Learning SAS: How to Split Strings Using Delimiters

Authored by

Mohammed looti

October 31, 2025

Introduction: Mastering String Manipulation in SAS

In the expansive realm of data preparation and statistical analysis, the ability to effectively manipulate character strings is not merely useful--it is foundational. Raw data often arrives in an unstructured or semi-structured format, where critical pieces of information are consolidated into a single textual string. Extracting these components, a process commonly referred to as string parsing, is essential for transforming messy, raw input into usable, structured datasets ready for robust analysis. Without precise parsing techniques, complex data fields such as addresses, concatenated codes, or full names remain inaccessible for segmentation and individual study.

For professionals utilizing SAS, one of the most powerful and versatile tools available for this transformation is the **SCAN function**. This function is specifically engineered to provide an efficient and straightforward methodology for extracting individual segments or 'words' from a character string. Its operation relies entirely on identifying a specified delimiter, which acts as the separator between the desired pieces of information. The inherent simplicity and remarkable flexibility of the SCAN function make it an indispensable utility across various data cleaning, ETL (Extract, Transform, Load), and transformation workflows within the SAS environment.

This comprehensive guide is designed to walk you through the practical application of the **SCAN function** in SAS. We will navigate a detailed, step-by-step example, illustrating precisely how to dissect a single, combined string into multiple distinct variables. Furthermore, we will cover crucial techniques for managing the resulting output and utilizing statements like DROP to optimize and refine your final datasets, ensuring they are perfectly structured for subsequent statistical modeling or reporting needs.

Dissecting the SCAN Function: Syntax and Parameters

The **SCAN function** stands out in SAS because of its precise focus on parsing character strings. Its primary purpose is to allow users to extract the *n*th word--or more accurately, the *n*th segment--from a given string. In this context, a "word" is explicitly defined as any sequence of characters that is separated from the next sequence by one or more occurrences of the specified delimiters. This capability is absolutely indispensable when confronting compound data fields that require segregation, such as parsing concatenated transaction IDs, geographical coordinates, or, as we will demonstrate, splitting full names into individual components.

To harness this functionality effectively, a thorough understanding of the function's fundamental syntax is required. The universal structure for calling the **SCAN function** is: `SCAN(argument, n, delimiters)`. Each of these three core parameters plays a distinct and crucial role in defining the operation and the resulting output. Mastering their usage ensures accurate and reliable parsing across diverse data types and complexities.

Below is a detailed breakdown of the required arguments for the SCAN function:

argument: This parameter represents the source material--it is the character string or the existing variable within your dataset from which you intend to extract a substring.

n: This is a numerical value that dictates the position of the segment to be extracted. Specifying `1` retrieves the first word, `2` retrieves the second, and so forth. An advanced feature allows the use of negative values, which instructs SAS to scan and extract components starting from the end of the string rather than the beginning.

delimiters: This final argument is a character string that explicitly lists the characters that act as delimiters. If this parameter is omitted, SAS defaults to a standard set of common delimiters (including blanks, periods, and special symbols). However, defining a custom delimiter, such as a comma or an underscore, grants the user precise, granular control over the parsing logic, which is critical for non-standardized data formats.

Setting Up the Data Environment: Creating a Sample Dataset

To provide a clear, illustrative example of the **SCAN function** in action, we must first establish a representative sample dataset within the SAS system. This dataset, which we will name `my_data1`, will simulate real-world data where components--in this case, parts of a person's name--are concatenated into a single field, using an underscore (_) character as the primary separator. Our objective is to meticulously split these combined names into separate, clean variables representing the first, middle, and last names, thereby facilitating easier subsequent analysis.

The following SAS code snippet details the creation of this initial dataset. We employ a DATA step to define the structure and populate the records using the DATALINES statement. Following the data creation, we utilize PROC PRINT to visually confirm the dataset's structure, ensuring the input variable is correctly established before parsing commences. This preparatory step is vital for demonstrating the transformation process clearly.

```
/*create dataset: Simulating concatenated data input*/
data my_data1;
input name $25.;
datalines;
Andy_Lincoln_Bernard
Barry_Michael
Chad_Simpson_Smith
Derrick_Parson_Henry
Eric_Miller
Frank_Giovanni_Goodwill
```

```
;
run;

/*print dataset to verify initial structure*/
proc print data=my_data1;
```

Obs	name
1	Andy_Lincoln_Bernard
2	Barry_Michael
3	Chad_Simpson_Smith
4	Derrick_Parson_Henry
5	Eric_Miller
6	Frank_Giovanni_Goodwill

As illustrated in the output image above, the initial dataset, `my_data1`, successfully contains a single variable named `name`. Every record in this variable is a single character string where multiple name components are joined together by the underscore character. This configuration precisely mirrors a common data quality challenge faced by analysts, establishing the perfect foundation for applying our string parsing solution.

Implementing SCAN for Field Extraction and Transformation

With the sample dataset prepared, we can now proceed to the core transformation stage: utilizing the **SCAN function** to achieve the desired segmentation. Our goal is to systematically parse the compound `name` variable and generate three new, distinct variables: `name1` (first name), `name2` (middle name), and `name3` (last name). This requires calling the SCAN function three separate times within a single DATA step, adjusting only the 'n' parameter for each call.

