How to Easily Open Text Files with VBA: A Step-by-Step Guide

Authored by

stats writer

November 19, 2025

Introduction to File Handling in VBA

File handling is a fundamental requirement in automated data processing using VBA (Visual Basic for Applications). When working within Microsoft Office applications like Excel, the ability to interact with external text files--whether for reading configuration settings, importing large datasets, or exporting reports--is indispensable. While there are several ways to manipulate files using VBA, this guide focuses on two primary methods: the legacy **Open Statement** and the more robust **FileSystemObject** model.

The core concept involves establishing a connection between your running macro and the external file on the disk. This connection requires specifying the file path, the intended mode of access (such as reading or writing), and assigning a unique file number for tracking purposes. Proper file management demands that every opened file must eventually be closed to release system resources and prevent data corruption.

Method 1: Utilizing the Legacy Open Statement

The traditional and simplest approach to file manipulation in VBA is through the **Open Statement**. This statement is part of the native VBA runtime environment and does not require enabling external references, making it highly accessible for basic file operations. The syntax is straightforward, allowing developers to quickly define how a file should be accessed.

The standard syntax for the **Open Statement** is as follows: `Open For As` . The is the full path to the text file. The defines the operation (e.g., Input, Output, Append). Finally, is a unique integer (typically between 1 and 255) assigned to the file handle during its active use.

For instance, to prepare a text file named "Test.txt" for reading data, the required code utilizes the `Input` mode: `Open "Test.txt" For Input As #1`. Once the file is opened using this method, subsequent commands like `Input #` or `Line Input #` are used to extract data from the file. After the data is read, the program should close the file using the **Close Statement**: `Close #1`.

Modes of Access: Input, Output, and Append

Understanding the different modes available in the **Open Statement** is critical, as the chosen mode dictates whether you can read from or write to the file, and how existing data is handled. These modes are specifically designed for sequential access of text files.

For Input: This mode is used strictly for reading data from an existing file. If the specified file does not exist, VBA will typically generate an error (Run-time error '53': File not found). Data read operations are performed sequentially from the beginning of the file.

For Output: This mode is used strictly for writing new data to a file. If the file already exists, opening it in `Output` mode will **truncate** (delete) all existing content before writing new data. If the file does not exist, it will be created.

For Append: Similar to `Output`, this mode is used for writing. However, if the file exists, the new data is written starting at the end of the existing file content, effectively adding to it instead of overwriting it. If the file does not exist, it is created, just like in `Output` mode.

For developers focused solely on reading data from a text file into Excel--which is the most common requirement for importing text-based logs or CSV files--the `For Input` mode is the appropriate choice when utilizing the legacy **Open Statement**. While this method is functional, many experts prefer the flexibility and object-oriented nature of the **FileSystemObject** model for complex file management tasks.

Method 2: The Modern Approach using FileSystemObject

For more advanced, robust, and object-oriented file operations, the **FileSystemObject** (FSO) model is highly recommended. FSO provides a comprehensive set of methods and properties for working with drives, folders, and files, including text file manipulation. It is part of the **Scripting Runtime Library**, which must be explicitly referenced in the VBA project before use.

The FSO method utilizes the **OpenTextFile Method**, which returns a `TextStream` object. This object then exposes specific methods like `ReadAll`, `ReadLine`, and `SkipLine`, offering granular control over reading and writing text data. This approach is generally cleaner and easier to read than the numerical file handling required by the native **Open Statement**.

You can use the **OpenTextFile** method in VBA to open a text file from a specific file path. Below is a practical example demonstrating the use of FSO to read an entire file into a string variable and display its contents in a cell.

Here is one common way to use this method in practice:

```
Sub ReadTextFile()

Dim FSO As New FileSystemObject
Set FSO = CreateObject("Scripting.FileSystemObject")

'specify path to text file
Set MyTextFile = FSO.OpenTextFile("C:UsersbobDesktopMyTextFile.txt", ForReading)

'open text file and display contents in cell A1
TxtString = MyTextFile.ReadAll
```

```
MyTextFile.Close
ThisWorkbook.Sheets(1).Range("A1").Value = TxtString

End Sub
```

This particular macro reads the text file called **MyTextFile.txt** located at the specified path (`C:UsersbobDesktop...`) and displays the entire contents of the file in cell **A1** of the first sheet. The use of `ReadAll` is efficient for single-cell imports.

