How to get the names of all the folders in the folder tree, starting from a specified folder

Terms of use Disclaimer

For instance, in order put the results into a document, or to print them

Article contributed by Thomas Gahler

The following procedures read the entire folder tree, starting from any folder you specify, into an array, and put the results into a document; or you could have it print the results, or whatever you want it to do with them.

Copy the functions below into a module in your VB Editor (press **Alt+F11**). You will need to delete the horizontal lines that have been used here to separate the functions.

You can call the functions with the following macro (or modify it as required)):

```
Sub Demo()

Dim FoldersArray As Variant
Dim i As Integer

'Read all subfolders of the specified folder into an array
'by calling the funcGetSubfolders function
FoldersArray = funcGetSubfolders("C:\Windows")

'Put the results (the array values) into the current document if it is blank,
'or else into a new document
If Len(ActiveDocument.Range.Text) > 1 Then
   Documents.Add
End If
For i = LBound(FoldersArray) To UBound(FoldersArray)
   ActiveDocument.Range.InsertAfter FoldersArray(i) & vbCr
Next i
ActiveDocument.Saved = True

End Sub
```

Or you could allow the user to type in a path for themselves, by displaying an input box or a UserForm. For instance, instead of the line:

```
FoldersArray = funcGetSubfolders("C:\Windows")
```

you could use:

```
Dim FolderToRead As String
Do
   FolderToRead = InputBox("Type path to folder you want to check")
   If Len(FolderToRead) = 0 Then
      Exit Sub
   End If
   If Len(Dir$(FolderToRead, vbDirectory)) = 0 Then
      MsgBox "Invalid folder name; please try again, or press Cancel to quit"
   End If
Loop Until Len(Dir$(FolderToRead, vbDirectory)) > 0

FoldersArray = funcGetSubfolders(FolderToRead)
```

Or see **How to allow the user to browse to and select a folder**, if you want to be *really* swish.

These are the functions you need. See also the Notes at the end.

```
Public Function funcGetSubfolders(FolderToRead As String) As Variant
```

```
'This function uses a string as a parameter and not an array.
'It translates this string to an array and then starts the main function, 'funcGetAllSubfolders'

Dim AllSubFolders(0) As Variant

On Error Resume Next
System.Cursor = wdCursorWait

'Add a backslash to  the end of the path, if not there already
If (Right$(FolderToRead, 1) <> "\") Then
   FolderToRead = FolderToRead & "\"
End If

'Set the path as the first entry in the array and pas the array to the main function
AllSubFolders(0) = FolderToRead
funcGetSubfolders = funcGetAllSubfolders(AllSubFolders)

System.Cursor = wdCursorNormal
StatusBar = ""
On Error GoTo 0

End Function
```

```
Private Function funcGetAllSubfolders(AllSubFoldersArray As Variant) As Variant

'This is a recursive function, that is, it keeps calling itself -
'which makes it a nightmare to step through!

Dim Counter As Integer

'The following string will contain the path of the folder which is currently being looked in
Dim CurFolderName As String

'The following string will contain the current value returned by Dir$().
Dim SubFolderName As String

'The following array will contain of the subfolders (if any) of 'CurFolderName'
Dim SubFolderList() As String

On Error Resume Next

'Get the last value we put into the AllSubFoldersArray Array variant,
'and convert it to a string so that we can assign it to the string
'variable CurFolderName
CurFolderName = CStr(AllSubFoldersArray(UBound(AllSubFoldersArray)))

'Read all subfolders of 'CurFolderName' and add them to 'SubFolderList'.
ReDim SubFolderList(0)
SubFolderName = Dir$(CurFolderName, vbDirectory)
Do While Len(SubFolderName) <> 0
   'Ignore the current directory and the encompassing directory.
   If SubFolderName <> "." And SubFolderName <> ".." Then
      'Unfortunately, calling Dir with the vbDirectory attribute
      'does not continually return subdirectories (only the first time);
      'so you have to use the GetAttr function (which is covered in Help)
      'to test, each time, that this is a folder and not a file
      If (GetAttr(CurFolderName & SubFolderName) _
            And vbDirectory) = vbDirectory Then
         'Up the array size by one
         ReDim Preserve SubFolderList(UBound(SubFolderList) + 1)
         'Add the new folder to the array
         SubFolderList(UBound(SubFolderList)) = SubFolderName
         StatusBar = "Reading Subfolders... (" _
               & CurFolderName & ": -> " & SubFolderName & ")"
      End If
   End If
   'Get the next directory
   SubFolderName = Dir$()
Loop

'Sort the list with the subfolders.
If UBound(SubFolderList) > 0 Then
   WordBasic.SortArray SubFolderList()
End If

'Now get all the subfolders of the current folder, then all the subfolders
'of each of those subfolders, and so on, up the directory tree,
'until there are no more subfolders. By recursively
'(repeatedly applying the procedure to successive results)
'calling the current function.

'If the current folder contains no subfolders, the following For .. Next loop gets skipped
```

```
For Counter = 1 To UBound(SubFolderList)

   'Up the size of the AllSubFoldersArray array by one
   ReDim Preserve AllSubFoldersArray(UBound(AllSubFoldersArray) + 1)

   'Set the next item in the AllSubFoldersArray to be
   'the next subfolder of the current folder
   AllSubFoldersArray(UBound(AllSubFoldersArray)) = _
         CurFolderName & "\" & SubFolderList(Counter) & "\"

   'Now run the this function recursively on that subfolder,
   'to get its subfolders, if it has any
   AllSubFoldersArray = funcGetAllSubfolders(AllSubFoldersArray)
Next Counter

'Set the complete directory structure as the function's  return value.
funcGetAllSubfolders = AllSubFoldersArray
On Error GoTo 0

End Function
```

Notes

1. The funcGetAllSubfolders function is recursive; (recursive means repeatedly applying the same procedure to successive results). So it reads the names of all the folders in the folder you specify; then for each of these folders in turn, it calls itself in order to read their subfolders, and while it's in the process of doing that, it repeatedly calls itself in order to read the subfolders of those subfolders, and so on. Therefore, many instances of the function run simultaneously, and in each instance of the function, all the variables have different values from every other instance. Somehow, the code manages to keep track of all this, but it's more or less impossible to keep track of them when stepping through the code; which makes recursive procedures a nightmare to debug Luckily, you don't need to, because the function works!

   Partly for his reason, and partly because recursive procedures can easily run out of memory, if not very well coded they are usually best avoided. But in this case, to do it any other way would be much more complicated, so it's justified. And the macro runs very fast!

2. Usually it's best to define array variables as a data type other than a Variant (such as a String), as this can save a lot of memory. However, when passing arrays to functions, it's much simpler to define them as a Variant.