Excel + VBA

Ordner und Dateien

Autor & Copyright: Dipl.-Ing. Harald Nahrstedt

Version: 2016 / 2019 / 2021 / 365

Erstellungsdatum: 04.01.2012

Überarbeitung: 01.12.2023

Beschreibung:

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

Anwendungs-Datei: AE-017_OrdnerDateien.xlsm

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 (*.xlsm) verwendet.

*Codeliste 1. Die Prozedur im Modul mod_01_FileExist prüft die Existenz einer Exeldatei vom Typ *.xlsm*

```
Sub CheckIfFileExists()
   If Dir("C:\Temp\Daten.xlsm") <> "" 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.

*Codeliste 2. Prozedur und Funktion im Modul mod_01_FileExist prüfen die Existenz einer Exeldatei vom Typ *.xlsm*

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

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

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 Standard-Wert *vbNormal* gesetzt, und verborgene Dateien werden nicht berücksichtigt. Sollen diese auch mitberücksichtigt 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.xlsm", 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.

Codeliste 3. Die Prozedur im Modul mod_01_FileExist zeigt Dateien des aktuellen Verzeichnisses

```
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 (Bild 1).

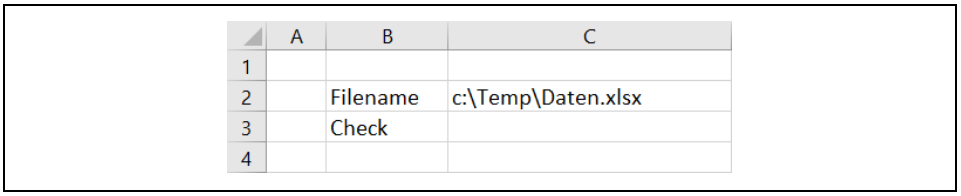

Bild 1. Vorgabe eines Dateinamens

Die folgende 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.

Codeliste 4. Die Prozedur im Modul mod_01_FileExist prüft die Existenz der vorgegebenen Datei

```
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.

Codeliste 5. Die Prozedur im Modul mod_02_FileOpen nutzt das Öffnen-Dialogfenster

```
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.

Codeliste 6. Die Prozedur im Modul mod_02_FileOpen nutzt das Öffnen-Dialogfenster mit einer Vorgabe

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

Codeliste 7. Interaktion mit dem Öffnen-Dialogfenster im Modul mod_02_FileOpen

```
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.

Codeliste 8. Anwendung des FindFile-Dialogs im Modul mod_02_FileOpen

```
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.

Codeliste 9. Anwendung der Methode FindFile des Application-Objekts im Modul mod_02_FileOpen

```
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.

Codeliste 10. Anwendung der Methode FindFile des Application-Objekts im Modul mod_02_FileOpen

```
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.

Codeliste 11. Anwendung der Methode GetOpenFileName des Application-Objekts im Modul mod_02_FileOpen

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

Voraussetzung für das Öffnen ist eine vorhandene Datei.

3 Mehrere Dateien öffnen

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

Codeliste 12. Die Prozedur im Modul mod_03_MultipleFiles verwendet die GetOpenFilename-Methode

```
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*.

Codeliste 13. Die Prozedur im Modul mod_04_ReadOnly überprüft das Read-Only-Attribut

```
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.

Tabelle 1. Dateiattribute

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.

Codeliste 14. Die Prozedur im Modul mod_04_ReadOnly überprüft den Schreibschutz einer Datei

```
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.

Codeliste 15. Die Prozedur im Modul mod_04_ReadOnly verwendet eine Textfunktion

```
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

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
```

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.

Codeliste 16. Die Prozedur im Modul mod_05_OrdnerExplorer verwendet die Shell-Methode

```
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).

Codeliste 17. Die Prozedur im Modul mod_05_OrdnerExplorer verwendet die Shell-Methode mit dem Attribut /e

```
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]
```

Tabelle 2. 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*.

Codeliste 18. Die Prozedur im Modul mod_05_OrdnerExplorer verwendet die Shell-Methode mit Laufwerksangabe

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

Die nächste Version nutzt die *FollowHyperelink*-Methode des aktiven Workbooks. Dabei zeigt der Explorer den Verzeichnisbaum links und die Dateiansicht rechts.

Codeliste 19. Die Prozedur im Modul mod_05_OrdnerExplorer verwendet die FollowHyperlink-Methode

```
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.

Codeliste 20. Die Prozedur im Modul mod_06_NewFolder verwendet die MkDir-Methode

```
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.

Codeliste 21. Die Prozedur im Modul mod_06_NewFolder verwendet die Add-Methode

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

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

Codeliste 22. Die Prozedur im Modul mod_06_NewFolder verwendet die CreateFolder-Methode

```
Sub CreateDirectory_3()
   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 Arbeitsmappe anlegen

7.1 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.

Codeliste 23. Die Prozedur im Modul mod_07_NewWorkbooks verwendet die Add-Methode

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

Sinnvoller ist die nachfolgende *OOP*-Version.

Codeliste 24. Die Prozedur im Modul mod_07_NewWorkbooks verwendet ebenfalls die Add-Methode