Prerequisites: Enabling the Microsoft Scripting Runtime Library

A crucial step when adopting the **FileSystemObject** is ensuring that the necessary library is accessible within your VBA Project. The FSO is contained within the Microsoft Scripting Runtime library. Although the `CreateObject` function allows for late binding, explicitly adding the reference (early binding) provides type safety and performance improvements.

Therefore, before using FSO methods effectively, you must enable the **Microsoft Scripting Runtime** within the Visual Basic Editor (VBE).

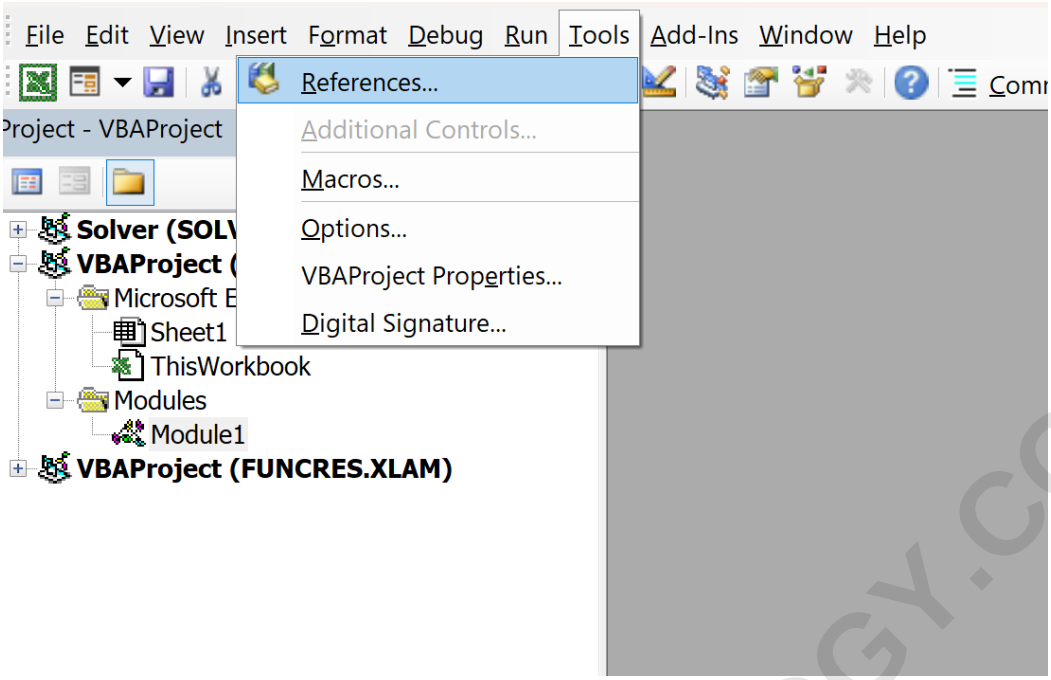
To successfully enable this reference, follow these sequential steps within the VBE:

Open the **Visual Basic Editor** (Alt + F11).

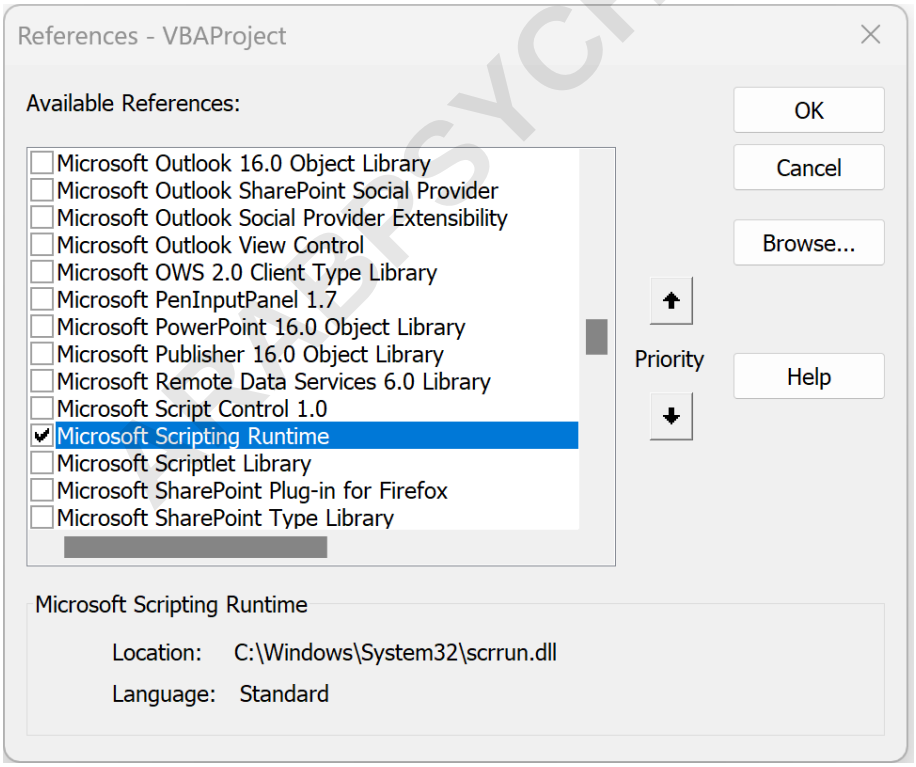
Navigate to the **Tools** menu.

