PhUSE US Connect 2019

Paper CT05

Perl functions in SAS: Perl functions can add pearl in your code

Kamlesh Patel, Jigar Patel, Dilip Patel, Vaishali Patel
Rang Technologies Inc, Piscataway, New Jersey

ABSTRACT

The wide variety of SAS functions give huge power to DATA step in manipulating various types of data. In text processing for data manipulation, there are many new functions available in SAS. Most of the programmers use traditional functions for achieving various data manipulation tasks in SAS. However, there are various string processing functions (like Perl regular expressions) in SAS which can offer a robust solution in place of long syntax with multiple functions. However, Perl regular expressions are least used in clinical programming due to its syntax and the steep learning curve on how to use them in day-to-day programming. We will explain to make the steep learning curve of Perl function into a smooth and easy curve for programmers. We will explain various tips on how to use them in day-to-day programming and make efficient programming.

KEYWORDS

SAS, PRX, Character manipulation, PRXCHANGE, PRXMATCH, PERL, DATA, regular expression

INTRODUCTION

SAS programmers employ different ways to search patterns in text strings and manipulate pieces of text strings. In order to achieve text string related operations efficiently, programmers need to make use of various SAS functions and technics available. In clinical industry, SAS programmers work with various types of character data; for example, a simple one-character variable like sex (M, F, U) to complex free text entered by the investigator (Adverse Event term). Here, we will discuss one of the efficient, but a less widely used family of functions, Perl Regular Expressions (PRX) functions, for handling character string manipulations.

Perl Regular Expressions (PRX) in SAS are based on Perl 5.6.1. Perl is one of the programming languages used in various platforms like UNIX scripting, etc. Perl Regular Expressions (PRX) looks nothing like SAS data step code; hence, it might look unfamiliar at the start to SAS programmers. Therefore, many SAS programmers do not bother to go out of the track to learn special PRX functions for day to day use.

To brief you a little bit about Perl language, Perl is similar to other expression languages like sed, grep, and awk. Perl provides text processing facilities without the arbitrary data length limits of many contemporary Unix command line tools, facilitating manipulation of text files. Perl 5 gained widespread popularity in the late 1990s as a Common Gateway Interface (CGI) scripting language, in part due to its then unsurpassed regular expression and string parsing abilities. In addition to CGI, Perl 5 is used for system administration, network programming, finance, bioinformatics, and other applications such as for GUIs.

The SAS has empowered itself by adding Perl functions and routines in character data processing. The power of Perl's regular expression is available in SAS since the SAS 9.0 release. This addition has given additional flexibility to SAS. In the past, SAS used procedures like INDEX, INDEXC, LENGTH, SUBSTR, SCAN, etc. for achieving this task. Now with the addition of PRX function, the task becomes simpler and more powerful. However, in clinical programming, PRX functions usage has been limited.

Power of PRX functions can be employed to –

- String search: Search for a specific string in character value
- Extract out substring: To take out a specific substring
- Search + Replace: Replace specific string in place of another string
- Parse string: Parse large amounts of text like a website or any other text data

In this article, we will look at the fundamentals of PRX functions and will try to provide a clear understanding of the clinical SAS programmer. The goal of this paper is to start using PRX function to make your code beautiful and add a pearl in your code.

FUNDAMENTALS AND BASICS OF PRX

1. USING CHARACTER STRING IN SLASHES

PERL language use slash for the string. The same applies in SAS PRX functions. Hence, any string constant should be written as –

/text string/

If text string, Hospital, should be written as –

/Hospital/

In SAS, character value should be quoted, hence, it above string we should use as below when we reference.

'/Hospital/'

2. USING TEXT STRINGS IN PRX FUNCTIONS

Two main ways –

A. Regular-Expression-ID (generated by PRXPARSE function):
   a. It is a text pattern identifier in numeric number form
   b. It is generated by passing a specific text string into PRXPARSE functions.
   c. SAS assigned each new identifier for every PRXPARSE functions encountered in same data step in increment from 1 to n. This also applies when same the step is iterated multiple times due to multiple records.
   d. Due to this reason, it is good programming practice to execute one string constant one time as shown in the example.
   e. The character string which we are passing (regular expressions) can be used with various metacharacters to customize the search.

