Learn How to Replace Characters in Strings Using SAS: A Comprehensive Guide

Authored by

Mohammed looti

October 31, 2025

In the expansive realm of data processing and advanced analytics, the ability to perform robust **string** manipulation is not merely a convenience--it is a foundational requirement. Data, particularly textual data, rarely arrives in a perfectly clean state, often necessitating the cleaning, standardization, or reformatting of specific **characters** or substrings. For professionals utilizing **SAS**, the industry-leading statistical software suite, mastering the correct functions for text transformation is paramount. Among the most efficient and powerful tools available for targeted text substitution is the **`TRANWRD()` function**, specifically designed to replace sequences of text rather than individual characters.

This authoritative guide offers a comprehensive examination of the **`TRANWRD()` function**, detailing its exact syntax, core functionality, and practical implementation within **SAS** programming environments. We will walk through detailed examples illustrating how to leverage this function to perform precise text substitutions within your **datasets**, whether your goal is to standardize inconsistent entries, correct pervasive spelling errors, or completely eliminate unwanted phrases. By the conclusion of this tutorial, you will possess a clear understanding of how to integrate the **`TRANWRD()` function** into your daily **SAS** workflows, significantly enhancing the quality and readiness of your character data for subsequent analysis.

Deep Dive into the TRANWRD() Function in SAS

The **`TRANWRD() function`** in **SAS** serves a highly specialized and vital purpose: replacing every instance of a designated target **string** within a source **string** with a specified replacement **string**. The function's name, derived from "TRANslate WoRD," differentiates it from other **SAS** character functions like `TRANSLATE()`, which operates character-by-character. `TRANWRD()` excels when dealing with entire words, phrases, or sequences of characters, making it the preferred choice for semantic data cleaning and standardization tasks.

One of the function's most powerful attributes is its flexibility regarding length. Unlike some fixed-length character replacement methods, the length of the replacement **string** in `TRANWRD()` does not need to match the length of the target **string**. **SAS** automatically manages the resulting length of the output **string**, ensuring that the transformation is seamless whether you are replacing a long phrase with a shorter abbreviation or vice versa. This dynamic resizing capability makes `TRANWRD()` an exceptionally versatile tool for data harmonization.

The syntax of the **`TRANWRD()` function** is elegantly simple, requiring three essential arguments to execute the replacement operation:

`source`: This is the mandatory initial argument, representing the original **string** or the character **variable** containing the text you wish to modify.
`target`: This argument defines the specific substring that the function will search for within the

`source`. It must be a literal **string** enclosed in quotation marks.
`replacement`: This is the literal **string** that will be substituted for every occurrence of the `target` **string**. It is also enclosed in quotes and can be an empty **string** if the goal is removal.

It is critical to note that the **`TRANWRD()`** operation is inherently case-sensitive. It will only replace exact matches, meaning "Wild" will not be replaced if the target is specified as "wild". If your data requires case-insensitive replacement, the standard practice involves nesting `TRANWRD()` within **SAS** functions like `UPCASE()` or `LOWCASE()` to enforce consistent capitalization before the substitution is performed.

Method 1: Substituting Substrings with New Characters

The primary use case for `TRANWRD()` is the direct substitution of one textual element with another. This method is indispensable when your data requires standardization--for instance, changing regional spellings, updating product codes, or correcting globally recognized errors across a text **variable**. By providing a non-empty **string** for the replacement argument, you ensure that the text is updated precisely as specified.

This process is typically executed within a **SAS DATA step**, where new **variables** are computed based on the existing **data**. The structure below illustrates the foundational syntax:

```
data new_data;
set original_data;
new_variable = tranwrd(old_variable, "OldString", "NewString");
run;
```

In this construction, the **DATA step** first reads the input **dataset** (`original_data`). The assignment statement then calls the `tranwrd` function, which systematically scans the contents of `old_variable`. For every instance where it identifies the `"OldString"`, it replaces it with the designated `"NewString"`, storing the result in the newly created `new_variable`. This approach ensures that the original source **variable** remains intact, allowing for easy verification and comparison of the transformation.

Method 2: Eliminating Substrings by Replacing with Blanks (Removal)

A second, equally critical application of the **`TRANWRD()` function** involves the complete removal of unwanted substrings, which is essential for detailed data cleansing. This is achieved by providing an empty **string** (represented as `""`) as the replacement argument. This technique allows analysts to strip away extraneous elements such as metadata tags, special **characters**, or numerical qualifiers that might clutter the text or interfere with subsequent text analytics procedures.

For example, if a **variable** contains entries like "ID-2023_Data" and you need only "Data", you would use this method to replace "ID-2023_" with an empty **string**. The implementation within a **SAS DATA step** is structured as follows:

```
data new_data;
set original_data;
new_variable = tranwrd(old_variable, "OldString", "");
run;
```

The use of `""` as the third argument is the crucial distinction here. When **SAS** encounters this, it substitutes the identified `"OldString"` with nothing, effectively deleting the substring and collapsing the resulting text. Analysts must be precise when defining the `target` **string**, particularly regarding surrounding whitespace. If the target includes a trailing space (e.g., `"Wild "`), that space will be removed along with the word, resulting in perfectly concatenated words. If the space is not included, the resulting output may contain double spaces where the word used to be.

Setting Up Our Illustrative Dataset

To provide clear and verifiable demonstrations of the `TRANWRD()` function in action, we will utilize a simple, self-contained **SAS dataset**. This setup ensures that the effects of our character replacement operations are immediately apparent and traceable back to the source data. Our sample **dataset**, named `original_data`, features a single character **variable** called `team`, which contains various descriptive animal team names.