Select **References...** from the dropdown list.

This action opens the References dialogue box, which lists all available libraries for your VBA project.



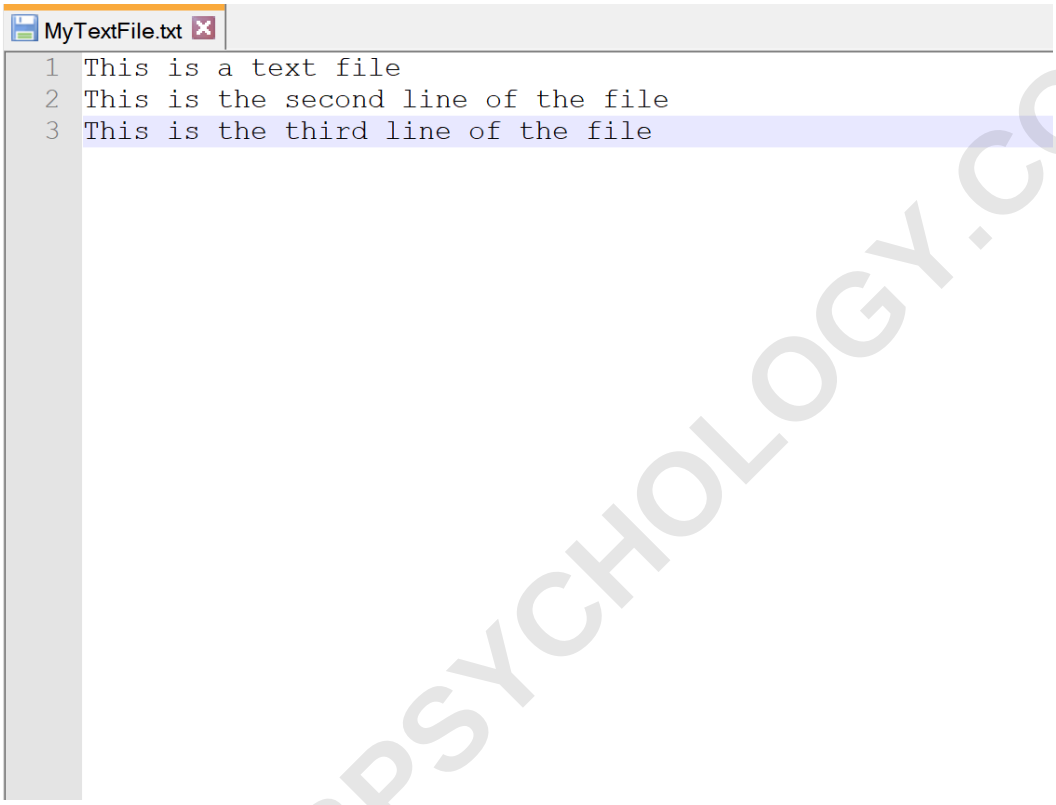
In the new window that appears, scroll down the list until you locate **Microsoft Scripting Runtime** and check the box next to it. Then click **OK** to finalize the reference inclusion.



Example: How to Open a Text File Using VBA

Suppose we have a text file called **MyTextFile.txt** located on the Desktop that we'd like to read into Excel using VBA. This file contains several lines of data that we want to consolidate into a single cell.