Please see sample code 2a and 2b in appendix 1.

B. Perl-Regular-Expression in PRX functions:
   a. It can be a character constant (e.g. '/Hospital/'), variable, or any DATA step expression which returns the value in the form of a Perl regular expression.
   b. There are many rules of making a regular expression with the help of metacharacters and options. Those are discussed below.

Please see sample code 2c in appendix 1.

3. MAKING PERL REGULAR EXPRESSIONS

   a. This is the power of PRX function!!!
   b. Can be customized and written to search VERY complex text strings in a character variable. Though we have covered basic level of PERL expressions in this article, there are so many things can be learned using references and support.sas.com.
   c. A Wide variety of metacharacters can be used to capture the desired text string. Those metacharacters are shown in below table.
   d. Tip: Capital character represents the negation of small letter characters.
   e. Tip: [ ] brackets can be used to group characters.

PRX Expression note	Metacharacter	Syntax (quotation needs to apply when we put in function)	Example of strings	Explanation
With slash		/Nausea/	Nausea	Basic expression
Alternation (OR) using Pipe (|)	|	/Nausea|Vomiting|Gastric Problem/	Nausea, nausea, NAUSEA	Similar to OR operator. It is similar to -Nausea OR Vomiting OR Gastric Problem.
With grouping for a specific character	[]	/[Nn]ausea/	Nausea, nausea	It will match for the character with 1st Character can be capital or small "N"/"n" word nausea
String with ANY ALPHA-NUMERIC character before targeted string	\w	/\w[Nn]ausea/	1Nausea, aNausea, Anausea	\w stands for any alpha-numeric character \w will match a word character (alphanumeric plus "_")
String with ANY NON-ALPHA-NUMERIC character before targeted string	\W	/\W[Nn]ausea/	~Nausea, @Nausea, #nausea	\W stands for any NON alpha-numeric character \W will match a Non-Word character
String with ANY SPACE character before targeted string	\s	\s[Nn]ausea	Nausea …	\s is for the string with a preceding space. This expression will look for a string with space before the targeted string. \s will match a White space character
String with ANY NON-SPACE character before targeted string	\S	\S[Nn]ausea	~Nausea, ANausea, 1nausea	This expression will look for a string with NO space before the targeted string. \S will match a non-whitespace character
String with ANY Digital character before targeted string	\d	/\d[Nn]ausea/	1Nausea, 2nausea	This expression will look for the string with digit before the targeted string. Will match for the string with the preceding digit. \d will match a digit character
String with ANY NON-Digital character before targeted string	\D	/\D[Nn]ausea	Nausea …	This expression will look for the string with NON digit before the targeted string. \D will match a non-digit character
Search CASE-INSENSITIVE	/i	/Nausea/i	Nausea, nausea, NAUSEA	Case Insensitive search This will make case insensitive for the targeted string.
Range of character	[a-z]	/[a-c]ausea/	aausea, bausea, causea	Take character from “a to c” range for 1st character [a-z] will match a character in the range
Start of the line	^	/^Nausea/	Nausea ….	Only Nausea which is 1st in line ^ will match the beginning of the line
End of the line	$	/Nausea$/	… Nausea	It will capture only Nausea which is at the end of the line $ will match the end of the line
Any character	*	/Nausea*/	Nausea /vomiting, Nausea and , Nausea?	Any character after Nausea * can represent no character to any character.

PRX FUNCTIONS FOR BEGINNERS

Now, we have learned some basics of PRX function to start using some other function in our day to day programming. There are various functions in PRX family; however, we will focus on a few functions which are more useful for clinical programmers.

1. PRXMATCH

USE: Search for a specific pattern and return with the location of the pattern in the string

NOTE: It is similar to INDEX function, but PRXMATCH has more flexibility.

