VBA: How do I use MATCH function with dates?

Authored by

stats writer

November 19, 2025

The VBA (Visual Basic for Applications) MATCH function is a powerful tool borrowed from Excel's native worksheet capabilities, allowing developers to search for a specific value within a defined column or row (array or range) and return its relative position. When working with numerical or text data, this function is straightforward; however, integrating it with dates requires careful handling due to how VBA and Excel fundamentally store date values. Excel does not store dates in a human-readable format, but rather as serial numbers representing the number of days elapsed since January 1, 1900. Therefore, to ensure a successful match operation, it is paramount that both the lookup value and the range being searched are treated consistently as these underlying serial numbers, avoiding discrepancies caused by conflicting data types or formatting issues.

The functionality relies on returning the relative position of the item that satisfies the matching criteria, whether it is an exact correspondence or an approximate fit, depending on the optional `Match_Type` parameter supplied to the function. When dealing with date values specifically, achieving precision often necessitates using the exact match type. Furthermore, proper date representation is critical; if the date being searched for is provided as a simple text string, it must first be converted into the internal numeric date format that Excel utilizes for comparison. This process involves utilizing specific VBA conversion functions, such as `CDate` and `CLng`, to bridge the gap between human-readable date strings and machine-interpretable serial numbers, thereby enabling the MATCH function to operate reliably and return the correct position within the data set. Mastering these data type conversions is the key to leveraging the full potential of date lookups within your VBA projects.

Structuring the Match: Essential Syntax for Date Lookups

When attempting to search for a specific date within a defined range using VBA, developers must interface with the Excel worksheet environment using the `WorksheetFunction.Match` method. This approach ensures access to the robust searching capabilities native to Excel. The standard syntax requires three critical arguments: the lookup value, the lookup array (range), and the match type. For dates, the lookup value requires specialized treatment to convert the input string into a recognized long integer representing the date's serial number. This conversion is achieved through a combination of the CDate function, which transforms the string into a valid date data type, and the CLng function, which converts that date data type into its underlying serial number (Long integer). Using these nested functions ensures that we are comparing two identical underlying data types--the serial number of the search date against the serial numbers stored in the target cells--thereby guaranteeing accuracy in the matching process.

The code snippet below illustrates the precise structure required for initiating a date lookup within a specified range, incorporating essential error handling. The use of `CLng(CDate("Date String"))` is mandatory for reliable matching against standard Excel date formatting. Furthermore, setting the `Match_Type` parameter to `0` signifies that we are seeking an exact match, which is usually the

desired behavior when looking up a specific date. This robust approach not only executes the search efficiently but also provides a mechanism to gracefully manage scenarios where the target date is not present in the defined range, preventing the script from crashing and instead offering informative feedback to the user.

```
Sub MatchDate()

'attempt to find date in range
On Error GoTo NoMatch
MyMatch = WorksheetFunction.Match(CLng(CDate("4/15/2023")), Range("A2:A10"), 0)
MsgBox (MyMatch)
End

'if no date found, create message box to tell user
NoMatch:
MsgBox ("No Match Found")
End

End:
End Sub
```

In this particular example, the code attempts to find the date corresponding to **4/15/2023** within the cell range **A2:A10**. The variable `MyMatch` will store the relative row position if the date is successfully located. If the date is present, a message box will display the positional number, indicating precisely where in the designated range the date resides. Should the MATCH function fail to locate an exact match, the script immediately jumps to the defined error handling routine, ensuring a clean execution path and providing the user with a descriptive message rather than a run-time error. This implementation highlights the necessity of coupling the calculation logic with robust error trapping mechanisms, a fundamental practice in professional VBA development.

Understanding Date Conversion: CDate and CLng Functions

The complexity of using the MATCH function with dates stems from the internal representation of time in Excel. Excel stores every date as a sequential serial number, where integers represent the day and the fractional part represents the time of day. When you input a date as a text string (e.g., "4/15/2023") into VBA, it must be converted into this standardized serial number format before it can be accurately compared against dates stored natively in a worksheet range. Failure to perform this conversion results in a data type mismatch, causing the lookup to fail, especially when utilizing the strict exact match parameter (0).

