How to Easily Convert a String to a Long Number in VBA

Authored by

stats writer

November 20, 2025

The process of converting one data type to another is a fundamental requirement in most programming environments, and VBA is no exception. Specifically, when handling numeric data sourced from spreadsheets or external files, it is often necessary to explicitly cast a text string into a numeric format suitable for calculation. In VBA, functions such as **CLng** (Convert to Long) and **CDbl** (Convert to Double) are provided for this purpose, allowing developers to precisely control the underlying representation of data.

While several functions exist for conversion, **CLng** is specifically designed to convert an expression into the Long data type. Both **CLng** and **CDbl** accept the source string as their required argument and return the calculated numeric value. It is critical to understand, however, that these functions demand strictly numeric input; if the provided argument cannot be interpreted as a number--for example, if it contains alphabetical characters or complex formatting--the function will execute a runtime error, potentially halting the entire macro.

Understanding the implications of type conversion and implementing robust error-checking mechanisms are paramount to writing reliable VBA code. The following guide explores the **CLng** function in depth, providing two proven methods for safely converting string values to the Long data type, complete with practical examples illustrating best practices for handling varied input data.

Understanding the Long Data Type in VBA

Before executing any conversion, it is essential to understand the target data type. The Long data type is a 32-bit integer, capable of storing whole numbers ranging from approximately -2,147,483,648 to 2,147,483,647. This substantial range makes the **Long** type the standard choice for handling large integer counts, indices, or numerical identifiers within VBA applications, often replacing the older 16-bit **Integer** type which has a much smaller capacity and is prone to overflow errors when dealing with modern spreadsheet sizes.

When you convert a string to a Long data type, you are essentially instructing VBA to interpret the sequence of characters as a whole number. If the string contains a fractional component (e.g., "123.45"), the **CLng** function will automatically round the value to the nearest whole number before assigning it to the **Long** variable. This automatic rounding behavior is another critical characteristic that developers must account for during precision-sensitive conversions.

The primary tool for this specific conversion is the **CLng** function. Using this function allows for explicit type casting, providing clarity and potentially improving performance compared to implicit conversion methods, which occur automatically but can sometimes lead to unexpected results. Utilizing **CLng** ensures the developer maintains strict control over how the string input is processed and stored as a numeric value within the VBA environment.

The CLng Function: Syntax and Core Application

The **CLng** function is one of several C-type conversion functions available in VBA (including **CInt**, **CStr**, CDbl, etc.). The syntax is straightforward: `CLng(expression)`. The `expression` argument is the value you wish to convert, which is typically a string variable or a reference to a cell containing text data.

A key difference between **CLng** and related functions like CDbl is the resulting format. CDbl converts the expression to a **Double** data type, which is a floating-point number suitable for calculations requiring high precision (including decimal places). Conversely, **CLng** strictly yields a 32-bit integer, truncating or rounding any fractional part of the input. Developers should choose the conversion function that aligns precisely with the required output format and memory usage goals.

It must be reiterated that **CLng** will fail if the input string contains non-numeric characters (excluding valid signs and decimal separators) or if the resulting numeric value falls outside the acceptable range of the Long data type. This potential for runtime errors necessitates the implementation of control flow structures or explicit error handling, which we address in the following methods, moving beyond simple, direct conversion to a more robust approach.

Method 1: Implementing Direct String to Long Conversion

The first and simplest approach to converting a string to a **Long** is by applying the **CLng** function directly to the input. This method assumes that the input data set is clean and all entries are guaranteed to be numeric, or that any fractional values can be safely rounded. This technique is fast and succinct, making it ideal for controlled environments where data validation has already occurred prior to execution of the conversion macro.

