How to use tab in VBA

Authored by

stats writer

November 18, 2025

Introduction: Understanding Tab Characters in VBA

As developers work with VBA (Visual Basic for Applications) to automate tasks within applications like Excel, they frequently encounter scenarios requiring structured output, data alignment, or specialized string manipulation. One fundamental requirement for formatting text output, especially in message boxes or delimited files, is the insertion of the tab character. This character, often invisible, provides consistent spacing and column alignment that simple spaces cannot replicate. Understanding how to correctly represent and implement this character is crucial for generating clean, readable results.

In the context of programming, a tab is not just a visual spacing element; it is a specific control character, often represented internally by the **ASCII** value 9. VBA provides two primary, highly reliable methods for inserting this required character into any string variable or output stream. These methods ensure that the compiled code is both easy to read and consistently functional across different environments where your **VBA macro** might run.

This guide delves into these two methods, demonstrating their usage through practical examples that involve merging data fields from an Excel worksheet and displaying them in a structured **message box** output. Mastering these techniques will significantly enhance your ability to format complex output strings efficiently within the **VBA** environment.

The Two Primary Methods for Tabulation in VBA

When defining a tab character within a VBA string, you have the choice between using an intrinsic constant or utilizing a built-in function to reference its underlying numerical code. Both options achieve the exact same result--the insertion of **ASCII code 9**--but they differ slightly in their readability and conventional use.

These two methods are the cornerstone of text formatting in **VBA** and should be familiar to any serious developer. They are:

vbTab: This is an intrinsic VBA constant designed specifically for readability and ease of use.
Chr(9): This method uses the generic `Chr()` function to return the character corresponding to its numerical **ASCII** index, which for a tab is 9.

While some developers prefer the self-documenting nature of **vbTab**, others who frequently work with various control characters (like line feeds, carriage returns, or form feeds) might find the consistency of the **Chr()** function appealing. We will explore both options in detail, ensuring you understand why they are interchangeable for this specific formatting task. Both of these constants effectively represent a tab character in **VBA**.

Deep Dive into the vbTab Constant

The **vbTab** constant is perhaps the most straightforward way to insert a tab character. As an intrinsic constant, **vbTab** is built directly into the VBA language. Intrinsic constants are essentially named placeholders for values that never change, making your code highly readable because the constant's name clearly describes its purpose.

When the **VBA** interpreter encounters **vbTab**, it is immediately replaced by the corresponding control character during runtime. This constant is part of the extensive library of predefined constants used for string manipulation and environment interaction, alongside others such as **vbCrLf** (Carriage Return and Line Feed) and **vbNullString**. Using these constants significantly improves the maintenance and comprehension of complex string operations, especially when concatenating multiple strings and control characters.

The primary benefit of using **vbTab** over its functional counterpart, **Chr(9)**, is developer efficiency. You don't need to memorize the specific **ASCII** numerical code for the tab character; you simply type the constant name, which is often auto-completed by the VBE editor, reducing the chance of typographical errors and increasing coding speed.

Utilizing the ASCII Character Function: Chr(9)

The alternative, and equally effective, method involves leveraging the `Chr()` function. This function accepts a numerical argument--a value corresponding to the character's position within the standard character set, typically the ASCII or **ANSI** character tables. Since the tab character is universally recognized by the decimal value 9 in the ASCII standard, calling **Chr(9)** returns the desired control character.

The ASCII standard is a foundational element of computing, assigning unique numerical codes to 128 characters, including letters, numbers, punctuation, and crucial control characters like the tab. Understanding the `Chr()` function is invaluable not just for tabs, but for inserting any non-printing or special characters for which a dedicated **VBA** constant might not exist or be known.

