How to Use the FINDW Function in SAS

Authored by

stats writer

November 19, 2025

The FINDW function in SAS is an indispensable utility for advanced text processing and data preparation. Unlike simpler character-finding routines, this powerful tool is specifically designed for accurately locating delimited words or specific phrases within a longer text string. Its ability to recognize word boundaries--defined by spaces, punctuation, or other delimiters--makes it far superior to basic character searches when precision in linguistic analysis is paramount.

This comprehensive guide delves into the mechanics of the FINDW function, exploring its precise syntax and providing detailed examples illustrating its diverse applications. We will examine how this function handles complex scenarios, such as searching for partial words, integrating variables into the search criteria, and managing delimiters. Understanding these nuances is essential for any SAS user aiming to efficiently clean, categorize, or analyze textual data embedded within their records.

By the end of this article, readers will possess a profound understanding of how to leverage the FINDW function to streamline data quality initiatives. We will highlight the critical distinction between searching for a full word versus a mere substring, demonstrating why this functional difference is key to avoiding erroneous matches in textual searches. Mastering this function unlocks significant potential for advanced text mining operations across various industries.

The primary purpose of the **FINDW** function in SAS is to pinpoint the starting position of the first character of a whole word that resides within a specified character string. If the word is found, the function returns an integer corresponding to that position; if not, it returns zero.

To properly execute this operation, the function requires two fundamental arguments defined by the following basic syntax:

FINDW(string, word)

The parameters are defined as follows:

string: This is the source text string that the function will scan. It is typically a character variable from a SAS dataset.
word: This is the specific sequence of characters that the function attempts to locate as a distinct word within the *string* argument.

While this two-argument format represents the simplest application, the **FINDW** function also supports additional arguments for controlling the search direction, modifying case sensitivity, and specifying custom word delimiters. However, for most introductory applications, the basic two-argument structure is sufficient to demonstrate the function's power, as shown in the subsequent example.

Example 1: Basic Application of the FINDW Function in SAS

To illustrate the practical usage of **FINDW**, let us first establish a sample SAS dataset containing various animal-related phrases. This dataset, named `original_data`, provides diverse contexts where the target word "pig" may appear, including instances where it is part of a larger word (e.g., "piglet", "piggie") or stands alone as a complete word.

```
/*create dataset for demonstration*/
data original_data;
input phrase $40.;
datalines;
A pig is my favorite animal
My name is piglet
Pigs are so cute
Here is a baby pig
His name is piggie
;
run;

/*view dataset structure and content*/
proc print data=original_data;
```

The resulting dataset, shown below, clearly lays out the distinct phrases that will be subjected to the word-finding operation. Note the variations in capitalization and word boundaries, which will be critical when comparing **FINDW** to other SAS functions later.

Obs	phrase
1	A pig is my favorite animal
2	My name is piglet
3	Pigs are so cute
4	Here is a baby pig
5	His name is piggie

Applying the FINDW Function to Locate Word Positions

We will now apply the **FINDW** function within a new DATA step to search for the specific word 'pig' in the **phrase** column. Crucially, the function will only register a match if 'pig' is surrounded by

delimiters (such as spaces or the start/end of the string), confirming it as a standalone word rather than just a sequence of characters within a larger word like 'piglet' or 'piggie'.

```
/*find position of first occurrence of 'pig' as a distinct word in phrase column*/
data new_data;
set original_data;
findw_pig = findw(phrase, 'pig');
run;

/*view resulting dataset with new positional column*/
proc print data=new_data;
```

Executing the above code generates a new dataset, `new_data`, which includes the original phrases and a new variable, **findw_pig**. This new column quantifies the search result by displaying the position of the first character of the identified word 'pig' within each respective phrase. If no standalone match is found, the value defaults to zero, providing a clear binary indicator of presence.

Obs	phrase	findw_pig
1	A pig is my favorite animal	3
2	My name is piglet	0
3	Pigs are so cute	0
4	Here is a baby pig	16
5	His name is piggie	0

Interpreting the Output of FINDW

The resultant column, **findw_pig**, provides critical insights into the word structure of the original data. For rows where the word 'pig' is present, the function returns a positive integer representing the byte position. For instance, in the phrase "A pig is my favorite animal," the word 'pig' starts at position 3, since the first position is 'A' and the second is a space. Conversely, if the searched term 'pig' does not occur as a distinct word within the phrase, the **FINDW** function consistently returns a value of **0**.

