Taming Your Character Data with Regular Expressions in SAS® – Part I
Mary McCracken, SRI International, Menlo Park, CA
James J. Van Campen, SRI International, Menlo Park, CA

ABSTRACT
Regular expressions (regexp) are a powerful tool for working with character data. This paper provides an introduction to the new Perl® regular expression capabilities available in SAS® 9. Part I will teach you the basic building blocks of regular expression syntax. In Part II you will learn more advanced techniques: using regular expressions to find (PRXSUBSTR), parse (PRXPOSN), and change (PRXCHANGE) text data strings.

INTRODUCTION
In Part I, we will explore the basic Perl syntax that enables you to search out specific text patterns. You will be able to define a text pattern to search for. This can include any combination of upper case, lower case, numeric, white space, and special characters. A pattern is defined using the PRXPARSE function. Once the pattern is defined, you can search your character data for the pattern using PRXMATCH to determine the exact location of the desired string.

In Part II we will expand on this topic by introducing 3 additional functions: PRXSUBSTR, PRXPOSN, and PRXCHANGE. CALL PRXSUBSTR returns the start position and the length of a match. CALL PRXPOSN returns the start position and length for a capture buffer. CALL PRXCHANGE substitutes one string for another. With these call routines specific strings can be located, extracted, and/or replaced.

DEFINING PERL REGULAR EXPRESSIONS
When confronted with messy strings of character data, we arm ourselves with older, proven SAS tools such as index, substring, and scan. In SAS 9, we are offered more powerful tools in the form of regular expressions.

Similar to prior SAS tools, regular expressions are used to identify and/or manipulate data by locating patterns within a string. More specifically, they can search large amounts of data conveniently and efficiently, locate patterns in a text string, or specify the absence of a string. Once a pattern is found you can obtain the position of the pattern and manipulate it.

Regular expressions have the same functionality as a collection of familiar SAS tools that we all fall back on to sort out our character mishmash. In addition, the new regular expressions feature allows you to combine several separate function calls into a single expression. Extolling the use of regular expressions, SAS OnlineDoc 9.1 states that this new code enhancement will result in code that is less likely to have errors, is clearer to read, is easier to maintain, and is more efficient in terms of improving system performance. Although SAS 9 has also introduced SAS regular expressions, for the purpose of this paper we will be focus exclusively on Perl regular expressions.

WHAT IS A PERL REGULAR EXPRESSION
A regular expression is a user-specified pattern that will be matched against a character string. Regular expressions are created using the PRXPARSE function. The syntax of the function is:

```
PRXPARSE(Perl-regular-expression)
```

It might help to see the syntax written and explained as such:

```
NewRegExp = PRXPARSE("/pattern/");
```

Where
- NewRegExp: the variable you are creating:
- PRXPARSE: the keyword
- (" "): the required SAS syntax
- //: defines the Perl expression, / is the default Perl delimiter
- pattern: what you are going to search for in the text string
WHAT DOES A PERL REGULAR EXPRESSION DO

A regular expression is compiled on the first iteration of the data step; therefore you want to initialize the regular expression when _n_ = 1. Be sure to add a retain statement to the dataset – this will ensure that the regular expression is carried through the dataset. Remember, at this point you are only defining a search string, not doing the actual search.

Once you’ve defined the search string, you’ll want to search your text for that pattern. To search the data, simply use the PRXMATCH function. The syntax of the function is:

\[
\text{PRXMATCH(defined-regular-expression, text_string)}
\]

OR

\[
\text{NewFindVar = PRXMATCH(NewRegExp, text_string)}
\]

Where NewFindVar is the variable you are creating to contain the match index:

PRXMATCH is the keyword
NewRegExp is the variable created in PRXPARSE to define your pattern
text_string is the variable name of your text data

Here is an overall example of how PRXPARSE and PRXMATCH function together:

```sas
data Candy;
set ILoveSugar;
retain chocolate taffy;

/**create regular expressions **;
if _n_ = 1
then do;
  chocolate = PRXPARSE("/Hershey/"); /**define Hershey**;
  taffy = PRXPARSE("/LaffyTaffy/"); /**define LaffyTaffy**;
end; /**end if first iteration of dataset**;

/**search the regular expressions in the text string **;
Chocolate_index = PRXMATCH(chocolate,text_candy_data);
Taffy_index = PRXMATCH(taffy,text_candy_data);
run;
```

