VBA Tutorial: Extracting Text Between Characters with Custom Functions in Excel

Authored by

Mohammed looti

November 9, 2025

The Power of Custom Functions in String Extraction

While **Microsoft Excel** provides a robust library of native tools for data handling, sophisticated parsing requirements often demand the flexibility of **VBA** (Visual Basic for Applications). A frequent task in data cleansing and preparation involves isolating specific text segments precisely positioned between two defined delimiter characters. Although this can technically be accomplished using complex nesting of native worksheet functions like `FIND`, `MID`, and `LEN`, developing a custom, reusable **VBA** function significantly streamlines the process. This dedicated approach clarifies the logic, enhances maintainability across multiple projects, and efficiently encapsulates the necessary steps for boundary location, length calculation, and result extraction.

The primary challenge in extracting delimited text lies in dynamically calculating both the starting index and the required length of the enclosed substring, as the positions of the delimiters constantly change based on the source text. By designing a custom function, we can simplify the interface: the user merely passes the source text and the two delimiters as arguments. The function then handles all complex, underlying calculations automatically. This abstraction results in substantially cleaner formulas directly within the **Excel** worksheet, greatly improving readability and reducing potential error points for collaborators analyzing the workbook.

The custom function introduced here, named `ExtractBetween`, is specifically engineered to solve this exact problem. It requires three critical parameters: the source text string (`this_text`), the character marking the beginning of the desired segment (`start_char`), and the character marking the end (`end_char`). This robust structure offers an immediate and powerful solution for advanced **string manipulation** tasks that would otherwise necessitate intricate, error-prone worksheet formula logic.

Implementation: Defining the ExtractBetween Function

To integrate this powerful text extraction capability, we must define the custom **VBA** procedure within the Visual Basic Editor (VBE). Once correctly implemented in a standard module, this function operates identically to any built-in **Excel** function, ready to be called from any cell in your worksheet. The architecture is straightforward yet highly effective, relying predominantly on two core **VBA** functions: `InStr` for locating character positions and `Mid` for performing the actual substring extraction.

The function definition begins with the `Function` keyword, establishing its scope and required parameters. Within the function body, the execution follows a logical sequence. Initially, the `InStr` function is invoked twice: first to determine the position of the starting character, and second for the ending character. These indices are stored in the variables `StartPosition` and `EndPosition`. It is crucial to understand that these positions represent the index numbers of the delimiter characters

themselves, calculated from the beginning of the input string.

The final, critical step is the extraction handled by the `Mid` function. The `Mid` function needs three inputs: the source text, the starting index for extraction, and the total character count (length) to extract. Since the desired text starts *after* the opening delimiter, we must offset the calculated start position by one (`StartPosition + 1`). The calculation of the length is the most precise element: we subtract the `StartPosition` from the `EndPosition`, and then subtract 1 again. This double subtraction ensures that both the starting and ending delimiters are excluded, returning only the desired segment nestled between them.

The complete code for the `ExtractBetween` custom function is provided below. For successful execution and accessibility throughout your workbook, this code must be accurately placed into a standard module within the **VBA** Editor:

Function ExtractBetween(this_text, start_char, end_char)

StartPosition = InStr(this_text, start_char)
EndPosition = InStr(this_text, end_char)

ExtractBetween = Mid(this_text, StartPosition + 1, EndPosition - StartPosition - 1)

End Function

The Engine: Detailed Mechanics of InStr and Mid

The effectiveness and reliability of the `ExtractBetween` function are entirely dependent on the seamless coordination between the essential **VBA** functions: **InStr** and **Mid**. A deep understanding of their individual roles is necessary for anyone looking to debug or modify this extraction logic for more advanced data structures. The `InStr` function, short for "In String," is the foundational tool for identifying the starting index of a specified substring within a larger string. It returns an integer index; if the substring cannot be located, `InStr` returns 0.

In our custom code, the lines `StartPosition = InStr(this_text, start_char)` and `EndPosition = InStr(this_text, end_char)` precisely locate the indices of the opening and closing delimiters, respectively. These indices are the anchors that define the boundaries of the text we intend to isolate. It is critical to recall that **VBA**, unlike some other programming languages, uses 1-based indexing for strings, meaning the first character is located at position 1. This convention is consistently applied by both **InStr** and **Mid**, simplifying string calculations.

