Learning VBA: Retrieving File Creation and Modification Dates with FileDateTime

Authored by

Mohammed looti

November 9, 2025

The **FileDateTime** function is a critical, built-in component within **VBA** (Visual Basic for Applications) designed specifically for file management and rigorous auditing procedures. This powerful function efficiently retrieves the exact date and time stamp corresponding to when a specified file was created or, more frequently, when it was last modified by the operating system. Implementing this function allows developers and advanced users to integrate sophisticated tracking mechanisms directly into their Excel applications, thereby ensuring the accurate monitoring of resource lifecycles and version control.

Utilizing this function is paramount when working with shared network resources or intricate data processing workflows where the currency and integrity of the source data are non-negotiable requirements. Before diving into advanced applications, we will first examine the fundamental syntax structure and then proceed to a practical demonstration showcasing its common usage for rapid file metadata retrieval.

```
Sub CheckLastModify()

Dim wb_name As String

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

MsgBox FileDateTime(wb_name)

End Sub
```

Understanding the FileDateTime Functionality

Programmatically accessing file properties is a foundational capability for achieving effective automation within any **VBA** environment. The primary objective of the **FileDateTime** function is to offer a swift, native method for obtaining timestamp data without the necessity of instantiating the more resource-intensive **FileSystemObject** (FSO) library. This makes it the perfect choice for creating lean, self-contained scripts that require only a single piece of critical file metadata. The function requires just one argument: the complete file path (pathname) of the resource you intend to query.

When the automated routine executes, the system initiates a prompt, typically utilizing the **InputBox** function, to request the exact location of the target file from the user. It is critically important that the user supplies the accurate, full path, including the necessary file extension, as the function generally cannot reliably resolve relative paths across different operational contexts. Upon successful path validation, the function returns a Variant subtype Date value, which is subsequently displayed to the user via the standard **MsgBox** function. This returned timestamp consistently reflects the last moment the file's content was saved or modified, making it highly

relevant for compliance auditing and data reconciliation activities.

This initial demonstration highlights the efficiency inherent in the **FileDateTime** function: it consolidates the typically multi-step process of checking file properties within the operating system into a single, concise line of code within the **macro**. This streamlined approach offers a significant advantage for developers seeking to embed rapid status checks into larger automation sequences, ensuring, for instance, that a prerequisite source file has been updated before initiating subsequent data processing stages.

Syntax Specification and Core Usage

The formal syntax required for implementing this utility is remarkably straightforward: `FileDateTime(pathname)`. The mandatory `pathname` argument must be provided as a String expression that unambiguously identifies the file intended for the query. To prevent runtime failures, this path must be absolute, encompassing the drive letter and all intermediate directory structures. Should the specified file not be found at the designated location, the **VBA** interpreter will typically raise Error 53 (File not found), underscoring the necessity for robust error handling mechanisms in any production-ready code.

Let us analyze the scenario presented in the initial code block. When the **macro** named `CheckLastModify` is executed, the **InputBox** appears, functioning as the crucial user interface element responsible for gathering the file reference. The resulting string provided by the user is stored temporarily in the variable `wb_name`. Subsequently, the **FileDateTime** function processes this string, initiating a query against the underlying file system metadata. The resulting date and time stamp is then immediately conveyed to the user through a pop-up window generated by the **MsgBox** function, detailing precisely when the referenced file--often an **Excel workbook** in this context--was either created or last modified.

This implementation method offers an immediate and highly focused feedback loop. While it admittedly lacks the granular capabilities of the **FileSystemObject**, such as distinguishing between creation date and last access date, its superior speed and ease of integration establish it as the preferred option for quick, transactional status checks. Furthermore, because the output is a standard Date data type, the retrieved timestamp can be effortlessly manipulated, custom formatted, or compared against other existing date values within the **VBA** environment to drive complex conditional execution logic.

A Detailed Walkthrough Example

To demonstrate the functional utility of **FileDateTime**, let us consider a practical application involving a specific file residing on a local machine. Imagine we are tasked with tracking an essential **Excel workbook** named **My_Workbook.xlsx**, which is located in a standard user

directory, specifically at **C:UsersbobDocumentsmy_workbook.xlsx**. Our primary goal is to ascertain the precise moment this file was last updated, ensuring absolute certainty that we are utilizing the most current iteration of the data.

