How to use the FINDC Function in SAS?

Authored by

stats writer

November 19, 2025

Welcome to this comprehensive guide dedicated to mastering the function of SAS string manipulation, specifically focusing on the powerful and often critical `FINDC` routine. In the world of data processing, analyzing and extracting information from text fields--or string data--is a fundamental requirement. Whether you are validating user input, parsing log files, or cleaning up messy datasets, having reliable tools to locate specific components within a text variable is essential for successful data governance and analysis. This article serves as an expert overview of the `FINDC` function, detailing its syntax, mechanism, and practical application within the data step environment.

We will begin by defining precisely what the `FINDC` function achieves, contrasting it sharply with similar functions like `FIND`, which often leads to user confusion. Following the foundational introduction, we will explore the mandatory and optional arguments necessary to execute `FINDC` successfully. The core of this tutorial involves practical, step-by-step examples demonstrating how to apply `FINDC` to search for specific sets of characters within text strings, accompanied by explanations of the resulting positional values. Finally, we will summarize the key benefits of incorporating this function into your SAS programming toolkit, emphasizing its efficiency when dealing with diverse character lists rather than fixed substrings. Understanding `FINDC` is vital for any programmer seeking to elevate their capability in handling complex textual data structures.

Understanding the FINDC Function in SAS

The `FINDC` function in SAS is a specialized routine designed for searching text data. Unlike functions that search for entire words or phrases, `FINDC` is engineered to return the position of the first occurrence of **any individual character** contained within a specified list (or **charlist**) inside a target string. This capability makes it exceptionally useful for scenarios where you need to detect the presence of one or more distinct characters, such as punctuation marks, delimiters, or a defined set of letters, within a variable. The value returned by `FINDC` is always a numeric integer representing the byte or character position of the match, depending on the session encoding.

The distinction between searching for a specific sequence of characters (a fixed substring) and searching for any single character from a group is the key reason for the existence of `FINDC`. When `FINDC` scans the input string, it evaluates each position sequentially. The moment it encounters a character that matches any element defined in the **charlist** argument, the function immediately stops searching and returns the index of that match. If the search progresses to the end of the input string without finding any match from the defined character list, the function returns a value of 0. This return mechanism is consistent across all SAS string functions that indicate positional location.

Furthermore, `FINDC` is highly versatile because it can be influenced by optional arguments that modify its behavior, such as ignoring case or treating certain character types (like digits or spaces)

specially. For instance, you can use built-in character constants, which simplify the process of searching for common patterns without manually listing every single character. By understanding how to structure the search list and interpret the resulting position index, SAS programmers gain precise control over text parsing tasks, ensuring that data quality checks and transformations are executed reliably and efficiently within the **SAS** environment.

Syntax and Arguments of FINDC

To effectively utilize `FINDC`, it is essential to grasp its core syntax structure, which consists of mandatory arguments and various optional modifiers that control the search process. The basic syntax is straightforward and requires the specification of two primary components: the source text and the characters being sought. However, in its full form, the function allows for much greater complexity and customization through additional parameters that dictate search direction and character set classification.

The function uses the following fundamental syntax structure:

FINDC(string, charlist)

Where the arguments are defined as follows:

string: This is the mandatory argument that represents the source string or variable that the function will analyze. This argument must evaluate to a character value.
charlist: This is the mandatory argument defining the list of individual characters to search for within the **string**. This list is treated as a set of unique characters, not a sequence, meaning the order of the characters in the list does not affect the search outcome.

While the simplified syntax above is the most common use case, `FINDC` also accepts optional arguments that extend its utility. For example, the **modifier** argument allows the programmer to specify options such as 'i' (ignore case), 't' (trim trailing blanks), or 'a' (search for special alphanumeric characters). Mastering these modifiers is crucial for tasks requiring case-insensitive searches or searches against predefined character classes, thereby significantly reducing the amount of manual coding required to achieve specific text processing goals. Always consult the official SAS documentation for a complete list of available modifiers and their exact effects on the search operation.

Practical Example: Searching for Specific Characters

To demonstrate the utility of `FINDC`, let us work through a practical example within the data step. Suppose we have a dataset containing employee names, and we are tasked with determining if any name contains any of the infrequently used letters 'x', 'y', or 'z'. This type of operation is perfect

for `FINDC`, as we are interested in locating any one of these individual characters, not the exact substring "xyz".

First, we establish our initial dataset in **SAS**:

```
/*create dataset*/
data original_data;
input name $25.;
datalines;
Andy Lincoln Bernard
Barren Michael Smith
Chad Simpson Arnolds
Derrick Smith Henrys
Eric Millerton Smith
Frank Giovanni Goode
;
run;

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

Executing this code generates the original dataset, which we can visualize:

Obs	name
1	Andy Lincoln Bernard
2	Barren Michael Smith
3	Chad Simpson Arnolds
4	Derrick Smith Henrys
5	Eric Millerton Smith
6	Frank Giovanni Goode

Now, we proceed to apply the `FINDC` function. Our goal is to create a new variable, **first_xyz**, that holds the position of the first occurrence of either 'x', 'y', or 'z' within the **name** variable. Notice how we pass 'xyz' as the **charlist** argument; **SAS** interprets this as a set of three independent search targets.

