How do I Use the COUNTW Function in SAS?

Authored by

stats writer

November 19, 2025

The COUNTW function in SAS is a fundamental tool for performing lexical analysis within character variables. Its primary utility is determining the number of "words" present within a specified character string. This function proves indispensable in data preparation and text mining tasks where quantifying the complexity or length of textual entries is necessary. When working with large data volumes, manually counting textual units is impractical, making the programmatic efficiency of **COUNTW** a significant asset. Furthermore, its versatility allows it to be integrated seamlessly into the **SAS Data Step** or various procedures, enabling complex data transformations and analyses.

The ability to accurately quantify words within records is critical for standardized reporting and quality control. For instance, researchers often need to enforce minimum or maximum word counts in survey responses or textual fields. The **COUNTW** function provides the analytical precision required to manage these constraints efficiently. Unlike simple character counting functions, **COUNTW** intelligently identifies boundaries, or delimiters, that separate linguistic units. This inherent capability allows users to go beyond basic space-separated analysis, offering customization through optional parameters to handle complex text structures involving punctuation, tabs, or user-defined symbols. Understanding this function is crucial for anyone performing advanced text manipulation within the **SAS environment**.

Introduction to the COUNTW Function in SAS

The **COUNTW** function stands as a powerful utility within the SAS programming language, specifically designed for quantitative analysis of textual data. Its core purpose is to calculate the number of words embedded within a designated character variable or literal string. This capability is vital for tasks such as linguistic analysis, content normalization, and ensuring data quality across various input fields. By providing a quick and reliable measure of textual density, **COUNTW** significantly streamlines the data cleaning process, particularly when dealing with unstructured or semi-structured data sources.

A "word" in the context of the **COUNTW** function is defined as a substring of characters delimited by one or more specified separation characters. By default, **COUNTW** treats a blank space as the standard separator. However, one of the function's greatest strengths lies in its flexibility; users can define custom separators, allowing it to correctly parse text that uses unconventional delimiters like commas, hyphens, or underscores. This adaptability makes it suitable for complex data structures where standard delimiters are not consistently applied, such as machine-generated logs or specialized coded strings. Mastering these customization options is key to leveraging the function's full potential in diverse data environments.

When integrated into a larger **SAS Data Step**, **COUNTW** can dynamically generate new variables that record word counts for every observation in a dataset. This procedural approach ensures that

the word counting logic is applied consistently across millions of records without manual intervention. The results derived from **COUNTW** can subsequently be used for filtering, grouping, or as input for further statistical analysis, emphasizing its role not just as a counting utility, but as an integral component of the data preparation pipeline. We will now examine the precise syntax required to invoke this versatile function.

Understanding the Core Syntax of COUNTW

The structure of the **COUNTW** function is designed for clarity and flexibility, utilizing three distinct arguments to control its operation. The basic invocation requires only the input string, while the optional arguments provide mechanisms for fine-tuning word separation rules. It is essential for SAS programmers to grasp the role of each argument to ensure accurate word counting based on specific data characteristics. The function returns a numeric value representing the total count of words identified in the input string after applying the separation logic defined by the optional parameters.

The standard syntax for utilizing this powerful function is presented as follows, where the primary argument is mandatory and the subsequent arguments are used for specialized counting requirements:

COUNTW(string, <character>, <modifier>)

This structure ensures that even complex text analysis requirements can be met within a single function call. The omission of the optional arguments triggers the default behavior, where standard blanks are used as separators. However, for data fields where words might be joined by non-standard characters, the optional arguments become crucial for achieving meaningful results. Utilizing these parameters allows the programmer to override default SAS parsing rules, tailoring the definition of a "word" to the specific needs of the analysis.

Exploring the Parameters: String, Character, and Modifier

The definition and purpose of each argument are critical for successful implementation of the **COUNTW** function:

string: This is the mandatory first argument, representing the source character string that contains the words intended to be counted. This can be a variable name from a SAS dataset, or a literal string enclosed in quotation marks. If the input string is missing or null, the **COUNTW** function will return a count of zero.
character: This optional character constant serves to initialize a list of characters that should be treated as word separators (delimiters). If this parameter is provided, the function overrides the default behavior (using only spaces) and uses the characters defined here to distinguish between

words. For example, supplying ' ,.-' would instruct **COUNTW** to treat spaces, commas, periods, and hyphens as word boundaries.
modifier: This optional argument consists of codes that alter how the counting and separation process operates. Modifiers are typically enclosed in single quotes and can control behaviors such as case sensitivity, the treatment of multiple delimiters, or whether to count strings that contain only digits. Using modifiers grants the user exceptional control over the word counting mechanism, allowing for highly specific and customized text analysis logic.

