How to use PRXCHANGE function in SAS?

Authored by

stats writer

November 19, 2025

The PRXCHANGE function is a cornerstone utility within the SAS Data Step, designed specifically for sophisticated string manipulation. This powerful function allows users to perform advanced pattern matching and subsequent replacement across textual data fields. Unlike simpler functions that rely solely on fixed string values, PRXCHANGE leverages the full capability of regular expressions, granting precise control over the identification and modification of complex textual patterns.

Data cleaning, standardization, and extraction are fundamental tasks in statistical programming and data analysis, and the efficiency of these tasks often hinges on robust text processing tools. PRXCHANGE is indispensable for scenarios requiring flexible replacements, such as normalizing varying address formats, cleaning non-standard punctuation, or ensuring consistency in categorical text variables. Mastery of this function significantly enhances a programmer's ability to handle unstructured or semi-structured data effectively within the SAS environment.

This comprehensive tutorial aims to demystify the PRXCHANGE function. We will meticulously examine its required syntax and parameters, delve into the intricacies of crafting effective regular expressions for pattern definition, and provide detailed, practical examples demonstrating its application in real-world data transformation tasks within SAS.

Understanding the PRXCHANGE Syntax and Parameters

The operational mechanism of the PRXCHANGE function mandates three distinct arguments to execute a successful pattern replacement. Grasping the precise role of each parameter is essential for constructing efficient and error-free string transformations. The standardized syntax structure is straightforward yet powerful, encapsulating the entire logic of search, scope, and target string within a single function call.

The general form for invoking the function is: **PRXCHANGE(regular expression, times, source)**. Each component serves a critical purpose: the first argument defines the transformation logic, the second determines the breadth of the replacement, and the final argument identifies the input string or variable upon which the operation is performed. It is crucial that the first argument, the regular expression, is compiled internally by SAS, typically by being enclosed within quotes and often prefixed with an 's/' structure to denote a substitution operation, though simple matching patterns can also be used depending on the specific application.

The three required parameters are detailed as follows:

regular expression: This parameter is a character string that defines both the pattern being searched for and the replacement string. It typically follows the standard Perl Compatible Regular Expression (PCRE) format, often utilizing delimiters such as slashes (e.g., s/pattern/replacement/flags). This argument dictates exactly what text sequence will be targeted for

modification.

times: This numerical parameter dictates the scope of the replacement process. If a positive integer (N) is provided, only the first N occurrences of the pattern in the source string will be replaced. The most common and useful value, **-1**, instructs SAS to continue searching for and replacing all possible occurrences of the pattern until the very end of the source string is reached, ensuring a global transformation.

source: This final parameter is the character variable or string literal upon which the pattern matching and replacement operation will be executed. The value returned by the **PRXCHANGE** function will be the modified string derived from this source input, following the rules established by the first two parameters.

A Deep Dive into Regular Expressions in SAS

The true power of the PRXCHANGE function stems entirely from its integration with regular expressions (Regex). A regular expression is essentially a sequence of characters that defines a search pattern, allowing for complex searches that go far beyond simple literal string matching. When used within SAS, these expressions enable tasks like conditional replacement based on proximity, capturing groups for reordering content, or defining complex character sets to be targeted.

Within the context of PRXCHANGE, the regex pattern usually takes the form of a substitution command, often structured as `/pattern/replacement/modifiers`. The leading character, typically 's' for substitution, and the delimiters (often '/') are essential components that structure the command for the regex engine. The `pattern` section defines the sequence of characters to be found, while the `replacement` section specifies the text that should replace the found pattern. Understanding basic regex metacharacters, such as `.` (any character), `*` (zero or more occurrences), and `^` / `$` (start/end of string), is critical for effective usage.

Furthermore, the optional `modifiers`, or flags, appended to the end of the regex string significantly alter the behavior of the search. For instance, the 'i' modifier, which we will use in our examples, specifies a case-insensitive search, meaning the pattern will match regardless of capitalization in the source string. Other useful modifiers might include 'g' (global replacement, though this is often handled by the 'times' parameter in SAS) or 'm' (multi-line mode). Utilizing these modifiers ensures that the pattern matching is as flexible and targeted as required for specific data manipulation needs.

Setting Up the Demonstration Dataset

To illustrate the practical application of the PRXCHANGE function, we will utilize a small, representative dataset containing various phrases. This dataset, named `my_data`, is designed to

include variations of the target pattern, including multiple occurrences, differing capitalization, and instances where the pattern is part of a larger word. This variability ensures that our examples thoroughly test the capabilities of the function, especially its case-insensitive matching feature.

The input data is defined using standard SAS data step procedures. We define a single character variable, `phrase`, capable of holding up to 40 characters, which stores the textual information we intend to process. Note the inclusion of phrases such as "That is a cool cool zebra" (multiple occurrences) and "Well now that is COOL" (uppercase occurrence) to demonstrate the necessity of the 'times' parameter set to -1 and the 'i' modifier in our regex pattern, respectively. This setup is crucial for demonstrating global, flexible pattern matching.

