Learning to Unhide All Rows in Excel with VBA: A Comprehensive Tutorial

Authored by

Mohammed looti

November 9, 2025

The ability to efficiently manage and manipulate data structure is a cornerstone of advanced Excel usage. One common requirement is the need to show rows that have been intentionally or accidentally concealed. While manual unhiding is straightforward for a few rows, managing extensive datasets where numerous rows are hidden sporadically demands a more systematic approach. This is where **Visual Basic for Applications (VBA)** proves indispensable. By leveraging the programmatic power of VBA, users can execute complex operations, such as unhiding every row in a sheet, with a single command. This guide details the structure and implementation of the necessary VBA code, focusing specifically on the critical **Hidden property** used for toggling row visibility.

In the VBA object model, every element within an Excel environment--ranging from cells and rows to entire worksheets--is treated as an object with modifiable properties. When dealing with row visibility, we interact with the **Rows** object and its associated **EntireRow** property. To reveal a row, we simply need to instruct Excel to set the state of its **Hidden** property to `False`. This action overrides any previous manual hiding, instantly making all affected rows visible again. Understanding this fundamental concept is the first step toward automating complex data management tasks within your spreadsheets.

Understanding the Core Syntax for Row Visibility

The core command for revealing all rows relies on manipulating the **Hidden** property, which is a Boolean property of the Range object. When applied to the entire collection of rows, it forces a state change across the board. Setting this property to `True` hides the rows, while setting it to `False` ensures they are displayed. This simple yet powerful syntax is the foundation of the macro we will construct.

To unhide all rows within the currently active worksheet, you must reference the entire collection of rows and explicitly define the desired state of their visibility. The following syntax accomplishes this task efficiently. Note that the command targets the entire collection of rows within the active scope, making it unnecessary to iterate through them individually.

Sub UnhideAllRows()

Rows.EntireRow.Hidden = False

End Sub

By executing this simple **Macro**, we signal to Excel that the **Hidden** property for every row should be set to `False`, thereby guaranteeing that all rows are unhidden in the current context. This method is the fastest way to reset row visibility for a single sheet when you are certain that no rows should remain concealed.

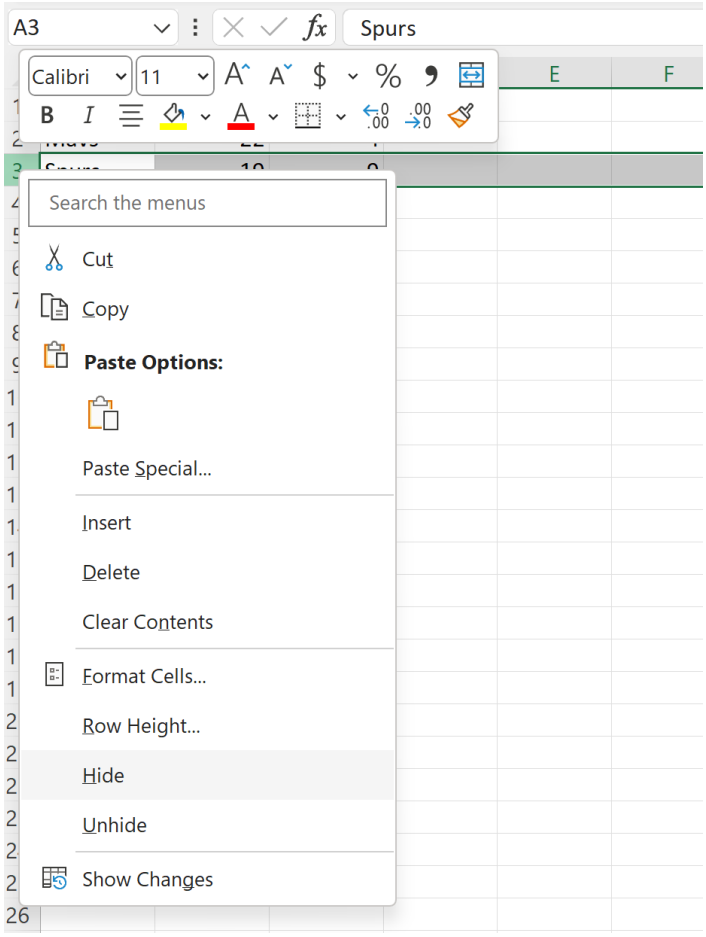
Step-by-Step Example: Implementing the Row Unhide Macro

To demonstrate the practical application of this code, consider a scenario involving a data sheet that tracks basketball player statistics. Data analysts often hide rows temporarily to focus on specific subsets of information or to prepare a simplified view for presentation purposes.

