Paper 006-2011

A better way to search text: Perl regular expressions in SAS

Kevin McGowan, Independent SAS Consultant
Brad Lagle, SRA International, Durham, NC

ABSTRACT

Many programmers use regular expressions to make their programs more efficient, especially when the program processes large amounts of text format data. One popular type of regular expressions are Perl regular expressions. Starting with version 8 of SAS®, the power of Perl regular expressions is now available to SAS programmers. This paper provides a short introduction to Perl regular expressions in SAS. The reader should already have a basic knowledge of SAS data step programming and some knowledge about how to create regular expressions.

INTRODUCTION

Pattern matching enables a program 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. The SAS Data step supports two types pattern-matching functions and CALL routines: SAS regular expressions and Perl regular expressions.

Regular expressions are a widely used pattern language which provides a programmer with tools for parsing large amounts of text. Regular expressions are composed of standard characters and special characters that are called metacharacters.

SAS introduced Perl regular expressions in version 8 to give SAS programmers the power to use the same type of regular expressions that are currently used in Perl and other programming languages. Some of the advantages of using Perl regular expressions are:

- Many users already know them from working in other languages.
- Perl regular expressions have more flexibility than the existing SAS regular expressions.
- Users can combine Perl code with SAS code in the same program.

Many times a SAS program performs the following tasks: validates data, replaces text, and extracts a substring from a string. Using Perl regular expressions allows the program to do all three tasks in one step, rather than in multiple steps. By reducing the number of steps, the program is easier to maintain and is also less prone to have mistakes.

Perl regular expressions have been widely used by many programs in other languages such as Java for a long time. SAS implements regular expressions using a modified version of Perl 5.6.1 via a set of new functions in SAS called the PRX functions. Only Perl regular expressions are available in SAS via the PRX functions, not the entire Perl language. The following parts of Perl regular expressions are not supported within SAS:

- Perl variables
- The regular expression options /c, /g, and /o and the /e option with substitutions
- Named characters, which use the \N{name} syntax
- The metacharacters \pP, \PP, and \X
- Executing Perl code within a regular expression. This includes the syntax (?{code}), (??{code}), and (?p{code}).
- Unicode pattern matching
- Using ?PATTERN?. ? is treated like an ordinary regular expression start and end delimiter.
- The metacharacter \G.
- Perl comments between a pattern and replacement text. For example: s{regexp} # perl comment {replacement}
- Matching backslashes with m/\\\\/. Instead m/\\/ should be used to match a backslash.

Perl regular expression basics

Perl regular expressions are composed of characters and special characters that are called metacharacters. To perform a match, SAS searches a source string for a substring that matches the Perl regular expression that you specify using the new PRX functions. Metacharacters enable SAS to perform special actions when searching for a match. The following is a short list of metacharacters that are used in Perl regular expressions:

\	Marks the next character as either a special character, a literal, a back reference or an octal escape
^	Matches the position at the beginning of the input string
$	Matches the position at the end of the input string
|	Specifies an or condition when you compare alpha numeric strings
*	Matches the preceding sub expression zero or more times
+	Matches the preceding sub expression 1 or more times
[abc]	a character set that matches any of the enclosed characters
[^123]	a character set that matches any character that is not enclosed
[1-4]	a character set that matches any character that is within the range enclosed
[^3-9]	a character set that matches any character that is not within the range enclosed

There are many more metacharacters that can be used in Perl regular expressions.

Basic SAS functions for Perl regular expressions

There are five primary functions that make up the language elements to implement Perl regular expressions in SAS. The five functions along with brief descriptions are:

- PRXPARSE – "Compiles" a Perl regular expression so that other PRX functions can use it for pattern matching of a character string. It returns a pattern matching number. If an error occurs in the parsing of the regular expression, SAS returns a missing. PRXPARSE constructs the Perl regular expression with met. Typically, this is the first function that will be used in a program. This function alone is not very helpful until its output is combined with other PRX functions.

- PRXPAREN – Returns the last bracket for which there is a match in the pattern. This function is used when there is more than one bracket in the regular expression.

- PRXMATCH – Searches for a pattern match and returns the position at which the pattern is found. You commonly include this function to search for a string within text.

- PRXCHANGE – This function actually performs a pattern matching replacement. PRXCHANGE is very similar to the standard SAS substring function when the function is used on the left side of the equal sign.

- PRXPOSN – Returns the value of the capture buffer. It is used to determine where in the string the pattern occurs. The capture buffer is created with one of the other PRX functions. This function does not directly return the substring. This must be done using the standard SAS substring function.

These additional SAS PRX functions are less commonly used:

- CALL PRXFREE – A call function that frees up the unneeded memory that was allocated to a Perl regular expression

- CALL PRXNEXT – Returns the position and length of a substring that matches a pattern and iterates over multiple matches within one string.

- CALL PRXDEBUG – This function enables PRX functions in a Data step to send debug output to the SAS log. When things are not working out correctly, this function can be very helpful.

- CALL PRXSUBSTR – Returns the position and length of a substring that matches a pattern.

