

How to Unhide All Rows in Excel Using VBA

Authored by
stats writer

February 27, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Unhide All Rows in Excel Using VBA*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=133042>

An Introduction to VBA and Spreadsheet Automation

Visual Basic for Applications, commonly referred to as VBA, serves as a powerful event-driven programming language developed by Microsoft. It is primarily utilized within the Microsoft Office suite to enhance functionality and automate repetitive tasks that would otherwise consume significant manual effort. In the context of Microsoft Excel, VBA allows users to create complex scripts known as **macros**, which can manipulate data, format cells, and interact with the application's core features. This capability is particularly essential for professionals who handle massive datasets and require a high degree of precision and speed in their daily workflows.

One of the most frequent challenges encountered by spreadsheet users is the management of row and column visibility. In large-scale data environments, rows are often hidden to simplify views or to isolate specific subsets of information for analysis. However, manually locating and unhiding these rows can become a tedious chore, especially if they are scattered across thousands of entries or multiple worksheets. By leveraging the **Object Model** in Excel, VBA provides a streamlined solution to restore visibility instantly. This programmatic approach ensures that no data remains accidentally obscured, maintaining the integrity of the reporting process.

To facilitate this, VBA employs specific properties that target the physical structure of a worksheet. The **Rows** property is the primary gateway for accessing horizontal ranges of cells. When combined with the **Hidden** property, it grants the developer total control over which parts of the dataset are exposed to the user. Understanding how these elements interact is the first step toward mastering spreadsheet automation. This guide will explore the technical nuances of unhiding rows, providing both simple and advanced code structures to optimize your Excel experience.

Moreover, the use of VBA for such tasks reflects a transition from basic data entry to sophisticated data management. By automating the "Unhide" function, users reduce the risk of human error, such as missing a hidden row that contains vital calculation inputs. As we delve deeper into the syntax and application of these commands, you will see how a few lines of code can replace minutes of manual clicking, thereby significantly boosting your **productivity** and technical proficiency within the Excel environment.

Exploring the Excel Object Model and the Rows Property

To effectively use VBA, one must understand the hierarchy of the **Excel Object Model**. At the top of this hierarchy is the Application object, followed by the Workbook, the Worksheet, and finally, the Range or Row objects. When we write a command to unhide rows, we are essentially communicating with these objects through their defined properties and methods. The **Rows** property specifically returns a Range object that represents all the rows on the specified

worksheet. If no specific worksheet is defined, VBA defaults to the currently active sheet, making the command highly versatile for immediate tasks.

Within this model, the **EntireRow** property acts as a modifier that ensures the subsequent command applies to the full horizontal extent of the grid. While simply calling the Rows property is often sufficient for basic visibility changes, using **EntireRow** is a best practice in VBA to avoid ambiguity. It explicitly tells the compiler that the entire row structure is being targeted, rather than just a specific selection of cells within those rows. This distinction is crucial when dealing with complex spreadsheets where certain cells might be merged or part of an Excel Table.

The technical implementation relies on assigning a Boolean value to the **Hidden** property. In programming, a Boolean value is a binary data type that can only be either **True** or **False**. When the Hidden property is set to True, the row is collapsed and removed from the visual interface. Conversely, setting it to False instructs Excel to expand the row and make it visible once more. This logic is the cornerstone of visibility management in VBA and is used across various objects, including columns and even entire worksheets.

The Syntax and Mechanics of the Hidden Property

The specific syntax required to unhide all rows in a worksheet is remarkably concise. By accessing the global Rows object, we can apply a state change to every row simultaneously. This is far more efficient than a manual selection process where a user might drag their mouse across the row headers or use the "Format" menu in the **Ribbon**. The efficiency of the VBA engine allows it to process thousands of rows in a fraction of a second, which is a testament to the power of programmatic **automation**.

The fundamental code block for this operation is encapsulated within a **Sub procedure**. A Sub, or subroutine, is a sequence of statements that perform an action but do not return a value. In our case, the action is changing the visibility state of the rows. The syntax is as follows:

Sub UnhideAllRows()

```
Rows.EntireRow.Hidden = FalseEnd Sub
```

In this code, the term **Sub** signals the beginning of the macro, followed by the name `UnhideAllRows()`. The core instruction `Rows.EntireRow.Hidden = False` is the active command. By assigning the value **False** to the **Hidden** property, we effectively "turn off" the hidden status for every row in the active sheet. Finally, `End Sub` marks the conclusion of the procedure. This simple structure is the building block for more complex data cleanup scripts used by data analysts and accountants worldwide.

