code kann prüfen, ob der angeschlossene Drucker ein Farbdrucker ist:

```
Option Explicit
Declare Function CreateICA Lib "GDI32" (ByVal driver As String, ByVal
device As String, ByVal Port As String, devmode As Long) As Long
Declare Function DeleteDC Lib "GDI32" (ByVal hdc As Long) As Boolean
Declare Function GetDeviceCaps Lib "GDI32" (ByVal hdc As Long, ByVal cap
As Integer) As Integer
Declare Function RegOpenKeyExA Lib "advapi32" (ByVal hkey As Long, ByVal
subkey As String, ByVal options As Long, ByVal access As Long, ByRef
newkey As Long) As Long
Declare Function RegCloseKey Lib "advapi32" (ByVal hkey As Long) As Long
Declare Function RegQueryValueExA Lib "advapi32" (ByVal hkey As Long,
ByVal entry As String, ByRef reserved As Long, ByRef dtype As Long,
ByVal retval As String, ByRef datalen As Long) As Long
Sub Demo()
'demo
MsgBox IsColourPrinter, , "Colour Printer?"
End Sub
Function IsColourPrinter() As Boolean
' *** Alan Warriner 1998 ***
'alan_warriner@bigfoot.com
'wrapper for GetPrinterColours function
'returns TRUE if a colour printer
'returns FALSE if not colour or an error occurred
IsColourPrinter = GetPrinterColours > 2
End Function
Function GetPrinterColours() As Integer
' *** Alan Warriner 1998 ***
'alan_warriner@bigfoot.com
'obtain the number of colours active printer is capable of printing
'2 colours (or less?) indicates mono printer
'a return value of zero indicates an error
'error return value
GetPrinterColours = 0
On Error GoTo errortrap
Dim PrinterName As String
Dim DriverName As String
Dim DriverFile As String
Dim Port As String
Dim newkey, datalen As Long
Dim OnLocation, tempval As Integer
Dim hdc As Long
hdc = 0
'constants for registry functions
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_NONE = 0
Const REG_SZ As Long = 1
'constant for device capablity function
Const NUMCOLORS = 24
'get active printer name
PrinterName = Application.ActivePrinter
'extract printer device name by getting last occurence of ' on '
```

```
OnLocation = 0
Do
tempval = InStr(OnLocation + 1, PrinterName, " on ")
If tempval > 0 Then
OnLocation = tempval
End If
Loop While tempval > 0
PrinterName = Left(PrinterName, OnLocation - 1)
'get printer driver name from registry
If Not GetRegistryEntry(HKEY_LOCAL_MACHINE,
"System\CurrentControlSet\Control\Print\Printers\" & PrinterName,
"Printer Driver", DriverName) Then
Exit Function
End If
'get printer port from registry
If Not GetRegistryEntry(HKEY_LOCAL_MACHINE,
"System\CurrentControlSet\Control\Print\Printers\" & PrinterName,
"Port", Port) Then
Exit Function
End If
'get printer driver file name from registry
If Not GetRegistryEntry(HKEY_LOCAL_MACHINE,
"System\CurrentControlSet\Control\Print\Environments\Windows
4.0\Drivers\" & DriverName, "Driver", DriverFile) Then
Exit Function
End If
'remove .xxx extension
If InStr(DriverFile, ".") Then
DriverFile = Left(DriverFile, InStr(DriverFile, ".") - 1)
End If
'get device context for printer
hdc = CreateICA(DriverFile, PrinterName, Port, 0&)
If hdc = 0 Then
Exit Function
End If
'get number of colours printer can use
GetPrinterColours = GetDeviceCaps(hdc, NUMCOLORS)
'handle errors
errortrap:
'dispose of device context
If hdc <> 0 Then
DeleteDC (hdc)
End If
Exit Function
End Function
Function GetRegistryEntry(ByVal hkey As Long, ByRef entry As String,
ByRef value As String, ByRef returnstring As String) As Boolean
' *** Alan Warriner 1998 ***
'alan_warriner@bigfoot.com
'get an entry from the registry
'return false if unable to
'otherwise
'return registry entry STRING in passed parameter 'returnstring'
'registry function constants
Const ERROR_NONE = 0
Const REG_SZ As Long = 1
Dim newkey, datalen As Long
'error return value
GetRegistryEntry = False
On Error GoTo errortrap
'try to open registry entry

If RegOpenKeyExA(hkey, entry, 0&, &H3F, newkey) <> ERROR_NONE Then
Exit Function
End If
```

```
'get length of registry entry & set passed string length to suit
RegQueryValueExA newkey, value, 0&, REG_SZ, 0&, datalen
returnstring = String(datalen, 0)
'read string data into passed parameter
If RegQueryValueExA(newkey, value, 0&, REG_SZ, returnstring, datalen) <>
ERROR_NONE Then
RegCloseKey newkey
Exit Function
End If
'close registry entry
RegCloseKey newkey
'return success value
GetRegistryEntry = True
'handle errors
errortrap:
Exit Function
End Function
```