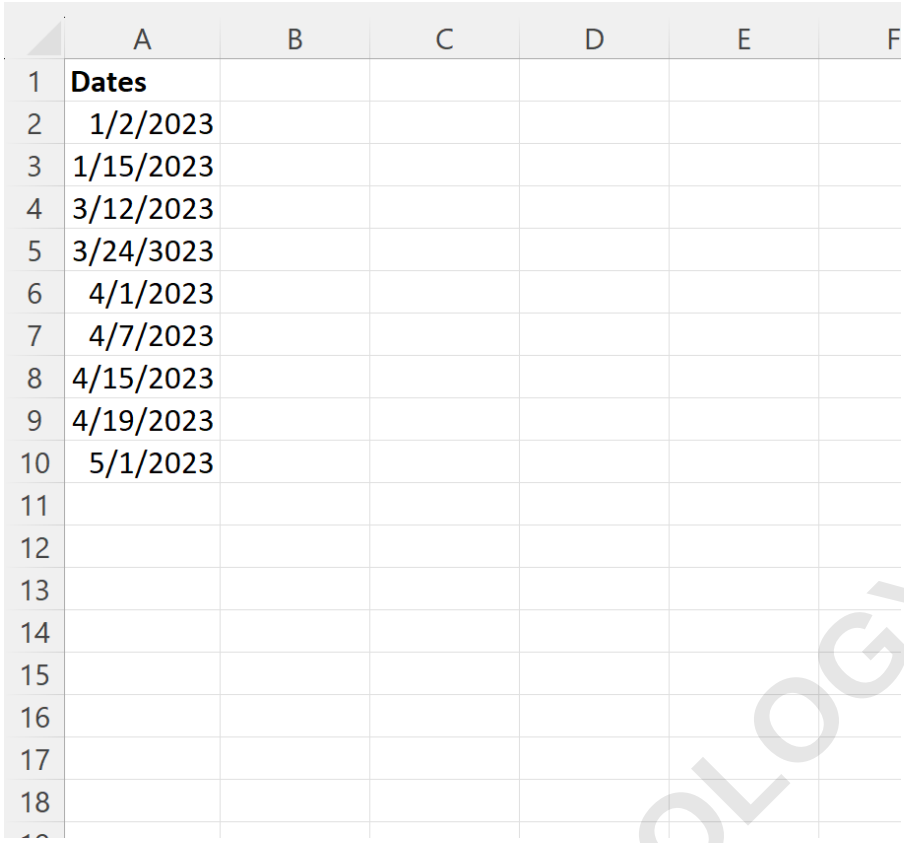
To overcome this, two crucial VBA conversion functions are employed in sequence. First, the

CDate function is used to interpret the provided text string and convert it into a VBA Date data type. This step recognizes the numerical components and separates them into their respective month, day, and year elements. Second, the CLng function (Convert to Long) is applied to the result of `CDate`. Since Excel serial dates are large integers, converting the VBA Date type into a Long Integer extracts only the whole number part of the date, effectively stripping away any potential time component (which is represented by the decimal part of the serial number). This conversion is vital because if the target cell range contains dates without time values, ensuring the lookup value also lacks a time component guarantees an exact match on the date itself.

In short, the expression `CLng(CDate("Date String"))` transforms the user-friendly date format into the underlying numerical value that the `WorksheetFunction.Match` can successfully process. This dual conversion process ensures type compatibility between the lookup argument and the values stored in the worksheet cells. By reliably converting the date search term into a Long Integer, we align the data type of the search value with the underlying type of the data stored in the Excel array, which significantly enhances the reliability and performance of date lookups within automation scripts. It is a nuanced but necessary step for any developer working extensively with temporal data in VBA.

Practical Example: Locating a Specific Date in a Range

To fully appreciate the mechanism of date matching, let us examine a specific scenario. Suppose a user has a column of dates, perhaps representing transaction times or project milestones, located in the range **A2:A10** of an active Excel worksheet. The goal is to programmatically determine the relative position of a specific date, **4/15/2023**, within this range. The following visualization depicts the data structure within the worksheet, demonstrating that the dates are stored sequentially, and the target date is visibly present within the data set.