Once the boundaries are successfully established, the **Mid** function assumes the role of the extractor. Its general syntax requires three arguments: `Mid(string, start, length)`. The

challenge is ensuring absolute accuracy for the `start` position and the `length` parameter. We set the start index to `StartPosition + 1` because the desired text begins immediately after the starting delimiter. The calculation for the length, `EndPosition - StartPosition - 1`, is simply the total distance between the delimiters minus the two characters they occupy. This meticulous calculation is essential to guarantee that the result includes only the intended text segment and prevents runtime errors caused by attempting to extract an invalid length.

Practical Demonstration: Applying the Function in Excel

To fully appreciate the utility of the `ExtractBetween` function, let us examine a typical data cleaning scenario. Structured metadata is often embedded within larger text strings, commonly delineated by characters such as parentheses, brackets, or custom separators. Consider a **dataset** in **Excel** where Column A contains mixed textual data, including a unique product ID consistently enclosed within parentheses. Our objective is to efficiently extract only these enclosed IDs into a dedicated Column C.

The scenario below illustrates data where Column A contains descriptions, IDs, and sales figures:

	A	B	C	D	E	F
1	**Product ID**	**Sales**				
2	AA(2500)	22				
3	RRA(1640)	30				
4	EET(123)	24				
5	RTT(150)	28				
6	HRR(1750)	25				
7	HHR(435)	37				
8	EHI(950)	15				
9	EGG(8152)	19				
10						
11						
12						
13						
14						
15						
16						
17						

Leveraging our custom **VBA** function, we aim to isolate the text residing between the opening parenthesis `(` and the closing parenthesis `)` for every row. Compared to manual parsing or

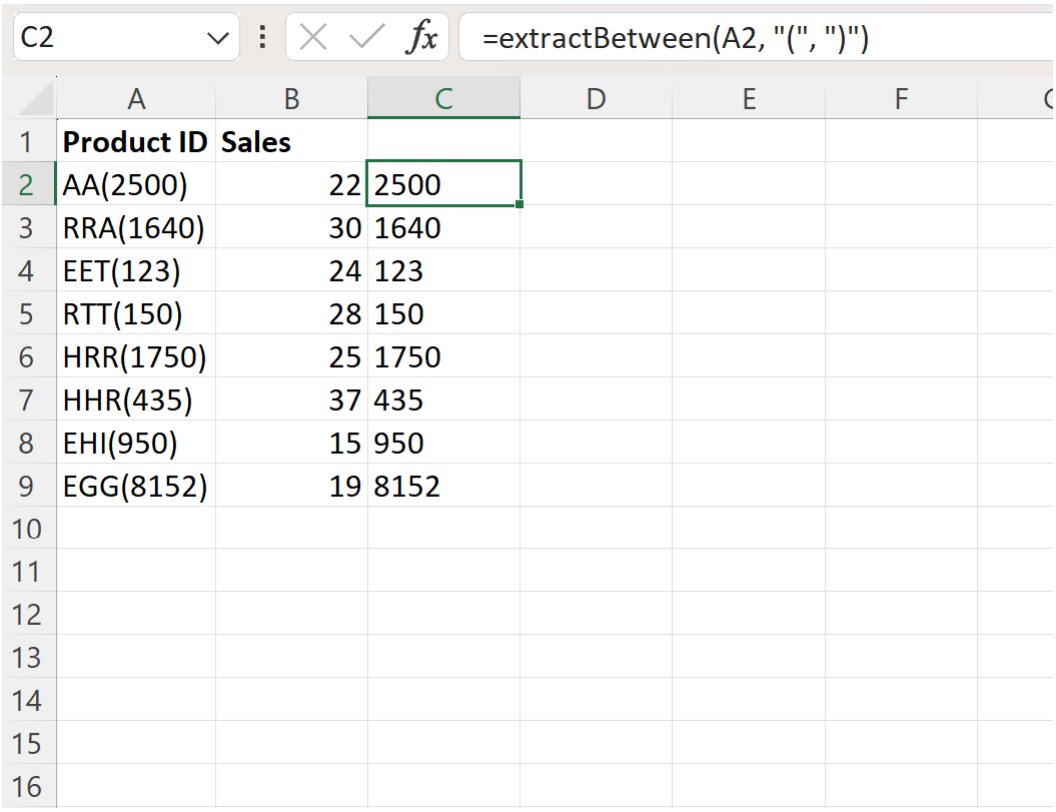
attempting to construct complex, volatile worksheet formulas, this automated solution provides enormous efficiency when handling hundreds or thousands of records.