The combination of these parameters allows the SAS programmer to fine-tune the definition of a word. For instance, the 'T' modifier is particularly useful, telling **COUNTW** to "trim" leading and trailing delimiters from the input string before counting, which prevents erroneous counts caused by extraneous whitespace. Understanding the full range of modifiers available in the SAS documentation is highly recommended for professionals dealing with irregular textual data, as they represent the key to reliable and precise word quantification.

Practical Example 1: Counting Words with Default Separators

To illustrate the basic application of the **COUNTW** function, consider a scenario where we need to determine the number of distinct words within a collection of conversational phrases. In this initial example, we will rely on the function's default behavior, which recognizes only spaces as the standard word separators. This is the most common use case when dealing with relatively clean, prose-like text.

Suppose we define the following SAS dataset named **my_data**, containing various phrases:

```
/*create dataset*/
data my_data;
input phrase $char50.;
datalines;
Hey_everyone
What's going on today
Wow, what a great day
Let's have fun
We should play basketball
This weather is so so awesome
;
run;

/*view dataset*/
proc print data=my_data;
```

Executing this code establishes the baseline data for our analysis, showing how the phrases are stored in the **phrase** column before any counting operations are applied. Note the presence of the underscore character in the first phrase, which will be crucial for understanding the default delimiter behavior.

Obs	phrase
1	Hey_everyone
2	What's going on today
3	Wow, what a great day
4	Let's have fun
5	We should play basketball
6	This weather is so so awesome

We now apply the **COUNTW** function within a new **Data Step** to calculate the word count for each record. We create a new variable, **word_count**, using the simplest form of the function call: `countw(phrase)`. This instructs SAS to analyze the content of the **phrase** column and return the count based on the implicit delimiter--the space character.

```
/*create new dataset that shows number of words in each row*/
data new_data;
set my_data;
word_count = countw(phrase);
run;

/*view new dataset*/
proc print data=new_data;
```

The resulting dataset, **new_data**, provides the calculated word counts alongside the original phrases, clearly demonstrating the function's output under default conditions.

Obs	phrase	word_count
1	Hey_everyone	1
2	What's going on today	4
3	Wow, what a great day	5
4	Let's have fun	3
5	We should play basketball	4
6	This weather is so so awesome	6

Analyzing Default Delimiter Behavior

The results from the first example highlight a critical characteristic of the default **COUNTW** operation: by default, it only recognizes the standard blank space as a boundary between words. Any characters other than a space, including common punctuation or symbols like underscores, are considered part of the word itself. This behavior leads to important distinctions in how phrases are counted:

In the phrase "Hey_everyone," there are absolutely no spaces. Consequently, the **COUNTW** function treats the entire string, including the embedded underscore, as a single unit, yielding a count of only **1** word.
For phrases like "What's going on today," which contain three distinct spaces, the function correctly identifies four separate words, resulting in a count of **4**.
Similarly, "Wow, what a great day" contains four spaces and is therefore counted as **5** words. Punctuation like the comma is generally ignored as a separator unless explicitly defined otherwise.

This strict reliance on the space character as the sole separator can sometimes lead to results that contradict a human reader's intuition, especially when dealing with data that uses alternative separators or compound words. Therefore, when data input quality is inconsistent, or when analyzing machine-generated text, it is almost always necessary to utilize the optional parameters to explicitly define all relevant delimiters. Ignoring this necessity can lead to significant counting errors and flawed downstream analysis.

Practical Example 2: Specifying Custom Delimiters (Underscore)

To overcome the limitations observed in the default counting method, particularly regarding strings joined by non-space characters like underscores, we must employ the optional **character** argument. By specifying a list of custom delimiters, we instruct **COUNTW** to broaden its definition

of what constitutes a word boundary, ensuring more accurate results for complex data inputs. This customization is essential for robust text processing.