We initiate the exact standard **macro** routine previously introduced. This routine is designed for maximum effectiveness and simplicity, requiring the user to supply the full, dynamic file path at the time of execution. This dynamic input prevents the reliance on hard-coded paths, which quickly become obsolete and difficult to maintain. The foundational code structure remains fully consistent, focusing solely on the capture of user input and the immediate delivery of the file metadata via the core **FileDateTime** function.

Sub CheckLastModify()

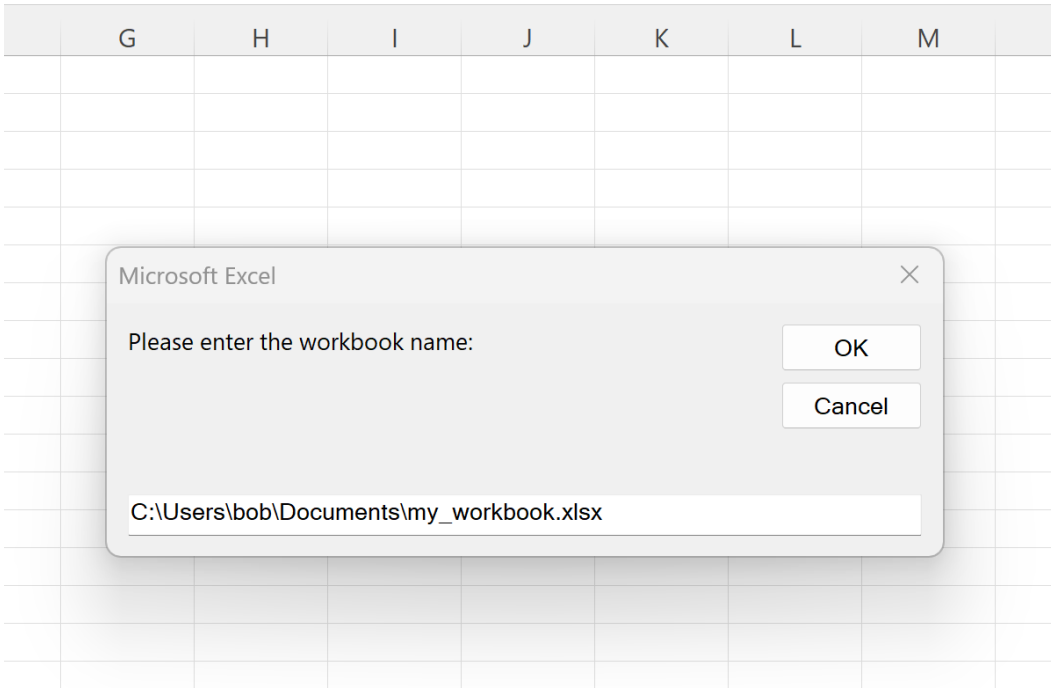
Dim wb_name As String

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

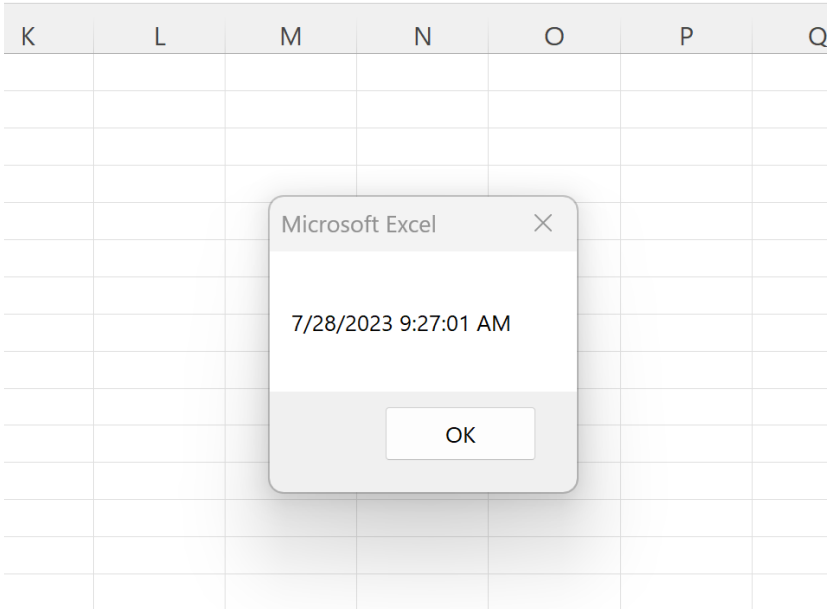
MsgBox FileDateTime(wb_name)

