Learning the SAS SCAN Function: Extracting Words from Strings

Authored by

Mohammed looti

October 27, 2025

Introduction to the SAS SCAN Function

The SAS system is a powerful platform for statistical programming and data management. When dealing with character data, one of the most essential tools available in the DATA step is the **SCAN** function. This function is specifically designed to parse a character string and efficiently extract the *n*th word, or token, based on system-defined or user-specified delimiters.

The ability to accurately and quickly segment combined data fields--such as full names, addresses, or complex codes--is crucial for data standardization and analysis. The **SCAN** function simplifies this process by automating the identification and extraction of tokens, treating spaces (by default) as separators between words. This article provides a comprehensive guide to using the SCAN function, detailing its syntax and illustrating its most common applications through practical examples in SAS.

Defining the SCAN Function Syntax

While the SCAN function supports advanced optional arguments (like specifying custom delimiters or modifiers), the core functionality relies on two main parameters. Understanding this basic structure is key to successfully implementing token extraction within your SAS programs.

The function utilizes the following basic syntax:

SCAN(string, count)

The required arguments are defined as follows:

string: This is the source character variable or literal string that the function will analyze.
count: This numerical integer specifies the position of the word to be extracted. A positive value (e.g., 1, 2, 3) extracts words counting from the left (start of the string). A negative value (e.g., -1, -2) extracts words counting from the right (end of the string).

Three Essential Methods for Using SCAN

The versatility of the **SCAN** function makes it suitable for various data cleaning tasks. We will focus on the three most common implementations used by SAS users to manipulate string variables, providing conceptual code examples for each method before diving into the live demonstration.

Method 1: Extract the *n*th Word (Positive Indexing)

This is the standard application, where a positive integer is used to retrieve a word based on its sequential position starting from the left. For example, to retrieve the second word, you simply pass the number 2 as the `count` argument.

```
data new_data;
set original_data;
second_word = scan(string_variable, 2);
run;
```

Method 2: Extract the Last Word (Negative Indexing)

A highly efficient feature of the SCAN function is its use of negative indexing. By setting the `count` to **-1**, you can quickly extract the last word in the string, regardless of how many words precede it. This is typically used for isolating surnames, file extensions, or final components of a path.

```
data new_data;
set original_data;
last_word = scan(string_variable, -1);
run;
```

Method 3: Extracting Multiple Words

To fully parse a single variable into several new variables (e.g., splitting a full name into first, middle, and last components), the **SCAN** function must be called individually for each desired output variable within the same DATA step, using the appropriate positive index for each call.

```
data new_data;
set original_data;
first_word = scan(string_variable, 1);
second_word = scan(string_variable, 2);
third_word = scan(string_variable, 3);
run;
```

Setting Up the Demonstration Dataset

Before executing the specific extraction methods, we first establish the working dataset that contains the character variable we intend to manipulate. This dataset, named `original_data`, includes a `name` variable that combines the first, middle, and last names of several individuals, along with a numerical `sales` variable.

The following DATA step code creates and displays the raw data:

```
/*create dataset*/
data original_data;
```

```
input name $20. sales;
datalines;
Andy Lincoln Bernard 55
Barren Michael Smith 41
Chad Simpson Arnolds 13
Derrick Parson Henry 29
Eric Miller Johansen 47
Frank Giovanni Goode 61
;
run;

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

As shown in the output image below, the `name` column consists of three tokens separated by spaces, which the **SCAN** function will use as default delimiters.

Obs	name	sales
1	Andy Lincoln Bernard	55
2	Barren Michael Smith	41
3	Chad Simpson Arnolds	13
4	Derrick Parson Henry	29
5	Eric Miller Johansen	47
6	Frank Giovanni Goode	61

Example 1: Isolating the Second Word

To demonstrate the utility of positive indexing, we will extract the second word from the `name` column, effectively isolating the middle name of each record. We use the index value **2** in the SCAN function call.

The following code creates a new dataset, `new_data`, which includes the original variables plus the newly calculated `second_word` variable:

```
/*extract second word in each row of name column*/
data new_data;
set original_data;
second_word = scan(name, 2);
```

```
run;

/*view results*/
proc print data=new_data;
```

After running the code, the output confirms the successful extraction. The new column `second_word` contains the middle name from each input string.

Obs	name	sales	second_word
1	Andy Lincoln Bernard	55	Lincoln
2	Barren Michael Smith	41	Michael
3	Chad Simpson Arnolds	13	Simpson
4	Derrick Parson Henry	29	Parson
5	Eric Miller Johansen	47	Miller
6	Frank Giovanni Goode	61	Giovanni

Example 2: Extracting the Final Word Using Negative Indexing

To isolate the last name (the final word) in the `name` column, we utilize the negative indexing capability of the **SCAN** function. By setting the index to **-1**, we instruct SAS to count backward from the end of the string, guaranteeing that the last token is retrieved regardless of whether the name has two, three, or more parts.

We execute the following DATA step code:

```
/*extract last word in each row of name column*/
data new_data;
set original_data;
last_word = scan(name, -1);
run;

/*view results*/
proc print data=new_data;
```

The resulting table demonstrates the efficiency of negative indexing. The new column `last_word` contains only the surname for each record, successfully extracted from the end of the input string.

Obs	name	sales	last_word
1	Andy Lincoln Bernard	55	Bernard
2	Barren Michael Smith	41	Smith
3	Chad Simpson Arnolds	13	Arnolds
4	Derrick Parson Henry	29	Henry
5	Eric Miller Johansen	47	Johansen
6	Frank Giovanni Goode	61	Goode

Example 3: Decomposing the String into Separate Fields

For scenarios requiring complete data normalization, we must separate all parts of the combined field. This involves using the **SCAN** function multiple times within the same data step, assigning a unique positive index (1, 2, 3) to retrieve each component (first, middle, last name) respectively.

The following code demonstrates this decomposition process:

```
/*extract each word in each row of name column*/
data new_data;
set original_data;
first_word = scan(name, 1);
second_word = scan(name, 2);
third_word = scan(name, 3);
run;

/*view results*/
proc print data=new_data;
```

The final output confirms that three new columns have been successfully created (`first_word`, `second_word`, and `third_word`), demonstrating how the **SCAN** function can be used repeatedly to break down complex character data into manageable, standardized fields.

Obs	name	sales	first_word	second_word	third_word
1	Andy Lincoln Bernard	55	Andy	Lincoln	Bernard
2	Barren Michael Smith	41	Barren	Michael	Smith
3	Chad Simpson Arnolds	13	Chad	Simpson	Arnolds
4	Derrick Parson Henry	29	Derrick	Parson	Henry
5	Eric Miller Johansen	47	Eric	Miller	Johansen
6	Frank Giovanni Goode	61	Frank	Giovanni	Goode

Additional Resources for SAS Programming

The **SCAN** function is a powerful tool in the character handling toolkit of SAS. Mastering its use for positive and negative indexing significantly enhances your ability to clean and standardize data. For further learning on related data manipulation techniques, consider exploring tutorials on:

Using the SUBSTR Function for fixed-position extraction.
The CATX Function for robust string concatenation.
Advanced uses of the SCAN function with custom delimiters (e.g., commas or pipes) and modifier arguments.