The following SAS code outlines this vital transformation step. We create the new dataset, `my_data2`, which inherits the records from `my_data1`. Within this step, we assign the output of the **SCAN function** to the new variables, explicitly setting the delimiter argument to `'_'`. This rigorous approach ensures that only the underscore character is used to define the boundaries of the segments we wish to extract.

```
/*create second dataset with name split into three columns*/
data my_data2;
set my_data1;
name1=scan(name, 1, '_');
```

```
name2=scan(name, 2, '_');
name3=scan(name, 3, '_');
run;

/*view second dataset*/
proc print data=my_data2;
```

Obs	name	name1	name2	name3
1	Andy_Lincoln_Bernard	Andy	Lincoln	Bernard
2	Barry_Michael	Barry	Michael	
3	Chad_Simpson_Smith	Chad	Simpson	Smith
4	Derrick_Parson_Henry	Derrick	Parson	Henry
5	Eric_Miller	Eric	Miller	
6	Frank_Giovanni_Goodwill	Frank	Giovanni	Goodwill

Observing the resulting output for `my_data2` confirms the successful parsing operation: the original `name` variable has been correctly dissected into `name1`, `name2`, and `name3`. Each new variable now holds the corresponding substring extracted based on the underscore delimiter. A crucial aspect of the **SCAN function**'s behavior is its inherent handling of missing components. For records that do not contain sufficient delimiters to satisfy all requested parts--such as "Barry_Michael," which only yields two segments--the corresponding variable (e.g., `name3`) will receive a blank value. This standard and predictable behavior is essential for maintaining data integrity when dealing with inconsistent or incomplete source data.

Refining Your Output: Managing Columns with the DROP Statement

Once the string splitting operation is successfully completed and the new, granular variables are created, the original concatenated variable often becomes redundant. Retaining unnecessary or superseded variables can introduce clutter, needlessly increase the size of the output dataset, and potentially confuse users during downstream analytical processes. SAS provides an elegant solution for managing this structural refinement through the use of the **DROP statement**, which is executed directly within the DATA step.

The **DROP statement** serves as a powerful tool for explicit variable management, allowing the user to specify precisely which variables should be excluded from the final output dataset. This process of intentional variable exclusion is critical for data governance and achieving optimal data organization. In our specific example, since the segmented `name1`, `name2`, and `name3` variables now provide the necessary information, the original `name` variable is extraneous and should be

dropped to maximize efficiency.

The following modified SAS code integrates the **DROP statement** into the DATA step that creates `my_data2`. By adding the line `drop name;`, we instruct SAS to perform the calculation and parsing using the original variable but omit it when writing the final data table.

```
/*create second dataset with name split into three columns, drop original name*/
data my_data2;
set my_data1;
name1=scan(name, 1, '_');
name2=scan(name, 2, '_');
name3=scan(name, 3, '_');
drop name;
run;

/*view second dataset*/
proc print data=my_data2;
```

Obs	name1	name2	name3
1	Andy	Lincoln	Bernard
2	Barry	Michael	
3	Chad	Simpson	Smith
4	Derrick	Parson	Henry
5	Eric	Miller	
6	Frank	Giovanni	Goodwill

The final dataset presented in the image confirms that the output is now streamlined. It exclusively contains the newly engineered variables: `name1`, `name2`, and `name3`. The redundant original `name` variable has been successfully removed, resulting in a clean, focused dataset that is immediately ready for subsequent analytical tasks. This combination of parsing and data management techniques exemplifies best practices in SAS data preparation.

Summary and Further Exploration

The **SCAN function** represents a fundamental and indispensable tool within SAS for character string manipulation. As demonstrated through our practical example, this function provides a high-efficiency yet simple method for segmenting complex strings into discrete, meaningful components

based on the explicit definition of delimiters. This capability is absolutely vital for common data processing requirements, including comprehensive data cleaning, feature engineering for machine learning models, and preparing raw textual data for rigorous statistical analysis.

Achieving mastery in data preparation in SAS involves skillfully combining powerful functions like SCAN with essential DATA step statements. By coupling the **SCAN function** with the structural control offered by the DROP statement, analysts gain total, granular control over both the content and the structural organization of their datasets. This holistic approach ensures that the output is not only accurate in its derived values but also optimally structured for efficiency and clarity, significantly enhancing the utility of the data for any subsequent analysis.

We strongly encourage users to move beyond this basic example and actively experiment with the full range of the **SCAN function**'s parameters. Try utilizing different delimiters, employing negative values for the 'n' parameter to scan from the right, or handling strings that contain multiple different types of delimiters simultaneously. Further exploration of these nuances will solidify your expertise in handling complex data scenarios within the SAS environment.

Additional Resources for SAS Data Preparation

For those dedicated to expanding their technical repertoire in SAS programming and advanced data manipulation techniques, the following resources and related tutorials offer deeper insights into essential programming tasks and statements: