How to Trigger Actions in Excel Based on Cell Color

Authored by

stats writer

February 8, 2026

The ability to trigger an automated action within a spreadsheet based on the visual attributes of a cell--specifically its fill color--is a highly sought-after capability in Excel. While built-in formulas do not directly read formatting data, this functionality is critical for efficient and automated data processing, particularly when data is highlighted using conditional formatting. Determining if a cell is red, for example, can instantly flag important or problematic data, significantly improving the accuracy and effectiveness of subsequent data analysis and manipulation steps.

Excel Formula: If Cell Color is Red Then Do Something

Understanding the Challenge of Cell Color Detection

Many intermediate and advanced Excel users frequently encounter the need to base a logical test on cell formatting, such as color. Standard worksheet functions, including the versatile IF statement, are designed to analyze data values (numbers, text, dates) but cannot directly retrieve non-data properties like background color or font style. This inherent limitation prevents formulas from executing actions simply because a cell is, for instance, filled with red.

To bypass this restriction and introduce formatting-aware logic into your spreadsheet, we must employ specialized, non-standard methods. The most reliable approach involves leveraging Excel's older macro language functions, specifically the `GET.CELL` function, which can be deployed through the modern interface using the Defined Name feature. This process effectively bridges the gap between formatting and formula logic, enabling precise control over data processing based on visual cues.

The Solution: Leveraging Defined Names and GET.CELL

The key to successfully querying a cell's color lies in the function `GET.CELL`. This function is part of the legacy Excel 4.0 Macro language, which, while obsolete for scripting, remains functional within the context of defined names for retrieving specific cell properties. We must wrap this function within a Defined Name because you cannot enter it directly into a standard cell formula.

The specific syntax we will use is `=GET.CELL(38, Sheet1!A2)`. The number **38** is a crucial argument for `GET.CELL`; it instructs the function to return the background fill color code (an integer) of the referenced cell. By setting this up as a named range, we create a pseudo-function that can be copied down a column, dynamically returning the color code for each corresponding cell.

Step 1: Setting up the Sample Data

For this practical demonstration, assume we are analyzing a roster of basketball players in Excel. We have already manually or programmatically applied a red background fill to the cells in column

A that correspond to players who have been designated as "All-Stars." Our goal is to automatically populate column B with the text "All-Star" or "Not All-Star" based solely on whether the adjacent cell in column A is red.

The initial dataset appears as follows:

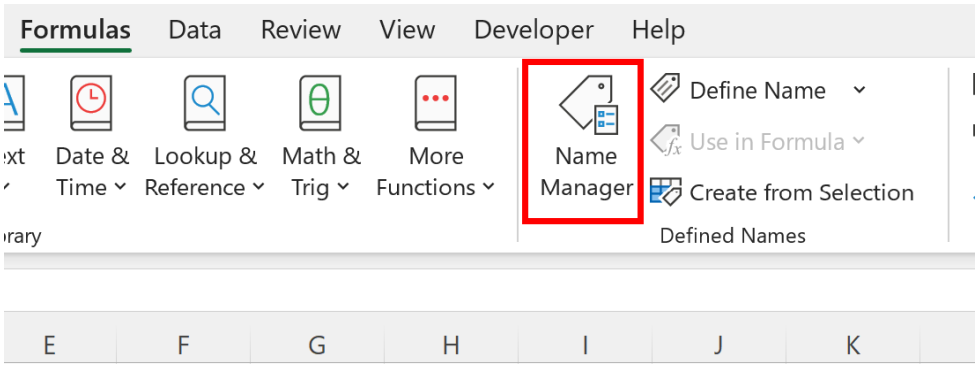
	A	B	C	D	E
1	**Athlete**				
2	Andy				
3	Bob				
4	Chad				
5	Doug				
6	Eric				
7	Frank				
8	Greg				
9	Henry				
10	Isaac				
11	John				
12	Kendall				
13	Luke				
14					
15					
16					

