An Introduction to PERL Regular Expression in SAS®

James J. Van Campen, SRI International, Menlo Park, CA

ABSTRACT

Regular expressions are a powerful technique for searching and manipulating text data. This paper introduces the concept of regular expressions and presents several SAS functions and call routines for working with them. Specifically, PERL regular expression syntax and the SAS functions for working with PERL regular expressions are discussed. Detailed examples are provided which illustrate how to write regular expressions and use the functions in SAS to find and manipulate a variety of text strings.

INTRODUCTION TO REGULAR EXPRESSIONS

Regular expressions are user-defined patterns that describe a set of character strings. Once defined, you can search for the pattern in character data. After locating a string, it can be extracted or modified as needed. SAS supports two types of regular expressions, SAS and PERL. PERL regular expressions are presented in this paper because the PERL syntax for defining regular expressions is a commonly used standard, and SAS provides a comprehensive set of tools for working with PERL regular expressions.

PERL SYNTAX FOR DEFINING REGULAR EXPRESSIONS

PERL has a rich syntax for defining regular expressions. A regular expression may consist of a specific string of characters such as "`/cat/`". Regular expressions may also contain wildcard characters or position parameters (meta-characters) such as "`/^c\wt/`". Below is a brief list summarizing some of the more common meta-characters used in PERL regular expressions.

*	Matches the previous sub expression zero or more times
+	Matches the previous sub expression one or more times
?	Matches the previous sub expression zero or one times
{n}	Matches the previous sub expression n times
{n,}	Matches the previous sub expression n or more times
{n,m}	Matches the previous sub expression n or more times but not more than m times
.	Matches exactly one character
^	Matches the beginning of a string
$	Matches the end of a string
\d	Matches a digit (0-9)
\D	Matches a non-digit
\w	Matches a word character (upper or lower case letter, blank, or underscore)
\s	Matches a white space character (space or tab)
a|b	Matches a or b
[abc]	Matches any of the characters in the brackets
[^abc]	Matches any characters except a, b, or c
[a-z]	Matches the letters a to z
[a-zA-Z]	Matches the letters a to z or A to Z
\(	Matches (
\)	Matches)
\\	Matches \

SAS SYNTAX FOR DEFINING REGULAR EXPRESSIONS

The SAS function for defining a PERL regular expression is PRXPARSE(). It has one argument which is the regular expression definition, and it returns a number which is the regular expression ID. Regular expressions only need to be defined once per data step. Usually, they are defined when processing the first record in a data file. The variable to which the regular expression ID is assigned is retained for use with each record. Here are some simple examples to illustrate the use of PRXPARSE(). Note that the "/ and /" simply delimit the beginning and the end of the regular expression.

```
Regexp1_ID = PRXPARSE("/cat/");
Regexp2_ID = PRXPARSE("/^Cat/");
Regexp3_ID = PRXPARSE("/c[a|o]t/");
```

The first expression will find the string “cat” anywhere in the data. The second expression will find the string “Cat” only at the beginning of the data. The third expression will find the strings “cat” or “cot anywhere in the data. These definitions are relatively simple to understand, but it is strongly recommended that all regular expression definitions be thoroughly documented, because they can be very difficult to decipher. In fact, some people have said regular expressions are “write only” because they are often so difficult to read.

CAPTURE BUFFERS

Sometimes the character data you are searching for can not be identified without using some of the surrounding characters. In these cases, a capture buffer is used when defining the regular expression. For example: if you are looking for a string enclosed in brackets. The brackets are a necessary part of the pattern definition, but the data of interest are the characters between the brackets. To add a capture buffer to a regular expression simply put parentheses around the part of the regular expression you want to locate. Suppose I had the following text string and wanted to extract the table number. I could not just look for digits because I might get the digit from the question code (B3a). I need to look for a number preceded by the word “Table”. The parentheses around \d+ identify it as a capture buffer. More than one capture buffer can be designated within a single regular expression.

```
data temp1;
    tabletitle= “Table 2. Gender of participant (B3a)”;
run;

data temp2;
    set temp1;
    if _n_=1 then do;
        regexp1_ID= prxparse(“/Table (\d+)/”);
    end;
    retain regexp1_ID;
run;
```