For instance, if we want both spaces and underscores to be treated as separators, we modify the function call to include the character argument, `' _'` (note that the space character itself must be included explicitly along with the underscore). The syntax below demonstrates how to achieve this precise control:

```
/*create new dataset that shows number of words in each row*/
data new_data;
set my_data;
word_count = countw(phrase, ' _');
run;

/*view new dataset*/
proc print data=new_data;
```

By executing this revised code, the **COUNTW** function now treats both the blank space and the underscore (`_`) as valid word separators. This change directly addresses the previous miscounting of the first phrase, "Hey_everyone," as the embedded underscore is now correctly recognized as a boundary, splitting the string into two distinct words.

Obs	phrase	word_count
1	Hey_everyone	2
2	What's going on today	4
3	Wow, what a great day	5
4	Let's have fun	3
5	We should play basketball	4
6	This weather is so so awesome	6

The new **word_count** column now accurately counts the number of words in the first phrase since we specified that an underscore should also be considered to be a separator between words. The first phrase, "Hey_everyone," is now correctly counted as **2** words, demonstrating the effectiveness of utilizing the optional character argument to tailor the counting logic. This technique is highly recommended whenever input data utilizes non-standard joining characters, such as pipes, tabs, or hyphens, as part of the data structure. It ensures that the definition of a "word" aligns precisely with the analytical goal.

Advanced Usage: Utilizing Modifiers for Complex Counting

While the character argument handles custom delimiters, the optional **modifier** argument provides granular control over the counting process itself. Modifiers are single characters enclosed in quotes that instruct the COUNTW function to handle specific textual nuances, such as leading/trailing delimiters, case sensitivity, or special character handling. Understanding these modifiers is essential for achieving reliable counts in highly variable data environments.

Some of the most frequently used modifiers include:

'T' (Trim): This modifier instructs **COUNTW** to ignore leading and trailing delimiters from the input string. This is crucial for preventing strings that start or end with excessive whitespace or specified delimiters from being misinterpreted as containing an extra, empty word.
'A' (Alphabetic and Numbers): When used with the character argument, 'A' specifies that characters that are alphabetic letters or digits should be included as delimiters. This is an advanced option often used in conjunction with the 'K' modifier.
'L' (Lowercase): This forces the string to be treated as lowercase during the counting process, primarily affecting modifiers that deal with specific character types, although **COUNTW** itself is generally case-insensitive regarding word separation.
'K' (Keep): This is the inverse of the default behavior. Instead of treating the characters listed in the character argument as delimiters, **COUNTW** treats them as characters to be kept and counted as part of the words. This essentially reverses the parsing logic.

For example, if a dataset contained extremely messy free text with uneven spacing and we only wanted to count non-empty words, we would combine the character argument with the 'T' modifier: `countw(phrase, ' ', 'T')`. This combination ensures that the counting is robust against accidental leading or trailing blanks, providing a more accurate word count reflective of meaningful content rather than formatting artifacts. Using modifiers effectively elevates the functionality of **COUNTW** beyond simple space counting into a sophisticated text analysis tool.

Summary and Further Resources

The **COUNTW** function is a powerful, flexible, and essential component of the SAS toolkit for anyone engaging in text processing or data cleaning operations. By understanding its three parameters--the input string, the optional character delimiters, and the behavioral modifiers--programmers can reliably quantify the word content of any character string, regardless of the complexity of the internal formatting. Whether relying on the default space delimiter or defining complex custom boundaries, **COUNTW** provides the necessary functionality to transform textual data into quantitative metrics.

The ability to specify custom word boundaries using the character argument is what distinguishes

COUNTW as a superior function for parsing non-standard data inputs compared to simpler counting methods. Always remember to explicitly define all characters that should serve as separators when the input data deviates from standard sentence structure. This proactive approach ensures accuracy and consistency across your analytical pipeline.

For the most comprehensive and detailed information regarding all modifiers and edge cases, users should consult the official **SAS documentation**. The documentation provides exhaustive definitions and examples that cover scenarios far beyond basic space counting, ensuring that you can harness the full potential of this function in specialized applications.

Note: You can find the complete documentation for the SAS **COUNTW** function here.

Related SAS Text Analysis Tutorials

To further enhance your skills in SAS text manipulation and data processing, consider exploring tutorials on related functions. Mastering these complementary tools allows for comprehensive handling of diverse data challenges.

The following tutorials explain how to perform other common tasks in SAS:

How to use the **SCAN** function to extract specific words based on delimiters.
Techniques for using the **SUBSTR** function for extracting portions of a character string.
Methods for applying **COMPRESS** or **TRANSLATE** functions for data cleaning prior to word counting.