Sample Print:

<table>
<thead>
<tr>
<th>chocolate</th>
<th>taffy</th>
<th>Chocolate_index</th>
<th>Taffy_index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>25</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>34</td>
<td>5</td>
</tr>
</tbody>
</table>

Each time you compile a regular expression using PRXPARSE, SAS assigns a sequential number to the new variable. In a print of the dataset, the value for every variable Chocolate is 1 since it is the first regular expression defined. The value for every variable Taffy is be 2 since it is the second regular expression defined. If SAS had been unable to compile the PRXPARSE it would return a value of numeric missing (.). The values for Chocolate_index and Taffy_index are the position in the text string where the specified pattern starts. A value of 0 for Chocolate_index or Taffy_index indicates that the string was not found.

SYNTAX

In the syntax and examples above we performed simple, explicit searches for a definite string or word. Regular expressions show their true strength in their flexibility of searching, not just with a defined word but with Metacharacters. Metacharacters are characters that have a special meaning and describe other characters. There are many of them - many more than will fit in this paper. We have therefore chosen a chunk that will be the most common and hopefully the most immediately useful.
BASICS

// Used to locate a specific text expression. This is sensitive to both case and spaces.
PRXPARSE("/Lemon/") will find the substring "Lemon" in "Lemon Pie"
PRXPARSE("/lemon/") will not find the substring "lemon" in "Lemon Pie"

\w Used to locate a word character. This will include any character A-Z, a-z, 0-9 and _ (underscore).
PRXPARSE("/\w/") searching for any word character will find the "M" in "M&M"
PRXPARSE("/\w/") searching for any word character will find the "W" in "tab tab Willy Wonka"

\W Used to locate any non-word character; anything other than A-Z,a-z,0-9 and _ (underscore) such as symbols like ",", ",", ",", and ",".
PRXPARSE("/\W/") searching for any non-word character will find the "&" in M&M

\d Used to locate any single digit from 0-9.
PRXPARSE("/\d/") searching for any digit will find the "3" in "3 Musketeers"

\D Used to locate any single non-digit.
PRXPARSE("/\D/") searching for a non-digit will find the "t" in "5th Avenue Bar"

\s Used to locate any whitespace such as a space or tab.
PRXPARSE("/\s/") searching for a space will find the space after d in "Almond Joy"

\S Used to locate any non-whitespace, excludes things such as a space or tab.
PRXPARSE("/\S/") searching for any non-white space will find the "C" in "tab Crunch"

ENHANCERS: ADD QUANTITY

+ Used to match the previous expression 1 or more times.
PRXPARSE("/\d\d+/") will search for 2 or more digits
PRXPARSE("/\s+/") will search for 1 or more spaces
PRXPARSE("/\d\d\d+/") searching for 3 or more digits will find the "100" in the "100 Grand Bar" but not the "3" in "3 Musketeers"

* Used to match the previous expression 0 or more times.
PRXPARSE("/\d\d\d*/") will search for 2,3 or more digits
PRXPARSE("/\s\s*/") will search for 1,2 or more spaces
PRXPARSE("/ooo*/") searching for 2, 3 or more o's will find the "oo" in "Goobers" or the "oooooo" in "I loooooove candy" but not the "o" in "Gobstopplers"

? Used to match the previous expression 0 or 1 times.
PRXPARSE("/\d\d\d?/") will search for 2 or 3 digits
PRXPARSE("/\s\s?/") will search for 1 or 2 spaces
PRXPARSE("/\d\d?/") searching for 1 or 2 digits will find the "3" in "3 Musketeers" and the "10" in "100 Grand Bar"

. Used to match exactly one character.
PRXPARSE("/c.c/") searching for the letter "c", followed by any single character, followed by the letter "c" will match "coc" in "coco puffs" but will not match "choc" in "chocolate bombs"

{} Used to locate the previous pattern X amount of times. This can be used with the following parameters {x,y} where it will match the previous expression x or more times but not more than y times.
PRXPARSE("/\sub{5}/") will find the pattern "sub" 5 times
PRXPARSE("/\sub{5, 9}/") will find the pattern "sub" 5 or more times but no more than 9 times
PRXPARSE("/\z{2, 5}/") searching for "z" 2 or more times but no more than 5 times will find the "zzz" in "Razzzleberry Taffy"
SPECIFY LOCATION

