

Some Simple Perl Regular Expressions Examples in SAS® 9

Selvaratnam Sridharma, Census Bureau, Washington, D.C.

ABSTRACT

Perl regular expressions allow you to locate patterns in a string. Many of the string processing tasks in SAS can be done using Perl regular expressions. These tasks can also be performed using traditional string functions in SAS. But, Perl regular expressions can sometimes provide a much more compact solution to a much more complicated string manipulation task. Here we will give some examples to show the usefulness of the Perl regular expressions in SAS programs.

INTRODUCTION

Perl regular expressions helps you to search for and extract multiple matching patterns from a character string in one step, as well as to make several substitutions in a string in one step. We will illustrate this using the following functions, which are some of the functions in SAS regular expressions.

PRXPARSE - Compiles a Perl regular expression that can be used for matching of a character value

PRXMATCH - Searches for a pattern match and returns the position at which the pattern is found

PRXCHANGE - Performs a pattern matching replacement

We give simple examples for the following:

- Search for a pattern of characters within a string.
- Search and replace a string with certain pattern with other text.

EXAMPLES

These examples use some basic tools in Perl regular expressions.

EXAMPLE 1

Let's start with a very simple example. In the following code **PRXPARSE** function uses a Perl regular expression syntax **"/cat/"**. **PRMATCH** will match the regular expression with a string. **POS** will give the location of the letter "c" in "cat" in each of the strings.

```
DATA _NULL_;
  IF _N_ =1 THEN PATTERN = PRXPARSE ( "/cat/" );
  RETAIN PATTERN;
  INPUT STRING $30. ;
  POS = PRXMATCH ( PATTERN, STRING );

  IF POS > 0 THEN PUT STRING;
DATALINES;
CAT is an animal.
There is a cat in the house.
It is beautiful      cat
cat is lovely.
It is a cat
This is a cat.
This is a CAT.
It is a CAT
This is a caterpillar.
It is mcat.
;
```

The output will be:

```
There is a cat in the house.
It is beautiful           cat
cat is lovely.
It is a cat
This is a cat.
This is a caterpillar.
It is mcat.
```

EXAMPLE 2

In the code in Example1, if we use **PRXPARSE ("/*cat* /")** instead of **PRXPARSE ("/*cat*"")**, the output will be:

```
There is a cat in the house.
It is a cat
```

Here expression "*/ cat /*" has a blank space in front and after "cat".

EXAMPLE 3

In the code in Example1, if we use **PRXPARSE ("^*cat* /")** instead of **PRXPARSE ("/*cat*"")**, the output will be:

```
cat is lovely.
```

Here the metacharacter '^' matches the beginning of a string.

EXAMPLE 4

In the code in Example1, if we use **PRXPARSE ("^*cat* | *cat* /")** instead of **PRXPARSE ("/*cat*"")**, the output will be:

```
There is a cat in the house.
It is a cat
cat is lovely.
```

In the above regular expression, '|' specifies an 'or' condition.

EXAMPLE 5

In the code in Example1, if we use **PRXPARSE ("^*cat* | *cat* /")** instead of **PRXPARSE ("/*cat*"")**, the output will be:

```
There is a cat in the house.
It is a cat
cat is lovely.
```

In the above regular expression, '|' specifies an 'or' condition.

EXAMPLE 6

If we also want to select the strings with '**cat.**' or '**CAT.**', we could use **PRXPARSE ("^*cat* | *cat* | *cat*\./i")** instead of **PRXPARSE ("^*cat* | *cat* /i")** as in Example 5 . Here we need to use the escape character '\' to mask the special character '.', which has a special meaning in regular expressions.

The output will be:

CAT is an animal.
 There is a cat in the house.
 cat is lovely.
 It is a cat
 This is a cat.
 This is a CAT.
 It is a CAT

EXAMPLE 7

We are still missing the sentence "It is beautiful cat" in the output. Since there is no space after the word 'cat' in this string, it was not selected. We can use PRXPARSE ("/^cat | cat | cat\.\| cat\\$/"i) instead of PRXPARSE ("/^cat | cat | cat\.\|") in Example 6 to select sentences of this type. The \$ operator indicates "end of string".

The output will be

CAT is an animal.
 There is a cat in the house.
 It is beautiful cat
 cat is lovely.
 It is a cat
 This is a cat.
 This is a CAT.
 It is a CAT

EXAMPLE 8

In Example 1, if we want to substitute 'rat' for 'cat', we can use the following code. PRXCHANGE function substitutes one text for another.

```
DATA _NULL_ ;
  IF _N_ =1 THEN PATTERN = PRXPARSE ( "s/cat/rat/" );
  RETAIN PATTERN;

INPUT STRING $30. ;
CALL PRXCHANGE(PATTERN, 5, STRING);

DATALINES;
CAT is an animal.
There is a cat in the house.
It is beautiful                   cat
cat is lovely.
It is a cat
This is a cat.
This is a CAT.
It is a CAT
This is a caterpillar.
It is mcat.
;
```

The output will be:

CAT is an animal.
 There is a rat in the house.
 It is beautiful rat
 rat is lovely.
 It is a rat
 This is a rat.
 This is a CAT.
 It is a CAT
 This is a raterpillar.
 It is mrat.

EXAMPLE 9

Let's look at another little more complicated example. In this example the string field could have 'PO Box' , 'P.O.Box' , 'Box' , 'P O Box' , etc. We need to select all the lines that have 'PO Box' , 'P.O.Box' , 'Box' , 'P O Box' , etc, and replace each one of them with 'P O BOX' in the strings. In the regular expression below, '*' matches the preceding sub expression zero or more times, and '?' matches the previous subexpression zero or one time. '\s' matches a white space character, including a space or a tab. Here 'PO Box' , 'P.O.Box' , 'Box' , 'P O Box' et are substituted by 'P O BOX' by PRXCHANGE function .

```
DATA _NULL_;
RETAIN PATTERN PATTERN1;
IF _N_ =1 THEN PATTERN =
PRXPARSE ("s/P?\s*\.*\s*O?\s*\.*\s*BOX\s*\.*\s*/P O BOX /i");
INPUT STRING $31. ;

CALL PRXCHANGE(PATTERN, 5, STRING);

PUT STRING;

DATALINES;
1250 Church street P.O.BOX 495
P O BOX 2235 15 North st.
PO BOX 20202020.
123 Mill st P BOX 223
11 prospect ave p o box
BOX 22282828
P Box 225
p box22152
pobox 2212
Pbox 2345
P. O. box. 256
;
```

The output will be:

```
1250 Church street P O BOX 495
P O BOX 2235 15 North st.
P O BOX 20202020.
123 Mill st P O BOX 223
11 prospect ave P O BOX
P O BOX 22282828
P O BOX 225
P O BOX 22152
P O BOX 2212
P O BOX 2345
P O BOX 256
```

CONCLUSION

If we use traditional character functions to do the above string manipulations, the codes could be very long. Even though code may be compact when we use SAS regular expression, it may sometimes be more efficient to use traditional SAS character functions to deal with string manipulations.

If you want to learn more about Perl regular expressions in SAS, refer to Cody's paper in reference 1, and/or the SAS documentation (version 9) on Perl regular expressions.

REFERENCES

1. Cody, Ronald (2004), "An Introduction to Perl Regular Expression in SAS 9", *Proceedings of the 29th Annual SAS Users Group International*

ACKNOWLEDGMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

CONTACT INFORMATION

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

Selvaratnam Sridharma
Economic Planning and Coordination Division
U.S. Bureau of the Census
Address
Washington, DC 20233-6100
301-763-6774
Email: selvaratnam.sridharma@census.gov