Step 2: Defining the Custom Color-Checking Name

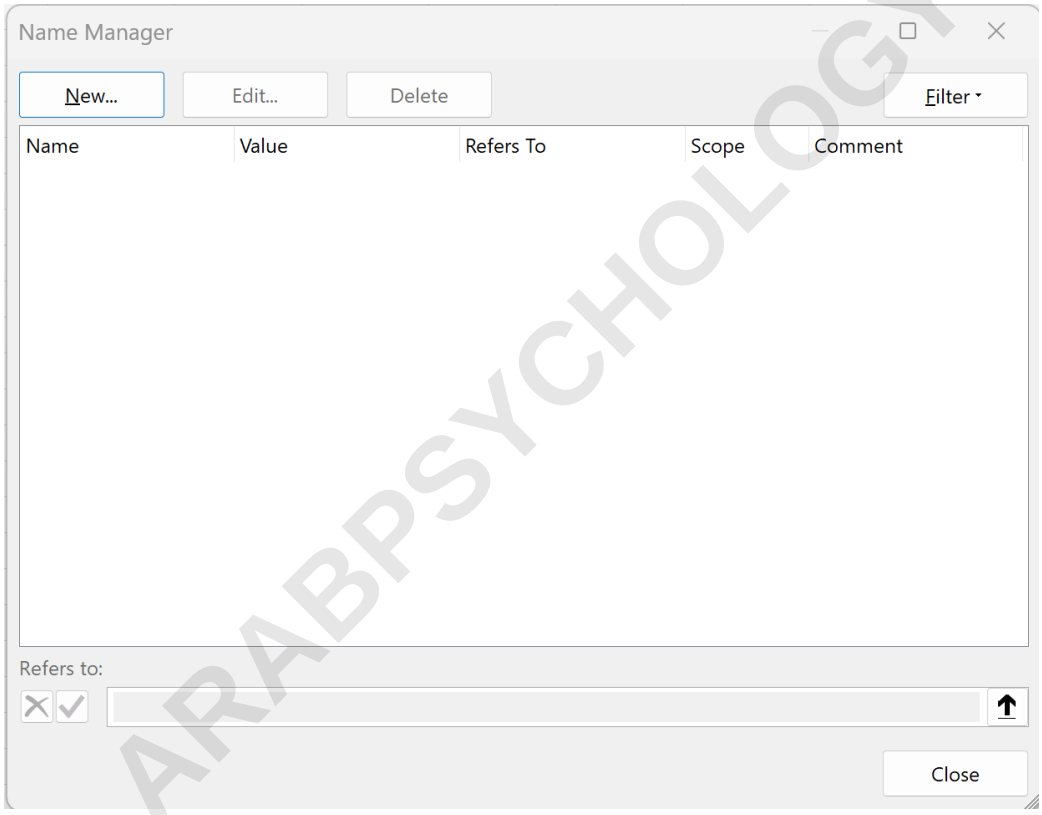
The first step in implementing this solution is navigating to the Name Manager to create our custom function.

Click the **Formulas** tab located in the top ribbon interface.
Select the **Name Manager** icon from the Defined Names group.

This action opens the Name Manager dialog box, which allows for the creation, editing, and deletion of named ranges and custom functions.



Within the Name Manager, proceed by clicking the **New** button located in the top left corner of the window. This will prompt the creation of a new Defined Name entry.



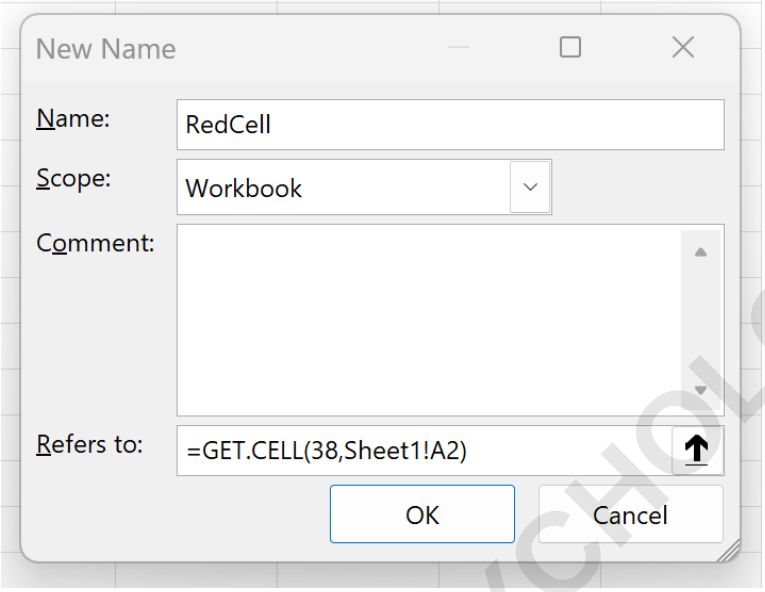
Step 3: Configuring the GET.CELL Function

In the "New Name" dialog box that appears, careful configuration is required to ensure the macro function operates correctly. This step is critical as it establishes the linkage between the defined name and the cell's color property.