It is important to note that this command affects every single row from row 1 to the very last row supported by Excel (which is 1,048,576). Because the **Rows** property is used without a specific index, it defaults to the global collection of rows. This "all or nothing" approach is exactly what is needed when you want to reset a spreadsheet to its fully expanded state, ensuring that no data points are overlooked during a final review or audit of the workbook.

Implementing the Unhide Macro in the Visual Basic Editor

To utilize the VBA code provided, users must interact with the Visual Basic Editor (VBE). The VBE is a separate development environment integrated into Excel where all macros are written, edited, and managed. To access it, one typically presses **Alt + F11** on their keyboard. Once inside, the user must insert a new **Module**, which serves as a container for the code. Modules are the standard location for general-purpose macros that need to be accessible across the entire workbook.

After inserting a module, the code can be copied and pasted directly into the text area. It is vital to ensure that the code is placed within a standard module rather than a worksheet or workbook object module, as this ensures the macro behaves correctly when called. Once the code is in place, it can be executed in several ways: by pressing **F5** while the cursor is within the Sub, by clicking the "Run" button in the toolbar, or by assigning the macro to a button on the actual Excel worksheet for easier access by non-technical users.

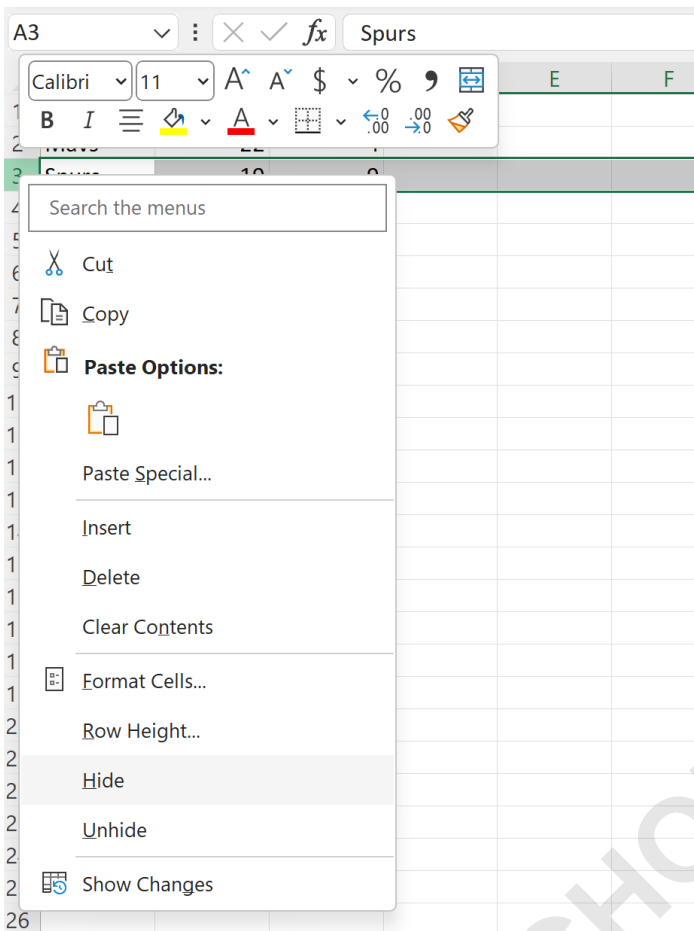
The following example demonstrates the practical application of this syntax. Imagine a scenario where a user is reviewing a list of basketball players, but several rows have been hidden to focus on specific statistics. To restore the full list, the macro can be invoked to immediately reveal every player in the dataset, regardless of how many rows were previously concealed. This demonstrates the immediate utility of the script in a real-world data management context.

A Practical Demonstration with Basketball Player Data

Suppose we have an Excel sheet that contains information about various basketball players, including their names, teams, and performance metrics. In a typical workflow, a user might filter this data or manually hide rows to create a cleaner view for a presentation or a specific calculation. The image below represents the initial state of such a spreadsheet, where all data is visible and organized for the user to see.

	A	B	C	D	E	F
1	Team	Points	Assists			
2	Mavs	22	4			
3	Spurs	19	9			
4	Rockets	15	3			
5	Kings	15	8			
6	Warriors	29	12			
7	Nets	24	10			
8	Lakers	40	8			
9	Thunder	35	3			
10	Blazers	23	6			
11	Jazz	33	2			
12						
13						
14						
15						
16						
17						

In many instances, a user might right-click on a specific row header--for example, row 3--and select the **Hide** option from the context menu. This action collapses the row, making it invisible to the eye and excluding it from standard view, though the data remains in the file's memory. This is a common practice when certain entries are temporarily irrelevant to the task at hand.



As the complexity of the task increases, more rows might be hidden. For example, the user might repeat the hiding process for rows 7 and 8. At this stage, the spreadsheet becomes fragmented, with jumps in the row numbering indicating that data is missing from the display. While this is useful for focus, it becomes a liability when the user needs to perform a global update or verify the entirety of the dataset before distribution.

