How to use FileDateTime Function in VBA (With Example)

Authored by

stats writer

November 18, 2025

The **FileDateTime** function in VBA (Visual Basic for Applications) is a fundamental tool for system interaction, designed specifically to retrieve the date and time stamp associated with a specified file. This stamp indicates precisely when the file was either **created or last modified**.

The primary utility of this function lies in auditing processes, verifying the recency of data, and ensuring that automation scripts are processing the most up-to-date resources available on the system. It is a powerful, yet simple, function provided by the VBA run-time library.

Understanding the FileDateTime Function Syntax and Usage

The **FileDateTime** function accepts a single, mandatory argument: the full path to the file you wish to query. The function returns a Variant of subtype Date, which holds the precise date and time information retrieved from the operating system's file metadata. This returned value is critical for determining the lifecycle status of a document or application file.

The basic syntax for utilizing this powerful function is straightforward:

Syntax: `FileDateTime(pathname)`
`pathname`: A required String expression specifying the complete name and absolute path of the target file.

It is crucial to remember that the file must exist at the specified location for the function to execute successfully. If the specified file cannot be located, the function will trigger a run-time error, necessitating proper error handling mechanisms for reliable macro execution.

Practical Application: Dynamic File Path Input

A highly common and effective way to use **FileDateTime** is by integrating it with user interface tools, such as the InputBox. This approach allows the user to dynamically specify the file they are interested in querying, making the macro highly reusable across different work environments or document sets without hardcoding file paths.

Consider the following macro, which demonstrates this dynamic interaction. The file path provided by the user is captured as a String variable and passed directly to the function:

Sub CheckLastModify()

Dim wb_name As String

wb_name = InputBox("Please enter the workbook name:")

MsgBox FileDateTime(wb_name)

End Sub

When a user executes this routine, the **InputBox** will appear, prompting them to enter the full file path for the target resource, such as an Excel workbook. The macro then retrieves the date and time stamp, which is subsequently presented to the user through a standard **message box**. This setup provides instant feedback regarding the file's modification history.

In-Depth Example: Querying a Specific File

To fully illustrate the utility of **FileDateTime**, let us consider a specific operational scenario where we need to verify the last update time of a critical data file. Assume we are analyzing a file named **My_Workbook.xlsx** residing in a user's document directory. The full file path is:

C:UsersbobDocumentsmy_workbook.xlsx

Our goal is to execute a VBA routine that efficiently retrieves this modification timestamp. We use the same macro structure, relying on the user to input the correct path:

Sub CheckLastModify()

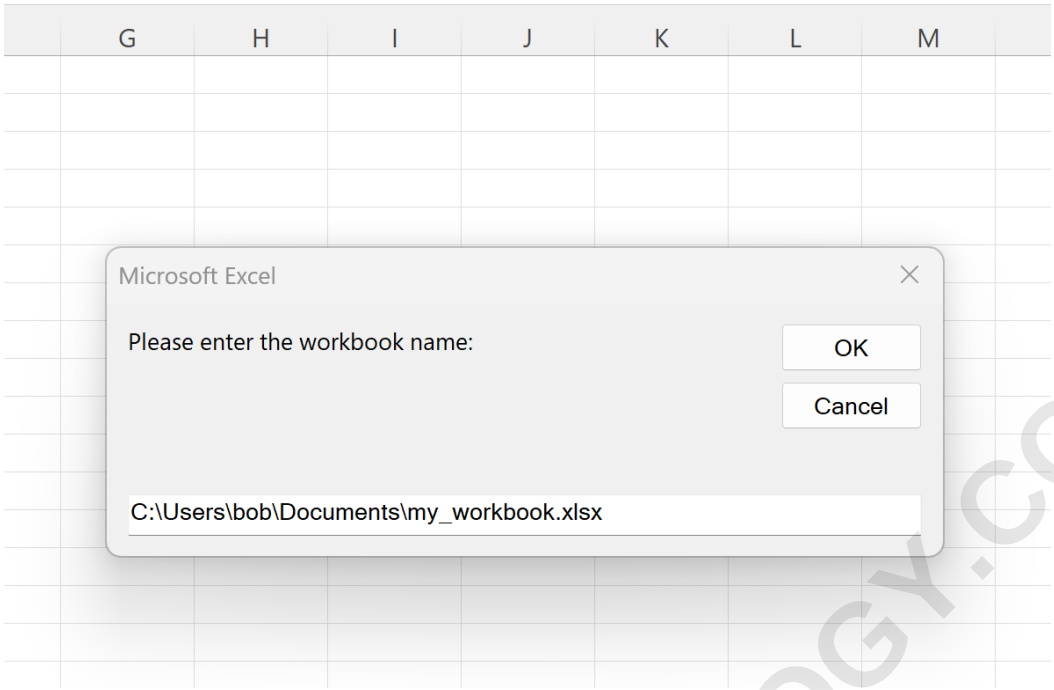
Dim wb_name As String

wb_name = InputBox("Please enter the workbook name:")