5. Bildschirmauflösung feststellen

Folgender API-Aufruf stellt die Bildschirmauflösung fest:

```
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
```

Im zweiten Beispiel kann die Bildschirmauflösung auch per Funktion abgefragt und in eine Zelle geschrieben werden:

```
Declare Function GetDeviceCaps Lib "gdi32" (ByVal hdc As Long, ByVal nIndex As Lor
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
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
```

6. Freier Festplattenplatz

Folgende API-Funktion gibt den freien Festplattenplatz in eine Zelle aus:

```
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA"
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&
dVal# = SecPerCluster& * BytesPerSector&
dVal# = dVal# * NumFreeClusters&
GetFreeSpace = dVal#
End Function
```

7. 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
```

Rech

8. Netzlaufwerke und Shares feststellen

Der folgende Code gibt als Ergebnis die verbundenen Netzlaufwerke und Shares an:

```
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
```

9. NumLock ein- und ausschalten

Mit folgendem Code lässt sich die Taste NumLock ein- und ausschalten:

```
Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal bScan As Byte, ByVa
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
```

10. Bildschirmauflösung ändern

Mit Hiilfe dieses API - Aufrufs lässt sich die Bildschirmauflösung ändern:

```
Private Declare Function EnumDisplaySettings Lib "user32" Alias "EnumDisplaySettin

Private Declare Function ChangeDisplaySettings Lib "user32" Alias "ChangeDisplaySe

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
```

11. CapsLock einschalten

Folgender Code schaltet CapsLock ein:

```
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
```

```
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
```

Arbeitsmappen- und Blattschutz

1. Blattschutz durch Makro aus- und einschalten

Durch folgendes Makro kann der Blattschutz ein- und wieder ausgeschaltet werden.

```
Sub SchutzAusEin()
    ActiveSheet.Unprotect "Test"
    MsgBox "Blattschutz ist aufgehoben!"
    ActiveSheet.Protect "Test"
    MsgBox "Blattschutz ist gesetzt!"
End Sub
```

Herber

2. Tastatureingaben abfangen

Generell können Tastatureingaben durch den SendKey-Ereignisse abgefangen werden. In der Beispielzeile wird die Tastenkombination Alt+F8 (Makro-Menü) abgefangen. TueDiesUndDas ist das auszuführende Makro:

```
Sub Auto_Open()
        Application.OnKey "%{F8}", "TueDiesUndDas"
End Sub
Sub TueDiesUndDas()
    MsgBox "Hallo!"
    MsgBox ActiveCell.Address
End Sub
```

Benutzerdefinierte Funktionen

Excel-Version auslesen

Die Folgende Funktion gibt die verwendete Excel-Version in eine Zelle zurück:

```
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
```

Die folgende Prozedur hat die gleiche Funktionalität wie die obige Funktion, gibt das ergebnis aber in einem Meldungsfenster aus:

```
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
```

Kalenderwoche berechnen

Mit der folgenden Funktion aus der MS KB kann die Kalenderwoche nach DIN 1355 berechnet werden:

```
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
```

Stellenzahl auslesen

Die folgende benutzerdefinierte Funktion gibt die Anzahl der Stellen einer Zahl zurück:

```
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
```

Ture

Datenimport- und Export

1. CSV-Datei schreiben

Der folgende Code schreibt den markierten Zellbereich in eine CSV-Datei. Pfad und Dateiname können angegeben werden:

```
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
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
```

Ogilvy

2. Zellen nach Datenimport aufbereiten

Nach dem Import von Daten (aus Access oder anderen Anwendungen, im Format TXT oder CSV) werden Zielzellen von Excel oft nicht richtig erkannt. Summen und andere Funktionen werden nicht berechnet, die manuelle Formatzuweisung ist wirkungslos. Nur durch manuelles Bearbeiten jeder einzelnen Zelle mit F2 und ENTER lässt sich dieses Verhalten abstellen.
Der folgende Code setzt alle Zellen der Spalte G (7) zurück. Bei Bedarf kann "Set MyRange" auch anders zugewiesen werden:

```
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
```

Busko

3. Existenz einer Datei prüfen

```
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
```

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
```

Ogilvy

Daten nach Access

Mit folgendem Code kann ein Datensatz an eine Access-Datenbank angefügt werden:

Annahmen:

Datenbank = Test.mdb
Tabelle = Test
Feld 1= Name
Feld 2 = Alter
Worksheet Range("A1") = Hr Schmitz
Worksheet Range("A2") = 30

```
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
```

Kommentare

1. Kommentare per Makro formatieren

Das folgende Makro weist der aktiven Zelle einen formatierten Kommentar zu:

```
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

Zur Zuweisung eines bestimmten Zellbereiches mit demselben Kommentar kann der folgende Code verwendet werden. Dazu den Code in ein Modulblatt kopieren und nach Auswahl des Zielbereiches ausführen. "Kommentar!" ist dabei der zugewiesene Kommentar.

```
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
```

Herber

3. Grösse des Kommentarfensters automatisch festlegen

Der folgende Code legt für alle Kommentare das Kommentarfenster automatisch auf die optimale Grösse fest:

```
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
```

Makroausführung beeinflussen

1. Makroausführung pausieren

Folgender Code hält die Makroausführung füe eine Sekunde an und startet dann das nächste Makro

```
Sub Pause()
        Application.OnTime Now+TimeValue("00:00:01"), "NextMacro"
End Sub
```

Manville

2. Makro durch Veränderung einer Zelle starten

Unter Excel kann das Worksheet_Change Ereignis verwendet werden. Dazu folgender Beispielcode:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    If Target.Address = "$A$1" Or Target.Address = "$A$3" Then
        If Range("A1").Value < Range("A3").Value Then
        Macro1
        End If
    End If
End Sub
```

Das Workbook_Change Ereignis wird nicht aufgrufen, wenn die Zelländerung durch eine (Neu)Berechnung, also durch das Berechnungsergebnis einer Formel verursacht wurde. In diesem Beispiel würde A1 und A3 also eine Formel enthalten.
Soll auch nach eine Berechnung das Makro aufgerufen werden, kann das Worksheet_Calculate Ereignis verwendet werden:

```
Private Worksheet_Calculate()
    If Range("A1").Value < Range("A3").Value Then
    Macro1
    End If
End Sub
```

Pearson

3. Makoausführung verbergen

```
Folgende Codezeile verhindert die Bildschirmaktualisierung, das Makro läuft schne
```

```
Sub Screen()
    Application.ScreenUpdating=False
End Sub
```

Bildschirmaktualisierung wieder einschalten:

```
Sub Screen()
    Application.ScreenUpdating=True
End Sub
```

4. Makroausführung, Unterbrechen verhindern

Folgende Codezeile verhindert das Unterbrechen der Makroausführung durch Betätigung der Esc-Taste:

```
Sub Abbruch()
    Application.EnableCancelKey = xlDisabled
End Sub
```

Folgende Codezeile verzweigt in den aktuellen Errorhandler:

```
Sub Abbruch()
    Application.EnableCancelKey = xlErrorHandler
End Sub
```

5. Makroausführung nach jeder Eingabe

Mit folgendem Code kann eine Makroausführung nach jeder Einageb ausgelöst werden:

```
Application.OnEntry = "MeinMakro"
```

Zum Ausschalten:

```
Application.OnEntry = ""
```

Dieser Code arbeitet global, d.h. in allen geöffneten Mappen und Tabellen.

6. Makroausführung nicht durch Sicherheitsabfragen unterbrechen

Durch folgende Codezeile kann die Anzeige von Sicherheitsabfragen (z.B. "Soll Zwischenablage gespeichert werden?", "Soll Blatt XYZ wirklich gelöscht werden?") abgeschaltet werden:

```
Application.DisplayAlerts = False
```

Diese Zeile in der ersten Zeile des Makros eintragen.

