How to Extract Text Between Two Characters in VBA with the Mid Function

Authored by

stats writer

February 28, 2026

The Role of VBA in Modern Data Management

In the contemporary landscape of digital operations, **VBA** (Visual Basic for Applications) remains a cornerstone for professionals seeking to enhance the capabilities of **Microsoft Office** applications. As a robust **programming language**, it provides the necessary tools to automate repetitive tasks, streamline complex workflows, and perform sophisticated **data analysis** that exceeds the standard limitations of out-of-the-box software features. By leveraging the power of automation, users can transform static spreadsheets into dynamic tools capable of processing vast amounts of information with minimal manual intervention, thereby reducing the margin for error and increasing organizational productivity.

One of the most frequent challenges encountered during **data processing** is the need to parse and isolate specific segments of information within a larger **string** of data. Whether you are dealing with log files, complex product identifiers, or structured text reports, the ability to extract content between two specific characters is an essential skill. This process requires a nuanced understanding of how **VBA** interprets text positions and how it utilizes built-in functions to navigate through character arrays to find exactly what the user needs for their specific reporting or analytical requirements.

The flexibility of **VBA** allows for the creation of **custom functions** that can be utilized directly within **Excel** cells, much like native formulas such as SUM or VLOOKUP. These User-Defined Functions (UDFs) provide a clean and reusable way to handle complex logic without cluttering the spreadsheet with nested formulas that are difficult to debug. By mastering the extraction of text, developers can ensure that their data remains clean, categorized, and ready for further downstream **manipulation** or visualization, making it a pivotal technique in any data scientist's or office power user's toolkit.

Understanding the Foundational Functions: InStr and Mid

To effectively isolate text between two delimiters, one must first master two primary **VBA** functions: **InStr** and **Mid**. The **InStr** function serves as a diagnostic tool that scans a **string** from left to right to identify the numerical position of a specific character or substring. Understanding the precise location of your starting and ending characters is the first step in the extraction logic, as it provides the coordinates necessary for the subsequent steps of the operation.

The **Mid** function, conversely, acts as the "extractor" in this partnership. It is designed to return a specific number of characters from a **string**, starting at a position specified by the user. While other functions like LEFT or RIGHT are useful for basic trimming, **Mid** offers the surgical precision required to reach into the middle of a text block and pull out only the relevant data. This is particularly useful when the data is wrapped in parentheses, brackets, or other unique delimiters

that signify a change in data context.

When combined, these functions allow a programmer to define a dynamic range for extraction. Instead of relying on fixed character counts, which fail when the length of the data varies, the combination of **InStr** and **Mid** creates a flexible solution. By calculating the distance between the starting character and the ending character, the **function** can adapt to any input **string**, ensuring that the output remains accurate regardless of whether the extracted text is three characters long or thirty characters long.

Constructing the ExtractBetween Custom Function

Creating a **custom function** is the most efficient way to implement this logic within **Excel**. A custom **function**, once defined in a module within the **VBA** editor, becomes accessible across the entire workbook. This approach is superior to writing long, complex formulas directly in the cell, as it centralizes the logic, making it easier to update or fix if the business requirements change in the future. The following code provides the structural foundation for this utility:

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 logic begins by assigning the position of the first delimiter to a variable named StartPosition and the position of the second delimiter to EndPosition. By using variables, the code remains readable and allows for easier debugging. The final calculation within the **Mid** function is critical: we add one to the StartPosition to ensure we do not include the starting character itself in our result, and we subtract the start position and an additional unit from the end position to calculate the exact length of the inner text.

This snippet demonstrates the elegance of **VBA** in solving common **Excel** hurdles. By wrapping this logic into a **function** named ExtractBetween, we provide an intuitive interface for the end user. They only need to know the target cell and the two characters they are looking for, while the technical complexity of position calculation and **string** slicing remains hidden safely within the **VBA** module.

Strategic Implementation of User-Defined Functions in Excel

To implement the ExtractBetween **function**, users must first access the Visual Basic Editor by pressing ALT + F11 in **Excel**. Within this environment, one must insert a new module and paste the provided code. This step is vital because functions written within worksheet objects or workbook objects may not be globally accessible in the same way as those stored in a dedicated module. Once the code is saved, the **Excel** environment recognizes ExtractBetween as a legitimate command, ready for use in any worksheet formula.