```
/*find position of first occurrence of either x, y or z in name*/
data new_data;
```

```
set original_data;
first_xyz = findc(name, 'xyz');
run;

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

The resulting dataset clearly illustrates how `FINDC` operates, returning a precise index for the located character.

Obs	name	first_xyz
1	Andy Lincoln Bernard	4
2	Barren Michael Smith	0
3	Chad Simpson Arnolds	0
4	Derrick Smith Henrys	19
5	Eric Millerton Smith	0
6	Frank Giovanni Goode	0

The newly generated column, **first_xyz**, displays the positional index of the first match found from the set {'x', 'y', 'z'}. For instance, in the first row, "Andy Lincoln Bernard," the search finds the 'y' in "Andy" at position 4. It is important to note that the function searches from left to right and reports only the first hit. If none of the specified characters are present in a given observation, such as in the name "Barren Michael Smith," the `FINDC` function gracefully returns a value of **0**, signaling that no match was found for that particular string. This zero-value return is crucial for subsequent logical processing, allowing programmers to easily filter observations that require special handling based on the presence or absence of these key characters.

FINDC vs. FIND: A Critical Distinction

A common point of confusion for new SAS programmers is the difference between the `FINDC` function and its closely related counterpart, the `FIND` function. While both functions aim to locate content within a source string and return a positional index, their methodologies are fundamentally distinct and serve entirely different purposes in text processing. Understanding this difference is essential for writing accurate and efficient SAS code.

The **FIND** function is designed to locate the starting position of a specific substring (a sequence of characters) within another string. When using **FIND('Source', 'Sub')**, SAS attempts to match the

complete sequence 'Sub' exactly as defined. If the search term is 'Smith', the function looks for 'S-m-i-t-h' occurring consecutively. Conversely, the **FINDC** function operates on individual characters. When using **FINDC('Source', 'Smith')**, the function searches for the first occurrence of 'S', or 'm', or 'i', or 't', or 'h'--treating the input as a character set, not a sequence. This differentiation dictates which function should be used depending on whether the requirement is to locate a fixed pattern or to check for the inclusion of any character from a defined group.