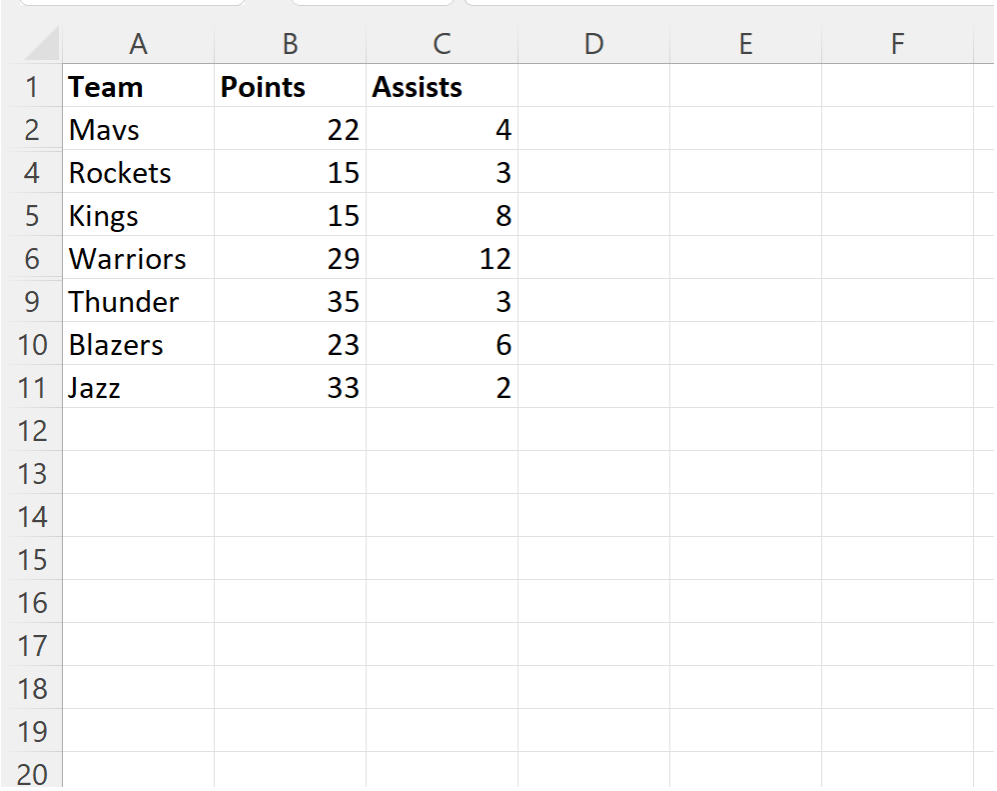
Suppose we begin with a complete Excel sheet containing detailed information, as illustrated below. This represents the original, fully visible dataset before any hiding operations take place.

	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						

Now, imagine that during the analysis phase, certain rows are manually hidden. For instance, Row 3 might be hidden by right-clicking the row header and selecting **Hide**. This manual process is acceptable for small adjustments but becomes tedious when managing large data tables.



If we repeat this hiding process for additional rows, such as Rows 7 and 8, the sheet begins to look segmented. This state of having multiple hidden rows often necessitates a quick, universal method to restore the sheet's full view without manually searching for the hidden breaks (indicated by the double lines between row numbers).



