Paper 4005-2019

Quick Tips and Tricks: Perl Regular Expressions in SAS®

Pratap S. Kunwar, Jinson Erinjeri, Emmes Corporation.

ABSTRACT

Programming with text strings or patterns in SAS® can be complicated without the knowledge of Perl regular expressions. Just knowing the basics of regular expressions (PRX functions) will sharpen anyone's programming skills. Having attended a few SAS conferences lately, we have noticed that there are few presentations on this topic and many programmers tend to avoid learning and applying the regular expressions. Also, many of them are not aware of the capabilities of these functions in SAS. In this presentation, we present quick tips on these expressions with various applications which will enable anyone learn this topic with ease.

INTRODUCTION

SAS has numerous character (string) functions which are very useful in manipulating character fields. Every SAS programmer is generally familiar with basic character functions such as SUBSTR, SCAN, STRIP, INDEX, UPCASE, LOWCASE, CAT, ANY, NOT, COMPARE, COMPBL, COMPRESS, FIND, TRANSLATE, TRANWRD etc. Though these common functions are very handy for simple string manipulations, they are not built for complex pattern matching and search-and-replace operations.

Regular expressions (RegEx) are both flexible and powerful and are widely used in popular programming languages such as Perl, Python, JavaScript, PHP, .NET and many more for pattern matching and translating character strings. Regular expressions skills can be easily ported to other languages like SQL., However, unlike SQL, RegEx itself is not a programming language, but simply defines a search pattern that describes text.

Learning regular expressions starts with understanding of character classes and metacharacters. Becoming skillful on this topic is not hard but RegEx can be intimidating at first as it is based on a system of symbols (metacharacters) to describe a text pattern to read text, and this can be an obvious reason for anyone to put it off.

<table>
<tr><td rowspan="2">Regular Expression
(Matching Text)
/search-string/source-string/</td><td colspan="2">prxmatch('/^[BDGKMNSTZ]{1}(OO)[0-9]{3}-\d{2}\s*$/', id);
→B00003-39</td></tr>
<tr><td>a) Metacharacter^ = Start
b) One of BDGKMNSTZ chars
c) {1} = quantifier 1 char
d) Followed by char group (OO)
e) [0-9]{3} char class 3 times</td><td>f) Followed by –
g) Followed by \d{2} 2 digits
h) $ End with #7 above or space \s 1 or more times *</td></tr>
<tr><td rowspan="2">Regular Expression
(Find and Replace)
s/regular-expression/replacement-string/</td><td colspan="2">prxchange('s/\d//',-1, 0001000254698ABCD)
→ABCD</td></tr>
<tr><td>a) s/ Substitution operator
b) | Alternation matching</td><td>c) \ escape character
d) -1 1 time -</td></tr>
</table>

CHARACTERS AND METACHARACTERS

Regular expressions are built up from metacharacters and their power comes from the use of these metacharacters, which allow the matching of types of text and sequences through systemic searches. There are different sets of characters and metacharacters used in Perl regular expressions as listed below.

Literal characters	This works the same way as normal find and replace without the use of symbols; more like SAS TRANWRD function. This is simple but inflexible.	
Character classes (sets and ranges)	[abc]	a, b, or c
	[^abc]	any but not a, b, or c
	[a-zA-Z]	character between a to z
	[0-9]	any digits
Predefined Character classes (Shorthand Character Sets)	.	any character
	\w, \d, \s	Word [0-9 a-z _), digit, whitespace
	\W, \D, \S	not word, digit, whitespace
Character sets (groups and look around)	This allows for a series of characters within a range defined by starting and ending characters.	
	() (abc)	capture everything enclosed
	(?:abc)	non-capturing group
	(?=abc)	positive lookahead
	(?!abc)	negative lookahead
	\1, \2	back reference to group #1, #2
Positional metacharacters (anchors/boundaries)	^abc$	start / end of the string
	\b\B	word, not-word boundary
Quantifiers metacharacters (Wildcards/Repetitions/Multipliers)	a* a+ a?	0 or more, 1 or more, 0 or 1
	a{5} a{2,}	exactly five, two or more
	a{1,3}	between one & three
	a+? a{2,}?	match as few as possible
	ab|cd	alternative matching ab or cd
	(July|Jul)	July or Jul equivalent to July?
Escaping Metacharacters (Metacharacters -> Literal Meaning)	When a metacharacter(s) itself is in the text, then the metacharacter needs to "escape" from its metacharacter meanings. This is done by putting a backslash in front of it for its literal meaning. \. \? * \+ \[\] | \(\) \{ \} \$ \^ \\	
Note: The '^' has a different meaning in character class [^abc] vs (^abc). Similarly, - has different meaning within [-a-z]. A character can have different meanings depending on where it is being used. For example, escaped metacharacters are not metacharacters.		