To highlight this critical difference, consider the following demonstration where we search for the sequence 'Smith' using both **FIND** and **FINDC** on our existing dataset. This example clearly shows how **FIND** requires an exact, contiguous match of the entire term, while **FINDC** only requires the presence of any single component character.

```
/*create new dataset*/
data new_data;
set original_data;
find_smith = find(name, 'Smith');
findc_smith = findc(name, 'Smith');
run;

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

The output below provides a powerful visualization of how the two functions approach the search task differently:

Obs	name	find_smith	findc_smith
1	Andy Lincoln Bernard	0	7
2	Barren Michael Smith	16	9
3	Chad Simpson Arnolds	0	2
4	Derrick Smith Henrys	9	5
5	Eric Millerton Smith	16	3
6	Frank Giovanni Goode	0	8

Analyzing the results, we observe that the **find_smith** column correctly returns **0** for the first row ("Andy Lincoln Bernard") because the complete substring 'Smith' is not present. However, the **findc_smith** column returns **7** for that same row. This result is achieved because **FINDC** found the

character 'i' (which is part of the 'Smith' character set) at position 7 (within the word "Lincoln"). For the second row, "Barren Michael Smith," **FIND** returns **16**, indicating where the fixed sequence 'Smith' begins, while **FINDC** returns **1**, having found the first character 'B' (which is not in 'Smith') then proceeding to the 'a' and so on, until it finds the first match from the list {'S', 'm', 'i', 't', 'h'}, which is the 'S' in 'Barren' at position 7, or the 'i' in Michael at position 9. Wait, let's re-examine the results for row 2: "Barren Michael Smith". 'S' is at position 7 in 'Barren' is incorrect. 'B' is 1, 'a' is 2, 'r' is 3, 'r' is 4, 'e' is 5, 'n' is 6, space is 7. M is 8. I is 9. C is 10. H is 11. A is 12. E is 13. L is 14. Space is 15. S is 16. The first character from {'S', 'm', 'i', 't', 'h'} is 'i' in "Michael" at position 9. Let's check the image output. For row 2, `findc_smith` is 9. This confirms 'i' at position 9 is the first match. This confirms that `FINDC` searches for any individual character from the list, making it highly effective for character-level parsing and validation within the data step.

Advanced Usage: Modifiers and Character Sets

The true power of the `FINDC` function is unlocked when utilizing its optional modifier argument, which allows programmers to control the search behavior without writing complex conditional logic. These modifiers are single characters placed within quotes and passed as the third argument to the function. They dictate properties such as case sensitivity, handling of trailing blanks, and the interpretation of the **charlist** argument itself, streamlining many common text cleansing tasks in SAS.

Key modifiers include: **'i'** for case-insensitive searching (e.g., finding 'a' regardless of whether it is 'A' or 'a'); **'t'** for trimming trailing blanks from the **string** argument before searching, which is often crucial for data read from fixed-length files; and **'v'** (for variable) which inverts the meaning of the **charlist**, turning it into a list of characters NOT to search for (i.e., searching for characters that are **not** in the list). Furthermore, the **'k'** modifier allows the specification of a character constant, such as **'k'** followed by **'d'** (for digits) or **'w'** (for word characters), enabling the search for predefined character classes. For example, using **FINDC(name, ' ', 'k')** would search for the first occurrence of a space.

The use of character constants with the **'k'** modifier is particularly valuable for identifying structural elements within a string. For instance, if you need to find the location of the first non-alphanumeric character in a user-provided identifier, you can use the syntax with appropriate constants. This eliminates the tedious process of manually listing every possible punctuation mark or special symbol. By leveraging these modifiers, SAS programmers can write highly efficient and reusable code that dynamically adapts to various data quality challenges, ensuring that text analysis is both robust and scalable across large datasets processed within the data step.

Benefits of Using FINDC for Data Validation

The `FINDC` function provides numerous benefits, making it an indispensable tool for data validation and preparation in the **SAS** environment. Its character-set approach is inherently superior to **FIND** when the goal is to confirm the presence or absence of multiple potential markers, delimiters, or restricted characters within a data field. This efficiency saves considerable time and processing power compared to writing multiple nested conditional statements or complex regular expressions for simple character checks.

One major benefit is its utility in data cleansing. If a field, such as a phone number or an ID code, is restricted to only numeric digits, `FINDC` can quickly identify if any prohibited non-numeric characters (like letters, punctuation, or symbols) are present. By searching for a character list containing all invalid characters, or by using the inverse search modifier ('v') along with a digit constant, the programmer receives an immediate index of the violation or a zero if the string is clean. This speed is crucial when processing billions of records where performance optimization is paramount.

Furthermore, `FINDC` simplifies the parsing of delimited data where the delimiter might not be consistent. For instance, if a log file uses either a comma, a semicolon, or a pipe symbol as a separator, `FINDC(text_variable, ',;|')` instantly returns the position of the first encountered separator, allowing for easy subsequent manipulation using functions like `SUBSTR` or `SCAN`. This adaptability, combined with the precision of positional indexing, establishes `FINDC` as a vital component in the data step programmer's arsenal for high-volume text manipulation and data quality assurance in SAS.

Summary of Key FINDC Features

In summary, the `FINDC` function is a specialized and powerful tool for character-level searching within text variables in **SAS**. It is designed specifically to identify the position of the first occurrence of **any individual character** belonging to a defined set, providing an index or returning **0** if no match is found. This functionality distinguishes it clearly from the `FIND` function, which requires an exact match of a fixed substring sequence.

Key features and applications include:

Character Set Searching: It treats the search list as a collection of independent characters, ideal for checking for multiple delimiters or invalid characters simultaneously.
Positional Indexing: Returns the byte or character index of the first match, facilitating precise data extraction using functions like `SUBSTR`.
Modifier Support: Offers robust control over the search via optional arguments, enabling case-insensitive searches, trailing blank trimming, and inverse searches.
Predefined Character Sets: Allows the use of constants (e.g., digits, uppercase letters) through the 'k' modifier, simplifying complex pattern matching without manual listing.

Mastering `FINDC` is essential for efficient and robust text processing. By leveraging its unique capabilities, **SAS** programmers can significantly streamline data validation, parsing, and cleansing processes, ensuring data integrity across various analytical tasks. It is a cornerstone function for anyone working extensively with character data within the data step.

Further Resources and Functions

While `FINDC` addresses character-level searches, **SAS** offers a rich ecosystem of string functions to handle every conceivable text manipulation requirement. For tasks involving regular expressions (RegEx) or pattern matching that goes beyond simple character lists, advanced functions like `PRXMATCH` and `PRXPOSN` are recommended. For sequence-based searches, `FIND` remains the appropriate choice. For cleaning characters from a string based on a character list, functions like `COMPRESS` are often used in conjunction with `FINDC` results.

To continue enhancing your SAS programming expertise, consider exploring the official documentation on other common string manipulation routines:

SCAN Function: Used for extracting specific words or tokens from a string based on delimiters.
SUBSTR Function: Used for extracting a specific substring based on start position and length.
COMPRESS Function: Used to remove or retain specific sets of characters from a string.
TRANSLATE Function: Used to convert specific characters to other characters.

By integrating `FINDC` with these complementary functions, you will possess a complete toolset for highly sophisticated and effective data preparation within the **SAS** system.