How to Easily Extract Substrings in SAS Using the SUBSTR Function

Authored by

stats writer

December 1, 2025

The SUBSTR function (short for Substring) is one of the most essential SAS character functions, designed specifically for manipulating text data. Its primary purpose is to efficiently extract a specific portion of a larger string or character value based on defined starting points and lengths. This capability is fundamental for data cleaning, parsing identifiers, and standardizing text fields.

Understanding how the SUBSTR function operates requires familiarity with its three mandatory arguments. These arguments precisely define which characters are to be retrieved from the source value. When executed, the function returns the specified segment as a new string, allowing analysts to create new variables or subset data based on textual patterns. Mastery of this function is vital for anyone working extensively with character data in a SAS environment.

The basic syntax for applying the SUBSTR function is straightforward, requiring the source text, the position where the extraction should begin, and the total number of characters to extract from that starting point. This functionality ensures high precision when handling character data.

Understanding the SUBSTR Function Syntax

The core structure of the SUBSTR function remains consistent across all applications within the DATA step. It requires three specific parameters to successfully isolate the desired substring. It is crucial to remember that character indexing in SAS starts at position 1, not 0.

The generic syntax is presented as:

SUBSTR(Source, Position, N)

Where the arguments are defined as follows:

Source: This is the mandatory input character variable or literal string from which characters will be extracted.
Position: This numeric value specifies the exact starting point (character number) within the source string where the extraction process should begin.
N: This numeric value determines the total **N**umber of characters, counting from the specified Position, that will be included in the resulting substring.

Four Essential Methods for Using SUBSTR

The versatility of the SUBSTR function allows it to handle various text manipulation tasks. While the syntax remains constant, adjusting the arguments enables four primary use cases that are frequently encountered in real-world SAS programming scenarios. These methods range from simple truncation to conditional variable creation.

Method 1: Extracting the First N Characters from a String

The simplest application of **SUBSTR** involves isolating characters starting from the very beginning of the source string. To achieve this, the **Position** argument must always be set to **1**. The **N** argument then dictates how many initial characters are to be retained. This is invaluable when dealing with fixed-width data or prefix identification.

For instance, if you need to extract the first four characters of a variable named `string_variable` to create a new variable called `first_four`, the syntax is clean and direct:

```
data new_data;
set original_data;
first_four = substr(string_variable, 1, 4);
run;
```

Method 2: Extracting Characters Within a Specific Range

When the desired characters are located somewhere in the middle of a longer string, both the **Position** and **N** arguments become crucial. The **Position** argument must be set to the exact index where the desired sequence begins. The **N** argument must then reflect the count of characters needed, starting from that defined position.

To extract characters from the second position up to (and including) the fifth position, you would start at position 2 (`Position=2`) and extract a total of four characters (`N=4`). This method is frequently used for extracting embedded codes, middle initials, or specific date components from concatenated strings.

```
data new_data;
set original_data;
two_through_five = substr(string_variable, 2, 4);
run;
```

Method 3: Extracting the Last N Characters from a String

Extracting characters from the end of a string is slightly more complex, as it requires dynamically calculating the starting position using the **LENGTH function**. This approach is necessary because string lengths vary across observations in a dataset. The calculation involves finding the total length of the source string and then subtracting the number of characters needed (minus one) to determine the correct starting index.

For example, if we want the last three characters (`N=3`), the starting position is calculated as

`LENGTH(string_variable) - 2`. If the string is 10 characters long, the calculation is 10 - 2 = 8. Starting at position 8 and taking 3 characters (8, 9, 10) retrieves the final three characters. This is essential for obtaining suffixes or file extensions.

```
data new_data;
set original_data;
last_three = substr(string_variable, length(string_variable)-2, 3);
run;
```

Method 4: Conditional Logic Using SUBSTR for Variable Creation

The **SUBSTR function** is not only used for creating new variables; it is also highly effective within conditional logic statements, such as **IF-THEN/ELSE statements** in the DATA step. This allows users to test whether a specific substring exists or matches a criterion, and then execute an action, such as setting a flag or classifying an observation.

By using `SUBSTR` on the left side of a comparison operator (e.g., equals sign), we can check if a segment of the source string meets a certain condition. If the condition is met (e.g., the first four characters equal 'some'), a new variable can be assigned a specified value, resulting in powerful data filtering or categorization based on text components.

```
data new_data;
set original_data;
if substr(string_variable, 1, 4) = 'some_string' then new_var = 'Yes';
else new_var = 'No';
run;
```

Setting Up the Sample Dataset in SAS

To demonstrate the practical application of the four methods outlined above, we will utilize a simple sample dataset in SAS. This dataset, named `original_data`, contains a single character variable, `team`, which stores various team names. This setup allows us to clearly observe how the `SUBSTR` function modifies or extracts components from these character strings.

