Learning to Use the FINDW Function in SAS for Word Location

Authored by

Mohammed looti

November 14, 2025

The Role of the FINDW Function in SAS Text Processing

The **FINDW function** in the SAS programming environment is an invaluable utility specifically engineered for highly precise string manipulation. Its primary function is to locate and report the starting position of a specific, complete word within a longer sequence of textual data. This capability is fundamentally different from generic character searching, as **FINDW** enforces strict word boundaries, ensuring that the detected match is indeed a whole word and not merely a sequence of characters embedded within another term.

This specialized focus makes **FINDW** exceptionally powerful for tasks involving natural language processing, complex data cleaning, and conditional logic executed within SAS Data Steps or procedures. When dealing with unstructured or semi-structured text, the ability to accurately distinguish between a complete word (like 'cat') and a substring within a larger word (like 'cat' in 'concatenate') is paramount for maintaining data integrity and accuracy in analysis.

By respecting standard word delimiters--such as spaces, punctuation, or other specified characters--the **FINDW function** streamlines the process of extracting meaningful linguistic units from raw data. This is a critical advantage over simpler searching methods, allowing analysts and programmers to build robust logic that relies on exact lexical matches rather than partial sequence matches. Understanding and leveraging this function is essential for maximizing efficiency during text analysis projects in SAS.

Mastering the Syntax and Required Parameters

To effectively integrate the word-finding capabilities into your data manipulation workflows, a thorough understanding of the **FINDW function**'s syntax is required. Although the function is powerful, its structure remains highly accessible and straightforward, designed for ease of use across various analytical scenarios. Mastering the input parameters ensures that the function executes precisely as intended, returning reliable position data.

The basic structure of the function call requires two primary arguments: the source text being searched and the target word being sought. While advanced optional arguments exist (for modifiers like case sensitivity or delimiters), the core functionality relies on these two mandatory inputs.

The mandatory syntax for the **FINDW** function is defined as follows:

FINDW(string, word)

Below is a detailed breakdown of each required parameter, illustrating how they interact to facilitate accurate word searching:

string: This mandatory first argument specifies the source string--the character variable or literal text--that the function will analyze. It is the comprehensive body of text within which the function attempts to locate the specified word.
word: This second mandatory argument defines the specific **word** that the function is searching for. Crucially, the search adheres strictly to word boundary definitions, meaning that only an exact, standalone match of this **word** will be recognized within the designated *string*.

Upon successful execution, the **FINDW** function returns a numeric value corresponding to the starting character position of the first identified occurrence of the target word. Should the specified word not be found as a complete lexical unit within the source string, the function reliably returns a value of **0**. This zero return value serves as a clear flag for subsequent conditional processing steps in the SAS program.

Case Study: Applying FINDW for Precise Word Identification

To fully grasp the practical utility of **FINDW**, let us walk through a typical scenario involving text data within SAS. Imagine we are tasked with analyzing a dataset containing descriptive phrases, and our goal is to pinpoint exactly where the word 'pig' appears as a standalone term, ignoring variations or embedded substrings like 'piglet'.

First, we must establish our base data structure. We create a sample dataset named `original_data`. This dataset includes a single character column, `phrase`, which holds the textual expressions we intend to analyze. This initial step sets the stage for demonstrating the function's behavior against mixed textual inputs.

```
/* Create and populate the initial dataset */
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 the structure and content of the source dataset */
proc print data=original_data;
```

After successfully executing the above Data Step, the `original_data` dataset is created, containing the five textual observations. These observations represent a mixture of phrases where 'pig' appears as a standalone word, embedded as a substring, or as a capitalized word. The resulting structure is confirmed by the following output:

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

The next step involves applying the **FINDW** function itself. We create a new dataset, `new_data`, where we use **FINDW** to search for the explicit word 'pig' within the `phrase` column. This process generates a new variable that precisely captures the starting position of the target word, providing clear numerical results based on word boundary recognition.

```
/* Locate position of first occurrence of 'pig' using word boundaries */
data new_data;
set original_data;
findw_pig = findw(phrase, 'pig');
run;

/* Review the results dataset */
proc print data=new_data;
```

Upon execution, the `new_data` dataset is populated, including the new column, `findw_pig`, which contains the results. Observing these results confirms the function's behavior, particularly its dedication to matching only whole words.