	A	B	C	D	E	F
1	**Dates**					
2	1/2/2023					
3	1/15/2023					
4	3/12/2023					
5	3/24/3023					
6	4/1/2023					
7	4/7/2023					
8	4/15/2023					
9	4/19/2023					
10	5/1/2023					
11						
12						
13						
14						
15						
16						
17						
18						

With this dataset in place, we can construct the macro designed to execute the search. The macro initializes the search using the exact MATCH function syntax detailed previously. The script is specifically configured to look for the long integer representation of **4/15/2023** only within the boundaries of the defined range **A2:A10**. The use of error handling ensures that if the date were missing, the process would terminate gracefully without interrupting the user's workflow. This practical setup provides a real-world demonstration of how date lookups are managed effectively in automated VBA environments.

The following code block is deployed to perform the search. Note that the structure remains identical to the initial syntax demonstration, emphasizing consistency and the criticality of the CLng function and CDate function nested conversion for the lookup value. This is the core engine that permits a successful comparison between the long integer representation of the search date and the underlying serial numbers stored in the Excel cells.

Sub MatchDate()

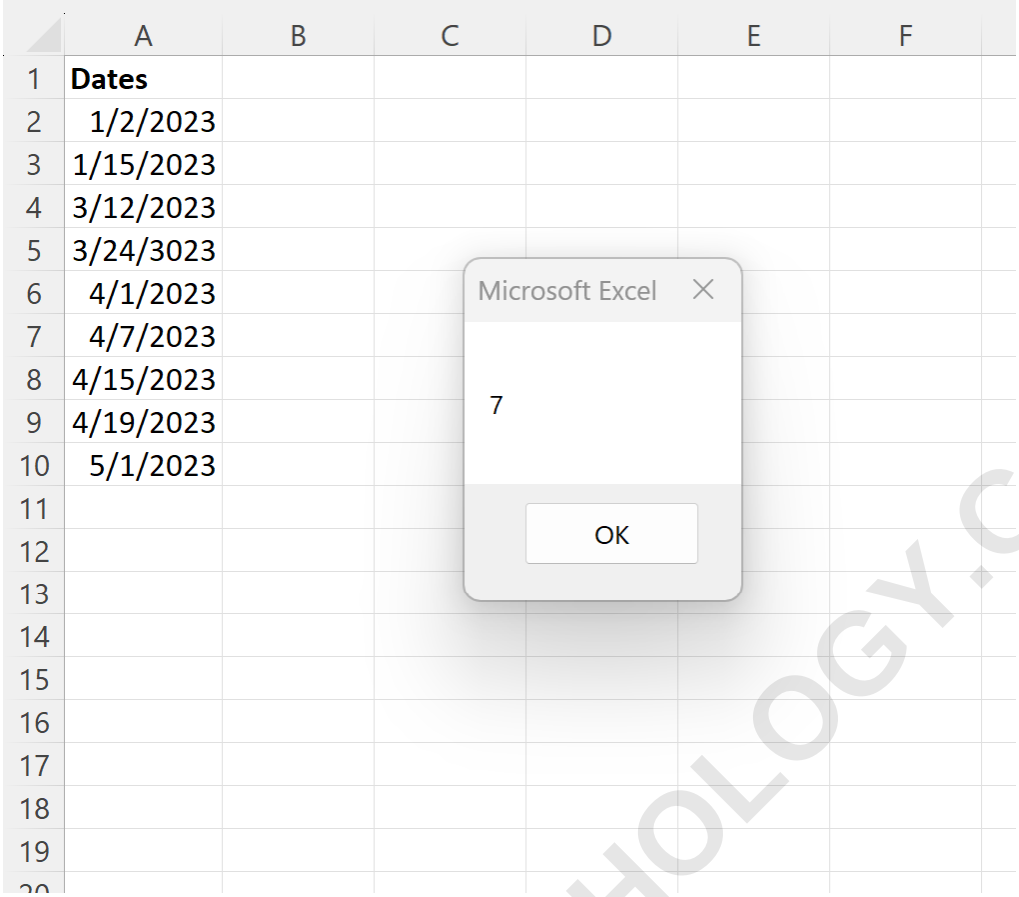
'attempt to find date in range
On Error GoTo NoMatch
MyMatch = WorksheetFunction.Match(CLng(CDate("4/15/2023")), Range("A2:A10"), 0)
MsgBox (MyMatch)

```
End

'if no date found, create message box to tell user
NoMatch:
MsgBox ("No Match Found")
End

End:
End Sub
```

