How to list files in folder?

Authored by

stats writer

November 18, 2025

Introduction to File Listing in VBA

Managing files and directories is a fundamental requirement for advanced spreadsheet automation. When working within Excel, the built-in functionality of VBA (Visual Basic for Applications) provides powerful tools to interact directly with the operating system's file structure. Specifically, listing the contents of a particular folder--whether to inventory datasets, process files in batches, or simply confirm existence--is a common task for developers and power users. This guide details two robust methods leveraging the FileSystemObject to achieve precise file listing within your VBA projects.

The ability to programmatically access and manipulate files opens up vast possibilities for automating repetitive data aggregation and reporting tasks. Instead of manually navigating folders and copying file names, VBA allows you to execute these operations swiftly and reliably. The core concept revolves around utilizing the external library known as the Microsoft Scripting Runtime, which grants access to the operating system's file management capabilities. Understanding how to instantiate and interact with objects from this library is crucial for any serious VBA developer aiming for folder automation.

We will explore two distinct approaches. The first method focuses on obtaining a complete inventory of every file residing within a specified folder path, regardless of its file type or extension. The second, more specialized technique demonstrates how to implement conditional logic within the loop to filter the results, allowing you to list only files matching specific criteria, such as all documents bearing the .xlsx extension. Both methods employ the same foundational objects but differ in their iterative logic.

Understanding the Scripting.FileSystemObject

The foundation of modern VBA file management lies in the FileSystemObject (FSO). The FSO is a key component of the Scripting Runtime library, designed specifically to provide an object-oriented interface for manipulating folders and files. By utilizing the FSO, developers can avoid complex Win32 API calls and instead interact with files using intuitive methods like `CreateObject`, `GetFolder`, and `Files` enumeration. This provides a clean, robust way to handle file system operations directly from your spreadsheet application.

To start interacting with the file system, you must first create an instance of the FileSystemObject. This is typically achieved using the `CreateObject("Scripting.FileSystemObject")` command, which dynamically binds the necessary libraries at runtime. Once the FSO object is instantiated, you can then use its methods--such as `GetFolder`--to obtain a reference to the specific directory you wish to analyze. This folder object then exposes the collection of files contained within it, which is essential for our listing process.

For listing files, the sequence of operations is standardized: first, declare your necessary variables (objects for FSO, the folder, and individual files, plus an integer counter); second, initialize the FSO object; third, define the target folder path and retrieve the folder object; and finally, iterate through the folder's `Files` collection using a `For Each...Next` loop. This iterative process allows us to systematically extract data points, such as the file name, and record them onto the Excel worksheet.

Method 1: Comprehensive Listing of All Folder Contents

The first and most straightforward method involves enumerating every single file present in the target directory. This is useful when you need a full audit of the folder or when you plan to process all files irrespective of their format. The core of this method relies on the `oFolder.Files` collection, which automatically includes all items directly within the specified path. We use a counter variable, conventionally named `i`, to manage the row placement within the Excel sheet as we loop through the collection.

The following macro demonstrates the necessary variable declarations and the complete structure required to execute this comprehensive listing. Note the use of `Dim oFSO As Object` and `Set oFSO = CreateObject("Scripting.FileSystemObject")`, which are boilerplate commands essential for initializing the file operations. The path specified, `"C:UsersbobDocumentscurrent_data"`, must be updated to reflect the actual directory you intend to scan.

The loop structure is simple: for every file found within the `oFolder.Files` collection, the code retrieves the file's `.Name` property and places it in the next available cell in column A (using `Cells(i + 1, 1)`). The inclusion of `i = i + 1` ensures that subsequent file names are listed sequentially down the column, maintaining a clean and organized output within your Excel workbook.

Code for Method 1: List All Files in Folder

This code snippet provides the complete VBA procedure for listing all files.

Sub ListFiles()

Dim i As Integer
Dim oFSO As Object
Dim oFolder As Object
Dim objFile As Object

Set oFSO = CreateObject("Scripting.FileSystemObject")

```
Set oFolder = oFSO.GetFolder("C:UsersbobDocumentscurrent_data")

For Each objFile In oFolder.Files
Cells(i + 1, 1) = objFile.Name
i = i + 1
Next objFile

End Sub
```

Method 2: Selective Filtering by File Extension

In many automation scenarios, you may only be interested in files of a specific type--for instance, only CSV files, only text documents, or only .xlsx workbooks. Method 2 builds upon the foundational code of Method 1 but incorporates an essential conditional statement (an `If...Then` block) inside the loop to filter the results. This significantly enhances the utility of the macro, ensuring that your output list is streamlined and relevant to the task at hand.

To implement filtering, we examine the file's name using string manipulation functions available in VBA. The most common technique is to use the `Right` function to check the last characters of the file name, which typically correspond to the file extension. For instance, to filter for .xlsx files, we check if the last four characters of `objFile.Name` are equal to `"xlsx"`. This check must be precise; for example, if we were looking for CSV files, we would check the last three characters for `"csv"`.