7. Makrounterbrechung abfangen

Folgendes Beispielcode fängt die Makrounterbrechung durch Drücken von Esc mit einem Errorhandler ab:

```
On Error GoTo EH
Application.EnableCancelKey = xlErrorHandler
While 1 = 1 'Schleife
X = X 'Schleife
Wend 'Schleife
Exit Sub
EH:
MsgBox "Break Key Hit"
Application.EnableCancelKey = xlInterrupt
```

➡Pearson

8. Makros dynamisch erstellen

Das folgende Beispiel zeigt wie eine Prozedur dynamisch in einem Modulblatt erstellt und wieder gelöscht werden kann:

```
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
```

Herber

8. Module und Code aus Arbeitsmappe entfernen

Mit folgendem Beispielcode können aus einer Arbeitsmappe alle Module und sonstiger Code entfernt werden. UserForms werden nicht berücksichtigt:

```
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
```

Die Funktion kann wie folgt aufgerufen werden

```
Sub PrepBook()
If Not bRemoveAllCode(ActiveWorkbook.Name) then MsgBox "An error _
occurred!", vbCritical,"bRemoveAllCode"
End sub
```

Rosenberg

Menüs verändern

1. Untermenüs durch Makro erstellen

Zum Erstellen einer eigenen Menüstruktur kann folgender Code eingesetzt werden (Excel 8):

```
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
```

Herber

2. Menü "Symbolleisten" deaktivieren/aktivieren

Diese Routine schaltet den Menüeintrag "Symbolleisten" ab. Auch durch Rechtsklick kann das Menü nicht aufgerufen werden. Damit kann verhinder werden, dass der Anwender Menüeinträgen verändert:

```
Sub DisableToolbarMenu()
        CommandBars("Toolbar List").Enabled = False
End Sub
```

Einschalten:

```
Sub DisableToolbarMenu()
    CommandBars("Toolbar List").Enabled = True
End Sub
```

Menüs dynamisch ein- und ausblenden

Durch die folgenden Makros wird beim Öffnen/Aktivieren der entsprechenden Arbeitsmappe eine benutzerdefinierte Menüleiste erstellt. Beim Schliessen/Deaktivieren der Arbeitsmappe werden die Standardmenüleisten wiederhergestellt.
Die Makros müssen in der Excel-Eentwickungsumgebung in das Modul "DieseArbeitsmappe" kopiert werden. Excel 8:

```
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
```

Held

Shortcut-Menü ein- und ausschalten

Die folgende Prozedur schaltet das Shortcut-Menü (Rechtsklick) aus bzw. ein (True):

```
Sub ShortCutOnOff()
Application.ShortcutMenus(xlWorksheetCell).Enabled = False
End Sub
```

Manville

Icons in Symbolleiste deaktivieren

Mit folgendem Makro, das auf eigene Bedürfnisse angepasst werden muss, können einzelne Symbole innerhalb einer Symbolleiste deaktiviert werden:

```
Sub SymbolGrauen()
CommandBars("Standard").Controls(1).Enabled = False
End Sub
```

Texte verändern

1. Zahl in Text

Folgende Funktion setzt numerisch gegebene Dollarbeträge in Text um:

```
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
```

Larson

2. Umlaute ersetzen

Der folgende Code erstezt die Umlaute in der aktuellen Zellauswahl:

```
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
```

Busko

2. Gross/Kleinschreibung tauschen

```
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!", vbI
OriginalSelection.Select
OriginalCell.Activate
End Sub
```

3. Minuszeichen umstellen

Folgendes Makro stellt ein angehängtes Minuszeichen nach links:

```
Sub MinusUmstellen()
Range("a1").Select
Do Until ActiveCell.Value = ""
altstring = ActiveCell.Value
längealtstring = Len(altstring)
längealtstring = längealtstring - 1
rechteszeichen = Right(altstring, 1)
If rechteszeichen = "-" Then neuerstring = Left(altstring, längealtstring): _
neuerstring = "-" + neuerstring: ActiveCell.Value = neuerstring
ActiveCell.Offset(1, 0).Range("A1").Select
Loop
End Sub
```

