VBA voor Doe het Zelvers deel 17

Handleiding van Helpmij.nl

Auteur: leofact

April 2015

Vorige aflevering

In de laatste aflevering werd besproken hoe je een werkboek deelt, wat daar de voor- en nadelen van zijn en wat de gevolgen zijn voor het gebruik van VBA. Daarnaast werd aangegeven wanneer je beter niet aan een gedeeld werkboek begint en onder welke omstandigheden het toch handig kan zijn om met een gedeeld werkboek te werken.

In deze aflevering

Dit keer komt het werken met bestanden aan bod. Naast het maken van een back-up wordt onder meer het aanmaken, kopiëren en het wissen van mappen en bestanden behandeld. Dit kan op verschillende manieren worden uitgevoerd; drie daarvan worden besproken met daarbij de voor- en nadelen. Een en ander wordt niet uitputtend behandeld, maar voldoende om zelf verder mee aan de slag te kunnen. Dat gaat heel goed met de informatie uit de aangeboden links. De code is ook voor deze aflevering uitgewerkt in een werkboek. Dit werkboek kan hier worden gedownload.

Bibliotheken

Bestanden kunnen met behulp van objecten uit verschillende bibliotheken worden gemanipuleerd. In dit artikel worden de objecten van zowel de VBA-bibliotheek als die van de Microsoft Scripting Runtime-bibliotheek behandeld.

De objecten van de eigen VBA-bibliotheek zijn vrij beperkt in functionaliteit. De basale bestandsbewerkingen zijn er echter goed mee uit te voeren. Een belangrijk voordeel is dat er geen aparte verwijzing voor gelegd hoeft te worden. Bovendien werken deze objecten ook onder Apple's OS X. Dit kan van de andere methodes niet worden gezegd. Omdat OS X een geheel ander bestandssysteem heeft, werkt lang niet alles hetzelfde als onder Windows. Daar komt nog bij dat het mis gaat als er met bestandsnamen wordt gewerkt die langer zijn dan 32 tekens. Deze beperkingen zijn te omzeilen door wat Applescript in te zetten, maar dat valt buiten het bereik van dit artikel.

Naast de VBA-Objecten kan de MS Scripting Runtime-bibliotheek worden gebruikt. Deze is uitgebreider, maar werkt alleen onder Windows. Er moet een verwijzing worden gelegd om bij het programmeren IntelliSense te kunnen gebruiken (Early binding).

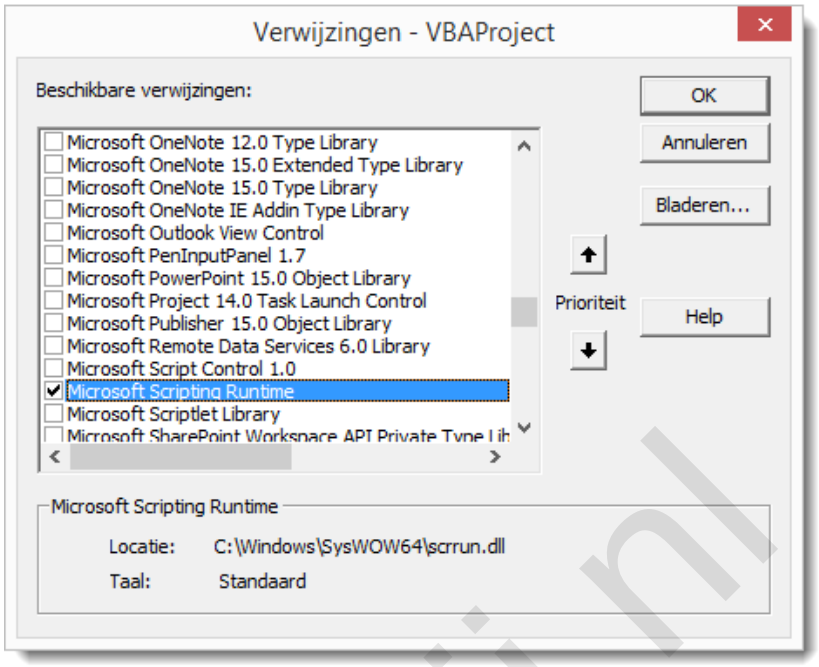
De Shell functie is een andere methode om bestanden te manipuleren. Met deze functie kunnen externe programma's worden opgestart en Windows commando's worden uitgevoerd, vergelijkbaar met het uitvoeren-venster van het Windows Startmenu (*Windowstoets* + *R*).

