Learning Word Counting in SAS: A Tutorial on Using the COUNTW Function

Authored by

Mohammed looti

November 14, 2025

In the dynamic field of advanced data preparation, especially when dealing with unstructured or textual information, the ability to accurately quantify elements within a character string is paramount. Analysts routinely face tasks such as processing raw log files, evaluating open-ended survey responses, or generating descriptive metrics from large text corpora. In these scenarios, determining the precise number of words in a given phrase forms a critical preliminary step in the data cleaning and analysis pipeline. For users of the SAS (Statistical Analysis System) environment, the COUNTW function offers an efficient and sophisticated solution. This function moves beyond basic space detection, providing the necessary flexibility to define what constitutes a "word" based on specific analytical needs, thereby simplifying complex word enumeration.

The central value proposition of the COUNTW function is its capacity to deliver a fast, concise numerical output representing the count of identified words within any designated character string. Its utility extends across a spectrum of data manipulation activities, from calculating fundamental descriptive statistics for text variables to underpinning complex text mining initiatives. By diligently integrating this function, SAS professionals can ensure their text-based data is meticulously characterized, significantly boosting the reliability and accuracy of subsequent statistical procedures. Fully understanding the subtleties of its optional arguments is essential for unlocking its complete potential, particularly when working with heterogeneous text environments that contain varying punctuation and special characters.

Deconstructing the COUNTW Function Syntax

To effectively harness the robust capabilities of the COUNTW function, a comprehensive understanding of its structure and the specific roles of its arguments is vital. The function is expertly engineered to manage both standard and highly customized word counting operations, allowing it to adapt effortlessly to different definitions of a "word" or its separator. While the general syntax structure is straightforward, it allows for considerable complexity through optional inputs, providing granular control over the counting process.

The standard syntax required for invoking the COUNTW function includes one required argument and two powerful optional arguments:

COUNTW(string, character, modifier)

We will now examine each parameter in detail to define its role within the overall word counting mechanism:

string: This is the mandatory first parameter. It explicitly defines the character string itself, or alternatively, the name of the SAS variable that holds the text data requiring analysis. The function rigorously processes this input to isolate and count the constituent words.

character: This is an optional parameter crucial for defining custom word separators. If this argument is omitted, SAS automatically defaults to treating only a single blank space as the primary word separator. However, the user can supply a string containing one or more characters (e.g., `',;.-_'`) to instruct the function to utilize these specific delimiters to distinguish word boundaries. This flexibility is indispensable when processing text that incorporates punctuation, hyphens, or underscores acting as intrinsic word separators.

modifier: This optional parameter accepts specific character codes that impose specialized processing rules on the COUNTW function. These codes directly influence how the input string and its specified separators are interpreted during the count operation. Key modifier codes include:

's' (Strip): This powerful code ensures that multiple consecutive separator characters are treated uniformly as a single delimiter. Critically, this code also automatically removes any leading and trailing blanks from the input string prior to the counting process, ensuring clean results.

't' (Trim): Explicitly instructs the function to remove leading and trailing blanks from the input string, which is essential for maintaining consistent word counts regardless of extraneous white space padding in the source data.

'i' (Ignore Case): Directs the function to disregard the case of the characters specified in the `character` parameter when scanning the input string for delimiters.

The judicious application of a modifier like **'s'** is generally considered a best practice in production environments, as it prevents repeated delimiters or excessive white space from artificially inflating the resulting word count. This intelligent use of optional parameters enables COUNTW to deliver highly precise and reliable word counts tailored precisely to the characteristics of your textual data.

Practical Demonstration: Default Word Counting Limitations

To fully illustrate the behavior of the COUNTW function, let us begin by establishing a sample SAS dataset containing various textual phrases. Our immediate goal is to compute the word count for each phrase using the function's default behavior, which strictly mandates that only a single blank space is recognized as a word separator.

We initiate this process by creating and populating the sample dataset in SAS using a standard Data Step:

```
/*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;
```

Within this Data Step, we define the `my_data` dataset and establish the `phrase` variable as a character string with a length of 50. The subsequent DATALINES statement facilitates the direct entry of our sample text data. The execution of PROC PRINT confirms the successful input of the data structure.

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

Following data entry, we apply the COUNTW function using its simplest, default configuration. This operation will systematically generate a new column, named `word_count`, which will store the calculated number of words for every observation contained within the `phrase` column:

```
/*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;
```

In this second Data Step, we create the `new_data` dataset and apply the function as `word_count = countw(phrase);`. By explicitly omitting the optional `character` and `modifier` parameters, SAS is instructed to recognize only blank spaces as delimiters.

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

A careful review of the resulting output immediately reveals the limitations inherent in the function's default operating mode:

For the entry "**Hey_everyone**," COUNTW incorrectly returns a count of **1**. This occurs because the underscore character is not pre-defined as a default separator, resulting in the entire sequence being processed and counted as a single, continuous word.

In the phrase "**Wow, what a great day**," the count is returned as **5**. Although a comma follows "Wow," this punctuation mark is not treated as a separator in the default mode; consequently, "Wow," is counted as one word alongside the other four.

This initial illustration clearly demonstrates that while the default behavior is adequate for text composed purely of standard, space-separated words, obtaining accurate word counts in complex, real-world textual data necessitates the definition of explicit, custom delimiters.

Customizing Delimiters for Enhanced Precision

As evidenced by the default processing, relying solely on a blank space as a separator frequently yields inaccurate word counts when the source data contains punctuation or special characters (such as underscores, slashes, or hyphens) that fundamentally function as legitimate word boundaries. The COUNTW function is designed to overcome this precise challenge by permitting users to specify custom separators via the optional `character` parameter.

To rectify the erroneous count for "Hey_everyone," we must explicitly instruct SAS to treat the underscore character as a delimiter, in addition to the standard space. This ensures that "Hey" and

"everyone" are correctly tallied as two separate lexical tokens. We achieve this critical adjustment by passing a character string containing all desired delimiters to the function's second argument.

We proceed to modify the previous Data Step to include both the blank space and the underscore within our delimiter list:

```
/*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 crucial functional change here is the expression `countw(phrase, ' _')`. The string literal `' _'` now functions as the comprehensive `character` parameter, compelling SAS to interpret both the blank space and the underscore as valid word delimiters. This calculated adjustment is essential for accurate parsing whenever non-standard separators are embedded within the source text.

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

With this modification successfully implemented, the `word_count` for "**Hey_everyone**" is now accurately returned as **2**. Similarly, to separate words from trailing punctuation, we would simply expand the delimiter list. For instance, using `countw(phrase, ' _.,')` would treat spaces, underscores, periods, and commas as word boundaries, ensuring that "Wow, what a great day" yields a count of 6 (as the comma after "Wow" is recognized as a separator). This example vividly underscores the critical role of the `character` parameter in adapting the function's behavior to the complexities inherent in real-world textual data structures.

Advanced Techniques and Robust Text Processing

While defining custom delimiters vastly improves counting accuracy, several advanced techniques and considerations can further optimize your usage of the COUNTW function within your SAS workflow. These practices are designed to ensure resilience against poorly formatted input and maximize processing efficiency.

Managing Punctuation and Whitespace: In rigorous text analysis, punctuation marks (such as commas, quotation marks, and parentheses) are typically not intended to be counted as part of a word itself. If the objective is a pure lexical count, these characters must be comprehensively included in the `character` parameter. A robust and inclusive list might appear as: `countw(phrase, ' _.,!?:;')`. Alternatively, for complex scenarios requiring intensive filtering, it is often more efficient to preprocess the input string using other dedicated functions. For example, using the `COMPRESS` function to remove specific character types entirely before applying `COUNTW` can simplify the subsequent counting logic: `word_count = countw(compress(phrase, ',.!?', 'p'), ' ');`. Here, the 'p' modifier in `COMPRESS` removes all standard punctuation, allowing COUNTW to reliably default back to space-only separation.

The Indispensable 'S' Modifier: When processing messy or user-generated data, instances of multiple consecutive spaces or repeated delimiters are common occurrences (e.g., "Word1___Word2"). Without the essential 's' modifier, some legacy COUNTW implementations might incorrectly interpret the gaps between these multiple delimiters as empty words, which artificially inflates or skews the final word count. By utilizing the 's' modifier--for example, `countw(phrase, ' _', 's')`--you explicitly instruct the function to treat all contiguous delimiter characters as a single separator. This guarantees that the final count reflects only the actual lexical tokens present in the text. Furthermore, the 's' modifier is particularly advantageous because it automatically incorporates the trimming of both leading and trailing whitespace, addressing common data cleanliness issues.

Edge Cases: Handling Empty Strings and Performance: It is important for analysts to recognize that the COUNTW function is engineered to correctly return 0 for an empty character string (`''`) or for a string composed entirely of delimiters (e.g., a string containing only multiple spaces or underscores). This standardized behavior is intuitive and ensures that null or meaningless inputs do not yield spurious non-zero counts. Regarding computational efficiency, COUNTW is highly optimized for seamless integration within the Data Step environment and consistently performs rapidly, even when applied across exceptionally large datasets.

Conclusion

The COUNTW function represents a fundamental and highly versatile utility within the SAS

ecosystem for all professionals engaged in the processing and analysis of textual data. Its core capability to accurately quantify words within a character string is significantly amplified by the flexibility offered through its optional arguments. By meticulously defining custom separators and applying powerful modifier rules, users can precisely tailor the function to handle the unique complexities of their specific data, thereby guaranteeing measurement precision across a wide array of text sources.

Mastering the function's syntax and the appropriate use of the `character` and `modifier` parameters empowers data analysts to transcend simple space-separated counts and conduct more rigorous, insightful, and reliable statistical analyses of textual variables. For the most comprehensive technical specifications and operational details, analysts should always refer directly to the Official SAS Documentation.

Additional Resources

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