	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						

To efficiently revert the worksheet to its original, fully visible state, we must execute the defined macro. To do this, open the VBA editor (usually via Alt + F11), insert a new module, and paste the code snippet below. Running the `UnhideAllRows` subroutine will instantly apply the `Hidden = False` command to all rows on the sheet.

```
Sub UnhideAllRows()

Rows.EntireRow.Hidden = False

End Sub
```

Upon successful execution of the macro, the sheet instantly restores all hidden rows, returning the dataset to its initial state, ready for further comprehensive review or analysis. The speed and reliability of this programmatic approach highlight the value of automating repetitive visibility tasks in Excel.

	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						

Extending Functionality: Unhiding Rows Across an Entire Workbook

While unhiding rows in the active sheet is valuable, advanced users often require a method to standardize row visibility across an entire **workbook** containing multiple sheets. Manually navigating through numerous worksheets to run the macro on each one is inefficient. Fortunately, VBA allows us to iterate through all sheets automatically using a structural loop.

To achieve this, we introduce the `For Each...Next` loop structure, a fundamental concept in VBA programming used for iterating over collections of objects. In this context, we define a variable (typically named `ws`) as a **Worksheet** object. The loop then systematically cycles through every **Worksheet** in the **Worksheets** collection of the active workbook.

Within the loop, the core unhiding command is applied, but instead of using the generic `Rows` collection (which defaults to the active sheet), we prefix it with the current `ws` variable (e.g., `ws.Rows.EntireRow.Hidden = False`). This ensures that the command is executed on the specific sheet currently being processed by the loop, guaranteeing universal row visibility restoration across the entire file.

Sub UnhideAllRowsAllSheets()

Dim ws As Worksheet

```
For Each ws In Worksheets
ws.Rows.EntireRow.Hidden = False
Next ws

End Sub
```

This macro, `UnhideAllRowsAllSheets`, provides a robust solution for ensuring data consistency across complex workbooks, making it an essential tool for auditors, data managers, and anyone handling multi-sheet reports. The efficiency gained by using the **For Each loop** drastically reduces the time spent on administrative data preparation tasks.

Advanced Considerations and Error Handling

While the basic unhiding macros are effective, professional VBA development often requires considering edge cases and potential errors. One common issue arises when a worksheet is protected. If a sheet is protected and the user attempts to execute the unhide macro, Excel will typically throw a runtime error, halting the macro's execution.

To handle protected sheets gracefully within the "Unhide All Sheets" macro, the code must be modified to temporarily unprotect the worksheet before applying the visibility change, and then promptly reprotect it afterward. This requires knowing the protection password (if one exists) or ensuring the code can handle sheets that are protected without a password. A more sophisticated macro would include conditional checks or utilize error handling statements like `On Error Resume Next`, although the latter should be used sparingly as it can mask critical issues.

Furthermore, it is important to understand the scope of the **Hidden** property. The code `Rows.EntireRow.Hidden = False` is comprehensive, but sometimes a user may only want to unhide a specific range (e.g., rows 10 through 20). In such cases, the syntax is adjusted to target the specific range object: `Range("10:20").EntireRow.Hidden = False`. Mastering the manipulation of the Range object in VBA allows for granular control over visibility, going beyond the simple "all or nothing" approach demonstrated here.

Further Resources for Excel VBA Mastery

Gaining proficiency in VBA is an ongoing process that benefits greatly from consulting official documentation and specialized tutorials. The commands used in this guide--specifically the **Hidden** property, the **For Each loop**, and the **Worksheet** object--are foundational elements of automating tasks in Excel.

For those interested in delving deeper into how Excel objects behave, reviewing the Microsoft Developer Network (MSDN) documentation for the Excel Object Model is highly recommended.

Understanding the hierarchy of objects--Workbook, Worksheet, Range, etc.--will unlock the potential to automate nearly any routine task, dramatically improving productivity and data accuracy.

The following tutorials explain how to perform other common tasks in VBA, building upon the principles of object property manipulation demonstrated in this guide:

How to Hide Columns: A parallel application of the `Hidden` property to the `Columns` object.

Working with Multiple Worksheets: Detailed methods for referencing and manipulating sheets within a single workbook structure.

Conditional Formatting using Macros: Applying logic to change cell properties based on data values, extending beyond simple visibility controls.