What is TimeValue Function in VBA (With Example)

Authored by

stats writer

November 18, 2025

Introduction to the TimeValue Function in VBA

The **TimeValue** function is a critical utility within Visual Basic for Applications (VBA), designed specifically for manipulating and extracting time components from textual data. In data processing and automation tasks within Excel, data often arrives as text strings, even when it represents dates and times. The primary purpose of the TimeValue function is to parse a string that represents a time and convert it into a valid internal Date/Time data type, specifically representing only the time fraction. This conversion is essential for performing accurate time-based calculations, comparisons, and formatting operations in your automated processes. Without this function, developers would have to resort to complex string parsing techniques, which are less efficient and more prone to errors.

When working with combined date and time stamps (datetimes), such as those commonly exported from databases or logged system files, it is frequently necessary to isolate the time element. The TimeValue function handles this separation seamlessly. It takes the entire date/time string, determines the time portion, and returns a double-precision floating-point number corresponding only to the time of day. This numerical representation is crucial because VBA stores dates and times as numbers: the integer portion represents the date, and the fractional portion represents the time. Therefore, TimeValue returns a value between 0 (representing 12:00:00 AM) and 0.99999... (just before 12:00:00 AM the next day), effectively discarding any underlying date component present in the original input string.

Understanding the utility of TimeValue is key to mastering VBA efficiency. It allows for the rapid standardization of time inputs, regardless of the format they were originally presented in (e.g., "10:30 AM," "22:30," or "1/1/2023 10:30:00"). While Excel offers similar built-in worksheet functions, utilizing TimeValue within a macro provides superior performance when processing large batches of data or integrating time extraction into complex, multi-step automation sequences. The function adheres to the locale settings of the host system when interpreting the input string, ensuring that regional time formats are handled correctly, although it is always best practice to use standard formats when possible.

The Essential Syntax and Arguments of TimeValue

The syntax for the **TimeValue** function is straightforward, requiring only one mandatory argument: the string expression representing the time. The structure is as follows: `TimeValue(time)`. The `time` argument must be a string expression that is recognizable as a time by VBA. This string can contain hours, minutes, and seconds, and may optionally include a date component. If a date component is included, the function ignores it entirely and extracts only the time information. It is crucial that the string is formatted in a way that VBA can successfully parse, which generally means conforming to standard time formats (e.g., military time or standard AM/PM format,

separated by colons).

When VBA processes the input string, it attempts to resolve the time based on the system's current culture settings. For instance, in a US locale, "1:30 PM" will be correctly interpreted as 13:30 in 24-hour format internally. However, if the input string is ambiguous or contains non-numeric characters that cannot be resolved as time separators (such as using a comma instead of a colon), the function will generate a runtime error, typically Error 13: Type Mismatch. Therefore, robust coding using TimeValue often involves input validation or utilizing helper functions to ensure the string conforms to expected time standards before processing.

The output of the TimeValue function is always a Date/Time data type, which, as previously noted, is internally stored as a Double. If the input string successfully represents a time, the returned value will be the fractional part of a day. This numerical value is extremely useful for mathematical comparisons; for example, if you need to check if one time falls after another, you can compare their respective TimeValue outputs directly, rather than comparing complex strings. Furthermore, the resulting time value can be directly assigned to an Excel cell, where Excel automatically applies the necessary time formatting for human readability, simplifying the presentation layer immensely.

Understanding VBA Date and Time Data Types

To fully appreciate the TimeValue function, it is essential to grasp how VBA handles the underlying Date/Time data type. Unlike some programming environments that use separate objects for dates and times, VBA stores both components within a single Double-precision floating-point number. This structure is inherited from Excel itself. The number is mathematically partitioned: the whole number (integer) part represents the date count starting from a specific base date (typically January 1, 1900, in Windows versions of Excel), and the decimal (fractional) part represents the time of day.

For instance, the number 45000.5 represents a specific moment in time. The integer 45000 signifies the 45,000th day since January 1, 1900, which corresponds to a specific date. The fractional part, .5, represents exactly half a day, or 12:00 PM (Noon). When the TimeValue function is executed, its singular task is to calculate and return this fractional component. If you input a string like "1/15/2025 06:00:00", the function discards the date (1/15/2025) and focuses only on 06:00:00, which is exactly one-quarter of a day, or 0.25. The returned numerical value will be 0.25, formatted by VBA as the Date/Time data type.

This numerical structure is highly advantageous for performing time arithmetic. If you need to calculate the duration between two times, you can simply subtract the two TimeValue outputs. The resulting difference will be a fraction representing the elapsed time. For example, subtracting the value for 9:00 AM (0.375) from the value for 10:00 AM (0.41666...) yields approximately

0.041666..., which is exactly 1/24th of a day, or one hour. This efficiency is why the TimeValue function is preferred over manual string manipulation when numerical operations on time are required in VBA.

Practical Application: Converting String Data to Time Values

A common scenario encountered by VBA developers involves handling raw data imports where date and time information are combined into a single text field. This combined format, often referred to as a datetime string, cannot be directly used for time-of-day specific analysis or calculations. The TimeValue function offers the most streamlined approach to isolate the time component when iterating through such data sets, particularly when the data resides within a consecutive block of cells in an Excel worksheet.

The following code snippet demonstrates a standard implementation pattern for extracting time values from a defined Range of cells. This method employs a simple loop construct, which is highly efficient for sequential data processing. The core logic involves reading the datetime string from column A, passing it to TimeValue for conversion, and then writing the resulting time value back into the corresponding row in column B. This process transforms non-numeric textual data into structured Date/Time data type elements suitable for downstream analysis.

```
Sub GetTimeValue()

Dim i As Integer

For i = 2 To 7
Range("B" & i) = TimeValue(Range("A" & i))
Next i

End Sub
```

This particular macro is designed to iterate through rows 2 through 7. For each row, it retrieves the contents of column A (e.g., A2, A3, ..., A7), converts the contained string into a time fraction using TimeValue, and then outputs this time value to the corresponding cell in column B (B2, B3, ..., B7). The power of this approach lies in its ability to handle variations in the input data format, provided the time component itself is unambiguous. This function effectively standardizes the representation, making subsequent time comparisons or duration calculations reliable and accurate within the Excel environment.

Example: How to Use TimeValue Function in VBA

Detailed Analysis of the Provided Macro Code

Let us consider a concrete example where we need to process a column containing various timestamps combined with dates. Suppose we have the following data set residing in Range A2:A7 of an Excel sheet. Our objective is to generate a derived column, column B, which contains only the time component for each entry.

	A	B	C	D
1	**Times**			
2	1/1/23 10:15:34 AM			
3	1/3/23 12:34:18 PM			
4	1/5/23 8:23:00 AM			
5	2/14/23 10:45:37 AM			
6	4/19/21 3:12:19 AM			
7	6/12/23 5:29:01 AM			
8				
9				
10				
11				
12				
13				
14				
15				
16				

The data structure clearly shows combined date and time information. To efficiently automate the extraction process, we utilize the previously detailed macro, which leverages the iterative capability of VBA and the specialized conversion power of the **TimeValue** function. The implementation begins with the declaration of a procedure and a loop counter.

The following macro, `GetTimeValue`, is designed to perform the necessary row-by-row data transformation:

Sub GetTimeValue()

Dim i As Integer

For i = 2 To 7
Range("B" & i) = TimeValue(Range("A" & i))
Next i

End Sub

In this code block, the loop variable `i` iterates from 2 to 7, targeting the rows containing data. Inside the loop, `Range("A" & i)` dynamically references the current input cell (e.g., A2, A3, etc.). This cell's value, which is a combined datetime string, is passed directly to the TimeValue function. The function executes the conversion, stripping the date information and returning the numerical fraction representing the time. This result is then assigned to the output cell, `Range("B" & i)`. Upon completion of the loop, column B will contain the extracted time values corresponding precisely to the times listed in column A, ready for any subsequent time-based analysis or reporting requirement.

Illustrative TimeValue Example in Excel

Once the `GetTimeValue` macro is executed in the Excel environment, the powerful conversion mechanism of the TimeValue function becomes visible. Column B is populated with the fractional time values, which Excel automatically displays using its default time formatting, ensuring immediate clarity for the user.

The execution of the script results in the following updated worksheet:

	A	B	C	D
1	**Times**			
2	1/1/23 10:15:34 AM	10:15:34 AM		
3	1/3/23 12:34:18 PM	12:34:18 PM		
4	1/5/23 8:23:00 AM	8:23:00 AM		
5	2/14/23 10:45:37 AM	10:45:37 AM		
6	4/19/21 3:12:19 AM	3:12:19 AM		
7	6/12/23 5:29:01 AM	5:29:01 AM		
8				
9				
10				
11				
12				
13				
14				
15				
16				

As clearly illustrated, Column B now holds the time component extracted from the corresponding datetime string in Column A. This output confirms that the TimeValue function successfully isolates

the time, regardless of the date or the exact structure of the input string, provided it is a valid time format. The consistency of this extraction process is what makes TimeValue an indispensable tool for data preparation in VBA.

We can analyze several specific rows to confirm the accuracy of the conversion and the underlying logic:

For the input **1/1/2023 10:15:34 AM**, the TimeValue function returns the time fraction representing **10:15:34 AM**.

When processing **1/3/2023 12:34:18 PM**, the function isolates and returns the time fraction corresponding to **12:34:18 PM**.

Given the input **1/5/2023 8:23:00 AM**, the function successfully converts and outputs the numerical value for **8:23:00 AM**.

This resulting column, B, is now suitable for complex calculations, such as calculating the time difference between subsequent events or aggregating data based on specific time slots, all of which rely on the numerical representation of time provided by the TimeValue function.

Handling TimeValue Limitations and Error Considerations

While the TimeValue function is highly effective, developers must be aware of its limitations, primarily concerning input validation and error handling. The function relies on the input string being parsable as a valid time format based on the system's locale settings. If the input string is malformed--for example, containing characters that cannot be interpreted as time separators, or representing an impossible time (like "25:00:00")--the function will trigger a runtime error (Type Mismatch, Error 13).

To create robust VBA macros, especially when dealing with external data sources, it is highly recommended to implement error trapping around the usage of TimeValue. Techniques such as using the `IsDate` function to pre-validate the string can significantly mitigate these risks. If `IsDate(Range("A" & i).Value)` returns `True`, then TimeValue is highly likely to succeed. Conversely, if `IsDate` returns `False`, the code should skip that entry or log an error instead of crashing the procedure.

Furthermore, developers should consider the locale settings where the code will run. A string formatted as "10.30" might be interpreted as 10:30 AM in a country using a period as the time separator, but it would fail or be misinterpreted in a US locale expecting a colon. For maximum portability and reliability in critical applications, it is often better to standardize the input format before calling TimeValue, or to use the more flexible `CDate` function if dealing with combined date and time stamps where the date portion must also be handled correctly, although `CDate` returns

the entire Date/Time data type, not just the time fraction. For cases strictly requiring the time fraction, TimeValue remains the most direct and efficient solution, provided proper error mitigation is in place.