The following **SAS** code block initiates the creation of this sample data. The `data` statement begins the **DATA step**. The `input` statement defines the structure of the `team` **variable**, specifying it as character type (`$`) with a length of 20 **characters**. The `datalines` statement embeds the actual data records directly within the program, followed by the `run` statement to execute the creation. Finally, `proc print` is used to display the initial state of the **dataset**, which serves as our critical baseline for comparison.

```
/*create dataset*/
data original_data;
input team $1-20;
datalines;
Angry Bees
Angry Hornets
Wild Mustangs
Kind Panthers
Kind Cobras
```

```
Wild Cheetahs
Wild Aardvarks
;
run;

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

Executing the setup code produces the initial output table. This table is essential as it confirms the exact starting values in the `team` **variable**. We will now use this original **data** to demonstrate the two core methods of character replacement, making it easy to observe which records are affected by the `TRANWRD()` function and how the text is ultimately transformed.

Obs	team
1	Angry Bees
2	Angry Hornets
3	Wild Mustangs
4	Kind Panthers
5	Kind Cobras
6	Wild Cheetahs
7	Wild Aardvarks

Example 1: Replacing a Substring with a New Word

This first practical example showcases Method 1: substituting a target substring with a completely new **string**. Our objective is to standardize the team nomenclature by replacing the adjective "Wild" with "Fast" across all relevant entries in the `team` **variable**. This operation highlights the targeted nature of `TRANWRD()`, affecting only the specified phrase while leaving all other text unchanged.

The transformation is performed using the following **SAS** code. Note the creation of a new **variable**, `new_team`, which holds the modified **data**, allowing the output to display both the original and the transformed values side-by-side.

```
/*replace "Wild" with "Fast" in team variable*/
data new_data;
set original_data;
new_team = tranwrd(team, "Wild", "Fast");
```

```
run;

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

The core statement, `new_team = tranwrd(team, "Wild", "Fast")`, instructs **SAS** to locate "Wild" and replace it with "Fast." Because the function is case-sensitive, it only targets the exact match. The resulting output confirms the successful execution of this targeted substitution, clearly showing the new descriptive variable alongside the original input.

Obs	team	new_team
1	Angry Bees	Angry Bees
2	Angry Hornets	Angry Hornets
3	Wild Mustangs	Fast Mustangs
4	Kind Panthers	Kind Panthers
5	Kind Cobras	Kind Cobras
6	Wild Cheetahs	Fast Cheetahs
7	Wild Aardvarks	Fast Aardvarks

As confirmed by the `proc print` output, entries such as "Wild Mustangs" and "Wild Cheetahs" are correctly transformed into "Fast Mustangs" and "Fast Cheetahs," respectively. Crucially, the records that did not contain the target **string** "Wild" (e.g., "Angry Bees") remain completely unaltered in the `new_team` **variable**, demonstrating the precision of the **`TRANWRD()` function**.

Example 2: Complete Removal of a Substring

Our second example demonstrates Method 2: leveraging `TRANWRD()` for data cleaning through the complete removal of a substring. We aim to strip the adjective "Wild" entirely from the team names, retaining only the animal name itself. This is achieved by substituting the target word with an empty **string**, thereby deleting the unwanted text. For this demonstration, we are explicitly targeting "Wild" (with no trailing space) to illustrate the direct substitution behavior.

The **SAS** code necessary for this removal is highly similar to the replacement example, with the critical difference lying in the third argument:

```
/*replace "Wild" with a blank in team variable*/
data new_data;
set original_data;
```

```
new_team = tranwrd(team, "Wild", "");
run;

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

By using `""` as the replacement value, the statement `new_team = tranwrd(team, "Wild", "")` instructs **SAS** to find "Wild" and replace it with zero **characters**. This effectively compacts the **string**, removing the targeted word and closing the gap it occupied. The subsequent output visualizes the result of this cleansing operation.

The output clearly displays the successful removal of the word "Wild" from the prefixes of the relevant team names:

Obs	team	new_team
1	Angry Bees	Angry Bees
2	Angry Hornets	Angry Hornets
3	Wild Mustangs	Mustangs
4	Kind Panthers	Kind Panthers
5	Kind Cobras	Kind Cobras
6	Wild Cheetahs	Cheetahs
7	Wild Aardvarks	Aardvarks

The resulting `new_team` **variable** now contains cleaner, simpler entries such as "Mustangs," "Cheetahs," and "Aardvarks." It is important to observe that because we targeted only "Wild" and not "Wild " (including the space), the output retains the space between the removed word and the animal name, resulting in a leading space for those entries (e.g., " Mustangs"). This detail underscores the exact, literal nature of the substitution performed by the **`TRANWRD()` function**.

Conclusion: Mastering String Replacement in SAS

The **`TRANWRD()` function** represents a cornerstone of efficient **string** manipulation within the **SAS** ecosystem. Through the practical demonstrations provided, we have established its utility in two primary areas: substituting specific substrings with new text for standardization, and completely removing unwanted phrases by replacing them with empty **strings** for data cleansing.

Its straightforward structure, combined with its ability to handle variable-length replacements,

makes it superior to character-by-character functions when dealing with word-level or phrase-level changes. By confidently integrating `TRANWRD()` into your **DATA step** workflows, **SAS** programmers can significantly reduce the time spent on manual data preparation, ensuring that textual **data** is consistently formatted, free of noise, and ready for rigorous statistical analysis and high-quality reporting. Mastering this function is an essential step toward becoming a proficient **SAS** data transformer.

Additional Resources

To further expand your knowledge of **SAS** programming and character data manipulation, we recommend exploring related official documentation and tutorials on complementary functions:

SAS String Functions (Official Documentation)
How to Use the TRANSLATE Function in SAS
Removing Blanks from Strings in SAS