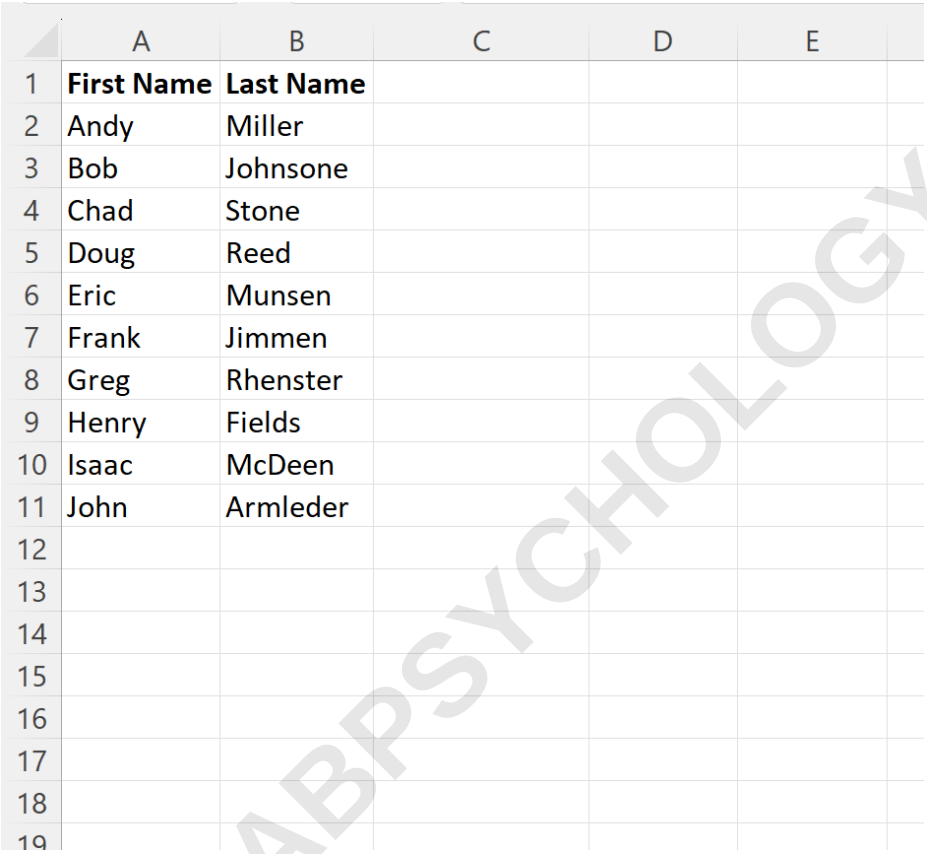
While **Chr(9)** requires recalling or referencing the number 9, it offers flexibility. Developers who frequently deal with file parsing or external system integration might find the numeric approach more intuitive, as it maps directly to file format specifications or protocol standards that reference character codes numerically. It is a powerful tool for intricate string construction and output formatting.

Practical Application: Concatenating Data with Tabs

To illustrate how these tab constants function in a real-world scenario, consider a common

requirement in data processing: extracting records from an Excel worksheet and compiling them into a single, formatted output string, such as for a log file or a standardized message box display. We will use a simple dataset consisting of names that need to be separated by a tab to ensure consistent alignment, which is critical for visual clarity.

Suppose we have the following list of first and last names in Excel, spanning rows 2 through 11. We would like to use **VBA** to create a message box that displays each of these first and last names with a tab between them, demonstrating how to use each of these constants in practice.



	A	B	C	D	E
1	**First Name**	**Last Name**			
2	Andy	Miller			
3	Bob	Johnsone			
4	Chad	Stone			
5	Doug	Reed			
6	Eric	Munsen			
7	Frank	Jimmen			
8	Greg	Rhenster			
9	Henry	Fields			
10	Isaac	McDeen			
11	John	Armleder			
12					
13					
14					
15					
16					
17					
18					
19					

We can create a VBA macro to perform this task, focusing on the string concatenation operation within a loop to build the complete string output.

Step-by-Step Code Implementation using vbTab

We begin by crafting a simple VBA macro named `UseTab`. This macro initializes necessary variables, iterates through the 10 rows of data, reads the values from columns A and B, and uses string concatenation to build the final output string, `AllNames`.