LOCATING STRINGS USING REGULAR EXPRESSIONS

There are one function and two call routines in SAS for locating matching strings. They are PRXMATCH(), CALL PRXSUBSTR(), and CALL PRXPOSN(). PRXMATCH() is the simplest of the three. It accepts two parameters. The first parameter is the regular expression ID if previously defined with PRXPARSE() or the regular expression definition if not previously defined. The second parameter is the character string to be searched. PRXMATCH() returns the start position of the matching string. CALL PRXSUBSTR() is similar to PRXMATCH() except that it can return the length of the matching string in addition to the start position. It accepts up to four parameters. The first is the regular expression ID. The second is the string to be searched. The third is the variable to which the start position will be assigned. The fourth is the variable to which the length will be assigned, and it is optional. CALL PRXPOSN() is used to identify the start position and length of a capture buffer. It is used in conjunction with the PRXMATCH() function. CALL PRXPOSN() has up to four parameters. The first is the regular expression ID. The second is the capture buffer ID. The third is the variable to which the start position of the capture buffer will be assigned. The fourth is the variable to which the length of the capture buffer will be assigned, and it is optional. Use of these functions and call routines will be illustrated in the example section of this paper.

MANIPULATING THE MATCHING TEXT ONCE IT IS LOCATED

The call routine, CALL PRXCHANGE() searches for matching strings and replaces them with another character string. It is similar to the function TRANWRD(), but it has the advantage of performing searches using wildcards. CALL PRXCHANGE() accepts up to seven parameters. The first parameter is the regular expression ID if previously defined with PRXPARSE() or the regular expression definition if not previously defined. The second parameter is the number of times to find and replace the character string. This parameter should be set to -1 if you want to replace all occurrences. The third parameter is the character string to be searched. The fourth parameter is the variable to which the new character string will be assigned. This is optional, and if it is omitted the new value is assigned to the original variable. The fifth parameter is the variable to which the result length is assigned, and it is optional. The sixth parameter is the variable to which the truncation flag is assigned, and it is optional. The truncation flag indicates if the result of the change exceeds the length of the variable to which it was assigned. The last parameter is a variable to which the number of changes is assigned, and it is optional.

The matching character strings may also be extracted and assigned to a new variable using the SUBSTRN() function. SUBSTRN() differs from SUBSTR() in that it will return a missing value if passed a length parameter equal to zero.

AN EXAMPLE THAT PUTS IT ALL TOGETHER

I once had to read in data from a series of Excel® worksheets using DDE. Each worksheet had a title cell that contained the table number, the table title, the year, and the question code. After reading in the data, all that information was contained in a single character variable. I used PERL regular expressions to parse the data into four separate variables. Here is a sample of ten strings that had to be parsed.

```
Obs  tabletitle
  1  Table 12a. Name of the extended content standards 2005-06 (A7d)
  2  Table 1a. Assessment title 2005-06 (A1a)
  3  Table 2b. Assessment developer 2006-07 (A1b)
  4  Table 4b. Purpose(s) of assessment 2006-07 (A1e)
  5  Table 59a. How are scoring conflicts resolved? 2005-06 (E102c, E102c1 part 3)
  6  Table 5a. Grades in which assessment was used 2005-06 (A3)
  7  Table 64b. What student criteria are used in scoring? 2006-07 (E108e part 2)
  8  Table 65a. What system criteria are used in scoring? 2005-06 (E108e part 3)
  9  Table 6a. Assessment approaches (structures/types of items used) 2005-06 (A4)
 10  Table 77b. How do the individual student reports express results? 2006-07 (H10)
```

