HARALD NAHRSTEDT

Excel + VBA

Ergänzungen

Kapitel

Einführung in VBA

Ordner und Excel-Dateien im Dialog

Erstellt am 04.01.2012

Beschreibung Dieses Kapitel beschreibt, wie Ordner und Dateien mit Dialog-Fenstern und VBA-Code sinnvoll gehandhabt werden können.

1 Prüfen, ob eine Datei existiert

Ein erster Schritt ist oft die Prüfung, ob eine bestimmte Datei vorhanden ist. Dies kann für jeden Dateityp durchgeführt werden. Da in diesem Kapitel der Focus auf Excel liegt, wird im nachfolgenden Beispiel das Extension (*.xlsx) verwendet.

```
Sub CheckIfFileExists()
   If Dir("C:\Temp\Daten.xlsx") <> "" Then
      MsgBox "Die Datei existiert."
   Else
      MsgBox "Die Datei existiert nicht."
   End If
End Sub
```

Da solche Überprüfungen oft an verschiedenen Stellen in einem Programm angewendet werden müssen, ist der Einsatz einer Funktion sinnvoll. Die Anweisung *On Error Resume Next* verhindert, dass innerhalb der Funktion Laufzeitfehler auftreten. Im Falle eines Fehlers gibt die Funktion generell ihren Default-Rückgabewert *False* zurück. Beispielsweise tritt der Laufzeitfehler 68 "Gerät nicht verfügbar" auf, wenn der angegebene Dateipfad auf ein nicht vorhandenes Laufwerk zeigt. Am Ende dieses Kapitels werden wir uns ausführlicher mit Laufzeitfehlern im Umgang mit Dateien beschäftigen.

```
Function FileExist(ByVal strFile As String) As Boolean
   On Error Resume Next
   FileExist = IIf(Dir(strFile) <> "", True, False)
End Function

Sub TestFileExist()
   If FileExist("C:\Temp\Daten.xlsx") = True Then
      MsgBox "Die Datei existiert."
   Else
      MsgBox "Die Datei existiert nicht."
   End If
End Sub
```

Die Funktion FileExist benutzt die Standard-Funktion

```
Function Dir([PathName], _
   [Attributes As VbFileAttribute = vbNormal]) As String
```

Wird darin der Parameter *Attributes* nicht gesetzt, wird der default-Wert *vbNormal* gesetzt, und verborgene Dateien werden nicht berücksichtigt. Sollen diese auch mit berücksichtigt werden, muss der Parameter auf *vbHidden* gesetzt werden.

```
   If Dir("C:\Temp\Daten.xlsx", vbHidden) <> "" Then
```

Die obige Codezeile prüft korrekt die Existenz der Datei "Daten.xlsx", welche verborgen sein könnte.

Der Parameter *Pathname* der Dir-Funktion akzeptiert auch eine leere Zeichenfolge (""). Dann gibt die Dir-Funktion eine Datei des aktuellen Verzeichnisses zurück.

```
Sub DirTest()
   MsgBox Dir("")
   MsgBox CurDir
End Sub
```

Das aktuelle Verzeichnis kann mit der VBA-Funktion *CurDir* abgefragt und mit *ChDir* gewechselt werden. Liegen im aktuellen Verzeichnis mehrere Dateien, so wird ein beliebiger Dateiname zurückgegeben. Gewöhnlich ist es der Name der ersten Datei (nach dem Alphabet). Das muss aber nicht unbedingt der Fall sein.

Für das nachfolgende Beispiel schreiben wir den Dateinamen in die Zelle C2 der aktuellen Tabelle.

	A	B	C
1			
2		Filename	C:\Temp\Daten.xlsx
3		Check	

Die nachfolgende Prozedur prüft wieder, ob die eingetragene Datei vorhanden ist. Ist der Inhalt der Zelle leer, wird der Name einer im aktuellen Verzeichnis abgelegten Datei ausgegeben. Enthält das aktuelle Verzeichnis keine Dateien, so wird ein Leertext "" ausgegeben.

```
Sub CheckFileExist()
  Dim strFileNameUser   As String
  Dim strFileNameCheck  As String

  strFileNameUser = ActiveSheet.Cells(2, 3)
  strFileNameCheck = Dir(strFileNameUser, vbHidden)
  ActiveSheet.Cells(3, 3) = strFileNameCheck
End Sub
```

2 Datei Öffnen-Dialog anzeigen

Es gibt eine Vielzahl verschiedener Möglichkeiten, den *Datei-Öffnen-Dialog* in Excel aufzubauen. Für welche Variante man sich letztlich entscheidet, kommt ganz auf den Verwendungszweck der im Dialogfenster ausgewählten Datei an. Einige Varianten öffnen automatisch die selektierte Datei, sobald der Benutzer auf die *Öffnen-Schaltfläche* klickt. Andere Varianten dagegen liefern lediglich den Dateipfad der selektieren Datei. Hier eine erste Übersicht:

- Dialogs-Auflistung mit Konstante *xlDialogOpen*
- Dialogs-Auflistung mit Konstante *xlDialogFindFile*

- *FindFile*-Methode von Application
- *GetOpenFilename*-Methode von Application
- *GetOpenFileName*-Funktion

2.1 Dialogs-Auflistung mit Konstante xlDialogOpen

Das *Öffnen-Dialogfenster* von Excel lässt sich komfortabel mittels Dialogs-Auflistung unter Verwendung der Konstante *xlDialogOpen* mit der Methode *Show* einblenden.

```
Sub ShowOpenDialog_1()
   Application.Dialogs(xlDialogOpen).Show
End Sub
```

Datei wird das Dialogfenster mit den default-Werten geöffnet. Es kann aber auch der zu suchende Dateiname vorgegeben werden. Er wird aber lediglich im Dialogfenster nur angezeigt.

```
Sub ShowOpenDialog_2()
   Application.Dialogs(xlDialogOpen).Show arg1:="Daten.xlsx"
End Sub
```

Das Dialogfeld vom Type xlDialogOpen besitzt mehrere Parameter. Der Parameter *arg1* ist für den vorzugebenden Dateinamen. Mittels Dialogfenster lässt sich über die Schaltflächen eine Interaktion aufbauen.

```
Sub ShowOpenDialog_3()
   If Application.Dialogs(xlDialogOpen).Show = True Then
      MsgBox "Benutzer hat [Öffnen] geklickt.", vbInformation
   Else
      MsgBox "Benutzer hat [Abbrechen] geklickt.", vbInformation
   End If
End Sub
```

2.2 Dialogs-Auflistung mit Konstante xlDialogFindFile

Das gleiche *Öffnen-Dialogfenster* von Excel lässt sich auch mittels Dialogs-Auflistung unter Verwendung der Konstante *xlDialogFindFile* mit der Methode *Show* einblenden.

```
Public Sub ShowFindFile()
   Application.Dialogs(xlDialogFindFile).Show
End Sub
```