```
Sub AddNewWorkbook_2()
  Dim wkbWorkbook As Workbook

  Set wkbWorkbook = Workbooks.Add

  'weitere Anweisungen

  Set wkbWorkbook = Nothing
End Sub
```

7.2 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.

Codeliste 25. Die Prozedur im Modul mod_07_NewWorkbooks verwendet die Add-Methode mit Vorlage

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

Und ebenfalls in der OOP-Version.

Codeliste 26. Die Prozedur im Modul mod_07_NewWorkbooks verwendet ebenfalls die Add-Methode mit Vorlage

```
Sub AddNewMusterBook_2()
   Dim wkbWorkbook As Workbook

   Set wkbWorkbook = Workbooks.Add("C:\Temp\Muster.xlsm")

'weitere Anweisungen

   Set wkbWorkbook = Nothing
End Sub
```

8 Arbeitsmappe öffnen

8.1 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.

Codeliste 27. Die Prozedur im Modul mod_08_WorkbookOpen verwendet die Open-Methode

```
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.

Codeliste 28. Die Prozedur im Modul mod_08_WorkbookOpen verwendet ebenfalls die Open-Methode

```
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.

Codeliste 29. Die Prozedur im Modul mod_08_WorkbookOpen öffnet die Arbeitsmappe schreibgeschützt

```
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).

Codeliste 30. Die Prozedur im Modul mod_08_WorkbookOpen nutzt die FollowHyperlink-Methode

```
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.

Codeliste 31. Die Prozedur im Modul mod_08_WorkbookOpen nutzt die Run-Methode

```
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*-Attributs 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.

Codeliste 32. Die Prozedur im Modul mod_08_WorkbookOpen nutzt ein CommandBar-Objekt

```
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.

Codeliste 33. Die Prozedur im Modul mod_08_WorkbookOpen nutzt die GetObject-Funktion

```
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.

Codeliste 34. Die Prozedur im Modul mod_08_WorkbookOpen nutzt die GetObject-Funktion in Kurzform

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

8.2 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.

9 Arbeitsmappe ist bereits geöffnet

9.1 Feststellen, ob eine Arbeitsmappe bereits geöffnet ist

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

Codeliste 35. Die Prozedur im Modul mod_09_IfOpen durchläuft alle Workbooks

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

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
```

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.

Codeliste 36. Die Prozedur im Modul mod_09_IfOpen prüft das Erstellen einer Arbeitsmappe

```
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
```

9.2 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.

Codeliste 37. Die Prozedur im Modul mod_09_IfOpen prüft den Zustand einer Arbeitsmappe

```
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

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
```

9.3 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.

Codeliste 38. Die Prozedur im Modul mod_09_IfOpen prüft den Schreibschutz einer Arbeitsmappe

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

10 Ereignisse beim Öffnen einer Arbeitsmappe

10.1 Die AutoOpen-Prozedur

Im Gegensatz zu den Ereignisprozeduren in einer Arbeitsmappe, wird beim Öffnen über die *Open*-Methode der *Workbooks*-Objektliste, die Prozedur *AutoOpen* nie ausgeführt.

Codeliste 39. Die Prozedur im Modul mod_10_Events prüft den Schreibschutz einer Arbeitsmappe

```
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.

Codeliste 40. Die Ereignis-Prozedur AutoOpenim Modul mod_10_Events

```
Sub AutoOpen()
   MsgBox "AutoOpen-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 *AutoOpen*-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 *AutoOpen*-Prozedur enthält.

Codeliste 41. Die Prozedur im Modul mod_10_Events nutzt die Methode RunAutoMacros

```
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 AutoOpen-Prozedur ausgeführt wird. Die nachfolgende Version umgeht dieses Problem durch die Verwendung einer Objekt-Variablen.

Codeliste 42. Die Prozedur im Modul mod_10_Events nutzt ebenfalls die Methode RunAutoMacros

```
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.

Codeliste 43. Die Prozedur im Modul mod_10_Events ist die beste Methode mit RunAutoMacros

```
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.

Codeliste 44. Die Prozedur im Modul mod_10_Events nutzt die Run-Methode

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

Die nächste Version fängt den Laufzeitfehler auf.

Codeliste 44. Die Prozedur im Modul mod_10_Events nutzt ebenfalls die Run-Methode

```
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
```

10.2 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 zurückgesetzt wird.

Codeliste 45. Die Prozedur im Modul mod_10_Events verwendet die EnableEvents-Methode

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

11 Darstellung der geöffneten Arbeitsmappe

11.1 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.

Codeliste 45. Die Prozedur im Modul mod_11_Visible verwendet die GetObject-Methode

```
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.

Codeliste 46. Die Prozedur im Modul mod_11_Visible verwendet ebenfalls die GetObject-Methode

```
Sub OpenWithHiddenWindow_2()
   Dim wkbWorkbook As Workbook

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

11.2 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.

Codeliste 47. Die Prozedur im Modul mod_11_Visible verwendet die Run-Methode

```
Sub OpenWithoutActivation()
   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

12 Laufzeitfehler zu Dateien

12.1 Übersicht der Laufzeitfehler beim Dateihandling

Tabelle 4. Laufzeitfehler

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	

12.2 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.

12.3 Fehlerbehandlungsroutinen für Dateizugriffe

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

Codeliste 48. Die Prozedur im Modul mod_12_Errorhandler ist ein Muster zur Fehlerbehandlung

```
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.

Codeliste 48. Die Fuktion im Modul mod_12_Errorhandler erklärt auftretende Fehler

```
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
```