Taming Your Character Data with Regular Expressions in SAS® – Part II

James J. Van Campen, SRI International, Menlo Park, CA
Mary McCracken, 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 regexp capabilities available in SAS 9 ®. Building on the foundation laid in Part I, Part II illustrates how to locate and manipulate character strings using the call routines PRXSUBSTR, PRXPOSN, and PRXCHANGE.

INTRODUCTION

In Part I, the syntax of PERL regexp was introduced. The PERL regexp syntax was used in conjunction with the SAS functions, PRXPARSE and PRXMATCH to define regular expressions and locate matches in character strings. This paper builds on those ideas by introducing and illustrating three SAS call routines, 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 replaced. The following three sections describe each call routine in detail, and the final section illustrates the application of these call routines with an example involving study subject contact data.

CALL PRXSUBSTR

CALL PRXSUBSTR is similar to the function PRXMATCH (see Part I), however, in addition to finding the start position of the match CALL PRXSUBSTR can also find the length of the match. The syntax for CALL PRXSUBSTR is as follows:

CALL PRXSUBSTR (*regular-expression-id, source, start <, length>*)

regular-expression-id: the identification number returned by the PRXPARSE function for the regular expression to be used.
source: the character expression to be searched
start: the start position in **source** where the matching pattern begins
length: the length of the matching pattern (optional)

Given a regular expression id and a source string CALL PRXSUBSTR will return the start position and optionally the length of the matching string. If there is no match CALL PRXSUBSTR will return zero for the start position and length. If there are multiple matches within the character expression CALL PRXSUBSTR will by default return the position and length of the longest match. The position and length information can be passed to another function such as SUBSTRN which can extract the matching string and assign it to a new variable.

CALL PRXPOSN

Sometimes the string of interest must be parsed into components. For instance, if the string can only be found by matching the surrounding characters or the components have value by themselves. In this situation CALL PRXPOSN can be of use. Given a regular-expression-id and a capture-buffer-id CALL PRXPOSN will return the start position and optionally the length of the capture-buffer. A capture-buffer is a portion of a regular expression defined by enclosing it with parentheses (see Part I). A regular expression may have multiple capture-buffers. The syntax for CALL PRXPOSN is as follows:

CALL PRXPOSN (*regular-expression-id, capture-buffer-id, start <, length>*)

regular-expression-id: the identification number returned by the PRXPARSE function for the regular expression to be used.
capture-buffer-id: the number identifying the capture-buffer for the regular expression to be used. The first set of parentheses (capture buffer) in the regexp is assigned 1, the second 2, etc.
start: the start position in **source** where the capture-buffer specified above begins
length: the length of the capture-buffer (optional)

If the capture-buffer-id is zero then CALL PRXPOSN will return the start position and length of the entire match. If the capture-buffer-id is greater than the number of capture-buffers defined then CALL PRXPOSN will return missing

values for start position and length. As with PRXSUBSTR, the position and length information can be passed to another function such as SUBSTRN which can extract the matching string and assign it to a new variable. To fill the capture buffers, a matching function such as PRXMATCH must be run prior to running CALL PRXPOSN. See example code on page 4.

CALL PRXCHANGE

CALL PRXCHANGE searches for regular expression matches and replaces them with another character string. CALL PRXCHANGE is similar to the function, TRANWRD, however, it has the advantage of performing searches with wildcards. The syntax for CALL PRXCHANGE is as follows:

CALL PRXCHANGE (*regular-expression-id* or ***regular-expression, times, old string <, new-string <, result-length <, truncation_value <, number-of-changes>>>>)**

regular-expression-id or **regular-expression**: the identification number returned by the PRXPARSE function for the regular expression to be used or a PERL regular expression.
times: the number of times to search for a match and replace the matching string. If times = -1 then CALL PRXCHANGE will find and replace all matches.
old-string: the character expression in which to search and replace. If new-string is omitted then all changes will be made to old-string. If new-string is specified then old-string will remain unchanged.
new-string: the variable to which the new character expression is assigned (optional)
result-length: the length of the new character expression (optional)
truncation value: a flag which is 1 if the result of the change operation exceeds the length of new-string and zero if the result is less than or equal to the length of new-string (optional)
number-of-changes: the number of substitutions made (optional)

When using CALL PRXCHANGE, it is necessary to use the swap (or substitution) operator when defining the regular expression and the replacement string with PRXPARSE. See example.