Here are the contents of the file we intend to read:

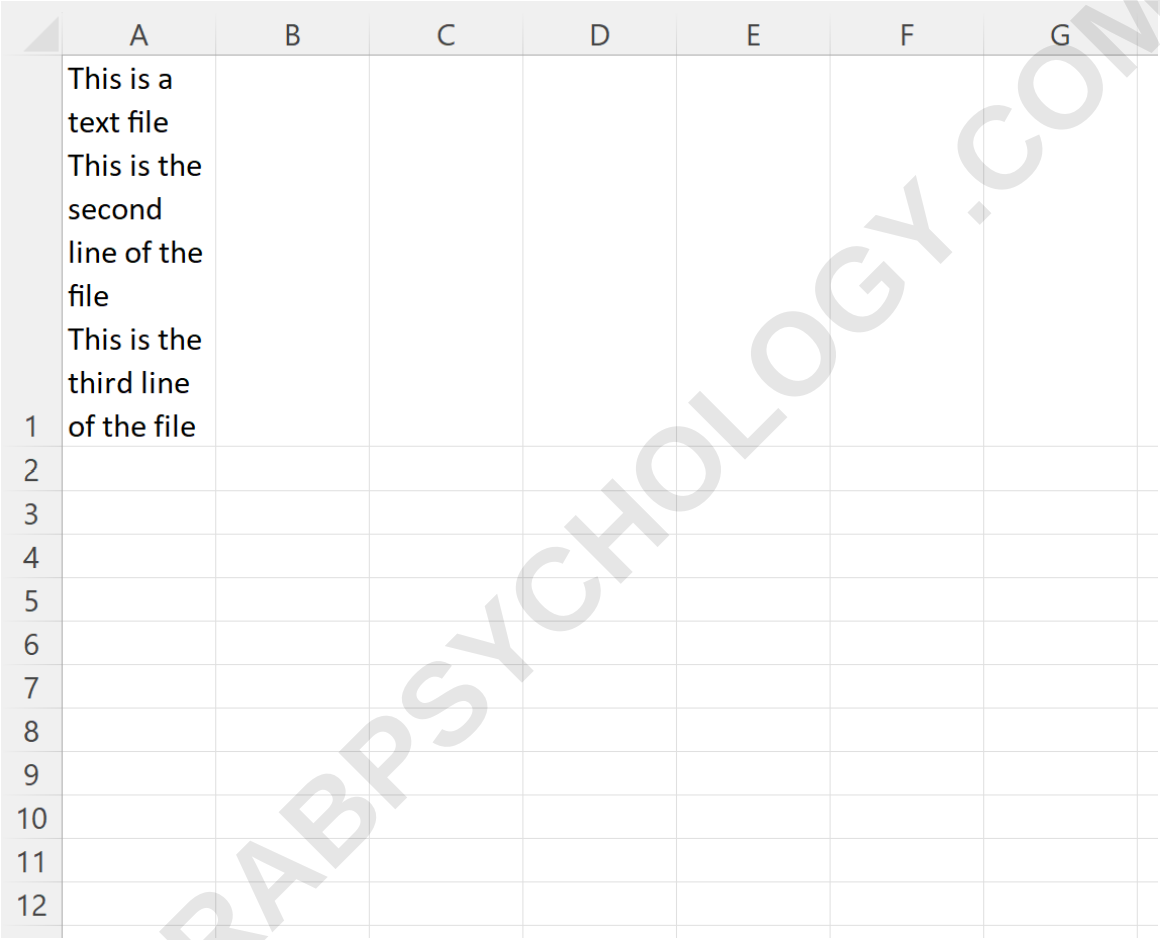


We have already enabled the necessary **Microsoft Scripting Runtime** library as detailed above. Next, we can create the following macro to read the text file efficiently using the **OpenTextFile Method** and the `ReadAll` function:

```
Sub ReadTextFile()

Dim FSO As New FileSystemObject
Set FSO = CreateObject("Scripting.FileSystemObject")

'specify path to text file
Set MyTextFile = FSO.OpenTextFile("C:UsersbobDesktopMyTextFile.txt", ForReading)

'open text file and display contents in cell A1
TxtString = MyTextFile.ReadAll
```

```
MyTextFile.Close
ThisWorkbook.Sheets(1).Range("A1").Value = TxtString

End Sub
```

Once we run this macro, the contents of the text file called **MyTextFile.txt** will be displayed entirely within cell **A1** of the active worksheet:



Notice that the contents of cell **A1** accurately match the contents read from the source text file, demonstrating the successful execution of the file reading operation using the **FileSystemObject**.

Conclusion and Resources

Mastering file handling in VBA, particularly for reading text files, is essential for automating many data import tasks within Excel. While the legacy **Open Statement** remains viable for simple tasks, the **FileSystemObject** provides a superior, object-oriented framework for robust and professional application development.

Always ensure that external references like the Microsoft Scripting Runtime are enabled when using FSO, and strictly follow the practice of closing all file objects immediately after use to maintain system stability.

Note: You can find the complete documentation for the **OpenTextFile Method** and other FSO components on the official Microsoft Developer Network (MSDN).