	A	B	C	D	E	F
1	Team	Points	Assists			
2	Mavs	22	4			
4	Rockets	15	3			
5	Kings	15	8			
6	Warriors	29	12			
9	Thunder	35	3			
10	Blazers	23	6			
11	Jazz	33	2			
12						
13						
14						
15						
16						
17						
18						
19						
20						

Executing the Macro to Restore Visibility

When the need arises to view the full dataset again, the VBA macro we discussed earlier becomes the most efficient tool. Rather than hunting for the small lines between row headers to manually unhide each section, we can simply run our `UnhideAllRows` macro. This programmatic approach ensures that every single hidden row--whether it was hidden manually or by a filter--is brought back into view with a single command.

The macro used for this specific scenario is identical to the one previously described. It targets the **Rows** collection and sets the **Hidden** property to **False**. This instruction is absolute and universal for the sheet in question. It acts as a "reset" switch for visibility, which is invaluable during the final stages of data preparation or when troubleshooting formula errors that may be linked to hidden values.

Sub UnhideAllRows()

```
Rows.EntireRow.Hidden = FalseEnd Sub
```

Once this macro is executed, the spreadsheet returns to its original, fully expanded state. As shown in the image below, rows 3, 7, and 8 have been restored. The continuity of the data is once again visible, allowing the user to proceed with their analysis without the fear of missing hidden

information. This cycle of hiding and unhiding is a standard part of high-level Excel usage, and VBA makes the transition between these states seamless.

	A	B	C	D	E	F
1	Team	Points	Assists			
2	Mavs	22	4			
3	Spurs	19	9			
4	Rockets	15	3			
5	Kings	15	8			
6	Warriors	29	12			
7	Nets	24	10			
8	Lakers	40	8			
9	Thunder	35	3			
10	Blazers	23	6			
11	Jazz	33	2			
12						
13						
14						
15						
16						
17						

Automating Visibility Across Multiple Worksheets

For users working with complex workbooks containing dozens of sheets, unhiding rows on just the active sheet might not be enough. In such cases, it is beneficial to create a macro that iterates through every worksheet in the workbook. This is achieved using a **For Each loop**, a fundamental control flow statement in programming that allows you to repeat a set of instructions for every object in a collection.

By defining a variable as a **Worksheet**, we can tell VBA to look at each sheet one by one. For every sheet it finds, it will apply the `Hidden = False` command to all rows. This global approach is a massive time-saver for monthly reporting or when consolidating data from various departments. The code structure for this advanced operation is as follows:

Sub UnhideAllRowsAllSheets()

```
Dim ws As Worksheet
```

```
For Each ws In Worksheets
```

```
ws.Rows.EntireRow.Hidden = FalseNext ws
```

End Sub

In this script, `Dim ws As Worksheet` declares our variable. The `For Each ws In Worksheets` line starts the loop, targeting every sheet in the current file. The line `ws.Rows.EntireRow.Hidden = False` performs the un hiding action specifically on the sheet currently being processed by the loop. Finally, `Next ws` tells VBA to move to the next worksheet until all of them have been processed. This ensures a comprehensive cleanup of the entire workbook in seconds.

Using a **loop** is a significant step up in VBA capability. It demonstrates how code can be scaled from performing a single task to managing an entire environment. Whether you have two sheets or two hundred, the execution time remains minimal. This type of automation is what separates advanced Excel users from beginners, providing a level of control that is simply not possible through the standard user interface.

Best Practices and Official Documentation

When working with VBA and the **Hidden** property, it is essential to follow best practices to ensure your macros are reliable. One such practice is to include error handling or to ensure that the workbook is not protected before running the script. If a worksheet is protected, VBA may be unable to change the **Hidden** property, resulting in a runtime error. Checking for these conditions can make your tools more robust and user-friendly.

Furthermore, documentation is an invaluable resource for any aspiring developer. Microsoft provides extensive guides on the [Range.Hidden](#) property and other related objects. Reviewing the official documentation allows you to discover additional parameters and related properties, such as **ColumnWidth** or **RowHeight**, which can further enhance your ability to format and present data programmatically.

In conclusion, mastering the ability to unhide rows using VBA is a vital skill for anyone looking to optimize their use of Excel. By understanding the **Rows** property, the **Boolean** nature of the **Hidden** property, and the implementation of **loops**, you can transform the way you interact with data. This not only saves time but also ensures that your spreadsheets are transparent, accurate, and professional. Whether you are managing a simple list of basketball players or a complex financial model, these VBA techniques provide the speed and reliability needed in today's data-driven world.