Examining the output, we observe the following important results: The first and fourth phrases contain 'pig' as a separate word, yielding positive positions (3 and 16, respectively). However, the second and fifth phrases, "My name is piglet" and "His name is piggie," return **0**. This result is

correct because 'pig' in those instances is merely a component of the longer words 'piglet' and 'piggie', and therefore does not qualify as a standalone word according to the function's strict word boundary definition. This confirms the function's utility in targeted word searches rather than generalized substring identification.

Understanding the Critical Difference: FIND vs. FINDW Functions

A common point of confusion for new SAS users is the operational difference between the **FIND** function and the **FINDW** function. While both are used for locating characters within a text string, their definitions of what constitutes a "match" are fundamentally distinct, leading to vastly different results in text analysis. The **FIND** function serves a broader purpose, returning the position of the first occurrence of a specified sequence of characters, or substring, anywhere within the target string.

In contrast, the **FINDW** function adheres to a strict definition of a "word." By default, SAS defines a **word** as a continuous sequence of characters bounded by word delimiters (usually spaces, tabs, punctuation, or the start/end of the string). This means **FINDW** will fail to find a match if the search term is embedded within a longer sequence of characters, such as finding 'cat' in 'caterpillar'. This distinction is absolutely vital for tasks requiring precise semantic matching.

To clearly illustrate this functional divergence, we modify our previous DATA step to include both the **FIND** and **FINDW** functions searching for the same term, 'pig', against the same set of phrases. This direct comparison highlights how the underlying logic of character search versus word search impacts the final output, particularly in cases involving compound words or partial matches.

```
/*create new dataset comparing both functions*/
data comparison_data;
set original_data;
find_pig = find(phrase, 'pig');
findw_pig = findw(phrase, 'pig');
run;

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

The resulting comparison dataset, shown below, demonstrates the dramatic differences in reported positions based on the function employed. Notice how the values in the **find_pig** column consistently show positive matches, whereas **findw_pig** yields 0 for 'piglet' and 'piggie' because they do not meet the word boundary criteria.

Obs	phrase	find_pig	findw_pig
1	A pig is my favorite animal	3	3
2	My name is piglet	12	0
3	Pigs are so cute	0	0
4	Here is a baby pig	16	16
5	His name is piggie	13	0

Analyzing Positional Results for Substrings vs. Words

The newly generated **find_pig** column accurately displays the position of the first occurrence of the character sequence 'pig' as a generic substring in the **phrase** column. For example, in the phrase "My name is piglet," 'pig' starts at position 12, even though it is part of a larger word. The **FIND** function is useful for pattern matching regardless of context.

Conversely, the **findw_pig** column strictly displays the position of the first occurrence of 'pig' when it is identified as a complete **word**. The phrases containing 'piglet' and 'piggie' result in 0s, confirming that **FINDW** ignored these entries because they lacked the requisite word delimiters. This highlights why **FINDW** is the preferred tool for linguistic analysis, keyword extraction, and ensuring data quality where full-word matching is non-negotiable.

Advanced Customization: Modifying Delimiters and Search Options

While the examples above utilize the default delimiters (space, period, etc.), the FINDW function supports optional arguments that allow users to define custom word boundaries and control search behavior. The full syntax includes modifiers for search direction, case sensitivity, and specifying alternative delimiters, which dramatically increases the function's flexibility in heterogeneous data environments.

For instance, the third argument allows for control options (e.g., 'i' for case-insensitive search, 'b' for backward search). More importantly, the fourth optional argument accepts a character variable specifying a list of characters that should be treated as delimiters, overriding the SAS defaults. This is crucial when processing data where words might be separated by non-standard characters like hyphens or underscores, which are typically ignored by default SAS word functions.

By utilizing these advanced options, analysts can fine-tune the search process to match the unique characteristics of their textual data. This level of control ensures that complex data cleaning tasks, such as standardizing terminology or extracting keywords from raw text files, can be executed with

precision and reliability, maximizing the integrity of the data processing pipeline.