How to use IsError Function in VBA (With Example)

Authored by

stats writer

November 18, 2025

Introduction to Error Handling in VBA and the IsError Function

Effective data validation and VBA programming rely heavily on robust error checking. When automating tasks in Microsoft Excel, it is common to encounter situations where formulas or operations result in calculation errors, known as error values (e.g., #DIV/0! or #REF!). Ignoring these errors can lead to runtime crashes in macros or result in the processing of inaccurate data. The native IsError function within VBA serves as a critical utility for preemptively identifying and managing these spreadsheet-generated issues.

The primary purpose of the **IsError** function is straightforward yet powerful: it determines if a specified expression evaluates to any of the standard Excel error types. This function returns a simple Boolean result--**TRUE** if the expression is an error, and **FALSE** otherwise. This clear binary output allows developers to implement conditional logic, such as skipping complex calculations on corrupted data points or flagging the cell location for later manual review. Understanding its proper implementation is fundamental for writing resilient and production-ready automated code.

By integrating **IsError** into your procedures, you ensure that your code handles data imperfections gracefully, preventing unexpected halts. This function acts as a safety barrier, allowing the rest of the subroutine to execute cleanly even when the source data contains various calculation flaws. This proactive approach elevates the quality and reliability of any data processing solution built using VBA.

Understanding the Syntax and Parameters of IsError

The syntax for the **IsError** function is unambiguous, adhering to the standard structure of VBA's built-in data inspection functions. It requires only one mandatory argument, which is the expression, variable, or cell value being evaluated. Since we are frequently checking cell contents within an Excel application, we typically access the function through the WorksheetFunction object when we need to mirror the behavior of the native Excel formula ISERROR.

The general syntax structure is defined as: `IsError(expression)`. The `expression` argument must resolve to a scalar value that the function can inspect. When dealing directly with cell ranges and assigning the result back into the worksheet, calling it through WorksheetFunction is the most reliable method, as it correctly translates the cell's error state into a VBA-compatible Boolean value. This ensures compatibility and predictable behavior when auditing spreadsheet data.

The following example illustrates how to correctly use the function within a loop to systematically validate a range of cell entries, effectively transforming the error status into a simple **TRUE** or **FALSE** indicator in an adjacent column:

Sub CheckIsError()

```
Dim i As Integer

For i = 2 To 11
Range("B" & i).Value = WorksheetFunction.IsError(Range("A" & i))
Next i

End Sub
```

This specific macro iterates through rows 2 through 11, checking the value of each cell in column A. The resulting Boolean output is then assigned to the corresponding cell in column B. This mechanism provides immediate, easily digestible feedback regarding the presence of error values across the specified range **A2:A11**, allowing developers to quickly pinpoint issues.

Why Use IsError? Common Error Values in Excel

The necessity of utilizing specialized inspection functions like **IsError** becomes paramount when processing datasets derived from complex financial models or large-scale data imports, where data integrity cannot be guaranteed. These Excel errors typically arise from invalid mathematical operations, referencing data that no longer exists, or supplying arguments of the wrong data type to a formula. By checking for these errors first, programmers can prevent runtime errors in VBA that might occur if the code attempts to perform numeric operations on an error string.

Excel defines a specific set of standard error values, all of which are recognized and result in a **TRUE** response from the **IsError** function. It is crucial to distinguish these structured error types from simple data validation failures or text values that merely look like errors. The function specifically targets calculation outputs designated by Excel as faults.

The comprehensive list of Excel error values that trigger **TRUE** when evaluated by **IsError** includes:

#DIV/0!: Caused by dividing a number by zero or an empty cell.
#VALUE!: Results from using a text argument in a function that expects a number or a range.
#NUM!: Occurs when an invalid number is used in a formula or when a function cannot find a valid result (e.g., trying to calculate the square root of a negative number).
#REF!: Indicates that a cell reference is invalid, often because the row or column containing the referenced cell was deleted.
#N/A: Signifies 'Not Available' and is often returned by lookup functions (like VLOOKUP) when the specified value cannot be found.
#NAME?: Appears when Excel does not recognize a formula name or text used in a formula (e.g., mistyping SUM as SMM).
#NULL!: Arises when two areas intersect but do not overlap, usually due to an incorrect range

operator.

Because **IsError** captures all these specific failure modes, it provides a single, unified method for checking the health of your spreadsheet data before attempting further manipulation.

Practical Application: Integrating IsError into a VBA Macro

To grasp the utility of **IsError**, let us examine a typical scenario where we must audit data containing calculated results. Imagine a spreadsheet where column A contains outputs from complex formulas, some of which invariably lead to errors due to dependencies on incomplete or flawed raw data. Using a dedicated VBA macro is the most scalable way to perform this validation, especially when processing datasets far larger than our example.

Our initial setup requires defining the input range (column A, rows 2 to 11) and the designated output range (column B, rows 2 to 11) where the validation results will be logged. This side-by-side reporting mechanism is essential for immediate visual feedback and facilitates rapid identification of problem rows. By automating this check, we eliminate the need for manual formula entry in the worksheet, making the process cleaner and faster.

Consider the provided initial dataset. Notice the mix of valid numbers, text entries, and various calculation errors in column A:

	A	B	C	D	E
1	**Values**				
2	12.5				
3	#DIV/0!				
4	15				
5	19				
6	22				
7	#VALUE!				
8	50				
9	#NUM!				
10					
11	13.2				
12					
13					
14					
15					
16					

Our objective is to execute the audit using the macro, yielding a column B populated only with

TRUE or **FALSE** flags that correspond directly to whether the adjacent cell in column A holds an error value. This validation process converts complex error reporting into a simple, standardized Boolean format suitable for subsequent automated decision-making.