In the **Name** field, type the descriptive name: **RedCell**.

In the **Refers to** box, input the following macro function: `=GET.CELL(38,Sheet1!A2)`.

It is vital to note the relative referencing utilized in the formula: `Sheet1!A2`. When defining a name that uses `GET.CELL`, the reference cell (A2 in this case) should be the first cell in the range you intend to evaluate (A2 is the first player name). Furthermore, the reference must be **relative** (i.e., not locked with dollar signs, like A2). If the dialog was opened while cell B2 was active, this relative reference ensures that when `RedCell` is used in B2, it correctly references A2, and when dragged to B3, it references A3, and so on.

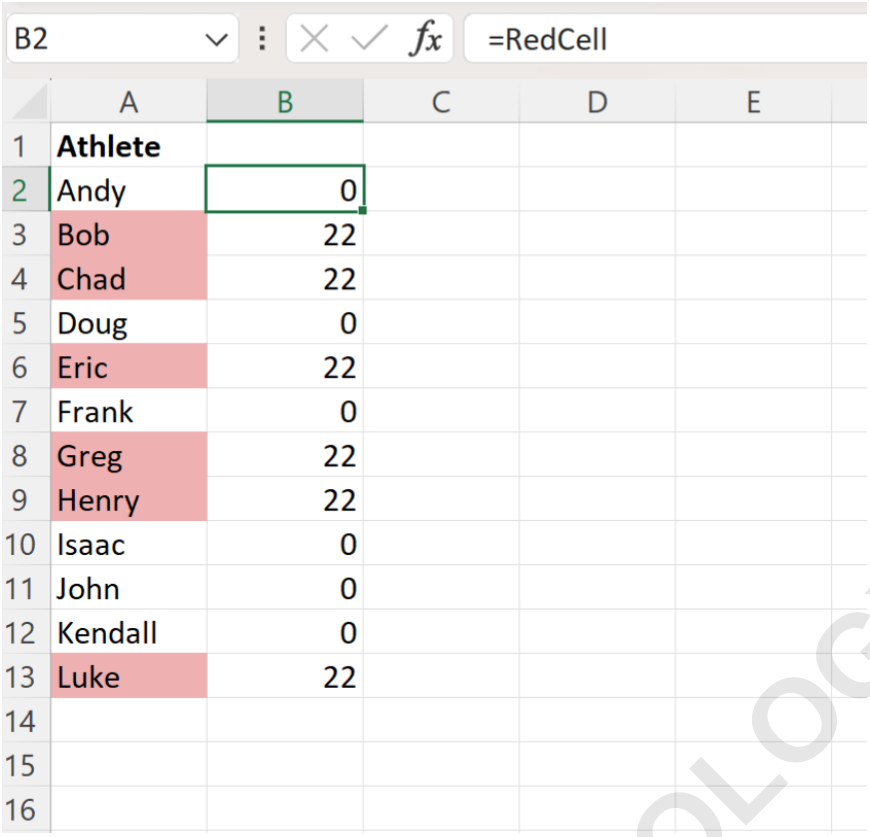


After confirming the settings by clicking **OK**, the custom name `RedCell` is now available to be used as a function within your Excel sheet.

Step 4: Determining the Specific Color Code

Before we can build the final logical test, we must determine the exact color code assigned to the specific shade of red used in column A. Excel uses a numerical system for colors, and different shades (even seemingly identical reds) often correspond to different integer codes.

To extract the code, simply type `=RedCell` into cell **B2**. Since `RedCell` is now defined as `=GET.CELL(38, A2)`, this formula retrieves the background color code of A2. Next, click and drag this formula down to apply it to all corresponding cells in column B.



By examining the results in column B, we observe that for every player whose name in column A is highlighted red, the formula returns the integer value **22**. This means that for the specific red shade used in this example, the color code is 22. Any cell that is not filled (or filled with white) will typically return a value of 0.

Step 5: Implementing the Final IF Statement Logic

With the color code identified, we can now construct the final, robust IF statement. This formula will utilize the custom `RedCell` function to perform a conditional test against the known color code (22) and return the appropriate text label.