In the common scenario of converting a range of values in an Excel worksheet, we typically iterate through the cells and apply the conversion within a loop. The following macro demonstrates this direct approach, reading values from column A and writing the converted Long data type result into the corresponding cell in column B:

```
Sub ConvertStringToLong()

Dim i As Integer

For i = 2 To 11
Range("B" & i) = CLng(Range("A" & i))
Next i

End Sub
```

This particular subroutine efficiently iterates from row 2 through row 11. In each iteration, it retrieves the value from the cell in column A, uses the **CLng** function to convert that value from a string (or implicit variant) into a Long data type, and then places the resulting number into the corresponding cell in column B. If any cell in the range **A2:A11** contains non-numeric text, this macro will generate a runtime error, highlighting the necessity of the next method for real-world data processing.

Method 2: Robust Conversion with Numeric Validation

When dealing with user input, imported spreadsheet data, or any data source where the numeric integrity of the string cannot be guaranteed, relying solely on direct conversion (Method 1) is risky. A much safer and more professional approach involves using the **IsNumeric** function before attempting the conversion. The IsNumeric function checks if an expression can be evaluated as a number; if it returns **True**, conversion is safe; if **False**, the input should be handled gracefully to prevent errors.

This conditional approach utilizes an `If...Then...Else` block to execute the conversion only when validated. If the cell value is numeric, the **CLng** conversion proceeds as normal. If the value is not numeric, we can specify an alternative action, such as logging the error, skipping the conversion, or, as demonstrated below, assigning a default value like zero (`0`) to the output cell, thus ensuring the macro runs to completion without interruption.

The following macro incorporates this vital validation step, making the code resilient to mixed data type inputs:

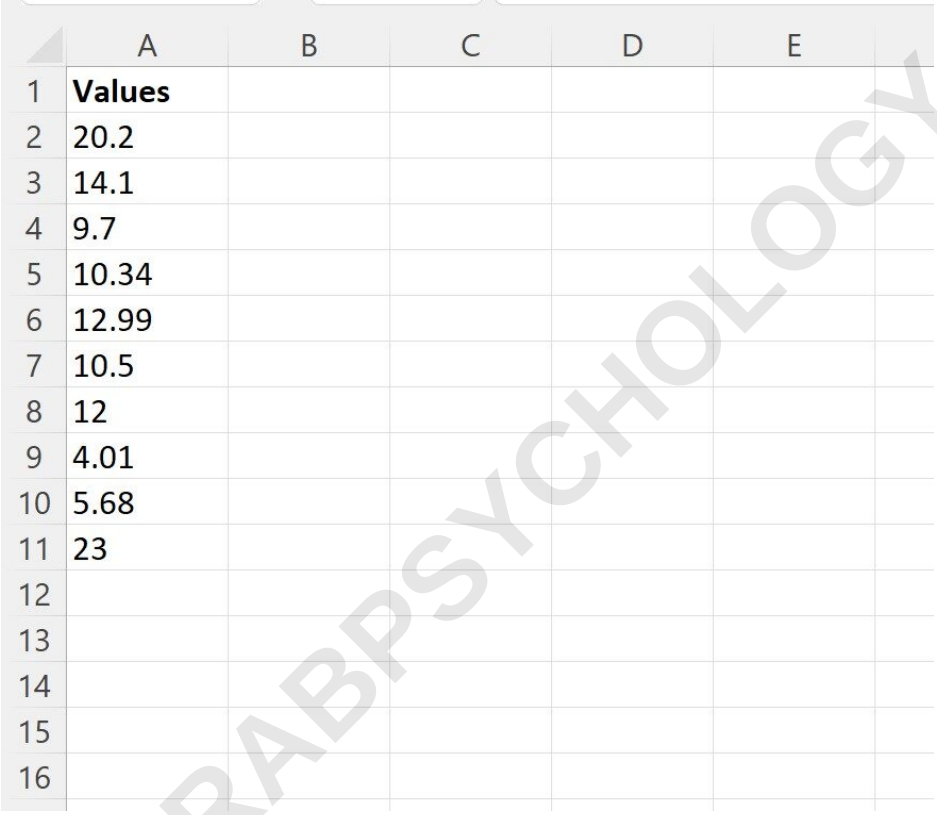
```
Sub ConvertStringToLong()

Dim i As Integer

For i = 2 To 11
If IsNumeric(Range("A" & i)) Then
Range("B" & i) = CLng(Range("A" & i))
Else
Range("B" & i) = 0
End If
Next i

End Sub
```

This implementation dictates that the conversion to the Long data type will only occur if the cell contents in column A pass the numeric check. In all other cases (e.g., if the cell contains "N/A" or

"Error"), the output cell in column B is explicitly set to zero. This ensures stable execution and provides a clear output value for subsequent processing, avoiding the abrupt termination associated with runtime errors.