SYNTAX:

PRXMATCH (*targeted-specific-string*, source)

Targeted-specific-string - > 1. Regular expression ID- generated from PRXPARSE function.

2. Regular expression- Character constant in form of regular expression, variable.

Source -> 1. Character string or character variable or expression that return character string

In the example code, we have shown various usage of PRXMATCH function step by step from simple to complex and we have explained it step by step.

1. One simple string – This is like INDEX functions. In this usage, there is no special advantage over INDEX functions.
2. Two or more string constant search – Using alternation (| - pipe) in a regular expression, we can search various strings in PRXMATCH compared to writing multiple times INDEX functions in DATA step.
3. Using Grouping in PRXMATCH – If we want to search for “Nausea” and “nausea”, you can do grouping using [] – bracket for 1st character like “/[Nn]ausea/”. Similarly, you can do it for any character.
4. 5. 6. 7. 8. 9. For any specific character (like alpha-numeric, space, digit) preceded or NOT preceded by a string can be controlled during PRXMATCH search string.
    a. \w - > Represents any Alpha-numeric value (e.g. A-z, 0-9)
    b. \W- > Represents NON-any Alpha-numeric value (e.g. ~, !, #, space, etc.)
    c. \s - > Represents any blank space value (e.g. blank, tab)
    d. \S- > Represents NON-any blank space value (e.g. alpha-numeric, special characters, etc.)
    e. \d - > Represents any digit value (e.g. 0-9)
    f. \D- > Represents NON-digit space value (e.g. alphabetic, special character, etc.)

    TIP: CAPITAL word (\W) makes negation (NON) for available characters represented by small letters (\w) character in the syntax.

10. Modifiers – Using modifiers in PRXMATCH can make efficient programming.
    a. /i – Case-insensitive search. It is very powerful for doing a case insensitive search for a string like nausea or Nausea or NAUSEA or nAuSea, all can be searched by adding modifier /i.

Please see sample code 3a to 3f in appendix 1.

2. PRXCHANGE

USE: Search for a specific pattern and perform replacement with a new string

NOTE: There are similar functions for replacement and matching pattern. However, it gives huge flexibility with flexible string search and replacement in the same function.

SYNTAX:

PRXCHANGE (*targeted-specific-string*, times, source)

Targeted-specific-string - > 1. *Regular expression ID*- generated from PRXPARSE function.

2. Regular expression- Character constant in form of regular expression, variable.

The basic syntax is simple -

SearchString/ReplaceString

With PERL (PRX) expression), there are slashes and quotes in SAS.

"/SearchString/ReplaceString/"

Though it is the default, "s" is for substitution.

"s/SearchString/ReplaceString/"

Times -> 1. It is a numeric value. It tells SAS that the number of times the operation of search/replacement to be performed.

(NOTE: if the value is -1, search and replacement will continue till the end of source string)

Source -> 1. Character string or character variable or expression that return character string

In the example code, we have shown various usage of PRXCHANGE function step by step from simple to complex and we have explained it step by step.

1. Simple string – with the simple example of passing string, we can get replacement of the targeted string.

2. Insert string – We can use some addition string constant (like 'and Vomiting') after the searched string. Please note that search string should be in the bracket for compiling buffer and put $1 to get the same text.

Please see the sample code 4a to 4b in appendix 1.

In addition to above-discussed functions, there are many other functions and call routines under the umbrella of PRX. To list some, PRXPOSN, PRXPARSE functions and Call PRXSUBSTR, Call PRXDEBUG, Call PRXNEXT etc. You can refer more from references about other functions.

SUMMARY

In a nutshell, PRX functions and call routines has been a valuable addition in SAS for character string manipulation. For clinical SAS programmers, it can be very helpful in programming in day to day if SAS programmers learn PRX fundamentals and functions. Even with the beginners' level of understanding, programmers can save a lot of time, make programming robust and add beautiful with PERL.