EXAMPLE

Given a text file containing contact information, use PERL regular expressions to convert it to a SAS dataset with the following variables: firstname, middleinitial, lastname, address, city, state, zip, area code, phonenumber, and email. To illustrate the use of the PRXCHANGE all three letter domain names (in the email addresses) starting with ‘s’ and ending in ‘i’ will be changed to ‘zap’.

```
John Q Smith 123 Elm St, Mountain View CA 94040 650-987-6543 john.smith@jones.com
Sandra J Jones 357 Main Ave, Sunnyvale CA 94263-4321 408-456-7891 sjjones@hotmail.com
Alfred P Dolittle 369 Dalrimple Blvd, London NV 74028 728-738-4567 slacker@slouch.com
Henry L Higgins 7541 El Camino Real, Redwood City CA 94064 (650)964-7895 nerd@linguist.net
James T Kirk 852 Asteroid Lane, Plutopolis NY 10026-0042 218.423.7536 jim@enterprise.com
Frances Z Smith 65487 Paco Drive, Los Altos CA 94022 (650)964-4678 fzs@smi.com
G Guzman 3636 California Ave Ste 100, Mountain View CA 94041 650-971-4523 gigi@sri.com
J B Halson 8956 Oak St #401, Tupelo MS 34254 321-654-9874 halsonator@elvis.org
Kathy H Parker 12 RR2, Owen Sound ND 45698 412-582-2468 khparker@scusd.k12.ca.us
June Joyworth 431 Haliburton Lane #56A, Washington DC 12345 245-951-6874 junebug@b52.com
```

The first step is to read the data into a SAS dataset. The data is stored as ten records with one character variable called string.

```
data contact_raw;
      infile "&data\contact.txt" missover pad;
      input string $ 1-100;
run;
```

The second step is to determine the regular expressions. The regexp for the domain name substitution will replace an ‘s’ followed by any word character followed by an ‘i’ and a period with ‘zap.’. The period in the regexp must be preceded by a slash since periods have special meaning in the PERL syntax.

```
      domain_prx= prxparse("s/s\wi\./zap./");
```

The first name is at the beginning of the string and has the first character capitalized followed by zero or more lower case letters.

```
      fname_prx= prxparse("/^[A-Z][a-z]*/");
```

The middle initial is a capital letter preceded and followed by a space character. Since what is needed is the letter without the spaces, a capture-buffer can be created by using parentheses around the letter portion of the regular expression definition.

```
mi_prx= prxparse("/ ([A-Z]) /");
```

The last name is preceded by a space, starts with a capital letter and is followed by one or more lower case letters, a space and a digit (the beginning of the address). Once again, a capture buffer is used to separate the portion of interest.

```
lname_prx= prxparse("/ ([A-Z][a-z]+) \d/");
```

The address starts with a digit followed by one or more word or non-word characters and ending with a comma. Use a capture buffer to exclude the comma.

```
address_prx= prxparse("/(\d(\w|\W)+),/");
```

The citystate regexp starts with the comma and a space, contains one or more word characters and spaces and ends with a space and two capital letters. Two capture buffers are created: the first for the city name and the second for the state abbreviation.

```
citystate_prx= prxparse("/, ([\w| ]+) ([A-Z]{2}) /");
```

The zip code regexp begins with the two capital letters for the state and a space followed by either 5 digits and a space or five digits a dash, four more digits and a space. Once again a capture buffer is used to separate the characters of interest.

```
zip_prx= prxparse("/[A-Z]{2} (\d{5}|\d{5}-\d{4}) /");
```

The phone number regexp starts with one or zero open parentheses followed by three digits, exactly one character, three digits again, exactly one character again and four more digits. Each sequence of three or four digits is assigned its own capture buffer.

```
phone_prx= prxparse("/ \(?(\d\d\d).(\d\d\d).(\d\d\d\d)/");
```

The email address is one or more letters, digits or periods, an @, and one or more letters, digits or periods. No capture buffer is necessary here.

```
email_prx= prxparse("/([A-Z]|[a-z]|\d|\.)+@([A-Z]|[a-z]|\d|\.)+/");
```

The program for defining the regular expressions, finding the lengths and positions of the strings and assigning values to the new variables is displayed on the following page. A proc print of the final dataset follows the code.

The third step is to perform the string substitution on the email addresses with CALL PRXCHANGE. No new-string is specified so the changes are made to the original string. A four character string is replacing a four character string, thus the length and truncation variables are not necessary. Two domain names were changed in the output following the code.

The fourth step is to determine the string start positions and lengths using CALL PRXSUBSTR and CALL PRXPOSN as necessary. If PRXPOSN is used to find a match then PRXMATCH must be run for the regexp first. CALL PRXSUBSTR is used for the first name since the regexp is exactly the first name. However, CALL PRXPOSN is needed for the middle initial since the regexp includes the surrounding spaces and a capture buffer is needed to get only the letter. Even though the length of the middle initial is known (1), CALL PRXPOSN returns the mi_len because in some cases the middle initial is missing; in those cases, mi_len is zero. Passing a zero length to SUBSTRN will cause it to return a missing value. Two capture buffers were used for the components of citystate and three capture buffers were used for the components of phone number.

The fifth step is to assign values to the new variables using SUBSTRN. For the phone number two capture buffers were concatenated with a hyphen between them.