SPECIFYING THE PERL REGULAR EXPRESSIONS

I had to define regular expressions to help me extract the four pieces of information from each title. I chose to use three regular expressions, one of which had two capture buffers and another that had one capture buffer. All the work was done in a single data step.

```
data temp2;
    length title $
    set temp1;
    if _n_=1 then do;
        prxid_numandtitle= prxparse("/Table (\d+)[ab]\. ([\w\W]+) 200[56]-0[67]/");
        prxid_year= prxparse("/200[56]-0[67]/");
        prxid_qcode= prxparse("/200[56]-0[67] \(([A-Z][\w\W]+)\)/");
    end;
    retain prxid_numandtitle prxid_year prxid_qcode;
    numandtitle_match= prxmatch(prxid_numandtitle,tabletitle);
    call prxposn(prxid_numandtitle,1,pos_num,len_num);
    call prxposn(prxid_numandtitle,2,pos_title,len_title);
    call prxsubstr(prxid_year,tabletitle,pos_year,len_year);
    qcode_match= prxmatch(prxid_qcode,tabletitle);
    call prxposn(prxid_qcode,1,pos_qcode,len_qcode);

    tablenumber= input(substrn(tabletitle,pos_num,len_num),8.);
    title= substrn(tabletitle,pos_title, len_title);
    year= substrn(tabletitle,pos_year, len_year);
    qcode= substrn(tabletitle,pos_qcode, len_qcode);
run;
```

While processing the first record (_n_=1), the regular expressions are defined using PRXPARSE(). The first regular expression has two capture buffers. The first is the table number and the second is the title. The table number is one or more digits preceded by the word “Table” and followed by the letter “a” or “b” and a period. The parentheses around the one or more digits delimit the first capture buffer – I did not want to keep the “a” or “b”. The title is one or more word or non-word characters followed the year (either 2005-06 or 2006-07). The parentheses around the one or more word or non-word characters delimit the second capture buffer. The regular expression for the year was pretty straightforward to define. The regular expression for the question code was the year followed by one upper case letter followed by one or more word or non-word characters. Once again the parentheses delimit the capture buffer. Note that the opening and closing parentheses surrounding the questioncode in the regular expression are preceded by slash. Also note that PRXMATCH() was always executed to locate the matching strings, prior to using CALL PRXPOSN() to locate the capture buffers. With regular expressions, as in life, there is more than one way to skin a cat. The regular expressions presented here are not the only way to get the job done, and they may not be the most efficient. They suffice, however, to illustrate how regular expressions work.

THE RESULTS

Once the position and length of the strings are identified, the four variables are extracted and assigned to new variables using the substrn() function. Tablenumber is a numeric variable, and the conversion is done with the input() function. The other three variables are character and are assigned directly. A proc print shows that all of the strings have been correctly parsed.

```
proc print data= temp2;
    var tablenumber title year qcode;
run;
```

```
Obs   Tablenumber   Title
 1       12         Name of the extended content standards
 2        1         Assessment title
 3        2         Assessment developer
 4        4         Purpose(s) of assessment
 5       59         How are scoring conflicts resolved?
 6        5         Grades in which assessment was used
 7       64         What student criteria are used in scoring?
 8       65         What system criteria are used in scoring?
 9        6         Assessment approaches (structures/types of items used)
10       77         How do the individual student reports express results?

Obs   Year      Qcode
 1    2005-06   A7d
 2    2005-06   A1a
 3    2006-07   A1b
 4    2006-07   A1e
 5    2005-06   E102c, E102c1 part 3
 6    2005-06   A3
 7    2006-07   E108e part 2
 8    2005-06   E108e part 3
 9    2005-06   A4
10    2006-07   H10
```

CONCLUSION

Regular expressions are powerful tools for working with character data. They make quick work of searching long and complicated character strings for matching substrings. Once matching strings are found, text functions such as substrn() can be used to extract them. The PERL regular expression syntax is rich but also somewhat cryptic, and documenting regular expressions is essential to help you and others understand them in the future. The above example illustrates the basics of using regular expression techniques with complex character data.

REFERENCES

Cody, Ron (2004) *SAS Functions by Example*, SAS Institute Inc., Cary, NC, USA

McCracken, Mary and Van Campen, James J., *Taming Your Character Data with Regular Expressions in SAS® - Part I*, WUSS 2004

Van Campen, James J. and McCracken, Mary, *Taming Your Character Data with Regular Expressions in SAS® - Part II*, WUSS 2004

CONTACT INFORMATION

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

James J. Van Campen
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025
james.vancampen@sri.com

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.