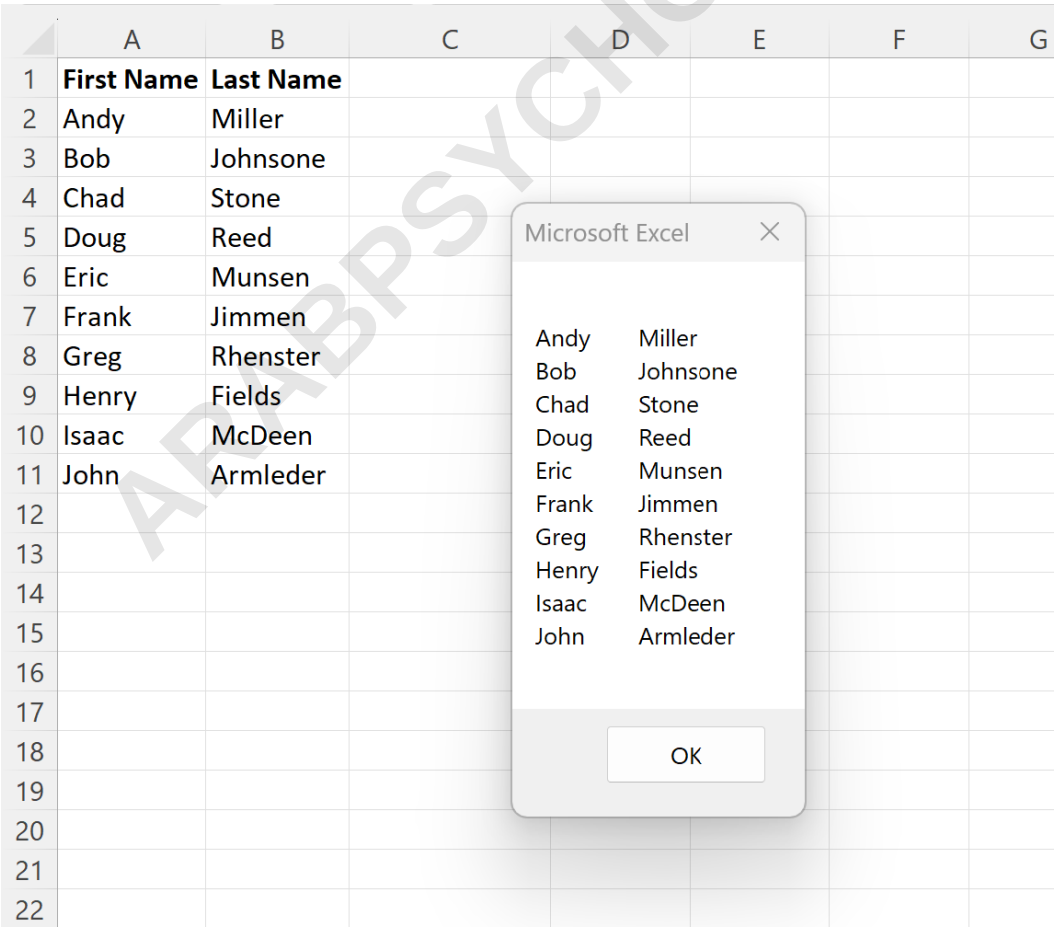
In the core loop, we utilize the **vbTab** constant to place the required horizontal separation between the first and last names. Furthermore, because we want each record to appear on its own line in

the final output box, we also include the **vbNewLine** constant at the end of each record string before appending it to `AllNames`.

We can create the following macro to do so:

```
Sub UseTab()

Dim i As Integer
Dim AllNames As String

For i = 2 To 11
AllNames = AllNames & Range("A" & i).Value & vbTab & Range("B" & i).Value & vbNewLine
Next i

MsgBox AllNames

End Sub
```

When we run this macro, we receive the following message box as output:



Notice that we used **vbTab** to concatenate the first and last names together in each row with a tab, ensuring visual separation and alignment.

The Importance of Line Breaks: Introducing vbNewLine

While the primary focus of this article is the tab character, it is critical to address the role of **vbNewLine** in achieving the multi-line output shown above. The **vbNewLine** constant represents the appropriate platform-specific sequence for starting a new line--typically a combination of carriage return and line feed (**vbCrLf**) on Windows systems.

Without **vbNewLine**, the concatenation process would simply string all names together continuously on a single, potentially massive line, making the message box output unreadable and defeating the purpose of separating the individual records. We used the **vbNewLine** constant to add each new name to a new line in the message box, ensuring vertical structure.

In the previous example, **vbTab** handled the horizontal spacing (alignment between fields), while **vbNewLine** handled the vertical spacing (separation between records). Understanding the distinction and correct application of both control characters is essential for generating highly structured text output in VBA.

Alternative Implementation using Chr(9)

If we'd prefer an alternative method, we could have also used **Chr(9)** to specify a tab character. This is functionally identical to **vbTab** but relies on referencing the ASCII code 9 directly through the `Chr()` function. The remainder of the macro structure remains completely unchanged, highlighting the interchangeability of the two methods.

This approach demonstrates the flexibility of **VBA** string manipulation. Whether you prefer the mnemonic constant or the universal **ASCII** reference, the compiler handles both correctly, delivering the same control character output. This ensures that regardless of your coding style, the resulting user interface element will maintain the desired formatting structure.