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

The output demonstrates the core capability of **FINDW**. In the first observation, "A pig is my favorite animal," the function correctly identifies the starting position of 'pig' as **3**. Conversely, for observations where 'pig' is not isolated--such as "My name is piglet" or "His name is piggie"--the function returns **0**. This zero return value is the definitive signal that the target word, bound by delimiters, was absent, highlighting the reliability of **FINDW** for precise lexical searching and data manipulation.

FIND vs. FINDW: Crucial Differences in String Searching

A frequent point of confusion for new SAS users lies in differentiating between the FIND function and the **FINDW function**. While both are designed for searching within character strings, their underlying mechanisms and search targets are fundamentally divergent, leading to vastly different results in text analysis tasks. Choosing the wrong function can compromise the integrity of text processing operations.

The primary distinction resides in their treatment of character sequences. The **FIND function** is a general-purpose search utility designed to locate a specific substring. A substring is simply any consecutive sequence of characters. Therefore, if you use **FIND** to look for 'car' in the string 'carpenter', it will return a positive position because the sequence 'c-a-r' exists, even though it is not a standalone word. It ignores lexical boundaries entirely.

In sharp contrast, the **FINDW function** is specifically engineered for linguistic accuracy. It only returns a position if the target text is found as a complete, delimited unit--a true word. If we searched for 'car' in 'carpenter' using **FINDW**, it would return **0**, indicating no match, because 'car' is not separated by spaces or other recognized delimiters. This boundary recognition makes **FINDW** indispensable for true lexical analysis, filtering, and conditional statements based on complete words.

We can demonstrate this critical difference by applying both functions concurrently to the same source dataset, `original_data`. We will create two new columns: one using **FIND** and one using

FINDW, both searching for the sequence 'pig'.

```
/* Comparing FIND (substring search) and FINDW (whole word search) */
data new_data;
set original_data;
find_pig = find(phrase, 'pig');
findw_pig = findw(phrase, 'pig');
run;

/* View the comparative results */
proc print data=new_data;
```

The visual output of this comparative analysis dramatically illustrates the functional difference between the two utilities, as seen below:

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

In this resulting dataset, the `find_pig` column shows a positive position (not 0) for every row containing the sequence 'pig', including 'piglet' and 'piggie'. This confirms that **FIND** operates purely on the basis of character sequence, treating 'pig' as a substring regardless of context.

Conversely, the `findw_pig` column returns **0** for 'My name is piglet' and 'His name is piggie', because in these cases, 'pig' is not delimited as a standalone word. This clear divergence underscores the importance of utilizing **FINDW** when the analytical requirement demands identification of complete lexical units rather than partial character matches.

Summary and Best Practices for Text Data Manipulation

The **FINDW** function is unquestionably an indispensable utility for advanced text analysis and reliable data manipulation within the SAS ecosystem. Its core strength lies in its ability to enforce linguistic precision by locating only whole words, effectively filtering out noise generated by partial matches or embedded substrings. This precision is vital in fields such as survey response coding,

regulatory text review, and any application where the context of a word dictates its relevance.

For SAS programmers and data scientists, incorporating **FINDW** into their toolkit allows for the development of more sophisticated and robust text processing routines. When designing extraction or filtering logic, always evaluate whether you need a character match (use FIND) or a complete lexical unit match (use **FINDW**). This deliberate choice is the hallmark of efficient and accurate text handling.

Furthermore, while this article focused on the basic syntax, remember that **FINDW** accepts optional arguments (like 'A' for case-insensitive search or user-defined delimiters) that greatly enhance its flexibility. Employing these modifiers can further fine-tune the search results to meet complex data cleaning requirements, ensuring that your SAS programs deliver optimal performance and analytical accuracy.

Additional Resources for SAS Programming Proficiency

To further enhance your SAS programming skills and explore other common functions, consider reviewing the following tutorials, which cover complementary text processing and data manipulation techniques:

This article provided a targeted look at the **FINDW** function, but the broader spectrum of SAS functions--including those for regular expressions and advanced character manipulation--can unlock even greater capabilities for handling complex textual data.