SAS PRX Functions

Find using PRXMATCH:	Find and Replace using PRXCHANGE:
PRXMATCH function can match the location of the search strings in the source strings. It has two parameters: the first is regular expression id (search string) and second parameter is character string to be searched (source string). Syntax: PRXMATCH(/regular-expression/, source) Ex. prxmatch('/world/', 'Hello world!'); The above example uses the PRXMATCH function to find the position (=7) of the search-string (world) in the source-string (Hello World)	PRXCHANGE is not only used to find strings but also to replace it using specified rules. PRXCHANGE expressions allow the programmer to choose part of text to replace and rest to keep. SAS has a simple function named TRANWRD which is very handy for a search-and-replace string, but TRANWRD works only with literal characters or words. Syntax: PRXCHANGE(s/regular-expression/replacement-string/, source) Ex. prxchange('s/world/planet/', 1, 'Hello world!'); The above example uses the PRXCHANGE function to replace 'world' in 'Hello world' with 'planet,' resulting in 'Hello planet'

APPLICATION 1: SIMPLE SEARCH

```
%let ptlist1=%str(HIVE?|HEPATITIS|TREPONEMA PALLIDUM|HTLV|CYCLOSPORA);
data a1a;
   text="HIV Positive";
   grade='GR3'; output;
run;
data a1;
   set a1a;
   if prxmatch("/(&ptlist1)/", text) then flag1='X'; /*check ptlist against text*/
   if prxmatch("/(GR3|SEVERE)/", grade) then flag2='X';/*GR3 or SEVERE*/
run;
```

text	grade	flag1	flag2
HIV Positive	GR3	X	X

APPLICATION 2: MULTIPLE SEARCHES

```
data a2;
   set sashelp.class;
   if prxmatch ("/^A/", name) then flag1='X'; /*start with A*/
   if prxmatch ("/d$/", strip(name)) then flag2='X'; /*end with d*/
   if prxmatch ("/d\s*$/", name) then flag3='X'; /*end with d or space*/
   if prxmatch ("/^J\w+y\s*$/i", name) then flag4='X'; /*start with J and end with y*/
   if prxmatch("/\w{2}(e|s)\s*$/i", name) then flag5='X'; /*end with e or s*/
   if prxmatch("/^\w{2,4}(e|s)\s*$/i", name) then flag6='X'; /*flag5 but 2 to 4 char)
   if prxmatch ("/\Janet?/", name) then flag7='X'; /*ending t is optional*/
   if prxmatch('/(\S)\1/', name ) then flag8='X'; /*2 continious white space*/
   if prxmatch("/[^Janet]/i", strip(name)) then flag9='X';/*Except J|a|n|e|t*/
   if prxmatch("/^[Janet]/i", strip(name)) then flag10='X'; /*start with J|a|n|e|t*/
run;
```

Name	flag1	flag2	flag3	flag4	flag5	flag6	flag7	flag8	flag9	flag10
Alfred	X	X	X						X	X
Alice	X				X	X			X	X
Barbara									X	
Carol									X	
Henry									X	
James					X	X			X	X
Jane					X	X	X			X
Janet							X			X
Jeffrey				X				X	X	X
John									X	X
Joyce					X	X			X	X
Judy				X					X	X
Louise					X				X	
Mary									X	
Philip									X	
Robert									X	
Ronald		X	X						X	
Thomas					X				X	X
William								X	X	