Meer informatie hierover is te vinden in de volgende links:

- snb - vba; VBA bestanden
- MSDN; Introduction to file and folder operations for the Apple Mac
- MS Office; Shell function
- Ron de Bruin; voorbeelden

Verwijzing

De eerste stap om een bibliotheek in Early Binding te kunnen gebruiken is om de verwijzing te leggen in het Verwijzingenvenster onder menu **Extra** van de VB-Editor. Zie hieronder:



Wanneer het werkboek ook aan andere gebruikers beschikbaar gesteld gaat worden is het wel zaak om de Early Binding in een Late Binding om te zetten. Dit om te voorkomen dat iedere gebuiker apart de verwijzing moeten leggen. Zie deel 13 voor meer informatie.

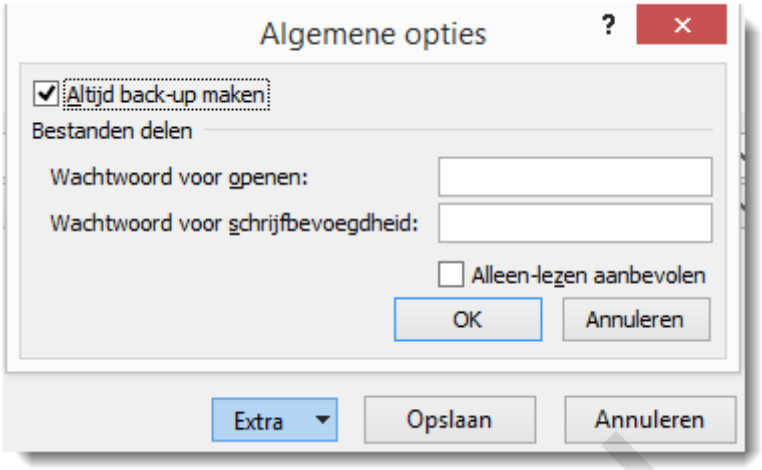
De VBA-bibliotheek bevat een aantal simpel in te zetten objecten voor basis-bestandsbeheer. In de volgende routine worden voorbeelden geven van vier veelgebruikte bestandsbewerkingen:

Tip: Open de map waarin je de bijlage hebt opgeslagen zichtbaar in de verkenner. Op deze manier zie je wat er wordt uitgevoerd door de code in de bijlage.

Er kan natuurlijk nog veel meer met VBA. Een aantal van de mogelijke bestandsbewerkingen worden daarom op verschillende manieren uitgewerkt.

Back-up

Het maken van een back-up is altijd een goed idee. Er zijn verschillende strategieën te bedenken om dit te automatiseren. De meest eenvoudige is het gebruikmaken van de ingebouwde back-upfunctie van Excel. Deze wordt geactiveerd door bij het opslaan in het venster **Opslaan als** voor **Extra** > **Algemene Opties** > **Altijd back-up maken** aan te vinken:



Het bestand wordt dan opgeslagen als “back-up van werkboeknaam.xlk”. Er wordt echter slechts één back-up bewaard en er is weinig controle over de opslaglocatie. Het back-upbestand wordt standaard in dezelfde map als het werkboek opgeslagen. Met behulp van VBA is er op eenvoudige wijze veel meer mogelijk. Dat kan bijvoorbeeld met de methode SaveCopyAs:

```
ActiveWorkbook.SaveCopyAs Strings.Format(Now, "yyyy-mm-dd") _
 & "_Facturen_Backup.xls"
```