Because the `ExtractBetween` function has been successfully defined in a standard module, it is immediately available for use within the spreadsheet environment. The syntax is highly intuitive, following the standard conventions of built-in **Excel** functions. For the initial data point in cell **A2**, we provide the cell reference, the opening delimiter, and the closing delimiter as arguments, instructing the function to execute the defined extraction logic.

We enter the following formula into cell **C2**:

=ExtractBetween(A2, "(", ")")

Upon entry, the first result instantly displays the correctly extracted text segment. The final, powerful step involves using **Excel's** auto-fill capability. By dragging the formula down through the remaining cells in Column C, the function is applied across the entire **dataset**, instantly providing the required, clean data points from Column A.



	A	B	C	D	E	F
1	**Product ID**	**Sales**				
2	AA(2500)	22	2500			
3	RRA(1640)	30	1640			
4	EET(123)	24	123			
5	RTT(150)	28	150			
6	HRR(1750)	25	1750			
7	HHR(435)	37	435			
8	EHI(950)	15	950			
9	EGG(8152)	19	8152			
10						
11						
12						
13						
14						
15						
16						

As clearly illustrated, Column C now contains only the product IDs that were originally embedded within the parentheses in Column A. This rapid and accurate transformation highlights how custom **VBA** functions streamline complex text parsing tasks, converting raw, unstructured inputs into

clean, actionable data suitable for subsequent analysis or reporting.

Enhancing Robustness and Addressing Edge Cases

While the current `ExtractBetween` function is highly effective for well-structured data where the delimiters are guaranteed to exist and appear only once, moving to production-level code demands careful consideration of potential errors and edge cases. Two critical scenarios must be addressed for robust functionality: handling missing delimiters and managing multiple delimiter occurrences.

If a required delimiter character is absent from the input string, the **InStr** function returns 0. If either `StartPosition` or `EndPosition` is 0, the subsequent `Mid` function calculation will attempt to use invalid indices or calculate a negative length, resulting in a visible runtime error (`#VALUE!`) within the **Excel** worksheet. To prevent this, professional code must incorporate a conditional check at the function's outset. If the start or end position is invalid, the function should gracefully return an empty string or a custom error message instead of failing. Furthermore, logic must confirm that the `StartPosition` is indeed less than the `EndPosition`, preventing failures if the delimiters are found in the wrong order (e.g., `)Text(`).

Another common complexity involves nested delimiters or instances where the same delimiter appears multiple times. The default behavior of **InStr** is to find only the *first* occurrence of the specified characters. If the extraction requirement changes--for example, needing text between the *last* instance of the starting character and the *first* instance of the ending character--the function logic requires modification. Advanced versions of this utility might utilize the optional `start` argument within `InStr` to dictate where the search should begin, or employ the `InStrRev` function to initiate the search backward from the end of the string. However, for the standard requirement of isolating text between two single, unique delimiters, the current simple structure remains the most efficient choice.

Conclusion and Resources for Advanced VBA Text Handling

The custom `ExtractBetween` function represents an elegant and powerful solution for a fundamental data processing requirement: precisely isolating text segments based on surrounding delimiters. By skillfully utilizing the core capabilities of the **VBA** **InStr** and **Mid** functions, we create a highly reusable utility that dramatically simplifies complex **string manipulation** tasks within **Microsoft** **Excel**. This methodology significantly improves the clarity and maintainability of your spreadsheets by centralizing the complex parsing logic within a dedicated code module, rather than cluttering cell formulas.

Implementation is straightforward--requiring only the placement of the function code into a standard module--and the benefits in data cleaning workflows are immediate. Whether you are dealing with product codes, extracting delimited metadata, or processing complex log entries, this

function ensures both accuracy and efficiency. It serves as a compelling demonstration that custom **VBA** functions are indispensable tools for advanced users and developers seeking streamlined automation.

For users committed to expanding their proficiency in **VBA** text handling, exploring related functions that address varying extraction requirements is highly recommended. Areas for focused study include:

The `InStrRev` function, which is optimized for searching a string backward, useful for finding the last occurrence of a character.
Effective use of the `Split` function for parsing text records that rely on common delimiters like commas, tabs, or pipe characters.
Implementing robust error trapping mechanisms (e.g., `On Error Resume Next`) to enhance function resilience against poorly structured or unexpected input data.

Mastering these advanced techniques will fully equip you to manage virtually any data extraction and transformation challenge encountered in modern data management and analysis.