Learning VBA: A Step-by-Step Guide to Getting the Workbook Name in Excel

Authored by

Mohammed looti

November 9, 2025

When building robust automated routines in VBA (Visual Basic for Applications), a common requirement is the ability to programmatically identify the name of the currently active Workbook. Retrieving this information is critically important for several back-end processes, including logging operations, establishing dynamic file paths for external resources, or generating output reports that need to clearly reference the source document. Fortunately, Excel offers streamlined methods within its Workbook object model to accomplish this task, giving developers flexibility depending on whether the file extension must be included in the result.

This expert guide details two distinct, fundamental techniques available in VBA for extracting the file identifier of an Excel document. We will cover how to retrieve the complete name (which includes the file extension) and, perhaps more often needed, how to isolate only the base name through targeted string manipulation techniques. Mastering these methods is foundational for advanced file handling and automation.

Prerequisites and Understanding the ActiveWorkbook Object

Before implementing any code, it is essential to understand the core object that enables access to the file properties we seek. All the operations discussed in this tutorial rely heavily on the use of the ActiveWorkbook property. This property returns a reference to the specific Workbook object that currently holds the focus within the Excel application, either because the user selected it or because code execution activated it most recently. A crucial consideration for developers is error handling: if no workbook is currently open, or if the user's focus is unexpectedly shifted to a non-workbook window, attempting to access this property will typically raise a runtime error. Therefore, ensuring a valid workbook context is always necessary before proceeding.

The ActiveWorkbook property is positioned high in the global Application object hierarchy, granting immediate and efficient access to the file currently selected. Once we secure this object reference, we can query its various associated properties, such as its Path (location on disk), FullName (path plus name), or, most pertinent to retrieving the identifier, its simple **Name**. This programmatic access forms the backbone for nearly all file-level automation tasks within VBA development, making the property a cornerstone concept.

For maximum utility and seamless integration, we will be implementing these file-retrieval methods as User Defined Functions (UDFs). UDFs are custom functions written using VBA that can be called directly from formulas entered into Excel worksheets. This powerful feature allows for dynamic data points, such as the file name, to be retrieved and displayed effortlessly alongside standard spreadsheet calculations.

Method 1: Retrieving the Full Workbook Name (Including Extension)

The simplest and most direct way to obtain the file name is by utilizing the built-in **Name** property

of the ActiveWorkbook object. This property returns the file name exactly as it is recognized and stored by the operating system, meaning it always includes the file extension (e.g., .xlsx, .xlsm, or .xlsb). This approach is highly recommended when the full, unambiguous identifier of the source file is required for critical system-level operations, such as file backups, copying procedures, or security verification checks.

To implement this retrieval mechanism, we define a UDF that simply assigns the value of the `ActiveWorkbook.Name` property to the function's return variable. Since file names are textual data, it is imperative that the function be explicitly declared to return a `String` data type, ensuring proper handling and storage of the file identifier.

The following structure illustrates the most concise and efficient way to retrieve the complete name of the Workbook currently active within the Excel application interface:

VBA Code for Full Name Retrieval

```
Function GetWorkbookName() As String
GetWorkbookName = ActiveWorkbook.Name
End Function
```

Execution of this function will yield the full file identifier, such as **my_vba_workbook.xlsm**. It is vital to remember that this function is entirely context-dependent, relying on the user's focus within Excel. Should the user switch focus to a different open workbook, subsequent calls to this UDF will immediately reflect the name of the newly focused document, highlighting the dynamic nature of the `ActiveWorkbook` property.

Walkthrough: Implementing Method 1 (Example 1)

Let us examine a practical application where we need to display the active filename directly on a worksheet for user reference. For this specific scenario, we assume the file currently open and active is named **my_vba_workbook.xlsm**. The first step involves inserting the simple function defined above into a standard VBA Module within the Visual Basic Editor (VBE).

The following function, designed to capture the complete name of the currently active workbook, must be placed into the module:

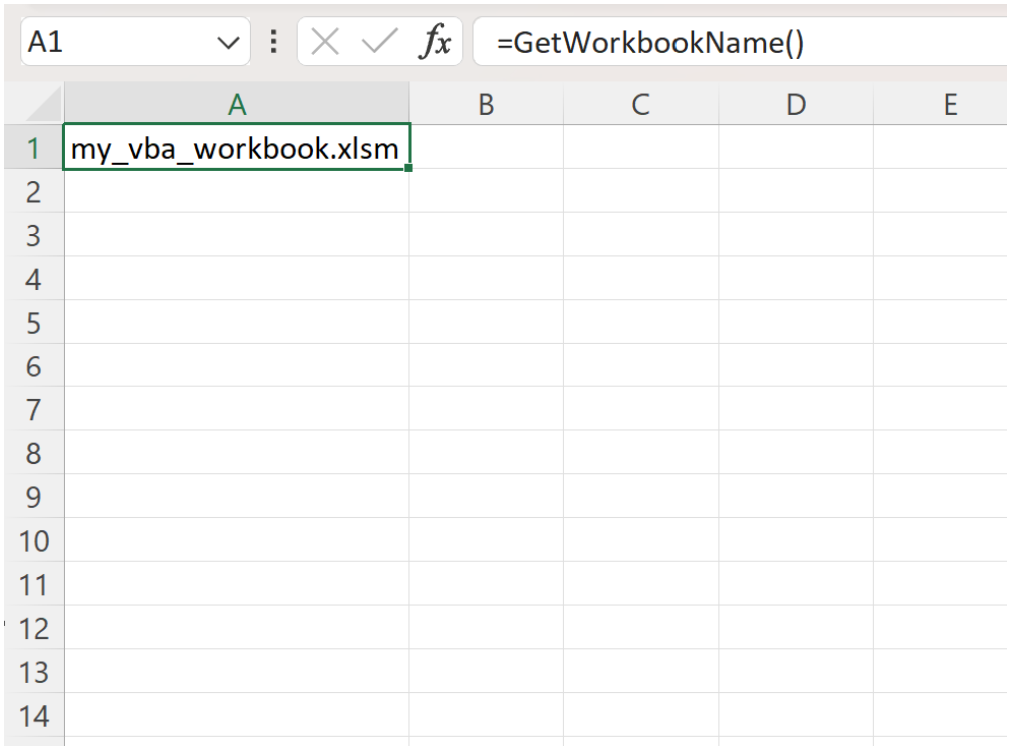
```
Function GetWorkbookName() As String
GetWorkbookName = ActiveWorkbook.Name
End Function
```

Once this UDF is successfully compiled, it becomes available for use just like any native Excel

formula. To invoke the function, the user simply references its name within the desired cell, for instance, cell **A1**:

=GetWorkbookName()

The subsequent screenshot graphically illustrates the execution of this formula, unequivocally demonstrating that the function successfully retrieves the full file name, including the requisite extension:



As clearly shown, the formula output is **my_vba_workbook.xlsm**, which accurately reflects the entire identifier of the currently active file. It is worth observing that the extension **.xlsm** specifically denotes that this particular file is a **macro-enabled Workbook**, confirming that it contains the VBA code modules necessary for the function to operate.

Method 2: Isolating the Workbook Name (Removing the Extension)

For many reporting, display, or internal referencing contexts, the inclusion of the file extension (e.g., .xlsx or .xlsm) is unnecessary and can clutter the output; often, only the clean, base file name is required. Achieving this cleaner output necessitates precise string manipulation techniques within VBA to parse the full name and isolate the desired portion. The industry standard approach involves combining two critical VBA functions: the InStr function and the Left function.

The core algorithmic strategy is executed in two steps. First, we must locate the exact position of the file extension separator--the period (`.`)--using the InStr function. This function searches a given string and returns the starting index of a specified substring (in this case, the period). Second, once the position of the period is known, we leverage the Left function to extract a specified number of characters starting from the beginning (left side) of the string. Crucially, we calculate the required number of characters by taking the position returned by InStr and subtracting 1; this ensures the period itself is excluded from the final output string.

This combination of string functions provides a highly robust and scalable way to handle any file name length, ensuring the dynamic and accurate removal of the extension regardless of its length (three or four characters). The resulting function effectively parses the value retrieved from the `ActiveWorkbook.Name` property:

VBA Code for Name Retrieval Without Extension

```
Function GetWorkbookName() As String
GetWorkbookName = Left(ActiveWorkbook.Name, InStr(ActiveWorkbook.Name, ".") - 1)
End Function
```

This refined function returns only the base name, such as **my_vba_workbook**, resulting in an output that is far cleaner for user interfaces or standardized internal reports where the file type is already implied. It is imperative that developers consider potential edge cases, such as files that may contain multiple periods in their name (e.g., "Report.2023.xlsx"). While the code above works perfectly for files where the last period demarcates the extension, more complex scenarios might require using the `InStrRev` function to ensure the search starts from the right side of the string.

Walkthrough: Implementing Method 2 (Example 2)

To demonstrate the practical benefits of advanced string manipulation, we now replace the simple `ActiveWorkbook.Name` function with the more complex UDF that incorporates the Left function and InStr function logic. This strategic approach guarantees that we capture only the user-defined name portion of the file, effectively stripping away the system-required file extension.