De bijzonderheid van SaveCopyAs is dat het, in tegenstelling tot SaveAs, standaard een al bestaand bestand overschrijft. Gebruik je in de naam de dag als kleinste eenheid, dan wordt er één back-up per dag bewaard. Een latere back-up gemaakt op diezelfde dag overschrijft automatisch eerder gemaakte back-upbestanden. Er kan ook worden gekozen om het uur, of zelfs de seconde als kleinste eenheid in de naam op te nemen:

```
ActiveWorkbook.SaveCopyAs Strings.Format(Now, "yyyy-mm-dd_hhnnss") _
& "Facturen_Backup.xls"
```

Er worden dan mogelijk wel veel back-upbestanden aangemaakt, met als voor de hand liggend nadeel dat er veel opslagruimte in beslag wordt genomen.

N.B. Er wordt steeds gebruik gemaakt van de extensie .xls voor een Excel 2003 werkboek. Vanaf Excel 2007 ligt het gebruik van .xlsm meer voor de hand bij een werkboek met macro’s, of .xlsx voor een werkboek zonder macro’s.

Verderop volgt een uitgewerkt voorbeeld van een back-uproutine waarbij onder meer ook de opslaglocatie geregeld wordt.

Mappen

Een map is eenvoudig aan te maken op de volgende wijze:

```
MkDir (ActiveWorkbook.Path & "Bestanden")
```

Hiermee wordt de map Bestanden aangemaakt in de map waarin het actieve werkboek is opgeslagen. Een lege map heeft echter weinig nut, we zorgen daarom dat er bestanden in worden opgeslagen. We

maken daarvoor een apart werkboek van ieder werkblad uit de bijlage met de volgende routine:

```
(Algemeen)                                    SlaWerkBladenOp
'---------------------------------------------------------------------------
' Procedure : SlaWerkBladenOp 20-3-2015
' Doel : Sla alle weeerkbalden op als bestand
'---------------------------------------------------------------------------
'
Sub SlaWerkBladenOp()
Dim Sh As Worksheet
Dim sPad As String
Application.ScreenUpdating = False
   'Stel pad samen.
    sPad = ActiveWorkbook.Path & "\Bestanden\"
   'Bestaat de map, wis deze dan.
   With New Scripting.FileSystemObject
        If .FolderExists(sPad) Then
                'Wis de bestanden als de map bestaat.
                Kill sPad & "*"
            Else
                'Maak de map
                MkDir (sPad)
            End If
    End With
        'doorloop de werkbladen
        For Each Sh In ActiveWorkbook.Worksheets
            'Maak een werkboek van ieder sheet.
            Sh.Copy
                'sla het op in de map Bestanden
                ActiveWorkbook.SaveAs sPad & Sh.Name
            'Sluit het opgeslagen werkboek
            ActiveWorkbook.Close
        'Doorloop alle werkbladen
        Next Sh
End Sub
```

De map is nu gevuld. Het zijn echter allemaal bestanden die verder niet meer voor je van nut zijn na deze demo. Gelukkig is een map eenvoudig te wissen.

RmDir (ActiveWorkbook.Path & “Bestanden”)

Wanneer je dit commando direct uitvoert merk je dat er een foutmelding volgt:

De map moet eerst leeg worden gemaakt voordat deze verwijderd kan worden. Ook dit gaat eenvoudig:

Kill ActiveWorkbook.Path & "Bestanden" & ".xls*"*

Hierbij wordt de wildcard “*” gebruikt. De wildcard vervangt ieder leesteken, of iedere reeks van leestekens. In het voorbeeld worden alle werkboeken, met een willekeurige naam met in de extensie .xls gewist. Dus ook .xlsm of .xlsx bestanden, enzovoorts. Was “*.*” gebruikt, dan werd ieder bestand in de map met één eenvoudig commando naar de prullenbak geveegd. Nu de map leeg is gemaakt

zal het verwijderen van de map met het eerder gegeven commando geen foutmelding meer opleveren.