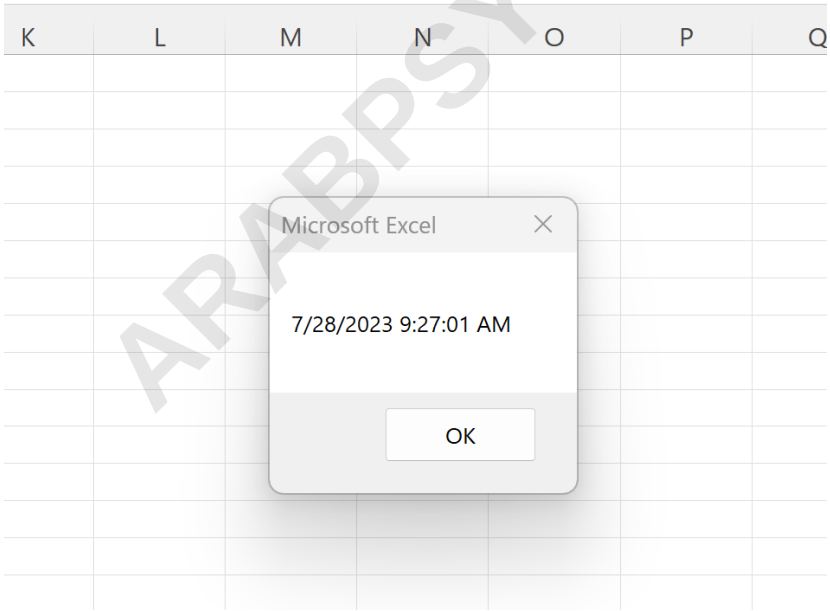
MsgBox FileDateTime(wb_name)

End Sub

Upon execution, the user receives the prompt and must input the exact, fully qualified path into the input field, including the drive letter and file extension. This ensures the **FileDateTime** function correctly targets the file system object:



After confirming the input by clicking **OK**, the macro processes the request and returns the metadata. The result, displayed in a subsequent **message box**, confirms the exact date and time of the last modification event:



From this output, we successfully ascertain that the workbook was last modified on **7/28/2023** at the precise time of **9:27:01 AM**. This combined value provides the maximum detail available through this specific function.

Refining the Output: Extracting Only the Date

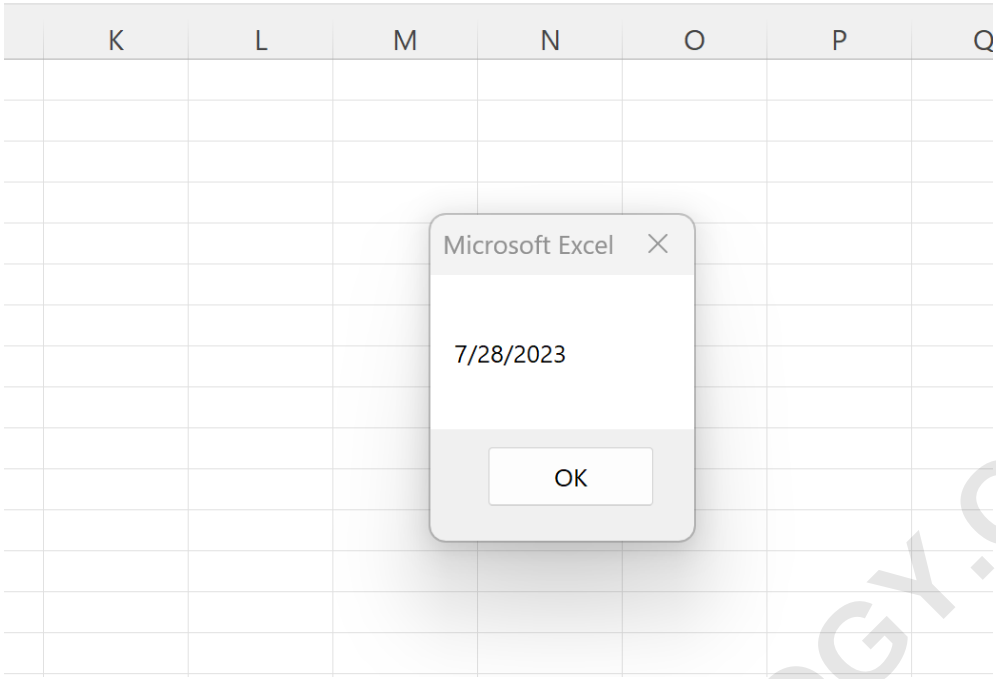
In many business scenarios, the modification time (down to the second) is unnecessary; only the modification date is required for logging or validation. To isolate the date component and suppress the time component, we can easily integrate the **FileDateTime** call within the DateValue function.

The **DateValue** function extracts the date portion from a Date variant, effectively setting the time component to midnight (12:00:00 AM). This streamlines the output, making it cleaner for reports that are date-centric.

Observe the refined macro structure that utilizes nested functions for precise output control:

```
Sub CheckLastModify()

Dim wb_name As String

wb_name = InputBox("Please enter the workbook name:")

MsgBox DateValue(FileDateTime(wb_name))

End Sub
```

When this revised macro is executed and the necessary file path is supplied, the resulting **message box** will display a simplified and focused result, showing only the date of the last file update:



This demonstrates the power of combining built-in VBA functions to manipulate system data into a format suitable for end-user requirements.

Considerations for Network Paths and Permissions

When implementing **FileDateTime** in enterprise applications, developers must account for networked resources. The function seamlessly handles network paths, provided they are specified using the full Universal Naming Convention (UNC) format, such as `\ServerNameShareFile.ext`. Relative paths or shortcuts will not be processed correctly.

Furthermore, security and permissions are paramount. If the user account executing the VBA macro lacks read permission for the target file or the directory structure containing it, the function will fail, typically generating a "Permission denied" error. Proper testing in a production environment is necessary to ensure the macro operates under the correct security context.

Error Handling and Robustness

The reliability of any macro relying on external file interaction depends heavily on adequate error handling. Since **FileDateTime** will raise an error if the file specified in the `pathname` argument does not exist (Error 53: File not found), developers should use defensive programming techniques.

A recommended approach is to check for the file's existence using the `Dir` function immediately before calling **FileDateTime**. If `Dir(pathname)` returns an empty string, the macro can gracefully

exit or prompt the user for a correction, preventing a fatal run-time error and enhancing the user experience.

Note: You can find the complete documentation for the **FileDateTime** function in VBA.