^ Used to restrict the search to the beginning of the string.
PRXPARSE("/^\D/") searching for a non-digit at the beginning of the data string will find the “S” in “Spree” but will not match on “3 Musketeers”

\A Used to restrict the search to the beginning of the string. This works the same way as ^.
PRXPARSE("/\A\D/") searching for a non-digit at the beginning of the data string will find the “S” in “Spree” but will not match on “3 Musketeers”

$ Used to restrict the search to the end of a string.
PRXPARSE("/Pop$/") searching for the pattern “Pop” at the end of the string will find the “Pop” in “BlowPop” but not in “Pop Rocks”

\Z Used to restrict the search to the end of a string. Can use either upper or lower case Z. This works the same way as the $.
PRXPARSE("/Pop\Z/") searching for the pattern “Pop” at the end of the string will find the “Pop” in “BlowPop” but not in “Pop Rocks”

\b Used to match at a word boundary. This will be one of the following places:
1) Before the first character of a string if the first character is a word character
2) After the last character of a string if the last character is a word character
3) Between a word character and a non-word character, right after the word character
4) Between a non-word character and a word character, right after the non-word character
PRXPARSE("/S\b/") searching for the character “S” at a word bound will find the “S” in “Snickers”
PRXPARSE("/T\b/") searching for the character “T” at a word bound will find the “T” in “Twix”

\B Used to match a non-word boundary. This will match in a word character other than to the first and last character of a word.
PRXPARSE("/ik\B/") searching for the pattern “ik” at a non-word boundary will find the “ik” in “Mike”
PRXPARSE("/Ik\B/") searching for the pattern “Ik” at a non-word boundary will not find the “Ik” in “Ikies”

\t Used to match the tab in a string.
PRXPARSE("/\t/") searching for a tab will find the tab in “Snickers   tab”

GROUPINGS

[] Used to match any of the list of characters included inside the bracket. This is case sensitive. For example:
PRXPARSE("/\[ZRS]/") searching for Z, R, or S will find the “Z” in “Zours”
PRXPARSE("/\[lfs]/") searching for l, f, or s will find the “f” in “LifeSavers”

[^] Used to match anything other than the list of characters inside the bracket. This is case sensitive. For example:
PRXPARSE("/^\^klnicre/") searching for something other than S, k, n, l, c, r, or e will find the “g” in “Snickers”

() Used to create and match a capture buffer. If you used multiple capture buffers in the same PRXPARSE, use CALL PRXPOSN to retrieve the value of the capture buffer. (PRXPOSN is discussed in length in Part II)
PRXPARSE("/(Ave)/") searching for the pattern Ave will find the “Ave” in “5th Avenue Bar”
PRXPARSE("/(world) (fine)/") searching for the patterns world and fine will find the “world” and the “fine” in “The world’s finest candy”
SPECIAL CHARACTERS
| Used as an alternation. This will match the pattern on either side of the pipe taking the first occurrence of either pattern. It works like an “or” statement.

PRXPARSE ("/k|t/") searching for the character k or t will find the “k” in “Skittles”
PRXPARSE ("/k|t/") searching for the character k or t will find the “t” in “Now or Laters”
\ Used to locate key symbols such as “(”, “)”, “,” and “/”. Simply put the \ character in front of any special characters you want to locate.
PRXPARSE ("/\(/\)“) searching for an open parentheses will find the “(“ in “Zots (my favorite)”

i Use to make the pattern not case sensitive. This character will appear after the closing / and before the closing quotes.
PRXPARSE ("/bu/i“) searching for any occurrence of bu regardless of case will find the “bu” in “BubbleYum”

SOME THINGS YOU NEED TO KNOW
Perl regular expressions are very flexible. Your search can be as simple or as complex as you like, depending on your skill level and sense of adventure. As you get more comfortable, keep in mind that there are even more Metacharacters that have not been covered here.

COMMENTS
Perl regular expressions were aptly described by Ron Cody in SAS Functions by Example as write only. They are easy to write intuitively but very challenging to read. To alleviate confusion, include descriptive comments on each PRXPARSE line so other programmers (or you) can read what you had intended to search for.