N.B. Kill en verwante opdrachten kunnen onverwachte gevolgen hebben als zij op systeemmappen (zoals bijv. System32) worden losgelaten. Gelukkig zijn systeembestanden veelal tegen onbedoeld wissen beschermd. Voorzichtigheid is echter geboden.

Bestandsinformatie kan met Objecten uit de VBA-bibliotheek op de volgende manier opgevraagd worden:

(Algemeen) | CheckBestand

```
'---------------------------------------------------------------
' Procedure : CheckBestand 18-3-2015
' Doel : Geeef info met VBA statements
'---------------------------------------------------------------
'
Sub CheckBestand()
With ActiveWorkbook
    Call MsgBox("Het pad van het werkboek: " _
       & vbCrLf & .Path _
        & vbCrLf & "De volledige naam met pad:" _
         & vbCrLf & .FullName _
        & vbCrLf & "Met het atribuut: " & GetAttr(.Name) _
       & vbCrLf & "En de bestandsgrootte= " & FileLen(.Name) _
      , vbExclamation, "CheckBestand")
End With
End Sub
```

Dit kan ook met behulp van het fileSystemObjectModel uit de Scripting Runtime-bibliotheek. Dat werkt niet altijd eenvoudiger, maar de mogelijkheden zijn wel uitgebreider:

(Algemeen) | DemoFSO_EB

```
'---------------------------------------------------------------
' Procedure : DemoFSO 18-3-2015
' Doel : Demonstreert de mogelijkheden van FSO (Earlybind)
'---------------------------------------------------------------
'
Sub DemoFSO_EB()

Dim FSO As New FileSystemObject
    With FSO.GetFile(ActiveWorkbook.FullName)
        Call MsgBox(.Name & vbCrLf _
          & "In map: " & .ParentFolder & vbCrLf _
          & "gemaakt: " & .DateCreated & vbCrLf _
           & "Last Accessed: " & .DateLastAccessed & vbCrLf _
            & "laatst gewijzigd: " & .DateLastModified & vbCrLf _
             & "Bestandstype: " & .Type & vbCrLf _
            & "Bestandgrootte: " & .Size & vbCrLf _
           & "Korte naam (DOS): " & .ShortName & vbCrLf _
          & "Kort pad (DOS): " & .ShortPath & vbCrLf _
         & "Attribuut: " & .Attributes _
        , vbInformation, "File Information")
    End With
End Sub
```

Dit voorbeeld is geschreven in Early Binding met de prettige ondersteuning van IntelliSense. Wanneer het werkboek aan een andere gebruiker beschikbaar wordt gesteld is het leggen van een Late Binding

natuurlijk beter; de gebruiker hoeft zich dan niet om een verwijzing te bekommeren:

(Algemeen) DemoFSO_LB

```
'---------------------------------------------------------------------
' Procedure : DemoFSO 18-3-2015
' Doel : Demonstreert de mogelijkheden van FSO (Late Binding)
'---------------------------------------------------------------------
'
Sub DemoFSO_LB()
Dim FSO
    Set FSO = CreateObject("Scripting.FileSystemObject")
        With FSO.GetFile(ActiveWorkbook.FullName)
        Call MsgBox(.Name & vbCrLf _
         & "In map: " & .ParentFolder & vbCrLf _
          & "gemaakt: " & .DateCreated & vbCrLf _
           & "Last Accessed: " & .DateLastAccessed & vbCrLf _
            & "laatst gewijzigd: " & .DateLastModified & vbCrLf _
             & "Bestandstype: " & .Type & vbCrLf _
            & "Bestandgrootte: " & .Size & vbCrLf _
           & "Korte naam (DOS): " & .ShortName & vbCrLf _
          & "Kort pad (DOS): " & .ShortPath & vbCrLf _
         & "Attribuut: " & .Attributes _
        , vbInformation, "File Information")
    End With
End Sub
```