2.3 FindFile-Methode von Application

Eine weitere Möglichkeit auf das Öffnen-Dialogfenster zuzugreifen, ist die FindFile-Methode der Applikation.

```
Public Sub ShowAppFindFile()
   Application.FindFile
End Sub
```

Doch auch über die Applikation kann auf die Dialog-Fenster zugegriffen werden. So lässt sich deren Aufruf genauer über den Applikationstyp steuern.

```
Public Sub ShowAppDialog()
   If InStr(Application.Name, "Excel") > 0 Then
      Application.Dialogs(1).Show
      '1=xlDialogOpen
   ElseIf InStr(Application.Name, "Word") > 0 Then
      Application.Dialogs(80).Show
      '80=wdDialogFileOpen
   End If
End Sub
```

2.4 GetOpenFilename-Methode von Application

Die Applikation hat noch eine weitere Methode für den Zugriff auf das Öffnen-Dialogfeld. Es ist die *GetOpenFilename*-Methode. Sie liefert sofort das Ergebnis der Interaktion.

```
Public Sub ShowGetOpenFile()
  MsgBox Application.GetOpenFileName("Excel Datei (*.xlsx), _
     *.xlsx")
End Sub
```

Voraussetzung für das Öffnen ist eine vorhandene Datei. Anders ist dies mit der *GetSaveAsFilename*-Methode (siehe Kapitel 1.5.18).

2.5 GetOpenFileName-Funktion

Die letzte Version ruft den Standard Windows-Dialog zum Öffnen einer Datei auf. Dazu ist die Deklaration eines Benutzertyps und der Bibliotheksfunktion GetOpenFileName erforderlich. Danach kann auf diese Strukturen zugegriffen werden.

```
'User Definition Type
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

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

'User Procedure
Sub ShowGetOpenFileName()
   Dim udtOFN  As OPENFILENAME
   Dim lngRC   As Long
   Dim strFile As String

   With udtOFN
      .lStructSize = Len(udtOFN)
      .lpstrTitle = "Datei Öffnen" & vbNullChar
      .lpstrFilter = "Exceldatei" & vbNullChar & "*.xlsx"
      .nFilterIndex = 1
      .lpstrFile = Space(256) & vbNullChar
      .nMaxFile = Len(.lpstrFile)
      .lpstrFileTitle = Space(256) & vbNullChar
      .nMaxFileTitle = Len(.lpstrFileTitle)
      .lpstrInitialDir = "C:\Temp" & vbNullChar
      lngRC = GetOpenFileName(udtOFN)
      If lngRC <> 0 Then
         strFile = Left$(.lpstrFile, InStr(1, _
            .lpstrFile, Chr(0)) - 1)
         MsgBox strFile, vbInformation
      End If
   End With
End Sub
```

3 Mehrere Dateien öffnen

Die *GetOpenFilename*-Methode der Applikation erlaubt auch das Selektieren und Öffnen mehrere Dateien. Dazu muss das Attribut der Mehrfachselektion auf *True* gesetzt werden.

```
Sub MultipleFilesOpen()
   Dim iCounter      As Integer
   Dim vFileNames    As Variant

   vFileNames = Application.GetOpenFileName(, , , , True)
   iCounter = 1
   While iCounter <= UBound(vFileNames)
      Workbooks.Open vFileNames(iCounter)
      MsgBox vFileNames(iCounter)
```

```
      iCounter = iCounter + 1
   Wend
End Sub
```

4 Schreibgeschützte Dateien

Eine Datei besitzt ein *Read-Only*-Attribut. Ist es gesetzt, kann die Datei nur gelesen, aber nicht veränderet werden. Die nachfolgende Prozedur überprüft dieses Attribut einer vorgegebenen Datei mithilfe der Konstanten *vbReadOnly*.

```
Sub CheckReadOnly_1()
   If GetAttr("C:\Temp\Daten.xlsx") And vbReadOnly Then
      MsgBox "Die Datei ist schreibgeschützt."
   Else
      MsgBox "Die Datei ist nicht schreibgeschützt."
   End If
End Sub
```

Das Attribut *GetAttr* kann aber auch noch andere Werte besitzen. Der von *GetAttr* zurückgegebene Wert ist immer die Summe der folgenden Attributwerte:

Konstante	Wert	Beschreibung
vbNormal	0	Normal
vbReadOnly	1	Schreibgeschützt
vbHidden	2	Versteckt
vbSystem	4	Systemdatei (beim Macintosh nicht verfügbar)
vbDirectory	16	Verzeichnis oder Ordner
vbArchive	32	Datei wurde seit dem letzten Sichern geändert (beim Macintosh nicht verfügbar)
vbAlias	64	Angegebener Dateiname ist ein Alias (nur beim Macintosh verfügbar)

Da der Schreibschutz immer den Wert 1 besitzt, egal welche Werte noch aufaddiert werden, kann auch der Attributwert mit einer ganzzahligen Division auf Rest = 1 abgefragt werden. Dieser Test funktioniert immer korrekt, weil die Summe der Attributwerte bei einer schreibgeschützten Datei nie ohne Rest durch 2 teilbar ist. Verwendet wird dazu die Modulo-Funktion.

```
Sub CheckReadOnly_2()
   If GetAttr("C:\Temp\Daten.xlsx") Mod 2 = 1 Then
      MsgBox "Die Datei ist schreibgeschützt."
   Else
      MsgBox "Die Datei ist nicht schreibgeschützt."
   End If
End Sub
```

Wie schon bei der Datei selbst, macht es auch für diese Abfrage Sinn, mit einer Funktion zu arbeiten.

```
Function CheckReadOnly(strFile As String) As Boolean
   'Konstante abfragen
   CheckReadOnly = GetAttr(strFile) And vbReadOnly
   'oder Attributwert testen
   CheckReadOnly = GetAttr(strFile) Mod 2 = 1
End Function

Sub TestReadOnly()
   If CheckReadOnly("C:\Temp\Daten.xlsx") Then
      MsgBox "Die Datei ist schreibgeschützt."
   Else
      MsgBox "Die Datei ist nicht schreibgeschützt."
   End If
End Sub
```

5 Ordner im Windows-Explorer anzeigen

Der Aufruf des Windows-Explorer eignet sich hervorrangend zur Anwahl des Arbeitsordners. Auch hier gibt es verschiedene Möglichkeiten. Die nachfolgende Version nutzt die Shell-Methode der Anwendung unter Vorgabe eines Startordners.

```
Sub ShowWindowsExplorer_1()
   Dim strFolder As String

   strFolder = "C:\Temp"
   Shell "Explorer.exe " & strFolder, vbNormalFocus
End Sub
```

Die nachfolgende Version nutzt ebenfalls die Shell-Methode, startet den Explorer aber mit dem Attribut /e in der Explorer-Ansicht (Verzeichnisbaum links, Dateiansicht rechts).

```
Sub ShowWindowsExplorer_2()
   Dim strFolder As String

   strFolder = "C:\Temp"
   Shell "Explorer.exe /e, " & strFolder, vbNormalFocus
End Sub
```

Syntax für den Explorer:

```
Explorer [/n] [/e][,/root,object][[,/select],subobject]
```

Parameterübersicht:

/root,object	Wenn Sie diesen Parameter angeben, wird der Pfad im Explorer als Root angezeigt (Sie können nicht weiter * blättern) Beispiel: Explorer /e,/root,c:\temp
/e	Auf der linken Seite wird ein Verzeichnisbaum angezeigt. Wenn Sie diesen Parameter weglassen, wird im Explorer nur der Ordnerinhalt angezeigt.
/n	Öffnet immer einen neuen Explorer, auch wenn schon einer geöffnet ist.
subobject	Legt den Ordner fest, der den Focus erhält, wenn der Explorer gestartet wird. Als Standard ist mit root definiert. (nicht in Verbindung mit /SELECT möglich)
/select	Legt fest, dass ein Ordner geöffnet wird und ein Objekt im Ordner selektiert ist.