LIMITATIONS
SAS 9 uses a modified version of Perl 5.6.1. Other components of the Perl language are not accessible. The modified version of Perl regular expressions does not support Perl variables, /c, /g/, /o, /e, /N, /pP, /PP, /X, /G or Perl comments

EXAMPLE
In a more practical example, the above Metacharacters are combined to search character data for an e-mail address, phone number, and customer id.

*create a temporary dataset of string data**;
data all;
length strings $250.;
strings="10009374 94041 tables@yahoo.com (408)971-9459"; output;
strings="Mary Joe 94042 data_stuff@svpal.org 408.971.9459 99 1st St"; output;
strings="10009376 Pat 94004 stats@hotmail.com 408-971-9459 box 98674"; output;
strings="10009377 Darcie 94044 all@directv.com 408 971-9459 po box 24"; output;
strings="10009378 Fiona 94045 science_dept@sjsu.edu 333 ravenswood ave"; output;
strings="0009379 Meme 94046 maven@sbcglobal.net (408) 971-9459 5 A St."; output;
strings="10009380 Mari 94047 408 971 9459 2000 rock suite 4"; output;
strings="10009381 Ann 94048 programmer@cvp.pf.org 4089719459 2387 main"; output;
strings="10009382 Linda teachers@mvwsd.k12.ca.us 99999999 854 north St"; output;
strings="10009383 Bea mathmatic@netzero.com 408 971.9459 37 sunset blv"; output;
; run;
*in new temp dataset, define and locate patterns using regular expressions**;

data play;
  set all;
  retain p_email p_id p_phone;

**initiate prxparse on the first iteration of the dataset**;
  if _n_ = 1 then do;
    p_email = prxparse("/\w*@\w*/"); *Email address: any number of word character, followed by @, followed by any number of word characters*;
    p_id = prxparse("/^\d{8}/"); *Check for 8 digit id at beginning of string*;
    p_phone = prxparse("/\d{3}\W*\d{3}\W?\d{4}/"); *Find phone number as starting with 3 digits, followed by 0 or more word characters, followed by 3 digits, followed by 0 or 1 word characters, followed by 4 digits*;
  end; **end if _n_ = 1**;

**search the text strings for the defined patterns**;
  find_email = prxmatch(p_email,strings);
  find_phone = prxmatch(p_phone,strings);
  check_id = prxmatch(p_id,strings);

  label
    find_email = "Position of start of email address"
    find_phone = "Position of start of phone number"
    check_id = "Problem with id if value is 0"
  ;
  run;

proc print data=play;
  run;

CONCLUSION
Those of us who deal with text data know how daunting it is to work with data that includes messy strings of text especially when you know you will need to make sense of it. Coding which would have been painfully long and tedious in previous versions of SAS can now be streamlined by utilizing regular expressions. At first it seems overwhelming for those of us unfamiliar with Perl, but rest assured that the components of these new tools are straightforward and relatively easy to pick up. Not only are they more efficient, but infinitely more powerful than the previous options available.

The options and possible combinations of these search tools are infinite. This is only a glimpse at the capabilities of regular expressions; a set of building blocks for you to piece together into skyscrapers. In Part II, you will start assembling these blocks to beat your wild and unruly character data into submission. We wish you well on your path to taming your character data.
REFERENCES
Schwartz, Randal L. & Tom Christensen (1997) Learning Perl, O'Reilly & Assoc., Sebastopol, CA, USA
Wall, Larry Tom Christiansen & Jon Orwant (2000) Programming Perl, O'Reilly & Assoc., Sebastopol, CA, USA
Perl Regular Expression Retrieved 7/26/04 from [http://piglet.uccs.edu/~cs301/perl/re.htm]

ACKNOWLEDGMENTS
The authors would like to lavish thanks on the following:
   Peter Godard for suggesting the topic of this paper
   Kathy Valdez for the gentle encouragement to present at WUSS and for editing expertise
   Steve McCracken for editing expertise and constructive criticism
   Cyndi Williamson for graciously shared and well timed pink boxes full of donuts
   Sugar – the ultimate fuel for programming

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:

   Mary McCracken
   SRI International
   333 Ravenswood Avenue, BS-165
   Menlo Park CA 94025-3493

   James J. Van Campen
   SRI International
   333 Ravenswood Avenue, BS-153
   Menlo Park CA 94025-3493

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.