Analyzing the Output of a Successful Match

Upon running the `MatchDate` macro with the sample data provided, the MATCH function successfully locates the target date, **4/15/2023**, within the range **A2:A10**. The function then returns the relative position of this date within that specified range. Since the range starts at cell A2, the first item in the range is considered position 1, the second is position 2, and so on. The output confirms that the function successfully processed the date serial number comparison and identified the location, presenting the result to the user via a message box.

When the code executes the line `MsgBox (MyMatch)`, the user receives the following visual confirmation:



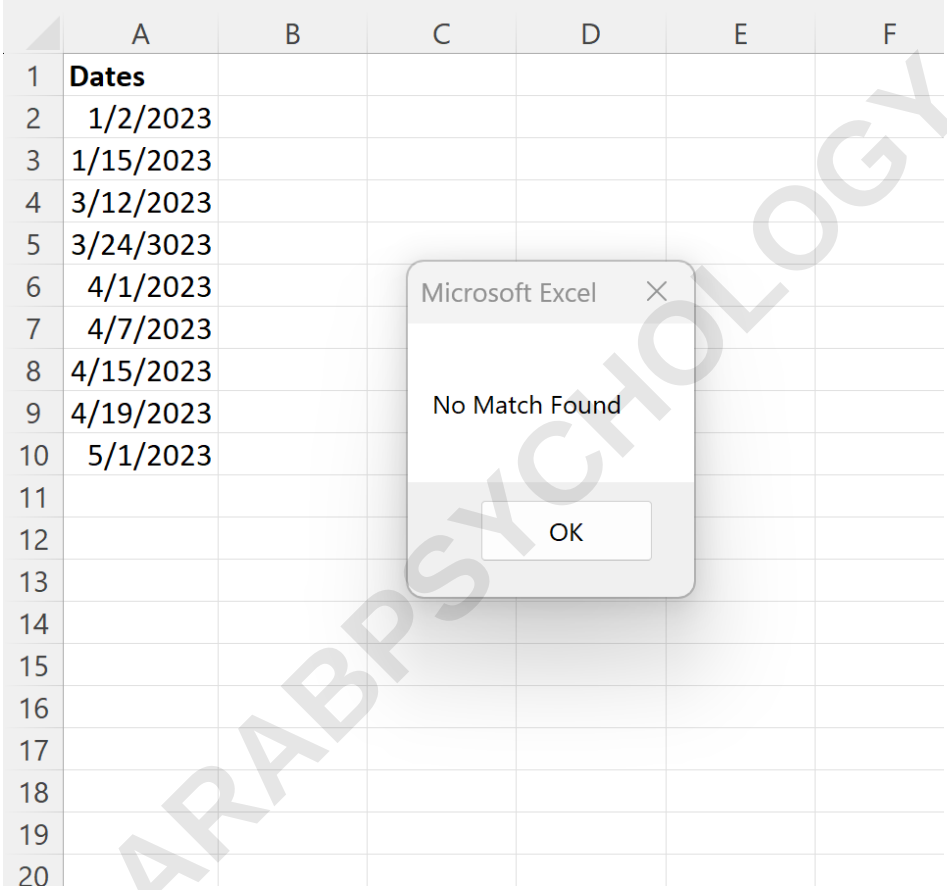
This message box clearly indicates that the target date was found in the **7th** position relative to the starting point of the search range, **A2**. If we count down the rows starting from A2 (which is position 1), the date 4/15/2023 is indeed located seven rows down, confirming the accuracy of the result. This successful outcome validates the necessity of correctly applying the CDate function and CLng function conversions, as any mismatch in data type would have inevitably resulted in the error handling routine being triggered, even if the date was visibly present in the worksheet.