End Sub

Upon execution of this **macro**, a prominent prompt appears, demanding that the user accurately type or paste the complete path to the intended target file. Precision in providing the path is paramount; even a minor typographical error will invariably trigger a runtime failure. Once the correct path (`C:UsersbobDocumentsmy_workbook.xlsx`) is entered into the **InputBox** and confirmed, the function immediately executes its query against the operating system's internal file properties registry.



Once the input is accepted by clicking **OK**, the **macro** provides instantaneous feedback by presenting the retrieved metadata in a subsequent message box. This output clearly displays the combined date and time of the last modification. This highly granular information is sourced directly from the file system, providing an official, auditable timestamp.



As clearly illustrated in the resulting message box, the **macro** successfully reports that the specific workbook was last modified on **7/28/2023** at **9:27:01 AM**. This precise level of detail confirms the

exact status of the file at the moment of the query, which is an indispensable requirement for environments demanding rigorous version control and comprehensive data accountability.

Refining Output: Extracting Only the Date Component

While the default behavior of the **FileDateTime** function is to return a comprehensive, combined date and time stamp, developers frequently need only the date component for simplified reporting or comparative analysis. For instance, if the primary goal is merely tracking the day a file was actively worked on, the exact time down to the hours, minutes, and seconds can often be viewed as redundant or distracting information. To accommodate this pervasive requirement, **VBA** offers several powerful intrinsic functions specifically designed to process and format Date data types.

The most streamlined approach for removing the time component is achieved by wrapping the **FileDateTime** function call within the **DateValue** function. The **DateValue** function is specifically engineered to parse a Variant (Date) input and subsequently return only the date portion, effectively discarding any time information that may be present. This critical refinement significantly streamlines the output, rendering the information immediately more digestible for users whose tracking needs are focused strictly on daily metrics.

By making a minor modification to the original code to incorporate this wrapper, the **macro**'s output is shifted from a precise timestamp to a clear, simple date indicator. This precise adjustment is executed by changing the existing line `MsgBox FileDateTime(wb_name)` to `MsgBox DateValue(FileDateTime(wb_name))`. This demonstrates the profound versatility inherent in **VBA**'s native functions, enabling rapid output customization without necessitating complex or cumbersome string manipulation routines.

Sub CheckLastModify()