Case Study 1: Direct Conversion of Guaranteed Numeric Strings

Consider a scenario where we have a column of values in an Excel worksheet that are currently formatted as text strings, perhaps imported from a CSV file. We know definitively that all these values represent integers, but they must be converted to the Long data type for use in specific calculations within VBA. The input data, covering rows 2 through 11, looks like this:



	A	B	C	D	E
1	**Values**				
2	20.2				
3	14.1				
4	9.7				
5	10.34				
6	12.99				
7	10.5				
8	12				
9	4.01				
10	5.68				
11	23				
12					
13					
14					
15					
16					

Since we are confident in the numeric quality of the input data, we can safely apply the direct conversion approach (Method 1) to convert each string to a **Long** and display the result in the corresponding row of column B. This approach minimizes code complexity while achieving the desired type cast. We will use the following macro:

Sub ConvertStringToLong()

Dim i As Integer

For i = 2 To 11

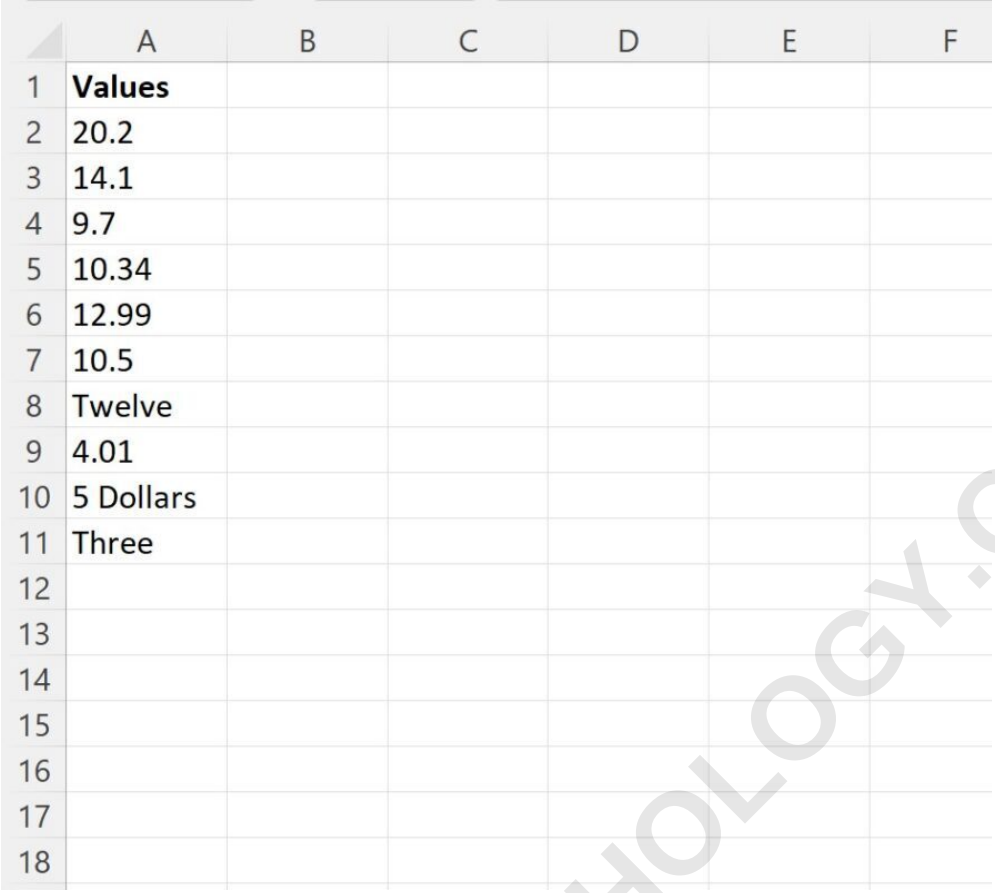
```
Range("B" & i) = CLng(Range("A" & i))
Next i

End Sub
```