Implementing Robust Error Handling with On Error GoTo

A crucial element of reliable VBA programming, especially when using worksheet functions that are prone to errors (such as `WorksheetFunction.Match` when no match is found), is the implementation of error handling. If the `WorksheetFunction.Match` method fails to find the specified lookup value, it typically generates a runtime error, halting the execution of the macro. To prevent this abrupt termination, the `On Error GoTo NoMatch` statement is included at the beginning of the macro. This command instructs VBA to bypass the standard error display and instead redirect execution to the predefined label, `NoMatch`, whenever an error occurs during the subsequent lines of code.

The section labeled `NoMatch:` is designed specifically to manage the scenario where the search date is not present in the range. Instead of crashing, the script executes the code under this label, displaying a user-friendly message: "No Match Found." This allows the user to understand the outcome without needing to debug a technical error. For instance, consider if the search date in the macro was changed to **4/25/2023**, a date not present in the range **A2:A10**. When the MATCH function attempts to execute the lookup for this non-existent value, it throws a runtime error, which is immediately intercepted by our error handler.

In the case of a failed lookup, the flow of execution jumps to the error label, resulting in the following output:



This result elegantly confirms that the specified date was not located. Including such error trapping is not merely optional; it is a fundamental requirement for creating stable and professional VBA applications that interact with potentially unpredictable worksheet data. It ensures that regardless of the data quality, the program responds predictably and helpfully.

Exploring Match Types for Date Ranges

While the exact match (using `0` as the third argument) is generally recommended for finding a

precise date, the MATCH function offers two other match types, which are particularly useful when dealing with date ranges or approximate lookups. The `Match_Type` parameter can accept three values: `0` (Exact Match), `1` (Less Than or Equal To), and `-1` (Greater Than or Equal To). Understanding the implications of these different match types is vital for advanced searching in VBA.

When using `1` (Less Than or Equal To), the function finds the largest value that is less than or equal to the lookup value. This requires the lookup array to be sorted in ascending order. When applied to dates, this allows a search for the most recent date occurring on or before the target date. Similarly, using `-1` (Greater Than or Equal To) requires the array to be sorted in descending order and finds the smallest value that is greater than or equal to the lookup value, effectively finding the earliest date occurring on or after the target date. Both approximate match types leverage the numerical nature of the date serial numbers, making them highly effective for time-series analysis where finding the closest boundary date is necessary.

It is important to reiterate that regardless of the match type chosen, the conversion of the lookup date using `CLng(CDate("Date"))` remains essential to ensure that the comparison is performed reliably against the underlying numerical serial numbers of the dates stored in the Excel range. Using `0` is safest for basic lookups, but advanced date operations greatly benefit from the flexibility offered by `1` and `-1`, provided the data is correctly sorted as a prerequisite for their operation.

Conclusion and Further Resources

Successfully utilizing the MATCH function with dates in VBA hinges entirely on recognizing and managing Excel's internal representation of dates as serial numbers. The crucial takeaway is the necessity of consistent data type conversion. By employing the CDate function followed by the CLng function, developers can reliably transform a text string into the precise numerical serial date required for accurate searching against worksheet data. This technique, coupled with robust error handling via `On Error GoTo`, ensures that your VBA code is both precise and resilient, delivering accurate results while managing potential lookup failures gracefully.

For those seeking to deepen their understanding of this powerful method and explore its full range of capabilities, including detailed documentation on optional parameters and specific return values, we recommend consulting the official Microsoft documentation. Mastering these techniques transforms date management in VBA from a potential frustration point into a reliable component of automated data processing workflows.

Note: You can find the complete documentation for the **Match** method in VBA here: WorksheetFunction.Match Method (Excel)

Resource Links and Documentation

For reference and further study, the following authoritative resources provide complete details on the functions and methods discussed in this guide:

VBA WorksheetFunction.Match Method
VBA CDate Function Official Documentation
VBA CLng Function Official Documentation
Excel Date Systems Explained