Een bestandseigenschap is op meerdere manieren op te vragen. Hier volgt nog een voorbeeld met de Shell functie:

(Algemeen) Shell_Demo

```
'---------------------------------------------------------------------
' Procedure : Shell_Demo 19-3-2015
' Doel : Simpel demo met gebruik van shell
'        Haalt het bestandstype van het werkboek op
'---------------------------------------------------------------------
'
Sub Shell_Demo()
Dim vResultaat As Variant
    With CreateObject("shell.application") _
            .Namespace(ActiveWorkbook.Path & "\")
            vResultaat = .getdetailsof _
            (.Items.Item(ActiveWorkbook.Name), 2)
    End With
MsgBox vResultaat
End Sub
```

Om het verschil tussen het gebruik van FSO en de eigen VBA-objecten te demonstreren volgt hier nog een voorbeeld om bestanden in de Windows System32 map te tellen. Met de VBA-objecten is het noodzakelijk om een lus te creëren die stap voor stap alle bestanden doorloopt:

```
(Algemeen)                                   BestandenTeller_System32
'---------------------------------------------------------------------
' Procedure : BestandenTeller_System32 18-3-2015
' Doel : Tel de bestanden van System32
'---------------------------------------------------------------------
'
Sub BestandenTeller_System32()
Dim sBestand As String
Dim lCount As Long
    sBestand = Dir("C:\Windows\System32\*.*")
        While (sBestand <> "")
            sBestand = Dir
            lCount = lCount + 1
        Wend
    MsgBox lCount
End Sub
```

Met behulp van FSO kan dat in één keer. Er is geen lus nodig:

```
(Algemeen)                                   BestandenTellerFSO_System32
' Procedure : BestandenTellerFSO 18-3-2015
' Doel : Telt het aantal bestanden in de System32 map
'---------------------------------------------------------------------
'
Sub BestandenTellerFSO_System32()
Dim oMap As Object
With CreateObject("Scripting.FileSystemObject")
    Set oMap = .GetFolder("C:\Windows\System32\")
MsgBox oMap.Files.Count
End Sub
```

Het verschil in uitkomst is te verklaren doordat in de VBA-lus alleen de bestanden worden geteld en met FSO worden ook de submappen meegeteld.

De eerst gegeven lus is wel eenvoudig om te bouwen tot een generator van een bestandenlijst. In dit voorbeeld van de bestanden uit System32-map. Dat zijn er nogal wat, de lijst wordt daarom beperkt tot de bestanden die beginnen met de letter “e”:

```
(Algemeen)                                   BestandsNamen
'---------------------------------------------------------------------
' Procedure : BestandsNamen 21-3-2015
' Doel : Haal de bestandsnamen op die beginnen met de letter "e"
'---------------------------------------------------------------------
'
Sub BestandsNamen()
Dim sBestand As String
Dim sTotaal As String
   sBestand = Dir("C:\Windows\System32\*.*")
   While (sBestand <> "")
     sBestand = Dir
     If Left(sBestand, 1) = "e" Then _
        sTotaal = sTotaal & sBestand & vbCrLf
  Wend
  MsgBox sTotaal
End Sub
```

Het resultaat wordt hier gegeven als Messagebox. Er kan natuurlijk ook worden gekozen om het resultaat in een werkbalk of in een arrayvariabele te verzamelen. De uitwerking hiervan wordt aan de eigen fantasie over gelaten.

N.B. De locatie van de System32-map is afhankelijk van de eigen Windows-installatie. Pas de code daarom aan als dat nodig is.