REFERENCES

1. Pattern Matching Using Perl Regular Expressions (PRX); SAS(R) 9.3 Functions and CALL Routines: Reference; http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#n13as9vjfj7aokn1syvfyrpaj7z5.htm
2. PRXMATCH Function; SAS(R) 9.3 Functions and CALL Routines: Reference; http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#n0bj9p4401w3n9n1gmv6tfshit9m.htm
3. PRXCHANGE Function; SAS(R) 9.3 Functions and CALL Routines: Reference; http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#n0r8h2fa8djqf1n1cnenrvm573br.htm
4. Cassell, David; The Basics of the PRX Functions, 2007, SAS Global Forum 2007.

ACKNOWLEDGMENT

I would like to thank Dhaivat for his valuable input for content.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kamlesh Patel
Rang Technologies Inc, NJ
Kamlesh.Patel1@rangtech.com

Jigar Patel
Rang Technologies Inc, NJ

jpatel@rangtech.com

Dilip Patel
Rang Technologies Inc, NJ
dilip@rangtech.com

Vaishali Patel
Rang Technologies Inc, NJ
vaishali.sas01@gmail.com

APPENDIX 1

```
*-----------------------------------------------------------------
* PhUSE US Connect 2019
* Paper CT05 – SAS Codes
* Perl functions in SAS: Perl functions can add pearl in your code
*-----------------------------------------------------------------*;
*Code 1;
*Sample Dataset used for examples;
Data AE ;
Infile cards;
Input  AEDECOD $ 1-40 ;
Cards;
Nausea
Occasional Soles of Feet Burning
Severe nausea
Heartburn
Mouth sore
Diarrhea
Fatigue
Fever with nausea
Weight loss
anausea
1nausea
      nausea
;
**********************************************************
*PRXPARSE;
/*Code 2a: Understanding how PRXPARSE gives increment of ID in same data step */
Data New;
PRX = prxparse ('/Hospital/');
Put "PRXPARSE: 1st Occurrence of PRXPARSE function in DATA step" PRX= ;
PRX = prxparse ('/Hospital/');
Put "PRXPARSE: 2nd Occurrence of PRXPARSE function in DATA step" PRX=;
PRX = prxparse ('/Hospital/');
Put "PRXPARSE: 3rd Occurrence of PRXPARSE function in DATA step" PRX=;
Run;
/*Code 2b: PRXPARSE - Common usage in data step */
Data New;
set AE;
*Executing only 1st time when many observations;
retain PRX;
if _N_=1 then PRX = prxparse ('/Nausea/');
*Once string is compiled and returned the value, you
can use same for other observations by retaining;
A2=PRXMATCH (PRX, aedecod);
Run;

/*Code 2c: Passing simple string into PRX functions. */
```

```
Data AE1;
Set AE;
*Simply passing regular expressions into PRX functions;
A2=PRXMATCH ("/Nausea/", aedecod);
Run;

************************************************************

/*Code 3a: PRXMATCH*/
Data AE1;
Set AE;
*Using PRXMATCH;
*****************************************
*1. Simple one string constant as argument.
 --> Commonly, Programmer use INDEX function.
*****************************************;
A1=Index ('Nausea', aedecod);
A2=PRXMATCH ("/Nausea/", aedecod);

IF Index ('Nausea', aedecod);
IF PRXMATCH ("/Nausea/", aedecod);
run;
/*Code 3b: PRXMATCH*/
Data AE1;
Set AE;
*****************************************
*2. Simple two or more string constants in one.
 --> Commonly, Programmers use INDEX function multiple times.
*****************************************;
IF Index ('Nausea', aedecod) OR Index ('Vomiting', aedecod) or Index ('Emesis ',
aedecod);
IF PRXMATCH ("/Nausea|Vomiting|Emesis/", aedecod); *Using ALTERNATION;
run;

/*Code 3c: PRXMATCH*/
Data AE1;
Set AE;
*****************************************
*3. Simple one string constant as argument.