The code below employs a DATA step with the `DATALINES` statement to quickly create the necessary input data. Following its creation, the `PROC PRINT` procedure is used to display the initial contents of the dataset, ensuring we have a baseline for comparison.

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

```
input team $1-10;
datalines;
Warriors
Wizards
Rockets
Celtics
Thunder
;
run;

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

Obs	team
1	Warriors
2	Wizards
3	Rockets
4	Celtics
5	Thunder

Example 1: Extracting the Initial Four Characters

In this first example, we apply Method 1 to truncate the team names, isolating only the first four characters from the `team` variable. We define the start position as **1** and the length (N) as **4**. This creates a new variable, `first_four`, containing the resulting substring.

This approach is effective for generating abbreviated codes or standardizing identifiers where only the leading characters are significant. Observe how the procedure processes each observation in the input dataset and successfully populates the new variable.

```
/*Applying SUBSTR to extract the first 4 characters*/
data new_data;
set original_data;
first_four = substr(team, 1, 4);
run;

/*Displaying the new dataset*/
```

proc print data=new_data;

Obs	team	first_four
1	Warriors	Warr
2	Wizards	Wiza
3	Rockets	Rock
4	Celtics	Celt
5	Thunder	Thun

As illustrated in the output, the `first_four` variable now accurately holds the leading four characters extracted from the original `team` variable for every record.

Example 2: Isolating Characters from the Middle of the String

Building on Method 2, this example demonstrates extracting a specific internal segment. We aim to retrieve characters starting from the second position (index 2) and continuing for a length of four characters. This means the resulting substring will include characters originally found at positions 2, 3, 4, and 5.

The resulting variable, `two_through_five`, isolates the internal sequence, which can be critical for extracting codes that follow a set pattern after an initial prefix. Note that the starting position determines the beginning of the extraction, and the length parameter (4) dictates how far the function reads from that point.

```
/*Extracting four characters starting from position 2*/
data new_data;
set original_data;
two_through_five = substr(team, 2, 4);
run;

/*Viewing the dataset with the new middle-string variable*/
proc print data=new_data;
```

Obs	team	two_through_five
1	Warriors	arri
2	Wizards	izar
3	Rockets	ocke
4	Celtics	elti
5	Thunder	hund

Example 3: Extracting Trailing Characters Using LENGTH

This example demonstrates Method 3, which requires combining the **SUBSTR** function with the **LENGTH function** to handle variable-length strings dynamically. We aim to extract the last three characters of the `team` name. Since the lengths of 'Warriors' (8 characters) and 'Thunder' (7 characters) differ, we cannot use a fixed starting position.

The expression `LENGTH(team) - 2` calculates the dynamic starting point. If 'Warriors' is 8 characters long, the starting position is 8 - 2 = 6. Starting at position 6 and taking 3 characters (`, 3`) correctly captures the final three letters (`ors`). This robust method ensures accurate extraction regardless of the source string's total length.

```
/*Calculating starting position using LENGTH function to get last 3 chars*/
data new_data;
set original_data;
last_three = substr(team, length(team)-2, 3);
run;

/*Viewing the dataset with the extracted suffix*/
proc print data=new_data;
```

Obs	team	last_three
1	Warriors	ors
2	Wizards	rds
3	Rockets	ets
4	Celtics	ics
5	Thunder	der

Example 4: Conditional Flagging Based on Substring Content

Our final example demonstrates Method 4: integrating `SUBSTR` within an `IF-THEN/ELSE` structure to perform logical comparisons. We are creating a new categorical variable, `W_Team`, which serves as a flag indicating whether the team name begins with the letter 'W'.

Inside the **DATA step**, we use `SUBSTR(team, 1, 1)` to extract the very first character. This extracted character is then compared against the literal value 'W'. If the condition is met, `W_Team` is assigned 'Yes'; otherwise, it defaults to 'No'. This is a powerful technique for data categorization and quality checking within a large dataset.

```
/*Using SUBSTR in conditional logic to create a flag variable*/
data new_data;
set original_data;
if substr(team, 1, 1) = 'W' then W_Team = 'Yes';
else W_Team = 'No';
run;

/*Viewing the final categorized dataset*/
proc print data=new_data;
```

Obs	team	W_Team
1	Warriors	Yes
2	Wizards	Yes
3	Rockets	No
4	Celtics	No
5	Thunder	No

Conclusion and Further Resources

The **SUBSTR** function is a foundational tool for character manipulation in SAS, offering precise control over substring extraction. By mastering the adjustment of its three arguments--Source, Position, and N--users can perform complex data cleaning, parsing, and classification tasks, as demonstrated through the four core methods.

For those interested in performing additional text processing or other common data transformations in SAS, the following resources provide guidance on related procedures and functions:

Explore tutorials on using the **FIND** function for locating specific text patterns.
Learn about the **SCAN** function for tokenizing strings based on delimiters.
Investigate methods for efficiently handling and cleaning large character fields in the SAS environment.