The key line of code here is `If Right(objFile.Name, 4) = "xlsx" Then`. Only when this condition evaluates to `True` does the code proceed to write the file name to the Excel sheet and increment the counter `i`. Files that do not meet the extension criterion are simply ignored, allowing the loop to continue to the next item without generating an entry in the output list. This provides a highly efficient way to target specific data types within large directories.

Code for Method 2: List Only .xlsx Files in Folder

This variation includes the conditional filtering logic, ensuring only files matching the specified extension are processed and listed.

```
Sub ListFiles()

Dim i As Integer
Dim oFSO As Object
Dim oFolder As Object
Dim objFile As Object
```

```
Set oFSO = CreateObject("Scripting.FileSystemObject")

Set oFolder = oFSO.GetFolder("C:UsersbobDocumentscurrent_data")

For Each objFile In oFolder.Files
If Right(objFile.Name, 4) = "xlsx" Then
Cells(i + 1, 1) = objFile.Name
i = i + 1
End If
Next objFile

End Sub
```

Setting the Stage: The Demonstration Environment

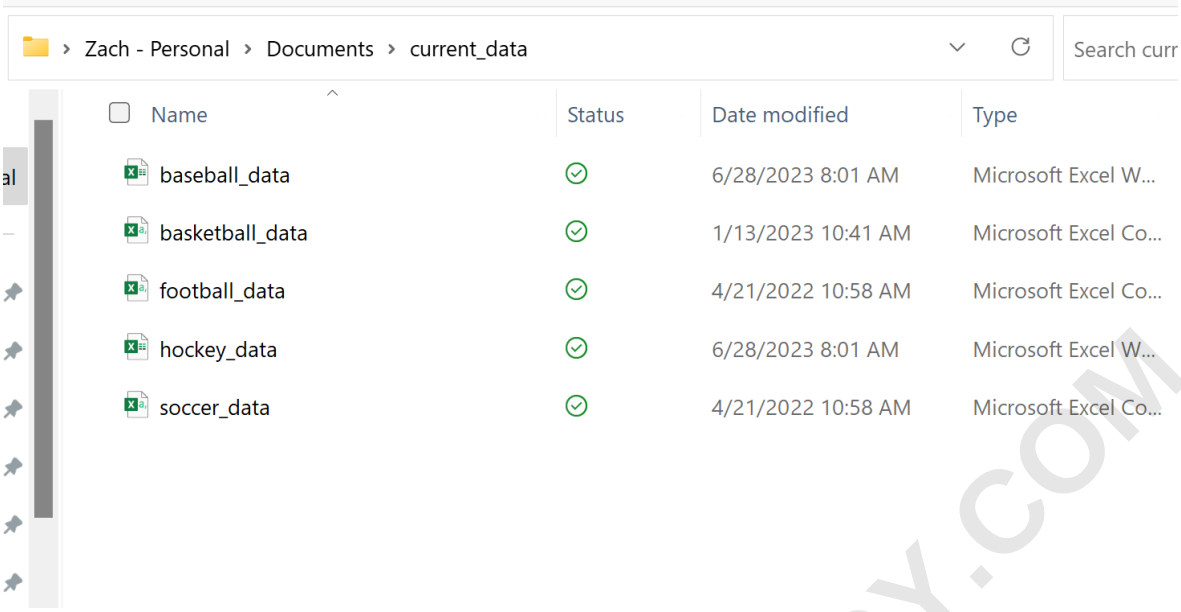
To clearly illustrate how these two methods function in a real-world context, we will use a specific folder structure for our practical examples. The file path chosen for this demonstration is a common structure found in Windows environments, representing a typical working directory for a user named 'bob'. It is imperative that when adapting this code for your own purposes, you replace this placeholder path with the actual location of your files.

The designated folder path for our demonstration is:

C:UsersbobDocumentscurrent_data

Crucially, this folder contains a mixture of files, simulating a real-world scenario where data often comes in various formats. Specifically, the folder includes two files with the .xlsx extension (standard Microsoft Excel workbooks) and three files with the **.csv** extension (Comma Separated Values), totaling five files that the FileSystemObject will iterate through.

Visualizing the contents of this folder provides a baseline expectation for the output of our VBA scripts. This image confirms the presence and naming conventions of all five files, serving as a reference point for comparing the results generated by the unfiltered and filtered listing methods.



Practical Demonstration 1: Listing Every File

Using the code from Method 1, we execute the macro designed to list every entry found within the `current_data` folder, regardless of its file extension. This approach serves to quickly inventory all available resources in the directory. The FileSystemObject treats all files equally in this scenario, collecting their names into the `Files` collection for processing.

We deploy the following exact code into the VBA Editor (Alt + F11), ensuring the path is correctly set.

Sub ListFiles()

Dim i As Integer
Dim oFSO As Object
Dim oFolder As Object
Dim objFile As Object

Set oFSO = CreateObject("Scripting.FileSystemObject")

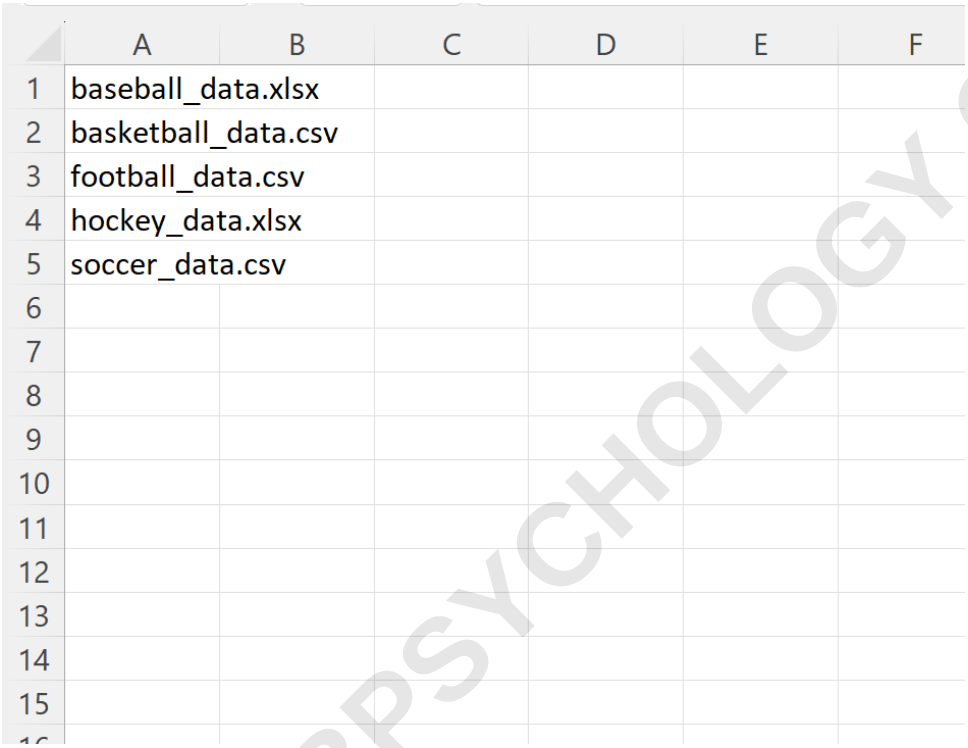
Set oFolder = oFSO.GetFolder("C:UsersbobDocumentscurrent_data")

For Each objFile In oFolder.Files
Cells(i + 1, 1) = objFile.Name
i = i + 1
Next objFile

End Sub

Upon successful execution of the subroutine, the results are populated starting in cell A1 of the active worksheet. As expected, since no filtering condition was applied, all five files--both the .xlsx and the **.csv** files--are displayed in a complete list. This confirms the successful iteration through the entire `Files` collection exposed by the folder object.

The resulting output provides a comprehensive list of all contents:



	A	B	C	D	E	F
1	baseball_data.xlsx					
2	basketball_data.csv					
3	football_data.csv					
4	hockey_data.xlsx					
5	soccer_data.csv					
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

We can clearly observe that the names of all files in the folder (regardless of their extension) are successfully listed in column A of our Excel sheet, fulfilling the objective of comprehensive directory listing.

Practical Demonstration 2: Filtering for Specific Data Types

In contrast to the previous example, this demonstration showcases the power of selective listing. We implement the code from Method 2, which incorporates the `If Right(objFile.Name, 4) = "xlsx" Then` condition. Our goal is to list only those files that are official Excel workbooks, effectively ignoring the three **.csv** files present in the same directory.

The following VBA code snippet is deployed, utilizing the `Right` function to perform the necessary file extension check during the iteration process.

Sub ListFiles()

Dim i As Integer
Dim oFSO As Object
Dim oFolder As Object
Dim objFile As Object

Set oFSO = CreateObject("Scripting.FileSystemObject")

Set oFolder = oFSO.GetFolder("C:UsersbobDocumentscurrent_data")

For Each objFile In oFolder.Files
If Right(objFile.Name, 4) = "xlsx" Then
Cells(i + 1, 1) = objFile.Name
i = i + 1
End If
Next objFile

End Sub

After running this refined macro, the output demonstrates the successful application of the filter. Instead of five entries, the list only contains the two files that satisfy the .xlsx extension requirement. The **.csv** files are correctly ignored, proving the effectiveness of using the `If...Then` structure combined with string functions for highly targeted file handling.

The resultant list, showing only the filtered files:

	A	B	C	D	E	F
1	baseball_data.xlsx					
2	hockey_data.xlsx					
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						

We can confirm that only the names of files possessing the **.xlsx** extension are now systematically listed in column A of the Excel sheet, providing a clean and targeted inventory for subsequent data processing tasks.

Conclusion and Further Reading

The utilization of the Scripting.FileSystemObject in VBA provides a powerful and reliable mechanism for interacting with the operating system's directories. Whether you require a full directory audit (Method 1) or a filtered list based on specific file extensions (Method 2), these techniques form the bedrock of sophisticated file automation within Excel. Mastery of these objects allows for the creation of scalable and efficient solutions that significantly reduce manual overhead in data management workflows.

For users looking to expand their knowledge of file system manipulation, the next logical step often involves not just listing files, but also managing the folder structure itself.

How to Create Folders Using VBA