APPLICATION 3: IN PROC SQL

```
/*search start with H|M|J and with y or space*/
proc sql;
  select *
    from sashelp.class
      where prxmatch('/^(h|m|j).*y\s*$/i', name);
quit;
```

Name	Sex	Age	Height	Weight
Henry	M	14	63.5	102.5
Jeffrey	M	13	62.5	84
Judy	F	14	64.3	90
Mary	F	15	66.5	112

APPLICATION 4: BOUNDARY

```
/*boundary before and after 4dr*/
data a4;
  set sashelp.cars (obs=4 keep=make model type);
  if prxmatch("/\b4dr\b/i", model) then flag3='X';
run;
```

Make	Model	Type	flag1
Acura	MDX	SUV	
Acura	RSX Type S 2dr	Sedan	
Acura	TSX 4dr	Sedan	X
Acura	TL 4dr	Sedan	X

APPLICATION 5: WITH ALTERNATION (|)

```
/*check of any of ids exist in text*/
%let idtext=%str(Z07IW001|Z07IW002|Z07IW003|Z07IW004|Z07IW005|Z07IW094);
%let text=%str(Continue from If Other, specify: was not obtained prior
tension notice and contacted ZZZZ. Potentially affected are: Z07IW098,
Z07IW094);

data a5;
  if prxmatch("/(&idtext)/", "&text") then flag=1;
run;
```

flag
1

APPLICATION 6: ID PATTERN

```
data a6a;
  input id $1-50;
  datalines;
MOCK-EXTRACT
G00011-39R
S00081-34
S00081-IS
T-11642-39
S00171 -42
G001054A
ZOO1054A
ZO1054A
B00003-39
;
run;
```

```
data a6;
    set a6a;
    if prxmatch( '/^( |KHANISILYE||MOCK-EXTRACT|MOCK23APR12|0|Z00722-6A-61)\s*$/', id) then flag1=1;
    else if prxmatch('/^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}[R]{1})\s*$/', id) then flag2=1; *G00011-39R -delete*;
    else if prxmatch('/^[TN]{1}[0]{2}[0-9]{3}-(\d{1}|C|M)(\d{1}|A)\s*$/', id) then flag3=1; *T00011-42*;
    else if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5}-(\d{1}|C|M)(\d{1}|A)\s*$/', id) then flag4=1;   *S00081-34 *;
    else if prxmatch('/^[BDGKMNSTZ]{1}[0-9]{5}-((\d{2}[\.]{1})|VTM|IS)\s*$/', id) then flag5=1; *S00081-IS K00081-VT*;
    else if prxmatch( '/^[BDGKMNSTZ]{1}( |-)[0-9]{5}-\d{2}\s*$/', id) then flag6=1; *T-11642-39*;
    else if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5} -\d{2}\s*$/', id) then flag7=1; *S00171 -42*;
    else if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5}(\d{1}|C)(\d{1}|A)\s*$/', id) then flag8=1; *G001054A G0000538**;
    else if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5}-\d{3}\s*$/', id) then flag9=1; *G001054A G0000538**;
    else if prxmatch( '/^[BDGKMNSTZ]{1}(O)[0-9]{4}-\d{2}\s*$/', id) then flag10=1; *ZO*;
    else if prxmatch( '/^[BDGKMNSTZ]{1}(OO)[0-9]{3}-\d{2}\s*$/', id) then flag11=1; *ZOO*;
    else if prxmatch( '/(PLEA     SE|H2O|NP\/OP IN|SWAB||NTULI|SAMPLE|ACADEMIC)/', id) then flag12=1;
run;
```