Het commando RmDir is alleen in staat tot het verwijderen van lege mappen. Met behulp van FSO is het echter mogelijk de map te verwijderen, ook als deze bestanden of submappen bevat. Zelfs als het mapattribuut op alleen lezen staat wordt deze nog verwijderd:

(Algemeen) | RuimOp

```
' Procedure : RuimOp 21-3-2015
' Doel :Ruim de map Bestanden op.
'---------------------------------------------------------------
'
Sub RuimOp()
Dim sPad As String
   'Stel pad samen.
    sPad = ActiveWorkbook.Path & "\Bestanden"
   'Bestaat de map, wis deze dan met inhoud.
   With New Scripting.FileSystemObject
        If .FolderExists(sPad) = True Then
                'Verwijder Map
                .DeleteFolder sPad, True
           Else
               MsgBox "Map bestaat niet"
        End If
    End With
End Sub
```

Als laatste voorbeeld wordt nog een routine gegeven waarmee een back-up van een werkboek kan worden gemaakt in een aangemaakte back-upmap. Door de routine te starten vanuit het Workbook_BeforeSave event, of het Workbook_BeforeClose event kan dit een geautomatiseerd proces worden:

(Algemeen) | Backup

```
'---------------------------------------------------------------
' Author    : Leo
' Date      : 21-02-2015 13:26
' Purpose   : Maakt automatisch een backup bij het afsluiten.
'---------------------------------------------------------------

Sub Backup()
Dim sDatum As String
Dim sPad  As String
Dim sMessage As String
'Geopend als alleen lezen? Dan geen back-up.
If ActiveWorkbook.ReadOnly = True Then Exit Sub

'Stel de string voor de Messagebox in.
sMessage = "De back-up map kan niet worden gevonden." _
    & vbCrLf & "Mogelijk is deze gewist." _
     & vbCrLf & "" _
      & vbCrLf & "Klik Ok om een nieuwe back-up map aan te maken." _
       & vbCrLf & "Klik Cancel om door te gaan zonder nieuwe map." _
        & vbCrLf & "Excel maakt dan geen back-up aan."
'maak datum en pad string.
    sDatum = Strings.Format(Now, "yyyy-mm-dd")
        sPad = ActiveWorkbook.Path & "\Back-up\"
'Controleer of de Back-up map bestaat
With CreateObject("Scripting.FileSystemObject")
    'Geen map? biedt aan deze te maken.
    If Not .FolderExists(sPad) Then
        'Select Case met resultaal Messagebox.
        Select Case MsgBox(sMessage _
                 , vbOKCancel Or vbExclamation Or _
                    vbDefaultButton1, "Back-up map mist")
                Case vbCancel
                        'Geen map aanmaken? Verlaat Sub.
                        Exit Sub
                Case vbOK
                        'Maak de map aan.
                        MkDir sPad
                End Select
        End If
End With
'Toon form Modeless (code loopt door) ter informatie
UfOpslaan.Show vbModeless 'Userform zichtbaar
    'Nodig om de inhoud van een modeless form zichtbaar te maken.
    UfOpslaan.Repaint
        'Sla op is de back-up map
        ActiveWorkbook.SaveCopyAs Filename:= _
            sPad & sDatum & "_Facturen_BU.xlsm"
                'verberg het userform.
                Unload UfOpslaan 'Userform sluiten
End Sub
```

Hiermee komt ook deze april-aflevering weer tot een eind.

Samenvatting

In deze aflevering kwam het werken met bestanden aan bod. Naast het maken van een back-up werd onder meer het aanmaken, kopiëren en het wissen van mappen en bestanden behandeld. Dit kan op verschillende manieren worden uitgevoerd; drie daarvan werden besproken met daarbij de voor- en nadelen. Een en ander werd zo besproken dat er een basis is gelegd waarop zelf verder geborduurd kan worden. Dit kan ondersteund worden met de informatie uit de aangeboden links. De code werd ook voor deze aflevering uitgewerkt in een bijlage.

Deel 18

In de volgende aflevering wordt aandacht geschonken aan werkboekbeveiliging. Dit om de inhoud te beschermen tegen ongewenste pottenkijkers en om de userinterface en opmaak intact te houden tijdens allerlei vormen van gebruik van het werkboek.