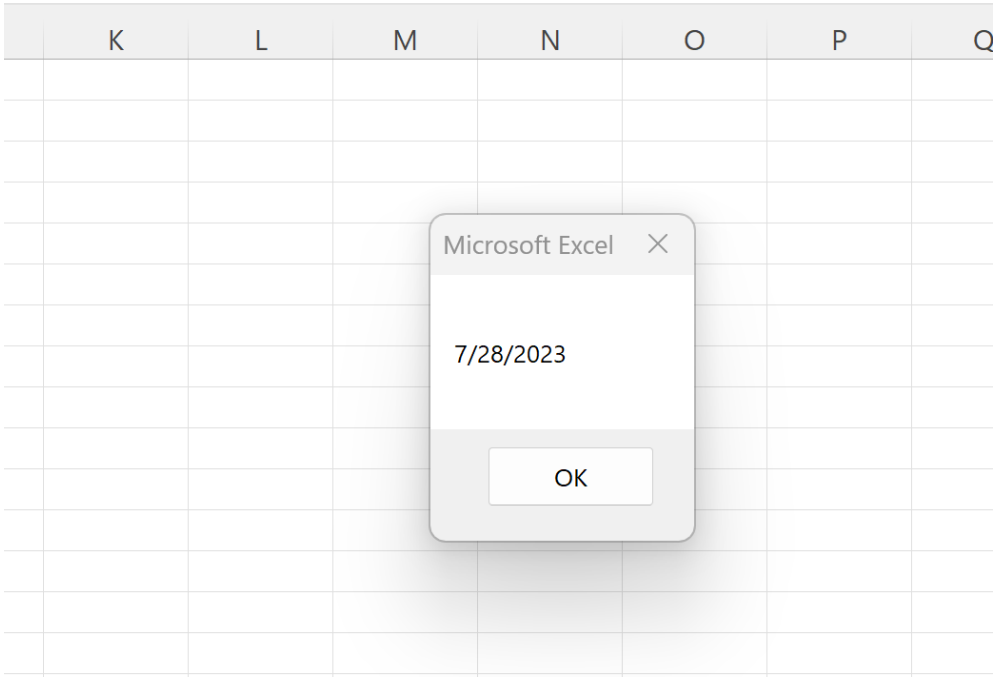
Dim wb_name As String

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 correctly supplied, the resulting message box now presents a distinctly cleaner output, displaying solely the date the file was last modified. This highly streamlined approach efficiently satisfies the requirements for generating daily logs or integrating with reporting systems where the time component is deemed extraneous detail.



Limitations and Advanced Alternatives in VBA File Management

While the **FileDateTime** function is exceptionally reliable and effective for its primary objective--retrieving the last modification timestamp--it is imperative to acknowledge certain inherent limitations, particularly when handling complex file system interactions. The main restriction is that it provides only one consolidated date/time value, which, by default, reflects the most recent write operation. Crucially, it lacks the ability to independently return the file creation date or the last access date, pieces of information frequently essential for thorough auditing processes. Furthermore, if the file resides on a shared network resource, potential discrepancies arising from server time zones or caching protocols might occasionally introduce minor inconsistencies in the reported time, although modern operating system standards generally mitigate this issue.

For developers who demand more granular and comprehensive control over file attributes, the definitive alternative is the utilization of the **FileSystemObject** (FSO), which is integrated within the Microsoft Scripting Runtime library. The FSO grants access to the `File` object, which in turn exposes distinct, separate properties such as `DateCreated`, `DateLastAccessed`, and `DateLastModified`. Implementing the FSO introduces a marginal increase in complexity--requiring the setting of a project reference to the library and subsequent object instantiation--but compensates by offering significantly greater functional depth for managing and querying all file metadata, including detailed file size and attributes.

Another vital consideration when employing straightforward file functions like **FileDateTime** is robust error handling. As previously noted, the function will inevitably fail and halt execution if the

provided file path is inaccurate or if the specified file does not exist. In professional, production-level code, the strategic use of `On Error Resume Next` followed by an explicit check for the specific error number (Error 53) is absolutely essential to prevent the **macro** from crashing unexpectedly. By integrating sophisticated error trapping, developers can ensure system stability and deliver user-friendly feedback, such as "Error: File not found. Please verify the path and attempt the query again," thereby significantly enhancing the overall reliability of the **VBA** application.

Conclusion and Resources for Continued Learning

The **FileDateTime** function stands as an invaluable, highly efficient tool within the **VBA** ecosystem for rapidly ascertaining the last modification time of any designated file. Its inherent simplicity, coupled with its native integration, makes it perfectly suited for rapid prototyping and critical auditing checks within **Excel workbook** automation projects. Regardless of whether you require the full, precise timestamp or choose to extract only the date component using the powerful **DateValue** wrapper, this function provides the critical metadata necessary for maintaining stringent data integrity and effective version control.

By achieving mastery over the core implementation techniques showcased in this guide, developers can significantly elevate both the efficiency and accountability built into their **VBA** scripts. While complex file system operations may eventually demand the expanded functionality of the FileSystemObject, **FileDateTime** remains the most direct, elegant, and expedient route for accessing fundamental file timestamp information.

For ongoing professional development and deeper technical exploration into **VBA** utilities, the complete, authoritative documentation for the **FileDateTime** function is readily accessible through Microsoft's official developer resources.

Further Learning Resources

To strategically expand your knowledge base concerning file manipulation and standard programming tasks within **VBA**, we recommend exploring the following related tutorials and official documentation:

Working with the **FileSystemObject** (FSO) for advanced retrieval of file attributes.
Implementing structured error handling techniques (`On Error GoTo`) for ensuring robust **macro** execution reliability.
Formatting Date and Time values using the versatile `Format` function in **VBA**.