The beauty of this implementation lies in its seamless integration. Users do not need to interact with the **VBA** backend once the initial setup is complete. They can simply navigate to their data sheet and begin applying the formula to their columns. This democratization of technical tools allows team members who may not be proficient in **programming** to benefit from the advanced automation created by a developer, fostering a more efficient and capable workspace.

Furthermore, using a User-Defined Function (UDF) ensures consistency across your **data analysis** projects. Instead of different users attempting to write their own complex "FIND" and "MID" combinations--which are often prone to "off-by-one" errors--everyone can use the standardized ExtractBetween **function**. This standardization is a hallmark of professional **data management** and is essential for maintaining the integrity of large datasets over time.

Practical Demonstration: Extracting Product Metadata

Consider a practical scenario where a company maintains a ledger of product sales. The product identifiers in column A contain both the common name of the item and a secondary internal code enclosed in parentheses. While the full **string** is necessary for some reports, other analytical tasks require only the code found within those parentheses. Manual extraction would be a monumental task prone to significant human error, especially as the dataset grows into the thousands of rows.

	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						

As shown in the initial dataset, the goal is to isolate the text within the parentheses for each entry in the ID column. By applying our custom **VBA** **function**, we can automate this entire process. The function will look at each cell, find the "(" character, find the ")" character, and return everything in between. This allows the sales data in the adjacent columns to be associated with a clean, specific identifier in a new column, facilitating better sorting and filtering capabilities.

To execute this in **Excel**, we use the following code structure within our **VBA** environment. This code is the engine that drives our extraction, ensuring that every cell is processed with identical logic and precision:

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

Deconstructing the Mathematical Logic of Substrings

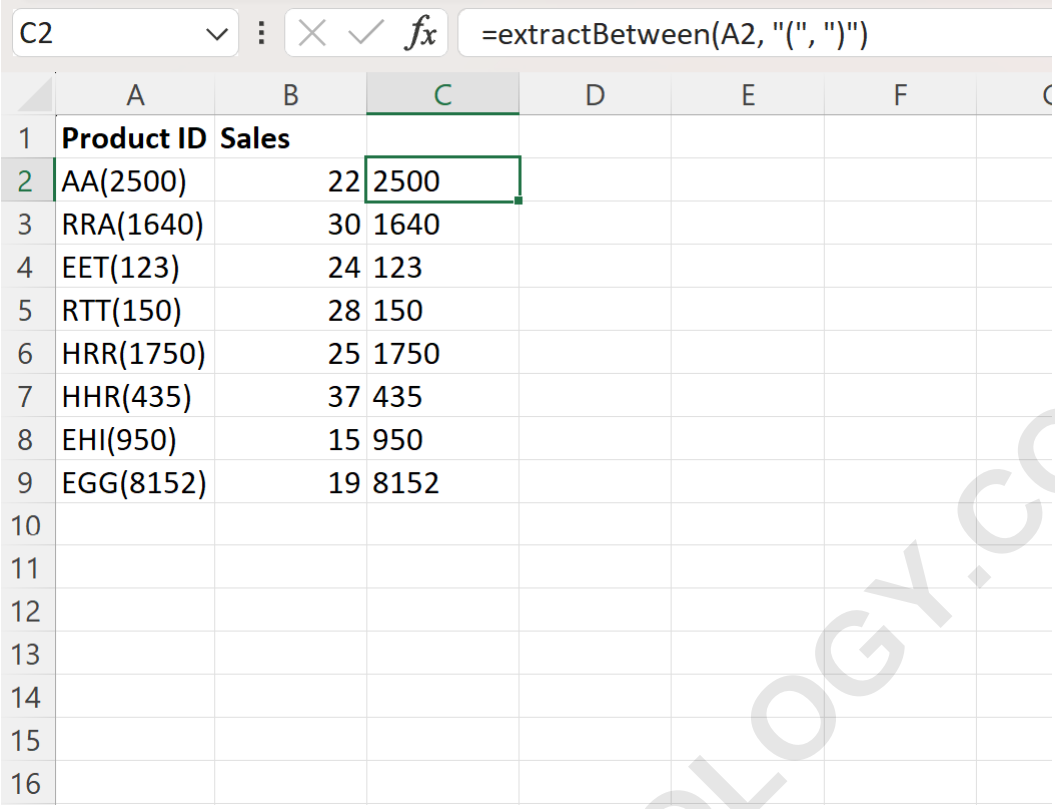
The core of the ExtractBetween **function** relies on a simple yet precise mathematical calculation. When **InStr** finds the start_char, it returns its index. If the "(" is at position 5, we don't want to start our extraction there, as that would include the parenthesis. Therefore, we use StartPosition + 1 to begin at position 6. This ensures the output is "clean" and contains only the internal metadata sought by the user.