Held

Angepasste Version, wenn der Zellbereich vor Start des Makros mit der Maus markiert werden soll:

```
Sub TrailingNegatives()
'to be used with selected ranges
     For Each Cell In Selection
     Cell.Select
     altstring = ActiveCell.Value
     längealtstring = Len(altstring)
     längealtstring2 = längealtstring - 1
     rechteszeichen = Right(altstring, 1)
     If rechteszeichen = "-" Then neuerstring = _
     Left(altstring, längealtstring2): _
     neuerstring = "-" + neuerstring: ActiveCell.Value = neuerstring
     Next

End Sub
```

Zellmanipulationen

1. Erste leere Zelle in einer Spalte finden

Beide Möglichkeiten liefert die erste leere Zelle eine Spalte

```
Sub Finde()
    Columns(MyColumnNumber).SpecialCells(xlCellTypeBlanks).Cells(1)
End Sub

Sub Finde()
    Cells(Application.WorksheetFunction.CountA(Columns(MyColumnNumber)) + 1,
    MyColumnNumber)
End Sub
```

2. Zellen im Makro ohne Zwischenablage kopieren

Bei der Programmierung von VBA-Makros sollte bei Zellmanipulationen Select, Copy, Paste vermieden werden. Hier ein Beispiel:

```
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
```

Herber

3. Zellen zeilenweise ausfüllen

Der folgende Code schreibt alle Eingaben in die Zelle A1 nacheinander in die Zeile C:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
If Target.Address = "$A$1" Then
Set actcell = [C1]
Do While actcell <> ""
Set actcell = actcell.Offset(0, 1)
Loop
actcell.Value = Target.Value
End If
End Sub
```

5. Erste leere Zelle finden

Folgender Code findet die erste leere Zelle eines Zellbereichs:

```
Sub Finde()
Selection.SpecialCells(xlBlanks).Areas(1).Cells(1).Select
End Sub
```

Sonstige

Fundstellen in Userform auflisten

Bei der Suche über ein ganzes Tabellenblatt sollen alle Fundstellen eines bestimmten Ausdrucks in einem UserForm angezeigt werden. Dazu folgenden Code verwenden:

```
Sub FundstellenSuchen()
    Dim C As Range
    Dim Gefunden()
```

```
    Dim i%
    For Each C In Tabelle1.Range("A1").CurrentRegion
        If InStr(C, "Zei") > 0 Then
            ReDim Preserve Gefunden(i)
            Gefunden(i) = C.Address(False, False)
            UserForm1.ListBox1.List = Gefunden
            i = i + 1
        End If
    Next C
    UserForm1.Show
End Sub
```

Herber

Beispiel

Mappe mit Dateinamen aus Zelle speichern

Der Folgende Code speichert eine Arbeitsmappe und verwendet dabei als Dateinamen den Eintag in einer bestimmten Zelle (hier A1, Tabelle 1):

```
Sub Auto_Close() 'unter Namen speichern, welcher in Zelle A1 steht
    Dim f As String; r As Integer
    f = ThisWorkbook.Sheets(1).Cells(1; 1).Value
    If f = "" Then
        f = Application.GetSaveAsFilename( _
        fileFilter:="Excel Workbook (*.xls), *.xls")
    If f = False Then
    Exit Sub
    End If
    End If
    r = ThisWorkbook.Sheets(1).Cells(1; 1).Characters.Count
    If ThisWorkbook.Sheets(1).Cells(1; 1).Characters(r - 3).Text <> ".xls" Then
        f = f & ".xls"
    End If
    ThisWorkbook.SaveAs Filename:=f
End Sub
```

Held

Inhalt der Zwischenablage löschen

Folgende Codezeile löscht den Inhalt der Zwischenablage:

```
Application.CutCopyMode= False
```

Zeilen ausblenden, mit Summe Nul l

Folgender Code blendet Zeilen mit Summe null aus:

```
Sub HideRows()
    For Each rngRow In ActiveSheet.UsedRange.Rows
    If Application.Sum(rngRow) = 0 Then
    rngRow.EntireRow.Hidden = True
    End If
    Next rngRow
End Sub
```

Green

Der folgende Code löscht Zeilen der aktuellen Auswahl, die in der Spalte A eine 0 haben. Der cCode löscht die Zeile nicht, wenn 0 das Berechnungsergebnis einer Formel ist. Soll auch in diesem Fall die Zeile gelöscht werden, muss der Ausdruck "And Not .HasFormula" entfernt werden:

```
Sub DeleteRow()
Dim N As Long
For N = Selection(1, 1).Row + Selection.Rows.Count - 1 _
        To Selection(1, 1).Row Step -1
    With Cells(N, 1)
    If .Value = 0 And Not .HasFormula Then
        .EntireRow.Delete
    End If
    End With
Next N
End sub
```

Pearson

Pfad in Fusszeile

Mit folgendem Makro kann der Speicherpfad in die Fusszeile einer Tabelle eingefügt werden:

```
Sub PfadInFusszeile()
     ActiveSheet.PageSetup.LeftFooter = ActiveSheet.Parent.fullname
End Sub
```

Held

Datei öffnen -Menü mit definiertem Pfad starten

Folgendes Makro einer Schaltfläche zuweisen. Der Dialog "Datei öffnen" starten im definierten Verzeichnis D:/MeineDaten

```
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
```

Herber

Tabellenname automatisch nach Zellinhalt benennen

Folgender Code benennt automatisch ein Tabellenblatt nach dem Inhalt einer bestimmten Zelle. Der Name der Zelle ist in diesem Beispiel "jobNumber". Der Code muss in das Alllgemeine Modulblatt der Arbeitsmappe kopiert werden:

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Excel.Range)
```

```
If Target.Address = Sh.Range("jobNumber").Address Then
Sh.Name = szRenameSheet(Sh, Target)
End If
End Sub
Private Function szRenameSheet(ByVal Sh As Worksheet, ByVal Target As Excel.Range)
Dim szName As String
If Not IsNull(Target) Then
szName = CStr(Target.Value)
With Application.WorksheetFunction
szName = .Substitute(szName, ":", "")
szName = .Substitute(szName, "/", "")
szName = .Substitute(szName, "\", "")
szName = .Substitute(szName, "?", "")
szName = .Substitute(szName, "*", "")
szName = .Substitute(szName, "[", "")
szName = .Substitute(szName, "]", "")
End With
szRenameSheet = Left$(szName, 31)
End If
End Function
```

Neuberechnung erzwingen

Generell kann eine Neuberechnung der Arbeitsmappe mit Ctrl+Alt+F9 erzwungen werden. In einem Makro ist dies durch ein Sendkey-Ereignis zu erzielen:

```
SendKeys "^%{F9}"
```

Manville

Formeln zählen

Der folgende Beispielcode gibt die Anzahl der Formeln im aktuellen Arbeitsblatt in einem Dialog aus:

```
Sub Countformula()
Dim R As Integer
R = 0
Range(Cells(1, 1), Selection.SpecialCells(xlLastCell)).Select
For Each Cell In Selection
If Left(Cell.Formula, 1) = "=" Then
R = R + 1
End If
Next Cell
Selection.SpecialCells(xlFormulas, 23).Select
MsgBox "Es sind " & R & " Formeln in der Tabelle " & _
ActiveSheet.Name & "enthalten"
End Sub
```

2. Möglichkeit:

```
Sub CountFormSub()
MsgBox ActiveSheet.UsedRange.SpecialCells(xlFormulas).Count
End Sub
```

Als Funktion:

```
Function countformulas() As Integer
Dim x As Range
Dim y As Integer
Application.Volatile
For Each x In ActiveSheet.UsedRange
If x.HasFormula Then y = y + 1
```

```
Next x
countformulas = y
End Function
```

Namen löschen

Der folgende Code löscht allen Namen aus der aktuellen Arbeitsmappe:

```
Sub DeleteAllNames()
        Dim Nm As Name
        For Each Nm In Names
                Nm.Delete
        Next
End Sub
```

Zufallszahlen

Die folgende Prozedur füllt die aktuelle Auswahl mit Zufallszahlen 0 < X < 100000. Es gibt keine Wiederholungen einzelner Zahlen. Mehrfachmarkierungen sind möglich:

```
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
```

Busko

Benutzernamen auslesen

Der folgenden Aufrufe liefern den Netzwerk-Anmeldenamen:

```
Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuf
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
```