Gelegentlich möchte man den Windows Explorer mit der Ordneransicht öffnen und den Inhalt des System- bzw. Home-Laufwerkes anzeigen. Dann muss man explizit den Pfad dieses Laufwerkes übergeben. Diesen erhält man unter anderem mit der Environ-Funktion von VBA. Der benötigte Bezeichner lautet *SYSTEMDRIVE* bzw. *HOMEDRIVE*.

```
Sub ShowWindowsExplorer_3()
  Shell "Explorer.exe " & Environ("SYSTEMDRIVE"), vbNormalFocus
End Sub
```

Die nächste Version nutzt die FollowHyperelink-Methode des Workbook-Objekts.Datei zeigt der Explorer den Verzeichnisbaum links und die Dateiansicht rechts.

```
Sub ShowWorkbookHyperlink()
   ActiveWorkbook.FollowHyperlink "C:\Temp"
End Sub
```

6 Ordner erstellen

Um einen neuen Ordner zu erstellen bzw. anzulegen, gib es ebenfalls unterschiedliche Methoden. Es kann die Make Directory-Methode (MkDir), die Add-Methode eines SubFolders oder die CreateFolder-Methode aus der FileSystemObject-Bibliothek verwendet werden.

Die nachfolgende Prozedur erstellt mit der *MkDir*-Methode ein vorgegebenes Verzeichnis einschließlich Pfadangabe.

```
Sub CreateDirectory_1()
   MkDir "C:\Temp\NeuesVerzeichnis"
End Sub
```

Wird der Pfad nicht mit angegeben, dann wird das neue Verzeichnis als Unterordner des aktuellen Verzeichnisses erstellt.

Die nächste Version benutzt benutzt die *Add*-Methode aus der FileSystemObject-Bibliothek.

```
Sub CreateDirectory_2()
   CreateObject("Scripting.FileSystemObject"). _
     GetFolder("C:\Temp").SubFolders.Add "NeuesVerzeichnis"
End Sub
```

Außerdem besitzt die FileSystemObject-Bibliothek noch die *CreateFolder*-Methode.

```
Sub CreateDirectory4()
   CreateObject("Scripting.FileSystemObject"). _
      CreateFolder "C:\Daten\NeuesVerzeichnis"
End Sub
```

Die fehlerhafte Nutzung der Versionen kann in unterschiedlichen Situationen unterschiedliche Fehlermeldungen auslösen. Mehr dazu in einem späteren Unterkapitel.

7 Verzeichnisbäume

In diesem Unterkapitel geht es darum, Ordner mit Unterordnern in einem Schritt zu erstellen. Sie vermuten richtig – auch hier gibt es verschiedene Methoden.

Die erste Version nutzt die API-Funktion M*akeSureDirectoryPathExists* aus der *ImageHlp.dll* Bibliothek zum Erstellen eines Verzeichnisbaumes.

```
Declare Function MakeSureDirectoryPathExists Lib _
   "imagehlp.dll" (ByVal DirPath As String) As Long

Sub CreateFolderTree()
   If MakeSureDirectoryPathExists("C:\Temp\Temp1\Temp2\") <> 0 Then
      MsgBox "Der Pfad wurde angelegt oder existiert bereits."
   Else
      MsgBox "Der Pfad konnte nicht angelegt werden."
   End If
End Sub
```

Wichtig bei dieser Funktion ist, dass als letztes Zeichen im Pfad unbedingt ein Backslash (\) angegeben werden muss. Andersfalls wird der Pfad ohne Fehlermeldung nicht angelegt.

Die gleiche API-Funktion verwendet die Funktion *CreateDirectoryTree*. Sie liefert den Wert *True*, wenn der Pfad erfolgreich angelegt wurde, oder wenn er bereits existiert.

```
Declare Function MakeSureDirectoryPathExists Lib _
   "imagehlp.dll" (ByVal DirPath As String) As Long

Function CreateDirectoryTree(ByVal strNewPath As String) As Boolean
   If Right$(strNewPath, 1) <> "\" Then
      strNewPath = strNewPath & "\"
   End If
```

```
   If MakeSureDirectoryPathExists(strNewPath) <> 0 Then
      CreateDirectoryTree = True
   Else
      CreateDirectoryTree = False
   End If
End Function

Sub TestCreateTree()
   If CreateDirectoryTree("C:\Temp\Temp1\Temp2") = True Then
      MsgBox "Der Pfad wurde angelegt oder existiert bereits."
   Else
      MsgBox "Der Pfad konnte nicht angelegt werden."
   End If
End Sub
```

Die nächste Variante benutzt den DOS-Befehl MD (Make Directory). Er wird in einem Konsolenfenster ausgeführt. Ein großer Nachteil dieser Version ist, dass keine Erfolgskontrolle stattfindet. So erfahren Sie nicht, ob die Aktion erfolgreich war.

```
Public Sub CreateDirectoryTree1()
   Shell "CMD /C MD C:\Temp\Temp1\Temp2", vbMinimizedNoFocus
End Sub
```

Die nachfolgende Version ist eine typische VBA-Methode, die mit einer *Do-Loop*-Schleife arbeitet. Der Pfad wird in einzelne Elemente zerlegt und abgearbeitet.

```
Public Sub CreateDirectoryTree_2()
   Const Path     As String = "C:\Temp\Temp1\Temp2"
   Dim strFolder  As String
   Dim intPos1    As Integer
   Dim intPos2    As Integer

   On Error GoTo ErrorHandler
   ChDrive Left$(Path, 1)
   ChDir Left$(Path, 3)
   intPos1 = InStr(Path, "\")
   If intPos1 > 0 Then
      Do
         intPos2 = InStr(intPos1 + 1, Path, "\")
         If intPos2 > 0 Then
            strFolder = Mid$(Path, intPos1 + 1, _
               intPos2 - intPos1 - 1)
            intPos1 = intPos2
         Else
            strFolder = Mid$(Path, intPos1 + 1)
         End If
         MkDir strFolder
         DoEvents
         If intPos2 > 0 Then
            ChDir Left$(Path, intPos2 - 1)
         Else
```

```
            ChDir Path
            Exit Do
         End If
      Loop
   End If
   Exit Sub

ErrorHandler:
   If Err.Number <> 75 Then
      MsgBox "Fehler beim Anlegen/Wechseln des Ordners '" & _
         strFolder & "'!", vbExclamation
   End If
   Resume Next
End Sub
```