The code block below outlines the creation and subsequent viewing of the initial dataset. We use the `DATALINES` statement for easy input of the sample records and then employ `PROC PRINT` to display the structure of `my_data` before any transformations are applied. This baseline view is essential for confirming the input state and verifying the output results generated by PRXCHANGE in the subsequent examples.

```
/*create dataset: my_data*/
data my_data;
input phrase $char40.;
datalines;
This is a cool name
That is a cool cool zebra
Oh hey there
Oh cool it's a cool-looking dog
Well now that is COOL
;
run;

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

Obs	phrase
1	This is a cool name
2	That is a cool cool zebra
3	Oh hey there
4	Oh cool it's a cool-looking dog
5	Well now that is COOL

Example 1: Comprehensive Substitution with New Content

The most frequent use case for PRXCHANGE involves performing a direct substitution--finding a specific string pattern and replacing it with new, desired content. In this example, our objective is to replace every instance of the word "cool" (regardless of capitalization) with the word "fun" across all records in our `my_data` dataset, storing the results in a new variable named `new_phrase` within a new dataset, `new_data`.

The core of this operation lies in the regex pattern: `'s/cool/fun/i'`. The 's' prefix signifies a substitution operation. The first pattern, `cool`, is the search target, while the second pattern, `fun`, is the replacement value. Crucially, the 'i' suffix acts as the case-insensitive modifier, ensuring that variations like "Cool," "cool," and "COOL" are all successfully matched. We set the `times` parameter to **-1**, guaranteeing that if a phrase contains multiple matches (like in the second record), every instance is replaced, not just the first one found.

Upon execution of the data step, we observe the transformative power of this function. For instance, the original phrase "That is a cool cool zebra" is transformed into "That is a fun fun zebra," demonstrating the global replacement enabled by the **-1** parameter. Similarly, the capitalization challenge posed by "Well now that is COOL" is handled effortlessly, resulting in "Well now that is fun," confirming the efficacy of the case-insensitive flag. The resulting code and output confirm the successful, comprehensive string manipulation.

```
/*create new dataset using PRXCHANGE*/
data new_data;
set my_data;
new_phrase = prxchange('s/cool/fun/i', -1, phrase);
run;

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

Obs	phrase	new_phrase
1	This is a cool name	This is a fun name
2	That is a cool cool zebra	That is a fun fun zebra
3	Oh hey there	Oh hey there
4	Oh cool it's a cool-looking dog	Oh fun it's a fun-looking dog
5	Well now that is COOL	Well now that is fun

Example 2: Leveraging PRXCHANGE for String Removal (Blank Replacement)

While the previous example focused on replacing a pattern with a different string, PRXCHANGE is equally powerful for data cleaning tasks, specifically the removal of unwanted characters or substrings. String removal is achieved by simply substituting the targeted pattern with an empty replacement string. In this demonstration, we aim to eliminate all instances of the word "cool" from the `phrase` column, effectively deleting the word wherever it appears.

The mechanism remains largely the same, but the regex structure is modified slightly: `'s/cool//i'`. Notice that the replacement field between the second and third slashes is left empty. This deliberate omission signals to the PRXCHANGE function that any matched pattern should be removed entirely, without leaving behind any characters, including blank spaces. Maintaining the **-1** parameter ensures all occurrences are removed globally, and the 'i' flag maintains the necessary case-insensitive matching capability.

Careful consideration must be given to potential side effects when performing removal. While "That is a cool cool zebra" correctly becomes "That is a zebra" (leaving two adjacent spaces which SAS handles), the phrase "Oh cool it's a cool-looking dog" becomes "Oh it's a -looking dog." This highlights that PRXCHANGE only targets the specified pattern ("cool") and does not automatically clean up surrounding punctuation or spaces unless the regex pattern explicitly includes them (e.g., matching " cool " including the spaces). The resulting dataset, viewed via `PROC PRINT`, clearly shows the elimination of the targeted pattern across all affected records.

```
/*create new dataset using PRXCHANGE for removal*/
data new_data;
set my_data;
new_phrase = prxchange('s/cool//i', -1, phrase);
run;

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

Obs	phrase	new_phrase
1	This is a cool name	This is a name
2	That is a cool cool zebra	That is a zebra
3	Oh hey there	Oh hey there
4	Oh cool it's a cool-looking dog	Oh it's a -looking dog
5	Well now that is COOL	Well now that is

Advanced Considerations and Conclusion

While the examples provided utilize simple literal text matching within the regular expression, the true potential of PRXCHANGE emerges when employing advanced PCRE features. Programmers can integrate lookaheads, lookbehinds, backreferences, and conditional logic within the regex string to handle highly complex data structures, such as extracting specific fields from log files or restructuring names formatted inconsistently (e.g., swapping "Last, First" to "First Last"). Mastering these advanced regex constructs is the key to unlocking the function's maximum utility in large-scale data cleansing operations.

It is important to differentiate PRXCHANGE from other related SAS functions. For instance, the PRXSUBSTR function is used only for pattern extraction, returning the matched substring rather than replacing it. Similarly, PRXPARSE is used to pre-compile a regular expression for repeated use, which can significantly improve performance when the same pattern must be applied across millions of observations. PRXCHANGE, by focusing specifically on pattern replacement, serves a unique and critical role in the SAS programmer's toolkit.

In conclusion, the PRXCHANGE function is an exceptionally robust tool for managing and transforming character data within SAS. By effectively combining the power of regular expressions with precise control over replacement scope (via the `times` parameter), users can execute complex string manipulations that are vital for data quality and preparation. Continuous practice with crafting nuanced regex patterns will further solidify expertise in utilizing this high-performance function for all text-based data requirements.

For continuing your journey into high-level data manipulation in SAS, consider exploring tutorials on related functions and advanced data step techniques:

How to use PRXMATCH for simple pattern detection without replacement.
Techniques for using PRXPARSE to optimize repeated regex calls.
Implementing conditional logic based on string properties using PRX functions.