Emails versenden (Outlook 98)

Der folgende Code stammt aus einer Microsoft-Hilfedatei und ermöglicht Mailversand mit Hilfe der anwendungsübergreifenden Möglichkeiten des Office97- Pakets.
Unter Extras/Optionen/Verweise der Excel-Entwicklungsumgebung (Alt+F11) muss zunächst ein Verweis zur Microsoft Outlook 8.0 Object Library hinzugefügt werden:

```
Sub Aufruf()
Call SendMessage(True)
End Sub
Sub SendMessage(DisplayMsg As Boolean, Optional AttachmentPath)
Dim objOutlook As Outlook.Application
Dim objOutlookMsg As Outlook.MailItem
Dim objOutlookRecip As Outlook.Recipient
Dim objOutlookAttach As Outlook.Attachment
' Create the Outlook session.
Set objOutlook = CreateObject("Outlook.Application")
' Create the message.
Set objOutlookMsg = objOutlook.CreateItem(olMailItem)
With objOutlookMsg
' Add the To recipient(s) to the message.
Set objOutlookRecip = .Recipients.Add("Nancy Davolio")
objOutlookRecip.Type = olTo
' Add the CC recipient(s) to the message.
Set objOutlookRecip = .Recipients.Add("Michael Suyama")
objOutlookRecip.Type = olCC
' Add the BCC recipient(s) to the message.
Set objOutlookRecip = .Recipients.Add("Andrew Fuller")
objOutlookRecip.Type = olBCC
' Set the Subject, Body, and Importance of the message.
.Subject = "This is an Automation test with Microsoft Outlook"
.Body = "This is the body of the message." & vbCrLf & vbCrLf
.Importance = olImportanceHigh 'High importance
' Add attachments to the message.
If Not IsMissing(AttachmentPath) Then
Set objOutlookAttach = .Attachments.Add(AttachmentPath)
End If
' Resolve each Recipient's name.
For Each objOutlookRecip In .Recipients
objOutlookRecip.Resolve
Next
' Should we display the message before sending?
If DisplayMsg Then
.Display
Else
.Send
End If
End With
Set objOutlook = Nothing
End Sub
```

Excel-Benutzernamen in Fenstertitel anzeigen

Folgende Prozedur fügt den Excel-Benutzernamen in die Titelleiste ein:

```
Sub FensterName()
ActiveWindow.Caption = ActiveWindow _
.Caption & " " & Application.UserName
End Sub
```

➡ Herber

Excel-Titelzeile ändern

Folgende Prozedur ändert die Excel-Titelleiste:

```
Sub TitelWechseln()
    Application.Caption = "Veränderte Titelleiste"
End Sub
```

Busko

Sound abspielen

Folgender Code verwendet einen Excel 4-Makroaufruf:

```
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
```

Green

Dasselbe kann durch Verwendung einer API - Funktion nach folgenden Beispiel erreicht werden:

```
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
```

Pearson

Datum als Dateiname

Folgender Code speichert die geöffnete Arbeitsmappe mit dem aktuellen Datum als Dateinamen:

```
Option Explicit
Sub DateAsFilename()
Dim sFileName As String
sFileName = Format(Now, "mmddyy") + ".xls"
ActiveWorkbook.SaveAs sFileName
End Sub
```

Seitenzahlen in Zelle

Anzeige der Seitennummer und -anzahl in einer Zelle. Lösung uber Excel 4 - Makrofunktion. Lauffähig auch unter Excel 8:

```
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 "
Else
ActiveCell.Formula = "Seite " & (H_Seite - 1) * V_Seitenanzahl + V_Seite & " von "
End If
End Sub
```

➜Steffens

A

➜API Aufrufe
➜Arbeitsmapenschutz

B

➜Blattschutz
➜Blattschutz , Makrosteuerung
➜Bildschirmauflösung feststellen
➜Bildschirmauflösung, ändern
➜Benutzerdefinierte Funktionen
➜Benutzernamen , auslesen, Netzwerk
➜Benutzernamen , auslesen, Excel

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Quellen

ERROR: undefinedresource
OFFENDING COMMAND: findresource

STACK:

/DefaultColorRendering
/ColorRendering
/DefaultColorRendering