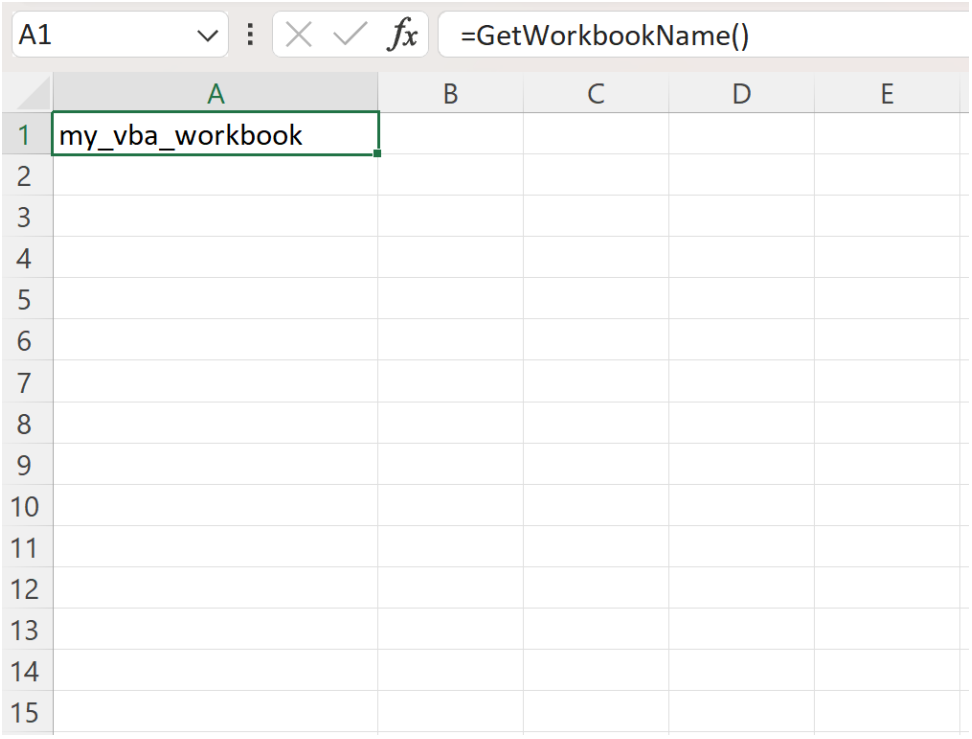
We must utilize the following string-parsing function within our VBA module to extract the active workbook name without the potentially distracting trailing file extension:

```
Function GetWorkbookName() As String
GetWorkbookName = Left(ActiveWorkbook.Name, InStr(ActiveWorkbook.Name, ".") - 1)
End Function
```

Identical to the first example, this UDF is called directly from the worksheet. By typing the exact same formula into cell **A1**, the Excel calculation engine executes the VBA code, applying the string manipulation before returning the result:

=GetWorkbookName()

The following screenshot validates that the string manipulation logic is completely successful, returning only the core file name without any extraneous characters:



The formula now returns the desired **my_vba_workbook**, having successfully truncated the **.xlsm** extension. This precise result is achieved by utilizing the Left function in conjunction with the InStr function. Specifically, InStr locates the period delimiter, and the Left function extracts all characters up to the position immediately preceding that delimiter, fulfilling the core requirement for a clean base file name.

Summary and Best Practices for File Referencing

Programmatically retrieving the active Workbook name in VBA represents a crucial, foundational skill for robust Excel automation projects. The choice between the two methods--retrieving the full name (Method 1) or isolating only the base name (Method 2)--must be driven entirely by the specific requirements of the automation task. The simple `ActiveWorkbook.Name` property is reliable and fast for system-level operations, whereas the combination of Left and InStr provides

the necessary string flexibility required for clean, user-facing outputs.

When implementing these solutions, always adhere to best practices: ensure that your code is placed in a standard module (not a sheet module or the `ThisWorkbook` module) if your intent is to use the function as a UDF directly in a worksheet cell. Furthermore, remember that the ActiveWorkbook property can change dynamically based on user interaction or code execution flow. If your procedure involves switching between multiple open files, you should use the `ThisWorkbook.Name` property instead if you specifically need the name of the file containing the executing VBA code, providing an absolute reference instead of a dynamic one.

By mastering these straightforward yet powerful techniques, developers can significantly enhance the robustness, clarity, and adaptability of their VBA projects, enabling sophisticated, dynamic file referencing and dramatically improving data logging capabilities within complex enterprise workflows.

Additional Resources for VBA Development

The following related tutorials provide instruction on how to perform other common file and system tasks in VBA:

How to Create Folders Using VBA