Die nachfolgende Version ist eine Abänderung der vorhergehenden in der Form, dass alle Möglichkeiten in einer Funktion gekapselt sind und dass der Funktionswert alle Alternativen liefert:

Funktionswert	Beschreibung
0	Ein Fehler ist aufgetreten
-1	Der Pfad existiert bereits
-2	Der Ordner konnte nicht angelegt werden
n (>0)	Der Pfad wurde angelegt (die Zahl n entspricht der Anzahl neu erstellter Ordner)

```
Function CreateTree(ByVal Path As String) As Integer
   Dim strFolder   As String
   Dim intPos1     As Integer
   Dim intPos2     As Integer
   Dim intMake     As Integer
   Dim intError    As Integer

   On Error GoTo ErrorHandler
   ChDrive Left$(Path, 1)
   ChDir Left$(Path, 3)
   intPos1 = InStr(Path, "\")
   If intPos1 > 0 Then
      Do
         intPos2 = InStr(intPos1 + 1, Path, "\")
         If intPos2 > 0 Then
            strFolder = Mid$(Path, intPos1 + 1, _
               intPos2 - intPos1 - 1)
            intPos1 = intPos2
         Else
            strFolder = Mid$(Path, intPos1 + 1)
         End If
         MkDir strFolder
         DoEvents
```

```
         intMake = intMake + 1
         If intPos2 > 0 Then
            ChDir Left$(Path, intPos2 - 1)
         Else
            ChDir Path
            Exit Do
         End If
      Loop
   End If
   If intMake = intError Then
      CreateTree = -1
   Else
      CreateTree = intMake - intError
   End If
   Exit Function

ErrorHandler:
   If Err.Number = 75 Then
      intError = intError + 1
   ElseIf Err.Number = 76 Then
      CreateTree = -2
      Exit Function
   Else
      CreateTree = 0
      Exit Function
   End If
   Resume Next
End Function

Sub TestCreateTree_2()
   Const strPath As String = "C:\Temp\Temp1\Temp2"
   Dim intRC As Integer

   intRC = CreateTree(strPath)
   If intRC = 0 Then
      MsgBox "Fehler beim Anlegen des Pfades " & _
         strPath
   ElseIf intRC = -1 Then
      MsgBox "Der anzulegende Pfad " & strPath & _
         " existiert bereits."
  ElseIf intRC = -2 Then
      MsgBox "Ein Ordner des anzulegenden Pfades " & _
         strPath & " konnte nicht erstellt werden."
  Else
      If intRC = 1 Then
         MsgBox "Der Pfad '" & strPath & _
            " wurde angelegt. Es wurde(n) " & intRC & _
            " neue(r) Ordner erstellt."
      Else
         MsgBox "Der Pfad " & strPath & _
            " wurde angelegt. Es wurde(n) " & intRC & _
            " neue(r) Ordner erstellt."
      End If
   End If
End Sub
```

Es gibt natürlich noch weitere Fehlermöglichkeiten, wie etwa ein schreibgeschützter Datenträger, eine nicht vorhandene Schreibberechtigung, und vieles andere mehr.

8 Kurze und lange Pfadnamen

Als Anwender freuen wir uns seit mehreren Jahren darüber, nicht mehr an die alte Dateikonvention gebunden zu sein, die bis zu Windows 95 zum Alltag gehörte. Als Entwickler ist es nicht so einfach. Immer wieder tauchen durch Tilde geprägte Abkürzungen langer Pfade auf. Meistens bei 16-Bit-Kommunokationen oder anderen unliebsamen Weggefährten aus der Vergangenheit.

Mit der API-Funktion OSGetShortPathName kann die kurze Schreibweise (hier C:\Temp\NEURO~1\NEURO~1\NEURO~1) eines langen Pfades (hier C:\Temp\Neuer Ordner 1\Neuer Ordner 2\Neuer Ordner 3) abgefragt werden.

```
Declare Function OSGetShortPathName Lib "kernel32" Alias _
   "GetShortPathNameA" (ByVal lpszLongPath As String, ByVal _
   lpszShortPath As String, ByVal cchBuffer As Long) As Long

Function GetShortPathName(ByVal strLongPath As String) As String
   Const cchBuffer = 300
   Dim strShortPath As String
   Dim lResult As Long
   Dim intZeroPos As Integer

   strShortPath = String(cchBuffer, Chr$(0))
   lResult    =    OSGetShortPathName(strLongPath,    strShortPath,
cchBuffer)
   If lResult = 0 Then
      MsgBox "Pfad nicht gefunden!", vbExclamation
      GetShortPathName = ""
   Else
      intZeroPos = InStr(strShortPath, Chr$(0))
      If intZeroPos > 0 Then
         GetShortPathName = Left$(strShortPath, intZeroPos - 1)
      Else
         GetShortPathName = strShortPath
      End If
   End If
End Function

Sub TestShortPathName()
   MsgBox GetShortPathName _
      ("C:\Temp\Neuer Ordner 1\Neuer Ordner 2\Neuer Ordner 3")
End Sub
```

Die nachfolgende Funktion zeigt, wie man einen langenPfadnamen (hier C:\Temp\Neuer Ordner 1\Neuer Ordner 2\Neuer Ordner 3) zu einem kurzen Pfadnamen (hier C:\Temp\NEURO~1\NEURO~1\NEURO~1) ermittelt.

```
Function GetLongPathName(ByVal strShortName As String) As String
   Dim strLongName As String
```

```
   Dim strTemp As String
   Dim intSlashPos As Integer

   On Error Resume Next
   strShortName = strShortName & "\"
   intSlashPos = InStr(4, strShortName, "\")
   While intSlashPos
      strTemp = Dir(Left$(strShortName, intSlashPos - 1), _
         vbNormal + vbHidden + vbSystem + vbDirectory)
      If strTemp = "" Then
         Err.Clear
         GetLongPathName = ""
         Exit Function
      End If
      strLongName = strLongName & "\" & strTemp
      intSlashPos = InStr(intSlashPos + 1, strShortName, "\")
   Wend
   GetLongPathName = Left$(strShortName, 2) & strLongName
End Function

Sub TestLongPathName()
  MsgBox GetLongPathName("C:\Temp\NEUERO~1\NEUERO~1\NEUERO~1")
End Sub
```

Beide Varianten funktionieren auch, wenn der Pfad die Dateinamen enthält. Hier sind dies

C:\Temp\Neuer Ordner 1\Neuer Ordner 2\Neuer Ordner 3\Daten.xlsx

bzw.

C:\Temp\NEUERO~1\NEUERO~1\NEUERO~1\DATEN~1.XLS

9 Arbeitsmappe anlegen

9.1 Arbeitsmappe mit Standardwerten anlegen

Da der Focus dieser Dokumentation auf Excel-Anwendungen liegt, dürfen Excel-Arbeitsmappen nicht vergessen werden. Arbeitsmappen lassen sich mit der Add-Methode der Workbooks-Objektliste erstellen. Die Methode besitzt ein *Template*-Attribut.