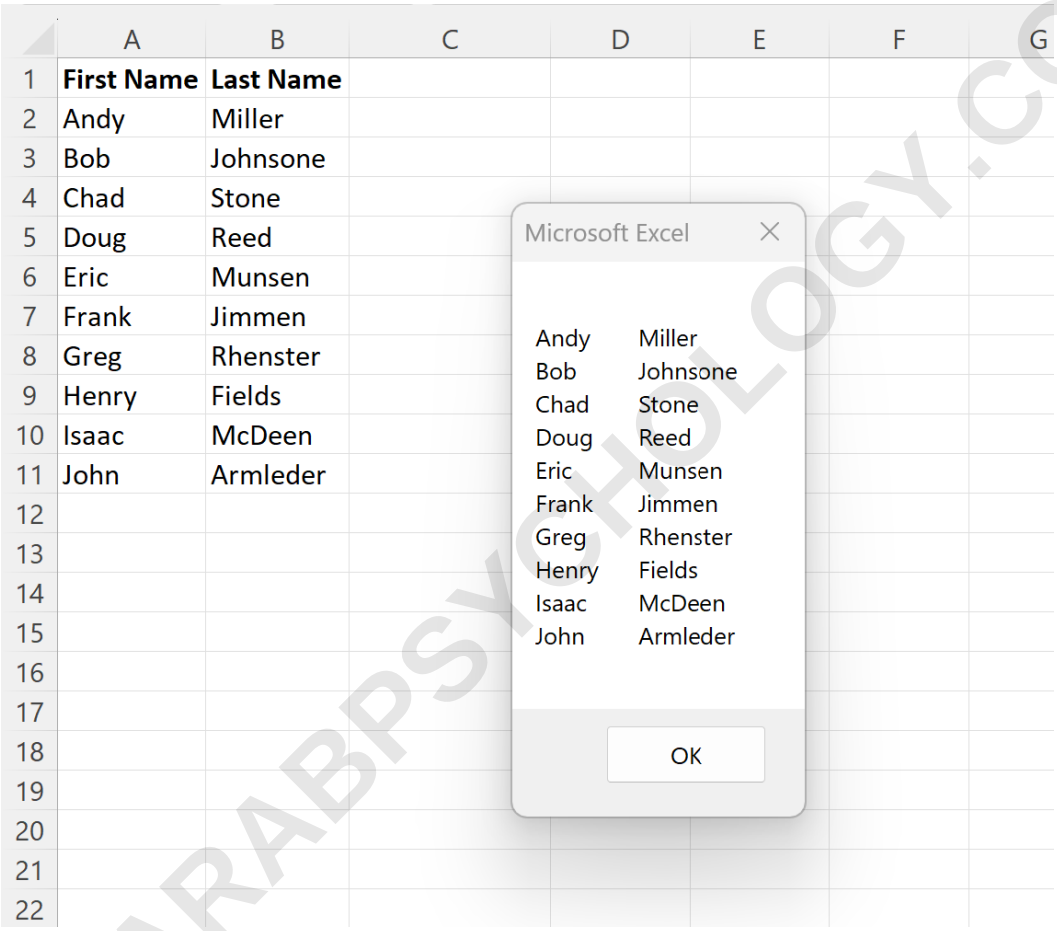
Here is the revised code block utilizing the **Chr(9)** function:

```
Sub UseTab()

Dim i As Integer
Dim AllNames As String

For i = 2 To 11
AllNames = AllNames & Range("A" & i).Value & chr(9) & Range("B" & i).Value & vbNewLine
Next i
```

```
MsgBox AllNames

End Sub
```

Comparing vbTab and Chr(9): Consistency in Output

When we run this macro, we receive the following message box as output, which is visually indistinguishable from the result generated by the **vbTab** example:



Notice that **vbTab** and **Chr(9)** both produce the exact same results. This consistency is guaranteed because **vbTab** is defined internally within the **VBA** library to be equivalent to the character represented by ASCII code 9.

Choosing between the two methods often comes down to internal coding standards or personal preference. If your team focuses heavily on code readability and uses many other **VBA** intrinsic constants, **vbTab** is likely the preferred choice. If your programming involves frequent manipulation of raw character codes or integration with systems defined by ASCII specifications, **Chr(9)** might offer better contextual clarity.

Ultimately, the programmer should select the method that makes the code easiest to maintain and debug, knowing that the resulting functionality for tabulation will be identical in every measurable way within the **VBA** environment.

Conclusion and Best Practices

The ability to correctly insert control characters is fundamental to robust string manipulation in VBA. Whether you are generating reports, formatting delimited files, or structuring user prompts via **MsgBox**, the tab character (ASCII 9) is essential for alignment.

We have confirmed that **vbTab** and **Chr(9)** are equally valid approaches to solving this formatting need. Both methods successfully handle string concatenation, allowing developers to combine data points (like first and last names) with the necessary horizontal separation.

For maximizing code readability and leveraging built-in language features, the best practice recommendation generally leans toward using the intrinsic constant **vbTab**. It requires no external numerical reference and clearly states its intent without requiring knowledge of the underlying **ASCII** code. Always remember to pair your tab characters with appropriate line breaks (**vbNewLine**) when formatting multi-record output to ensure vertical structure and clarity.