With grouping, programmers can add strings search with more types of words.
 --> Commonly, Programmers use INDEX function multiple times.
*****************************************;
If PRXMATCH ("/[Nn]ausea/", aedecod);
D2=PRXMATCH ("/[Nn]ausea/", aedecod);
run;
/*Code 3d: PRXMATCH*/
Data AE1;
Set AE;
*****************************************
*4. \w :: String with ANY ALPHA-NUMERIC character around targeted string.
E.g. Get strings Nausea or nausea which is preceded by one ALPHA-NUMERIC character.
*****************************************;
*If PRXMATCH ("/\w[Nn]ausea/", aedecod) > 0;
E2=PRXMATCH ("/\w[Nn]ausea/", aedecod) ;

*****************************************
NOTE: For understanding, Capitalization is NOT of small case. i.e. if w is any
alphanumeric
character, then W is NO alphanumeric character in PRX.
*5. \W :: String with NO ANY ALPHA-NUMERIC character around targeted string.
E.g. Get strings Nausea or nausea which is preceded by one NON-ALPHA-NUMERIC character
(like space).
*****************************************;
*If PRXMATCH ("/\W[Nn]ausea/", aedecod) > 0;
```

```
F2=PRXMATCH ("/\W[Nn]ausea/", aedecod) ;
Run;

/*Code 3e: PRXMATCH*/
Data AE1;
Set AE;
*****************************************
*6. \s :: String with ANY SPACE character around targeted string.
Space characters are - regular whitespace, tab etc.
E.g. Get strings Nausea or nausea which is preceded by one SPACE character.
*****************************************;
*If PRXMATCH ("/\s[Nn]ausea/", aedecod) > 0;
G2=PRXMATCH ("/\s[Nn]ausea/", aedecod) ;
*****************************************
NOTE: For understanding, Capitalization is NOT of small case. i.e. if s is any SPACE
character, then S is NO SPACE character in PRX.
*7. \S :: String with NO ANY SPACE character around targeted string.
E.g. Get strings Nausea or nausea which is preceded by one NON-SPACE character.
*****************************************;
*If PRXMATCH ("/\S[Nn]ausea/", aedecod) > 0;
H2=PRXMATCH ("/\S[Nn]ausea/", aedecod) ;
*****************************************
Same way for DIGIT -
*8. \d :: For digit character like 0-1.
*9. \D :: For NON-digit character like 0-1.
*****************************************;
I2 =PRXMATCH ("/\d[Nn]ausea/", aedecod) ;
J2 =PRXMATCH ("/\D[Nn]ausea/", aedecod) ;
Run;
/*Code 3f: PRXMATCH*/
Data AE1;
Set AE;
*****************************************
*10. \i :: Can I make search CASE-INSENSITIVE.
E.g. Get strings Nausea or nausea in any case.
The programmer can do grouping. However, the simple way is to put i.
*****************************************;
K2=PRXMATCH ("s/Nausea/i", aedecod) ;
*If PRXMATCH ("/Nausea/i", aedecod) > 0;
Run;

************************************************************
/*Code 4a: PRXCHANGE*/
Data AE2;
Set AE;
*****************************************
*1. Simple Usage of PRXCHANGE with character constant.
*****************************************;
K1=PRXCHANGE ("s/Nausea/NAUSEA/", -1 , aedecod) ;
*Making Search CASE Insensitive by adding i;
K2=PRXCHANGE ("s/Nausea/NAUSEA/i", -1, aedecod) ;
*NOTE: when you put -1 in times argument, it will go till end of string;
Run;

/*Code 4b: PRXCHANGE*/
Data AE2;
Set AE;
*****************************************
*2. Inserting some constant i.e. replacing the existing value with value + character
constant
*****************************************;
K1=PRXCHANGE ("s/Nausea/Nausea and Vomiting/i", -1 , aedecod) ;
K2=PRXCHANGE ("s/(Nausea)/$1 and Vomiting/i", -1 , aedecod) ;
Run;
```