Wird das Attribut nicht angegeben, dann wird eine neue Arbeitsmappe mit der vorgegebenen Anzahl Registerblätter erzeugt. Diese Anzahl kann über das Attribut *SheetsInNewWorkbook* gesetzt und abgefragt werden.

Eine Mustervorlagemappe wird beim nachfolgenden Aufruf nicht verwendet.

```
Sub AddNewWorkbook_1()
  Workbooks.Add
End Sub
```

Sinnvoller ist die nachfolgende OOP-Version.

```
Sub AddNewWorkbook_2()
  Dim wkbWorkbook As Workbook

  Set wkbWorkbook = Workbooks.Add

  'weitere Anweisungen

  Set wkbWorkbook = Nothing
End Sub
```

9.2 Arbeitsmappe nach Vorlage anlegen

Soll die neue Arbeitsmappe nach einer Vorlagenmappe (hier Muster.xlsm) erstellt werden, so muss lediglich der Name der Vorlage in der Add-Methode mit angegeben werden. Ich habe deshalb als Muster eine Mappe mit Prozeduren gewählt, weil dies der häufigste Fall ist. Natürlich kann auch eine Mappe ohne Prozeduren (*.xlsx) gewählt werden.

```
Sub AddNewMusterBook_1()
   Workbooks.Add ("C:\Temp\Muster.xlsm")
End Sub
```

Und ebenfalls in der OOP-Version.

```
Sub AddNewMusterBook_2()
   Dim wkbWorkbook As Workbook

   Set wkbWorkbook = Workbooks.Add("C:\Temp\Muster.xlsm")
'weitere Anweisungen
   Set wkbWorkbook = Nothing
End Sub
```

10 Arbeitsmappe öffnen

10.1 Arbeitsmappe nicht geöffnet

Eine Arbeitsmappe wird in der Regel mit der Open-Methode des Workbooks-Listenobjekts geöffnet. Doch es gibt auch noch einige andere Methoden. Diese sollen nachfolgend behandelt werden.

Die erste Version öffnet eine Arbeitsmappe mit den default-Werten. Das Filename-Attribut kann auch weggelassen werden. Die Open-Methode akzeptiert auch relative Pfade, UNC-Pfade und File-URLs.

```
Sub OpenWorkbook_1()
   Workbooks.Open Filename:="C:\Temp\Muster.xlsm"
End Sub
```

Die nächste Version unterdrückt die Rückfrage *Externe Verknüpfungen aktualisieren* durch Verwendung des *UpdateLinks*-Attributes. Wird dieses Attribut auf *False* (oder auch 0) gesetzt, werden vorhandene Verknüpfungen nicht aktualisiert.

```
Sub OpenWorkbook_2()
   Workbooks.Open Filename:="C:\Temp\Muster.xlsm", UpdateLinks:=0
End Sub
```

Die nächste Version öffnet eine Arbeitsmappe schreibgeschützt. Dazu wird das ReadOnly-Attribut auf True (oder auch 1) gesetzt.

```
Sub OpenWorkbook_3()
   Workbooks.Open Filename:="C:\Temp\Muster.xlsm", ReadOnly:=True
End Sub
```

Die nächste Version benutzt die *FollowHyperlink*-Methode des Workbook-Objekts. Sie funktioniert genauso wie die *Open*-Methode. Bedingung für die Nutzung ist, wie aus dem Code zu ersehen, dass es bereits eine geöffnete Arbeitsmappe gibt (ActiveWorkbook).

```
Sub OpenWorkbook_4()
   ActiveWorkbook.FollowHyperlink "C:\Temp\Muster.xlsm"
End Sub
```

Die nächste Version öffnet die Arbeitsmappe mithilfe der *Run*-Methode des Application-Objekts. Zu beachten ist, dass nach dem Dateinamen eine Ausrufungszeichen und ein beliebiges Zeichen oder Begriff stehen muss. In der Regel verwendet man ein einzelnes Leerzeichen. Diese Konstruktion beruht auf der Tatsache, dass die *Run*-Methode eigentlich zum Aufruf eines Makros gedacht ist, dessen Name nach dem Ausrufungszeichen stehen sollte. Da aber ein Leerzeichen garantiert kein Makroname ist, tritt der Laufzeitfehler 1004 auf, der die Fehlermeldung: Microsoft Excel kann das Makro … nicht finden, generiert. Das wird aber durch die Anweisung *On Error Resume Next* verhindert.

```
Sub OpenWorkbook_5()
   On Error Resume Next
   Application.Run "C:\Temp\Muster.xlsm! "
End Sub
```

Bei dieser Version wird die zu öffnende Arbeitsmappe nicht zur aktiven Arbeitsmappe. Außer, es gab vorher keine aktive Arbeitsmappe. Die zu öffnende Arbeitsmappe wird beim Öffnungsvorgang kurzzeitig aktiviert und dann gleich deaktiviert. Dadurch werden die Workbook-Ereignisse ausgelöst und aufgeführt. Das gilt nicht für eine *Auto_Open*-Prozedur dieser Arbeitsmappe. Die Ausführung von Ereignissen kann durch Setzen des EnableEvents-Attribut der Applikation auf *False* unterbunden werden.

Die nächste Version sucht das Kombinationsfeld Adresse der Web-Symbolleiste. Dieses Steuerelement besitzt die ID 1740. In dessen Text-Attribut wird der Name der Arbeitsmappe

eingetragen, wodurch diese automatisch geöffnet wird. Eine Bestätigung der Eingabe über die *Enter*-Taste ist nicht erforderlich. Der Dateipfad muss nicht als File URL angegeben werden. Dieser Code funktioniert in allen gängigen Office Applikationen, und auch bei nicht eingeblendeter Web-Symbolleiste.

```
Sub OpenWorkbook_6()
   Application.CommandBars.FindControl(ID:=1740).Text = _
      "file:///C:/Temp/Muster.xlsm"
End Sub
```

In der nächsten Version wird die Arbeitsmappe mit der *GetObject*-Funktion geöffnet. Damit ist die Arbeitsmappe aber nicht automatisch sichtbar. Dies besorgt das *Visible*-Attribut des Workbook.Windows-Objekts.

```
Sub OpenWorkbook_7()
   Dim wkbWorkbook As Workbook

   Set wkbWorkbook = GetObject("C:\Temp\Muster.xlsm")
   wkbWorkbook.Windows(1).Visible = True

   Set wkbWorkbook = Nothing
End Sub
```

Diese Prozedur gibt es auch noch in Kurzform.

```
Sub OpenWorkbook_8()
   GetObject("C:\Temp\Muster.xlsm").Windows(1).Visible = True
End Sub
```

10.2 Eine bereits geöffnete Arbeitsmappe erneut öffnen

Eine bereits geöffnete Arbeitsmappe, kann mit der *Open*-Methode der Workbooks-Objektliste erneut zum Öffnen aufgerufen werden. Dabei wird automatisch die geöffnete Arbeitsmappe geschlossen und neu vom Datenträger geladen. Enthält die geöffnete Arbeitsmappe jedoch noch nicht gespeicherte Änderungen, dann erscheint zuerst ein Hinweistext, dass alle Änderungen verloren gehen. Der Anwender kann entscheiden, ober er den Vorgang abbrechen oder fortsetzen will.