Examples of three basic uses of Perl Regular Expressions in SAS

There are typically three cases in which Perl regular expressions are handy in SAS programming in a data step: replacing text, extracting a substring, and validating data. One important note about the use of regular expressions in SAS is that they normally only need to be created one time. If the regular expression is going to be used in a data step, in most cases it can be done while the first data set record is being processed and then stored in a SAS variable using the retain function for later use. By using this technique all of the processing for regular expression compilation will only be done one time instead of potentially thousands of times. The following examples cover each of those three situations.

Example 1 – Text replacement

This example uses macro variables and regular expressions to replace two occurrences of symbols: the less-than character (<) is replaced by &lt; and the two occurrences of the greater-than character (>) are replaced by &gt;.

```
data _null_;
  if _N_ = 1 then do;
      retain abclt abcgt;
      abclt = prxparse('s/</&lt;/');      /*  make a regular expression   */
      abcgt_ = prxparse('s/>/&gt;/');   /*  make a regular expression   */
      if missing(abclt) or missing(abcgt) then
        do;
          putlog "ERROR: Invalid regexp.";     /* check to make sure the
regular expressions compiled OK */
          stop;
        end;
    end;
  input;
  call prxchange(abclt, -1, _infile_);    /* perform the replacements using
                                              PRXCHANGE */
  call prxchange(abcgt, -1, _infile_);
  put _infile_;
  datalines4;
```

Example 2 – Data validation

This example shows how to use Perl regular expressions to validate data to make sure it is in the correct format. In this case, we are checking to make sure the data is in the format of a standard US phone number. The program checks for two formats for phone numbers – with parentheses or with dashes.

```
data _null_;
  if _N_ = 1 then do;  /* only do this one time , for first record */
  parenf = "\([2-9]\d\d\) ?[2-9]\d\d-\d\d\d\d";  /*check for format w parens */
  dashf = "[2-9]\d\d-[2-9]\d\d-\d\d\d\d";    /* check for format with dash */
```

```
     phregexp = "/(" || parenf || ")|(" || dashf || ")/";    /* look for both patterns */
         retain re;
         re = prxparse(phregexp);  /* compile the regular expression */
        if missing(re) then  /* check to make sure expression is valid */
           do;
             putlog "ERROR: Invalid regexp " phregexp;   /* print if error found */
              stop;
           end;
      end;

    length first last home business $ 16;
    input first last home business;   /* read in the data */

    if ^prxmatch(re, home) then   /* check for valid home number */
      putlog "NOTE: Invalid home phone number for " first last home;

    if ^prxmatch(re, business) then  /* check for valid business number */
      putlog "NOTE: Invalid business phone number for " first last business;

    datalines;
Jonas Grumby (949)369-1890 (910)446-2167
Monty Hall 800-899-2164 360-973-6201
Peter Parker (508)852-2146 (508)366-9821
Elvis Presley . 919-782-3199
David Thompson . .
Renee Foster 7042982145 .
Thunderclap Newman 209/963/2764 2099-66-8474
;
```

Example 3:   Sub string extraction

This example shows how to use Perl regular expressions to take sub strings out of text. In this case, we are checking to make sure the data is in a North Carolina area code number set that we specify. This example also uses validation when it checks the area codes against a list of numbers that is stored in the program. The first part of this example is a repeat of the last example, so there are no comments for that section.

```
data _null_;
   if _N_ = 1 then
      do;
         parenf = "\(([2-9]\d\d)\) ?[2-9]\d\d-\d\d\d\d";
         dashf = "([2-9]\d\d)-[2-9]\d\d-\d\d\d\d";
```

```
        phregexp = "/(" || parenf || ")|(" || dashf || ")/";
        retain re;
        re = prxparse(phregexp);
        if missing(re) then
          do;
            putlog "ERROR: Invalid regexp " regexp;
            stop;
          end;

       retain areacode_re;  /* use retain since the data does not change */
      areacode_re = prxparse("/828|336|704|910|919|252/");   /* list of codes */
        if missing(areacode_re) then
          do;
            putlog "ERROR: Invalid area code regexp";
            stop;
          end;
      end;

    length first last home business $ 16;
    length areacode $ 3;
    input first last home business;  /* read in data */

    if ^prxmatch(re, home) then  /* check for home format */
      putlog "NOTE: Invalid home phone number for " first last home;

    if prxmatch(re, business) then  /* found a business number */
      do;
        which_format = prxparen(re);   /* determine which format */
        call prxposn(re, which_format, pos, len);  /* determine position */
        areacode = substr(business, pos, len);   /* get area code from number */
        if prxmatch(areacode_re, areacode) then   /* check area code vs. list */
          put "In North Carolina: " first last business;
      end;
      else
        putlog "NOTE: Invalid business phone number for " first last business;
    datalines;
    [data would be placed here]
```

CONCLUSION

Good programmers are always looking for ways to make their programs better and more efficent. Any SAS programmer who processes text data on a regular basis should strongly consider adding Perl regular expressions to their SAS programming toolbag.

CONTACT INFORMATION

Kevin McGowan
Kpmnc24@yahoo.com

Brad Lagle
Brad_Lagle@sra.com
SRA International
2605 Meridian Parkway
Durham, NC 27707
919-313-7707