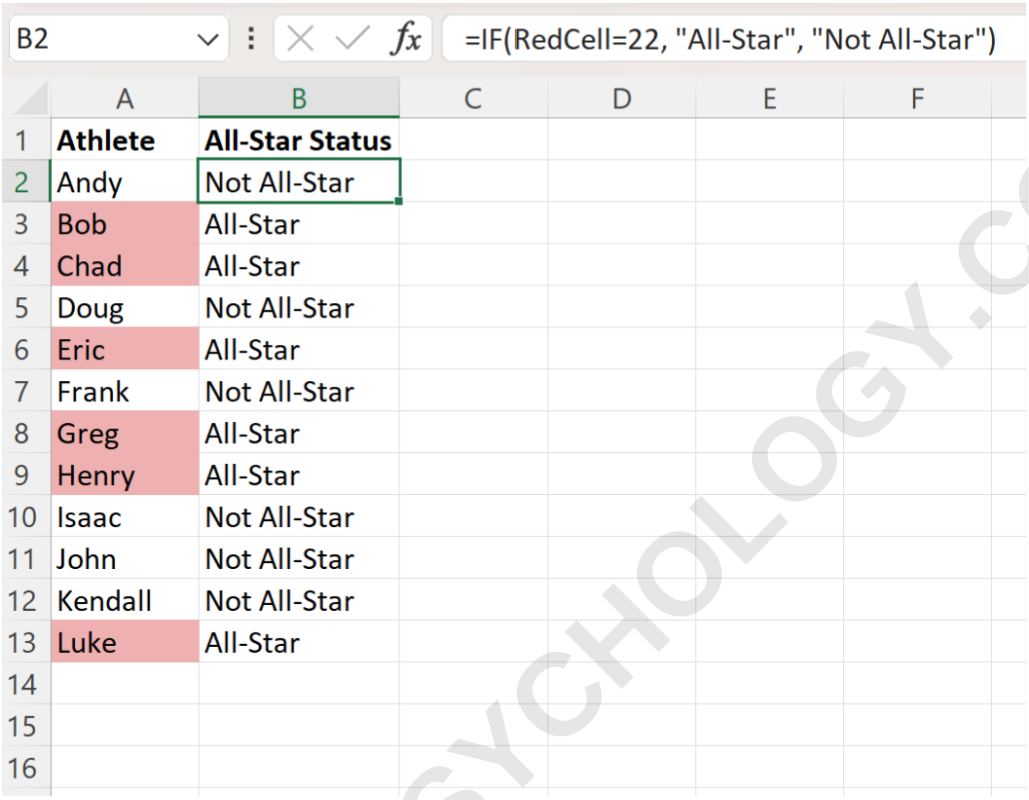
Type the following formula into cell **B2**:

=IF(RedCell=22, "All-Star", "Not All-Star")

This formula translates to: "If the color code retrieved by the `RedCell` function for the adjacent cell (A2) is equal to 22 (our specific red code), then return the text 'All-Star'; otherwise, return 'Not All-Star'."

Step 6: Reviewing the Automated Results

Once the formula is entered into B2, click and drag the fill handle down to apply this logic to every remaining player in the dataset. Column B will instantly update, providing an accurate, automated classification based purely on the visual background color of the names in column A.



	A	B	C	D	E	F
1	**Athlete**	**All-Star Status**				
2	Andy	Not All-Star				
3	Bob	All-Star				
4	Chad	All-Star				
5	Doug	Not All-Star				
6	Eric	All-Star				
7	Frank	Not All-Star				
8	Greg	All-Star				
9	Henry	All-Star				
10	Isaac	Not All-Star				
11	John	Not All-Star				
12	Kendall	Not All-Star				
13	Luke	All-Star				
14						
15						
16						

The resulting table now clearly shows which players are All-Stars, determined by the color of their respective cells. This demonstrates the power of combining legacy macro functions with modern Excel features to overcome inherent limitations in formula functionality.

Important Considerations for Color Codes

It is essential to remember that the color code is specific to the shade used. If you use a different shade of red, or if the spreadsheet is opened on a system with different color palette settings (though less common now), the resulting color code from `RedCell` may change.

This is precisely why the initial step of using `=RedCell` alone was necessary. You must always extract the color code first to ensure your IF statement compares against the correct integer value. If you receive an unexpected result, simply re-run the color extraction step (Step 4) to verify the current code for your specific fill color.

Conclusion and Further Reading

By utilizing the creation of a custom Defined Name that employs the `GET.CELL` macro function, we can effectively introduce conditional logic based on cell formatting into standard worksheet calculations. This technique is invaluable for users who rely on visual indicators like color to manage complex datasets.

The following tutorials explain how to perform other common operations in Excel:

Techniques for using VLOOKUP with multiple criteria.
How to calculate running totals in a spreadsheet.
Advanced uses of conditional formatting rules.