Wenn diese Meldung nicht erscheinen soll, müssen die Systemmeldungen anhand des DisplayAlerts-Attributes vorübergehend deaktiviert werden.

11 Arbeitsmappe ist bereits geöffnet

11.1 Feststellen, ob eine Arbeitsmappe bereits geöffnet ist

Die nachfolgend dargestellt Funktion durchläuft alle Objekte der Workbook-Objektliste und vergleicht die Arbeitsmappen-Namen.

```
Public Function CheckIfOpen(strFilename As String) As Boolean
   Dim wbkWorkbook As Workbook

   For Each wbkWorkbook In Application.Workbooks
      If UCase(wbkWorkbook.Name) = UCase(strFilename) Then
         CheckIfOpen = True
         Exit Function
      End If
   Next wbkWorkbook
   CheckIfOpen = False
End Function

Sub TestBookIfOpen()
   If CheckIfOpen("Muster.xlsm") Then
      MsgBox "Die Mappe ist geöffnet."
   Else
      MsgBox "Die Mappe ist nicht geöffnet."
   End If
End Sub
```

Die nächste Version folgt einem anderen Ansatz. Sie versucht die gesuchte Arbeitsmappe anzulegen. Schlägt der Versuch fehl, dann wird der Laufzeitfehler 9 ausgelöst. Dieser wird mit der *On Error Resume Next*-Anweisung abgefangen und ausgewertet.

```
Sub CheckIfBookIsOpen()
   Dim wkbWorkbook As Workbook

   On Error Resume Next
   Set wkbWorkbook = Workbooks("Muster.xlsm")
   If Err.Number = 9 Then
      MsgBox "Die Mappe ist nicht geöffnet."
   ElseIf Err.Number <> 0 Then
      MsgBox "Ein Fehler ist aufgetreten."
   Else
      MsgBox "Die Mappe ist geöffnet."
      Set wkbWorkbook = Nothing
   End If
End Sub
```

11.2 Arbeitsmappe in Bearbeitung

Die nächste Version zeigt eine Möglichkeit um festzustellen, ob eine Arbeitsmappe bereits in Bearbeitung ist. Egal, ob in der aktuellen oder einer anderen Sitzung. Dazu wird die Arbeitsmappe mit der Open-Methode exklusiv im Lesezugriff-Modus (For Input Lock Read) geöffnet. Tritt dabei der Laufzeitfehler 70 auf, dann ist die Arbeitsmappe in Bearbeitung.

```
Function IsFileOpen(strFilename As String) As Boolean
   Dim intErrorNum As Integer

   On Error Resume Next
   Open strFilename For Input Lock Read As #1
```

```
   Close #1
   intErrorNum = Err.Number
   On Error GoTo 0
   Select Case intErrorNum
   Case 0
      'Kein Fehler, Datei ist nicht geöffnet
      IsFileOpen = False
   Case 70
      'Fehler Zugriff verweigert, Datei ist in Bearbeitung
      IsFileOpen = True
   Case Else
      'Anderer Fehler: Fehlermeldung anzeigen
      Error intErrorNum
   End Select
End Function

Sub TestBookInWork()
   If IsFileOpen("C:\Temp\Muster.xlsm") Then
      MsgBox "Die Datei wird momentan bearbeitet."
   Else
      MsgBox "Die Datei wird momentan nicht bearbeitet."
   End If
End Sub
```

11.3 Arbeitsmappe schreibgeschützt geöffnet

Die nächste Version prüft, ob eine Arbeitsmappe schreibgeschützt geöffnet wurde. Zwar erscheint im Fenstertitel der Hinweis [Schreibgeschützt], per VBA kann dies durch eine Überprüfung des ReadOnly-Attributes erfolgen.

```
Sub CheckIfOpenAsReadOnly()
   If ActiveWorkbook.ReadOnly = True Then
      MsgBox "Mappe wurde schreibgeschützt geöffnet."
   Else
      MsgBox "Mappe wurde nicht schreibgeschützt geöffnet."
   End If
End Sub
```

12 Ereignisse beim Öffnen einer Arbeitsmappe

12.1 Die Auto_Open-Prozedur

Im Gegensatz zu den Ereignisprozeduren in einer Arbeitsmappe, wird beim Öffnen über die *Open*-Methode der Workbooks-Objektliste (siehe Unterkapitel 1.5.22.11), die Prozedur *Auto_Open* nie ausgeführt.

```
Sub OpenWithoutAutoOpen()
   Workbooks.Open "C:\Temp\Muster.xlsm"
End Sub
```

Zur Überprüfung stellen Sie in der Arbeitsmappe *Muster.xlsm* die nachfolgende Prozedur in einem Modul ein.

```
Sub Auto_Open()
   MsgBox "Auto_Open-Prozedur wurde aufgerufen!"
End Sub
```

Wenn Sie nun die *OpenWithoutAutoOpen*-Prozedur aufrufen, erscheint keine Meldung. Im Gegensatz erhalten Sie die Meldung, wenn Sie die Arbeitsmappe *Muster.xlsm* manuell öffnen.

Will man jedoch, dass die *Auto_Open*-Prozedur ausgeführt wird, dann muss man sie explizit aufrufen. Dazu wird die *RunAutoMacros*-Methode des Workbook-Objekts genutzt und die Konstante *xlAutoOpen* als Parameter übergeben. Es tritt selbst dann kein Laufzeitfehler auf, wenn die Arbeitsmappe keine *Auto_Open*-Prozedur enthält.

```
Sub OpenWorkbookWithAutoOpen_1()
   Workbooks.Open "C:\Temp\Muster.xlsm"
   ActiveWorkbook.RunAutoMacros xlAutoOpen
End Sub
```

Das Problem dieser Version ist, dass, wenn die geöffnete Arbeitsmappe nicht automatisch zur aktiven Arbeitsmappe wird, die falsche Auto_Open-Prozedur ausgeführt wird. Die nachfolgende Version umgeht dieses Problem durch die Verwendung einer Objekt-Variablen.

```
Sub OpenWorkbookWithAutoOpen_2()
   Dim wkbWorkbook As Workbook

   Set wkbWorkbook = Workbooks.Open("C:\Temp\Muster.xlsm")
   wkbWorkbook.RunAutoMacros xlAutoOpen
   Set wkbWorkbook = Nothing
End Sub
```