Upon execution of this macro, the conversion is successful across all rows. The values in column B are now stored as true numeric Long data type integers, ready for computational use. The output confirms that the direct application of **CLng** handled the transformation flawlessly when the input data was clean:

	A	B	C	D	E	F
1	**Values**					
2	20.2	20				
3	14.1	14				
4	9.7	10				
5	10.34	10				
6	12.99	13				
7	10.5	10				
8	12	12				
9	4.01	4				
10	5.68	6				
11	23	23				
12						
13						
14						
15						
16						
17						

Case Study 2: Conditional Conversion and Error Mitigation

Now consider a more complex scenario where the input column contains a mixture of valid numeric strings, fractional numbers, and explicit text entries that are not numerical identifiers. This is common when processing raw user inputs or integrating data from heterogeneous sources. Our initial column, formatted as text strings, now contains entries like "Text A" and "123.45":



	A	B	C	D	E	F
1	**Values**					
2	20.2					
3	14.1					
4	9.7					
5	10.34					
6	12.99					
7	10.5					
8	Twelve					
9	4.01					
10	5 Dollars					
11	Three					
12						
13						
14						
15						
16						
17						
18						

To safely convert these values, we must ensure that only the text strings that represent actual numbers are processed by **CLng**. The objective is to convert these numeric strings to the Long data type, while setting any non-numeric entry to zero, thereby maintaining the stability of our macro. We employ Method 2, utilizing the IsNumeric function for rigorous validation:

Sub ConvertStringToLong()

Dim i As Integer

For i = 2 To 11
If IsNumeric(Range("A" & i)) Then
Range("B" & i) = CLng(Range("A" & i))
Else
Range("B" & i) = 0
End If
Next i

End Sub

Running this validated macro produces the following output. Note how the entries "Text A" (in row 5) and "ERROR" (in row 10) are successfully ignored by the **CLng** function and instead default to the assigned value of zero in column B. Additionally, the fractional value "123.45" (in row 6) is correctly rounded down to the nearest integer, 123, due to the nature of the **CLng** function.

	A	B	C	D	E	F
1	**Values**					
2	20.2	20				
3	14.1	14				
4	9.7	10				
5	10.34	10				
6	12.99	13				
7	10.5	10				
8	Twelve	0				
9	4.01	4				
10	5 Dollars	0				
11	Three	0				
12						
13						
14						
15						
16						
17						

Only the text strings in column A that are definitively numeric are converted to the Long data type in column B. Otherwise, the non-numeric strings are gracefully converted to a value of zero, thereby successfully mitigating potential runtime conversion errors.

Advanced Considerations in VBA Type Casting

While **CLng** is the definitive function for explicitly converting to the Long data type, developers sometimes rely on alternative conversion methods. For instance, the **Val** function can extract a number from the beginning of a string, ignoring subsequent non-numeric characters. However, **Val** is often less reliable for strict type checking than **CLng**, especially since it does not raise an error on truncation or non-numeric input, potentially masking data issues.

For inputs that might exceed the range of the **Long** data type (i.e., numbers greater than 2.1 billion), developers should use **CDec** (Convert to Decimal) or CDbl (Convert to Double). If the desired output is an extremely large whole number, the **CDec** function provides the highest

precision and range for integer values within VBA.

Finally, for mission-critical applications where data integrity must be absolute, relying solely on **IsNumeric** might not suffice. A more advanced strategy involves implementing explicit error trapping using the `On Error GoTo` statement. This approach captures the specific error code raised by the **CLng** function, allowing the developer to differentiate between a non-numeric input error (Type Mismatch) and an overflow error (Number Out of Range), providing a higher degree of control over the conversion process.

Note: You can find the complete documentation for the VBA **CLng** function on the Microsoft documentation website.