id	flag1	flag2	flag3	flag4	flag5	flag6	flag7	flag8	flag9	flag10	flag11	flag1
MOCK-EXTRACT	1	.	.	.	.	.	.	.	.	.	.	
G00011-39R	.	1	.	.	.	.	.	.	.	.	.	
S00081-34	.	.	.	1	.	.	.	.	.	.	.	
S00081-IS	.	.	.	.	1	.	.	.	.	.	.	
T-11642-39	.	.	.	.	.	1	.	.	.	.	.	
S00171 -42	.	.	.	.	.	.	1	.	.	.	.	
G001054A	.	.	.	.	.	.	.	1	.	.	.	
ZOO1054A	.	.	.	.	.	.	.	.	.	.	.	
ZO1054A	.	.	.	.	.	.	.	.	.	.	.	
B00003-39	.	.	.	1	.	.	.	.	.	.	.	

APPLICATION 7: SIMPLE REPLACE

```
data a7;
   set sashelp.class (obs=3);
   name2=prxchange("s/(Alfred)/Alex/i",-1,name); /*replace Akfred with Alex*/
run;
```

Name	Sex	Age	Height	Weight	name2
Alfred	M	14	69.0	112.5	Alex
Alice	F	13	56.5	84.0	Alice
Barbara	F	13	65.3	98.0	Barbara

APPLICATION 8: REMOVE NUMBERS

```
data a8;
   text="0001000254698ABCD";
   alpha=prxchange('s/\d//',-1, text); /*remove digits*/
   num=prxchange('s/[a-z]//i',-1, text); /*remove alphabets*/
run;
```

text	alpha	num
0001000254698ABCD	ABCD	0001000254698

APPLICATION 9: REMOVE DIGITS OR ALPHABETS

```
data a9;
   text="0001000254698ABCD";
   alpha=prxchange('s/\d//',-1, text); /*remove digits*/
   num=prxchange('s/[a-z]//i',-1, text); /*remove alphabets*/
run;
```

text	alpha	num
0001000254698ABCD	ABCD	0001000254698

APPLICATION 10: REMOVE LEADING ZEROS

```
data a10a;
   x='000asd1234'; output;
   x='123AA'; output;
   x='0009876A0'; output;
run;

data a10;
   set a10a;
   L0 =prxchange('s/^0+//',-1,x);
run;
```

x	L0
000asd1234	asd1234
123AA	123AA
0009876A0	9876A0

APPLICATION 11: REMOVE DUPLICATES

```
data a11;
   clist = "ALEX ALEX Aaa B C D E F E G H B I Aaa Bb D J K TIM TIM";
      do i=1 to countw(clist);
         Nondups=prxchange('s/(\b\w+?\b)(.*?)(?=\b\1{1,}\b)(.?)/$2$3/i',-
1,compbl(clist));
      end;
run;
```

clist	i	Nondups
ALEX ALEX Aaa B C D E F E G H B I Aaa Bb D J K TIM TIM	21	ALEX B C D E F E G H B I Aaa Bb D J K TIM

CONCLUSION

In this paper, we stated that learning regular expressions requires understanding of various types of metacharacters and presented some simple examples, ranging from finding simple literals to finding complex string patterns and replacing them as well. The examples presented in this paper show regular expressions are powerful and convenient, which makes it worth learning.

Learning RegEx requires mastering the use of metacharacters, which requires a trial and error approach. Further fine-tuning can be performed by practicing the use of a free text editor like Atom in an interactive mode by placing source-string in the text buffer and search-string in the find buffer respectively.

ACKNOWLEDGMENTS

The authors would like to thank their colleagues at the Emmes Corporation for their feedback and encouragement. The author would also like to thank Ryan Bratt for reviewing this paper.

RECOMMENDED READING

- *https://www.lexjansen.com/*
- https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

CONTACT INFORMATION

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

Pratap S. Kunwar
The Emmes Corporation
401 N Washington St.
E-mail: pkunwar@emmes.com

Jinson Erinjeri
The Emmes Corporation
401 N Washington St.
E-mail: jerinjeri@emmes.com