Da es auch hier zu Problemen kommen kann, wenn die Arbeitsmappe nicht geöffnet werden konnte, ist die nächste Version die beste Lösung.

```
Sub OpenWorkbookWithAutoOpen_3()
   Dim wkbWorkbook As Workbook

   On Error Resume Next
   Set wkbWorkbook = Workbooks.Open("C:\Temp\Muster.xlsm")
   If Err.Number = 1004 Then
      MsgBox "Die Arbeitsmappe konnte nicht geöffnet werden!", _
         vbExclamation
   Else
      wkbWorkbook.RunAutoMacros xlAutoOpen
      Set wkbWorkbook = Nothing
   End If
End Sub
```

Die nächste Version nutzt die Run-Methode des Application-Objekts. Hinter dem vollständigen Namen der Arbeitsmappe in Hochkommas und einem Ausrufungszeichen, wird der Name der auszuführenden Prozedur, hier Auto_Open übergeben. Der vollständige Name der Arbeitsmappe muss immer in Hochkommas erfolgen, andernfalls wird der Laufzeitfehler 1004 ausgelöst.

```
Sub OpenWorkbookWithAutoOpen_4()
   Application.Run "'C:\Temp\Muster.xlsm'!Auto_Open"
End Sub
```

Die nächste Version fängt den Laufzeitfehler auf.

```
Sub OpenWorkbookWithAutoOpen_5()
   On Error Resume Next
   Application.Run "'C:\Temp\Muster.xlsm'!Auto_Open"
   If Err.Number = 1004 And _
      InStr(Err.Description, "!Auto_Open") > 0 Then
      MsgBox "Die Mappe enthält kein Auto_Open-Prozedur.", _
         vbInformation
   End If
End Sub
```

12.2 Das Workbook_Open-Ereignis

Durch das Deaktivieren aller auftretenden Ereignisse über das *EnableEvents*-Attribut der Applikation, wird erreicht, dass beim Öffnen einer Arbeitsmappe, der darin enthaltene Code nicht ausgeführt wird. Aber auch die Ereignisse der aktiven Arbeitsmappe sind deaktiviert. Daher ist es auch wichtig, dass diese Einstellung nach dem Öffnen wieder zurück gesetzt wird.

```
Sub OpenWorkbookWithoutEvents()
   Application.EnableEvents = False
   Workbooks.Open "C:\Temp\Muster.xlsm"
   Application.EnableEvents = True
End Sub
```

13 Darstellung der geöffneten Arbeitsmappe

13.1 Arbeitsmappe mit ausgeblendetem Fenster öffnen

Sowohl beim Öffnen einer Arbeitsmappe mit der Open-Methode der Workbooks-Objektliste, als auch die FollowHyperlink-Methode verhindern nicht, dass die neue Arbeitsmappe in einem Fenster dargestellt wird. Lediglich die GetObject-Funktion erlaubt, dass die Arbeitsmappe nach dem Öffnen ausgeblendet bleibt, wenn nicht das Vissible-Attribut gesetzt wird.

```
Sub OpenWithHiddenWindow_1()
  GetObject "C:\Temp\Muster.xlsm"
End Sub
```

Die nächste Version nutzt Objekt-Variablen. Die geöffnete Arbeitsmappe wird nicht zur aktiven Arbeitsmappe, da ausgeblendete Mappen nie aktiv sein können.

```
Sub OpenWithHiddenWindow_2()
   Dim wkbWorkbook As Workbook

   Set wkbWorkbook = GetObject("C:\Temp\Muster.xlsm")
   Set wkbWorkbook = Nothing
End Sub
```

13.2 Arbeitsmappe öffnen ohne sie zu aktivieren

Auch wenn es sich hier um eine Wiederholung handelt, soll der Aspekt einer inaktiven Arbeitsmappe noch einmal verdeutlicht werden. Die *Run*-Methode wurde bereits oben beschrieben.

```
Sub OpenWithoutActivation_1()
   On Error Resume Next
   Application.Run "'C:\Temp\Muster.xlsm'! "
End Sub
```

Nach der Ausführung ist die neue Arbeitsmappe nicht aktiv. Dennoch werden die Workbook-Ereignisse (ohne *Auto_Open*) ausgelöst. Der zeitliche Ablauf ist wie folgt:

1. *Open* der neuen Mappe
2. *WindowDeactivate* der alten Mappe
3. *Deactivate* der alten Mappe
4. *Activate* der neuen Mappe
5. *WindowActivate* der neuen Mappe
6. *WindowDeactivate* der neuen Mappe
7. *Deactivate* der neuen Mappe
8. *Activate* der alten Mappe
9. *WindowActivate* der alten Mappe

14 Laufzeitfehler zu Dateien

14.1 Übersicht der Laufzeitfehler beim Dateihandling

Code	Fehlerbeschreibung	Bemerkung
52	Dateiname oder -nummer falsch	
53	Datei nicht gefunden	
54	Falscher Dateimodus	
55	Datei bereits geöffnet	
57	Fehler beim Lesen von/Schreiben auf Gerät	
58	Datei existiert bereits	
59	Falsche Datensatzlänge	
61	Datenträger voll	

62	Einlesen hinter Dateiende	
63	Falsche Datensatznummer	
67	Zu viele Dateien	
68	Gerät nicht verfügbar	Laufwerk existiert nicht
70	Zugriff verweigert	
71	Datenträger nicht bereit	z.B. Datenträger nicht eingelegt
74	Umbenennen bei Angabe unterschiedlicher Laufwerke nicht möglich	
75	Fehler beim Zugriff auf Pfad/Datei	
76	Pfad nicht gefunden	

14.2 Die Präzedenz der Dateizugriff-Laufzeitfehler

Am einfachsten lässt sich die Präzedenz (Rangfolge, Vorrang) von Laufzeitfehlern anhand eines Beispiels erläutern.

Die Arbeitsmappe Muster.xlsm soll in ein anderes Verzeichnis kopiert werden. Die entsprechende VBA-Anweisung lautet:

```
FileCopy "C:\Temp\Muster.xlsm", "C:\Ablage\Daten.xlsm"
```

Dabei können folgende Probleme auftreten:

1. Die Zieldatei existiert bereits, dann wird kein Laufzeitfehler auftreten.
2. Die Zieldatei existiert bereits und ist ReadOnly gesetzt, dann wird der Laufzeitfehler 75 auftreten.
3. Die Zieldatei existiert bereits und ist gesperrt, da sie bereits geöffnet ist. Dann wird der Laufzeitfehler 70 auftreten.

Welche Meldung wird aber auftreten, wenn die Zieldatei ReadOnly und gesperrt ist? In diesem Fall ist es der Laufzeitfehler 70, denn dieser hat Vorrang vor dem Laufzeitfehler 75.

Dieses Beispiel macht deutlich, wie kompliziert das Erkennen und Behandeln von Laufzeitfehlern sein kann. Es ist jedoch wichtig, dass sämtliche potentiellen Fehler abgefangen, ausgewertet und behoben werden.