Example program

```
data contacts (keep= firstname middleinitial lastname address city state zip areacode
phonenumber email);
       set contact_raw;
* Define the PERL regular expressions using PRXPARSE;
       if _n_ = 1 then do;
              domain_prx= prxparse("s/s\wi\./zap./");
              fname_prx= prxparse("/^[A-Z][a-z]*/");
              mi_prx= prxparse("/ ([A-Z]) /");
              lname_prx= prxparse("/ ([A-Z][a-z]+) \d/");
              address_prx= prxparse("/(\d(\w|\W)+),/");
              citystate_prx= prxparse("/, ([\w ]+) ([A-Z]{2}) /");
              zip_prx= prxparse("/[A-Z]{2} (\d{5}|\d{5}-\d{4}) /");
              phone_prx= prxparse("/ \(?(\d\d\d).(\d\d\d).(\d\d\d\d)/");
              email_prx= prxparse("/([A-Z]|[a-z]|\d|\.)+@([A-Z]|[a-z]|\d|\.)+/");
       end;
       retain domain_prx fname_prx mi_prx lname_prx address_prx citystate_prx zip_prx
phone_prx email_prx;
* Perform string substitution on the email domain names;
       call prxchange(domain_prx, 1, string);
* Use call prxsubstr and call prxposn to determine start position and length of
strings;
* PRXMATCH must be run to fill the capture buffers prior to executing CALL PRXPOSN;
       call prxsubstr(fname_prx, string, fname_pos, fname_len);
       pos_mi= prxmatch(mi_prx,string);
       call prxposn(mi_prx, 1, mi_pos, mi_len);
       pos_lname= prxmatch(lname_prx,string);
       call prxposn(lname_prx, 1, lname_pos, lname_len);
       pos_address= prxmatch(address_prx,string);
       call prxposn(address_prx, 1, address_pos, address_len);
       pos_citystate= prxmatch(citystate_prx,string);
       call prxposn(citystate_prx, 1, city_pos, city_len);
       call prxposn(citystate_prx, 2, state_pos);
       pos_zip= prxmatch(zip_prx,string);
       call prxposn(zip_prx, 1, zip_pos, zip_len);
       pos_phone= prxmatch(phone_prx,string);
       call prxposn(phone_prx, 1, areacode_pos);
       call prxposn(phone_prx, 2, phone1_pos);
       call prxposn(phone_prx, 3, phone2_pos);
       call prxsubstr(email_prx, string, email_pos, email_len);
* Set lengths of new variables;
       length  firstname $30
                     middleinitial $1
                     lastname $30
                     address $40
                     city $30
                     state $2
                     zip $10
                     areacode $3
                     phonenumber $8
                     email $40;
* Use SUBSTRN function to extract strings into variables;
       firstname= substrn(string,fname_pos,fname_len);
       middleinitial= substrn(string, mi_pos, mi_len);
       lastname= substrn(string, lname_pos, lname_len);
       address= substrn(string, address_pos, address_len);
       city= substrn(string, city_pos, city_len);
       state= substrn(string, state_pos, 2);
       zip= substrn(string, zip_pos, zip_len);
       areacode= substrn(string, areacode_pos, 3);
       phonenumber= substrn(string, phone1_pos, 3) || "-" || substrn(string,
phone2_pos, 4);
       email= substrn(string, email_pos, email_len);
run;
```

```
proc print data= contacts noobs;
       title "Contact information stored as ten variables";
run;
```

Contact information stored as ten variables

firstname	middleinitial	lastname	address	city
John	Q	Smith	123 Elm St	Mountain View
Sandra	J	Jones	357 Main Ave	Sunnyvale
Alfred	P	Dolittle	369 Dalrimple Blvd	London
Henry	L	Higgins	7541 El Camino Real	Redwood City
James	T	Kirk	852 Asteroid Lane	Plutopolis
Frances	Z	Smith	65487 Paco Drive	Los Altos
G		Guzman	3636 California Ave Ste 100	Mountain View
J	B	Halson	8956 Oak St #401	Tupelo
Kathy	H	Parker	12 RR2	Owen Sound
June		Joyworth	431 Haliburton Lane #56A	Washington

state	zip	areacode	phonenumber	email
CA	94040	650	987-6543	john.smith@jones.com
CA	94263-4321	408	456-7891	sjjones@hotmail.com
NV	74028	728	738-4567	slacker@slouch.com
CA	94064	650	964-7895	nerd@linguist.net
NY	10026-0042	218	423-7536	jim@enterprise.com
CA	94022	650	964-4678	fzs@zap.com
CA	94041	650	971-4523	gigi@zap.com
MS	34254	321	654-9874	halsonator@elvis.org
ND	45698	412	582-2468	khparker@scusd.k12.ca.us
DC	12345	245	951-6874	junebug@b52.com

CONCLUSION

The output above illustrates the remarkable ability of PERL regexp to impose order on long, messy strings of characters. The strings have been properly parsed into ten variables. In summary, we have combined the PERL regexp syntax and the SAS functions PRXPARSE and PRXMATCH presented in Part I with the call routines CALL PRXSUBSTR, CALL PRXPOSN and CALL PRXCHANGE to locate, replace and manipulate text. PERL regexp capabilities of SAS 9 can help programmers tame even the wildest character data.

REFERENCES

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

CONTACT INFORMATION

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

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

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