Step-by-Step Example Walkthrough

The core of our validation process is the iterative procedure implemented via the `For...Next` loop in VBA. This ensures that every cell within the specified range receives an individual assessment by the **IsError** function, preventing any data point from being overlooked.

We begin by declaring the integer variable `i` to manage the row count. The loop is explicitly set to cover rows 2 through 11. Within this loop, the critical line of code performs the validation and assignment: `Range("B" & i).Value = WorksheetFunction.IsError(Range("A" & i))`. This structure is highly efficient for transferring the error status from one column to another.

The complete macro used to execute this audit is provided below:

Sub CheckIsError()

Dim i As Integer

For i = 2 To 11
Range("B" & i).Value = WorksheetFunction.IsError(Range("A" & i))
Next i

End Sub

When this subroutine runs, the WorksheetFunction.**IsError** method evaluates the content of cell A&i. If the cell contains an error (such as #DIV/0!), the function returns the Boolean value **TRUE**. Conversely, if the cell contains any valid data type, including zero, numbers, dates, or text strings, the function returns **FALSE**. This result is immediately written into the corresponding row of column B, delivering the validation map.

Analyzing the Results and Interpreting Boolean Output

After the macro successfully completes execution, column B is fully populated with **TRUE** or **FALSE** outcomes, providing a clear binary representation of the data quality in column A. This binary output simplifies filtering, conditional formatting, and subsequent data cleansing operations within the spreadsheet environment.

The resulting output, with the validation flags placed in column B, appears as follows:

	A	B	C	D	E
1	**Values**				
2	12.5	FALSE			
3	#DIV/0!	TRUE			
4	15	FALSE			
5	19	FALSE			
6	22	FALSE			
7	#VALUE!	TRUE			
8	50	FALSE			
9	#NUM!	TRUE			
10		FALSE			
11	13.2	FALSE			
12					
13					
14					
15					
16					
17					
18					

In this final state, the values in column B serve as definitive indicators. A **TRUE** result confirms the presence of an Excel error value in column A, thereby indicating a flaw in the underlying calculation or data. A **FALSE** result confirms that the adjacent cell contains valid data that can be safely processed by subsequent VBA code.

Specifically, the following rows were flagged with a **TRUE** output, correctly identifying the calculation errors:

The cell containing **#DIV/0!** returned **TRUE**.
The cell containing **#VALUE!** returned **TRUE**.
The cell containing **#NUM!** returned **TRUE**.
The cell containing **#N/A** returned **TRUE**.

All other values--including the numerical entries and text strings--correctly returned **FALSE**, confirming they are not flagged as error values by Excel's internal system. This powerful visual audit capability is a foundational element of data governance in automated reporting systems.

Comparing IsError with Other VBA Error Checking Methods

While **IsError** is highly effective for identifying Excel calculation errors within cells, it is essential for

developers to understand its place relative to other error handling structures in VBA. The two main categories of error management are data inspection (checking existing cell values) and runtime handling (managing code execution faults).

The distinction between the **WorksheetFunction.IsError** and other inspection functions like `IsErr` is subtle but important. While both `IsError` (when used as a worksheet function via WorksheetFunction) and the standard VBA `IsErr` function check for error states, the former is specialized to recognize the specific set of seven standard Excel error strings, making it the preferred tool for auditing spreadsheet data integrity. Relying on the worksheet function interface ensures consistent behavior with the Excel application environment.

More critically, **IsError** must not be confused with the mandatory runtime error handling statement `On Error GoTo`. The `On Error GoTo` structure is used to manage unexpected errors that occur during the execution of the VBA code itself (e.g., trying to divide by zero within the macro, attempting to access a file that doesn't exist, or object instantiation failures). In contrast, **IsError** is a data validation function; it detects errors that already reside in the cells, allowing the code to proceed logically. A robust VBA application typically employs both: `On Error GoTo` to prevent macro crashes, and **IsError** to gracefully handle flawed input data.

Advanced Use Cases and Best Practices for Robust Code

Strategic integration of **IsError** significantly improves code robustness, particularly in complex data pipelines. An advanced use case involves embedding the **IsError** check within an `If...Then...Else` block to execute complex error mitigation logic rather than simply reporting **TRUE** or **FALSE** to the worksheet. For instance, if an error is detected, the code can skip a calculation step, substitute a predetermined default value, or trigger a detailed logging routine that captures the error type and timestamp for later analysis.

A powerful pattern for data aggregation utilizes this conditional structure:

Check the input cell value using `If WorksheetFunction.IsError(TargetCell.Value) Then`.
If **TRUE**, execute remediation steps, such as setting the output to zero or incrementing an error counter variable for summary reporting.
If **FALSE**, proceed with the main business logic (e.g., calculation, formatting, or transferring data).

This methodology ensures that calculation errors in source data are quarantined and do not poison downstream results, resulting in much cleaner and more reliable output tables. Furthermore, because **IsError** returns a clean Boolean, it can be easily used in conjunction with advanced filtering techniques or conditional logic structures involving arrays and dictionaries for high-speed data processing.

For developers seeking to utilize the function in the most efficient manner, it is a best practice to pair **IsError** with an appropriate error display or handling mechanism, such as formatting the erroneous cell red, or writing a specific descriptive note into an adjacent column, rather than just the generic **TRUE** flag. This adds valuable context to the data audit, speeding up the manual correction phase.