14.3 Fehlerbehandlungsroutinen für Dateizugriffe

Die nachfolgende VBA-Prozedur soll als Muster für eine eigene Fehlerbehandlungsroutine dienen.

```
Sub FileErrorHandler()
   Dim strFileName As String

   strFileName = "C:\Temp\Muster.xlsm"
   On Error GoTo ErrorHandler
```

```
    'Raum für Prozeduranweisungen

    Exit Sub

ErrorHandler:
    Select Case Err.Number
    Case 52
        'Dateiname oder -nummer falsch
        MsgBox "Der Dateiname " & strFileName & _
            " oder die verwendete Dateinummer ist ungültig!", _
            vbExclamation
    Case 53
        'Datei nicht gefunden
        MsgBox "Die Datei " & strFileName & _
            " konnte nicht gefunden werden!", vbExclamation
    Case 54
        'Falscher Dateimodus
        MsgBox "Der zum Lesen/Schreiben der Datei " & _
            strFileName & _
            " verwendete Dateizugriffsmodus ist ungültig!", _
            vbExclamation
    Case 55
        'Datei bereits geöffnet
        MsgBox "Die Datei " & strFileName & _
            " ist bereits geöffnet!", vbExclamation
    Case 57
        'Fehler beim Lesen von/Schreiben auf Gerät
        MsgBox "Das Gerät der Datei " & strFileName & _
            " kann nicht angesprochen werden!", _
            vbExclamation
    Case 58
        'Datei existiert bereits
        MsgBox "Die Datei " & strFileName & _
            " existiert bereits!", vbExclamation
    Case 59
        'Falsche Datensatzlänge
        MsgBox "Die zum Lesen/Schreiben der Datei " & _
            strFileName & _
            " verwendete Datensatzlänge ist falsch!", _
            vbExclamation
    Case 61
        'Datenträger voll
        MsgBox "Die Datei " & strFileName & _
            " kann nicht gespeichert werden, weil" & _
            " der Datenträger voll ist!", vbExclamation
    Case 62
        'Einlesen hinter Dateiende
        MsgBox "Es wurde versucht, hinter dem Dateiende" & _
            " der Datei " & strFileName & " einzulesen!", _
            vbExclamation
    Case 63
        'Falsche Datensatznummer
        MsgBox "Die beim Zugriff auf die Datei " & _
            strFileName & _
            " verwendete Datensatznummer ist falsch!", _
```

```
         vbExclamation
   Case 67
      'Zu viele Dateien offen
      MsgBox "Auf die Datei " & strFileName & _
         " kann nicht zugegriffen werden, weil" & _
         " zu viele Dateien geöffnet sind!", _
         vbExclamation
   Case 68
      'Gerät nicht verfügbar
      MsgBox "Das Gerät der Datei " & strFileName & _
         " ist nicht verfügbar!", vbExclamation
   Case 70
      'Zugriff verweigert
      MsgBox "Der Zugriff auf die Datei " & _
         strFileName & _
         " wurde verweigert!", vbExclamation
   Case 71
      'Datenträger nicht bereit
      MsgBox "Der Datenträger für die Datei " & _
         strFileName & _
         " ist nicht bereit (Diskette, CD ROM, " & _
         "Wechselplatte o.ä.)!", vbExclamation
   Case 74
      'Umbenennen bei Angabe
      'unterschiedlicher Laufwerke nicht möglich
      MsgBox "Die Datei " & strFileName & _
         " kann nicht umbenannt werden, weil zwei" & _
         " verschiedene Laufwerke angegeben wurden!", _
         vbExclamation
   Case 75
      'Fehler beim Zugriff auf Pfad/Datei
      MsgBox "Die Datei " & strFileName & _
         " ist bereits geöffnet!", vbExclamation
   Case 76
      'Pfad nicht gefunden
      MsgBox "Der Pfad der Datei " & strFileName & _
         " konnte nicht gefunden werden!", vbExclamation
   Case Else
      'Sonstige Laufzeitfehler
      MsgBox "Beim Zugriff auf die Datei " & _
         strFileName & _
         " ist ein Fehler aufgetreten!" & vbCrLf & vbCrLf & _
          "Laufzeitfehler " & Err.Number & vbCrLf & _
          Err.Description, vbExclamation
   End Select
End Sub
```

Mit dem Aufruf

```
MsgBox GetFileErrorMessage(Err.Number, Err.Description, strFile)
```

kann auch individueller Hinweistext zu einem Dateizugriffsfehler über die nachfolgende Funktion zurückgeben werden.

```
Public Function GetFileErrorMessage _
   (ByVal intError As Integer, ByVal strError As String, _
   ByVal strFile As String) As String
   If intError = 52 Then
      'Bad file name or number
      GetFileErrorMessage = _
      "Der angegebene Pfad-/Dateiname '" & _
      strFile & "' ist ungültig!"
   ElseIf intError = 53 Then
      'File not found
      GetFileErrorMessage = _
      "Die angegebene Datei '" & strFile & _
      "' konnte nicht gefunden werden!"
   ElseIf intError = 55 Then
      'File already open
      GetFileErrorMessage = _
      "Die angegebene Datei '" & strFile & _
      "' ist bereits geöffnet!"
   ElseIf intError = 57 Then
      'Device I/O error
      GetFileErrorMessage = _
      "Auf das Gerät mit der angegebenen Datei '" & _
      strFile & "' kann nicht zugegriffen werden!"
   ElseIf intError = 68 Then
      'Device unavailable
      GetFileErrorMessage = _
      "Das Gerät mit der angegebenen Datei '" & _
      strFile & "' ist nicht verfügbar!"
   ElseIf intError = 70 Then
      'Permission denied/File locked
      'File is already opened by another user
      GetFileErrorMessage = _
      "Die angegebene Datei '" & strFile & _
      "' ist nicht verfügbar!"
   ElseIf intError = 71 Then
      'Disk not ready
      GetFileErrorMessage = _
      "Der Datenträger mit der angegebenen Datei '" & _
      strFile & "' ist nicht bereit (Diskette, CD-ROM," & _
      " Wechselplatte usw. ist nicht eingelegt)!"
   ElseIf intError = 75 Then
      'Path/File access error
      'OLD: If Dir(strFileName, vbDirectory) <> "" Then
      If Val(GetAttr(strFile) And vbDirectory) <> 16 Then
         GetFileErrorMessage = _
         "Bei der angegebenen Datei '" & strFile & _
         "' handelt es sich um einen Ordner!"
      Else
         GetFileErrorMessage = _
         "Auf die angegebene Datei '" & strFile & _
         "' kann aus einem unbekannten Grund" & _
```

```
         " nicht zugegriffen werden!"
      End If
   ElseIf intError = 76 Then
      'Path not found
      GetFileErrorMessage = _
      "Das Verzeichnis der angegebenen Datei '" & _
      strFile & "' konnte nicht gefunden werden!"
   ElseIf intError <> 0 Then
      'All other errors
      GetFileErrorMessage = _
      "Beim Zugriff auf die angegebene Datei '" & strFile & _
      "' ist ein nicht näher bekannter Fehler aufgetreten!" & _
      vbCrLf & vbCrLf & "Fehler " & intError & vbCrLf & strError
   Else
      'No error
      GetFileErrorMessage = _
      "Es trat kein Fehler auf."
   End If
End Function
```