The most complex part for many beginners is determining the length of the text to be extracted. In **VBA**, the **Mid** function's third argument is the number of characters to return, not the ending index. To calculate this length, we take the EndPosition (the index of the closing character) and subtract the StartPosition. However, because we also want to exclude the closing character itself, we subtract an additional 1. This formula (EndPosition - StartPosition - 1) accurately captures the distance between the two delimiters.

Once the logic is established, we can apply it to a specific cell. For example, by typing the following formula into cell **C2**, we instruct **Excel** to look at cell **A2** and find the text located between the opening and closing parentheses:

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

After entering the formula, the user can utilize **Excel**'s "Fill Handle" to drag the formula down the entire column. This action applies the **custom function** to every row, instantly populating Column C with the extracted codes as illustrated in the following image:



	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						

Advanced Considerations for Robust Character Extraction

While the basic ExtractBetween **function** is highly effective for standard datasets, advanced users might consider adding **error handling** to manage unexpected input. For instance, if a cell is missing one or both of the delimiters, the **InStr** function will return zero. Without proper checks, this could cause the **Mid** function to fail or return an error value. Adding an "If" statement to verify that both StartPosition and EndPosition are greater than zero is a best practice for professional-grade **VBA** development.

Another consideration is the handling of multiple occurrences of the same character. The **InStr** function, by default, finds the first occurrence of a character. If your **string** contains multiple sets of parentheses and you need to extract text from the second or third set, you would need to modify the **function** to use the optional start argument of the **InStr** function. This allows the search to begin after the first instance is found, enabling the extraction of nested or sequential data blocks.

Finally, it is worth noting that **VBA** is case-sensitive by default when performing **string** searches. While this is rarely an issue when searching for symbols like parentheses or brackets, it becomes relevant if you are using letters as delimiters. You can use the "Option Compare Text" statement at the top of your module or specify the "vbTextCompare" argument within the **InStr** function to ensure that your extraction logic is case-insensitive, making your **function** even more versatile.

Optimizing Workflow Through VBA Automation

The integration of **VBA** for text extraction is more than just a technical trick; it is a strategic approach to **workflow optimization**. By automating the isolation of relevant data, organizations can significantly reduce the time spent on data preparation. This allows analysts to focus more on interpreting the results and making data-driven decisions rather than getting bogged down in the minutiae of **data cleansing**. The efficiency gained from a well-written **function** can save hundreds of man-hours over the course of a large-scale project.

Moreover, the use of **VBA** encourages a deeper understanding of the underlying **data structures** within **Excel**. As users become more comfortable writing and implementing custom functions, they develop a mindset geared towards **efficiency** and scalability. They begin to see patterns in data challenges and realize that most manual tasks can be solved through logical **programming** solutions. This growth in technical literacy is a valuable asset in any data-centric career path.

In conclusion, the ability to extract text between two characters using **VBA** is a fundamental skill that bridges the gap between basic spreadsheet usage and advanced **data engineering**. By utilizing the **InStr** and **Mid** functions within a custom-built **function**, you can create a powerful, reusable tool that simplifies complex **extraction** tasks. This not only ensures data accuracy but also empowers you to handle larger and more complex datasets with confidence and ease, ultimately leading to more professional and reliable outputs.