PharmaSUG 2012 - Paper AD05

The Application of SAS Perl Regular Expression in Clinical Trial Studies - Batch Processing SAS Programs

Zhong Yan, PharmaNet/i3, Indianapolis, IN
Kimberly Jones, PharmaNet/i3, Austin, TX

ABSTRACT

In clinical trial research, it is sometimes necessary to batch process SAS programs within a study in order to ensure consistent updates. The process of manually opening each program and searching for the places that need to be updated is time consuming and prone to errors. SAS Perl Regular Expression is a powerful tool that can be used to scan files for matches with an identifiable pattern and replace them with customized choices. This paper introduces the application of SAS Perl Regular Expression to batch process SAS programs using SDD SAS programs as an example.

INTRODUCTION

Regular expression is a single expression or pattern used to describe different pieces of text. The power of regular expression is that it can specify a complicated pattern, rather than just fixed characters. Based on the matched pattern, it is very easy to replace any part of or the entire matched pattern with a new string. SAS version 9 introduces Perl regular expressions via a set of functions - PRX functions (for example, PRXPARSE, PRXMATCH, PRXCHANGE, PRXPOSN, PRXDEBUG, etc.). Only Perl regular expressions are available in SAS instead of the entire Perl programming language.

In clinical trials, there can be scenarios that require consistent updates across all programs for a study. For example, input data references may need to be updated to point to the final data lock location, additional titles or footnotes may need to be inserted, or the titles/footnotes may need to be updated when the programs are promoted to the production area. It is time consuming to manually open each program and update each piece of information. For SAS programs in SDD, updating an input reference involves updating a SDD parameter. This is typically a manual process requiring the user to open the program in SDD and click / browse the specific SDD parameter to point to the new location. This manual process is both time consuming and inconvenient. . SAS Perl regular expressions can be used to make these kinds of updates, making the process much easier, due to their unique power of text processing. In this paper, we will introduce metacharacters used in regular expressions, the SAS Perl regular expression functions that will be used in the following case study, and the case study to batch process SAS programs using SAS Perl regular expressions.

METACHARACTERS USED IN SAS PERL REGULAR EXPRESSIONS

Metacharacters (see table 1) are characters and special characters that are used in Perl regular expressions. The paired forward slashes (/…/) are the default delimiters of Perl regular expressions. When SAS searches a source string for any match of the Perl regular expression, metacharacters give SAS flexibility to start the search at a particular location for a match of a particular set of characters.

Metacharacter	Behavior	Example
.	Match any character (one character)	
\d	Match a digit character	/\d\d\d\d\d/ matches any five digits such as 5 digit zip code
\w	Match a word character (upper- and lowercase letters, digits, and underscore)	/\w\w\w\w\w/ matches “ab_5D” but not “ab dd”
\s	Match a whitespace character	/\d\d\s\d/ matches “12 3” but not “123”
\D	Match a non-digit character	/\D/ matches “x” or “,” but not “1”
\W	Match a non-word character	
\S	Match a non-whitespace character	

Metacharacter	Behavior	Example
*	Match >=0 times for the previous expression	/ant*/ is like /an/ and /ant+/. /ant*/ matches “ant”, “and”, “antelope”, and “antttabc”
+	Match >=1 times for the previous expression	/ant+/ matches “antelope”, “anttq”, but not “and”
?	Match 1 or 0 times for the previous expression	/Males?/ matches “Male” or “Males”
{n}	Match exactly n times for the previous expression	/\d{5}/ is same as /\d\d\d\d\d/
{n,}	Match at least n times for the previous expression	/\d{2,}/ matches “12ab”, “123aaa”, but not “1a”
{n,m}	Match at least n but not more than m times	/\d{1,3}done/ matches “1done”, “12done”, “123done”, but not “1234done”
^	Match beginning of line	/^Hi/ matches any line beginning with “Hi”
$	Match end of line	/\d$/ matches any line ends with any digit /^Warning:$/ performs exact match of any line “Warning:” only
[ ]	Matches any single character inside the square bracket. An “^” immediately following the opening square bracket means “Anything but”	/[AEIOUaeiou]/ matches any one of the vowels /[^AEIOUaeiou] matches any one of the non-vowels /[a-zA-Z]/ matches any of the letters of either case.
|	Alternation	/(B/b)ob/ matches Bob or bob /^(Note:|Warning:|Error:)/ detects any line starting with any o “Note:”, ”Warning:”, or “Error:”
( )	Use it for a group of the metacharacters. Two major uses of the grouping: 1. Allows for later retrieving of selected text 2. Allows for alternative phrases	/(Red|Blue|Green)/ matches any of the words “Red”, “Blue”, and “Green” and stores it in a scalar for later retrieval when needed
i	Use after the last delimiter to indicate case-insensitive match	/Hi/i matches “Hi”, “hi”, “HI”, or “hI”
\	Escape character that can be used to match the next metacharacter such as any of: {}[]()^$.|*+?\	/\(/ matches “(abc123“ /\\/ matches “\abc123”

Table 1. Metacharacters in SAS Perl Regular Expressions

SAS PERL REGULAR EXPRESSION FUNCTIONS

Table 2 introduces three frequently used SAS Perl regular expression (PRX) functions: PRXPARSE, PRXMATCH, and PRXCHANGE. Other SAS PRX functions such as PRXPOSN and PRXSUBSTR also have unique features. Users can read SAS online documentation for their usage details.

PRX Function	Description
PRXPARSE	To be used to create a Perl regular expression and return a sequential number as pattern ID for future usage

PRX Function	Description
PRXMATCH	To be used for pattern searching based on pattern ID or Perl regular expression and return the position at which the pattern is found
PRXCHANGE	To do pattern matching and replacement and return the new string

Table 2. Frequently Used SAS PRX Functions

CASE STUDY USING SAS PRX FUNCTIONS IN A CLINICAL TRIAL

Sometimes, during the course of a clinical trial, it is necessary to make a consistent update to all SAS programs stored within a particular study folder. One example of this would be when all of the programs need to be promoted from a QA area to a production area. Update patterns need to be defined across all of the programs. One of the challenges faced in defining such a pattern is that different programs might use different coding styles. Here we have used a case study to show, step by step, how SAS Perl regular expressions can make pattern creation an easy task, thus automating the process of updating a group of SAS programs.

READ SAS PROGRAM NAMES FROM A DIRECTORY INTO SAS MACRO VARIABLES

In order to batch process SAS programs, SAS needs to know the program names to be processed. The FILENAME statement in the code below uses a DOS command, “DIR” with “/B” option, to list the file names. The SAS keyword FILENAME points to the list via the PIPE option. The first DATA step uses an IF statement to exclude SAS programs with names beginning with “out_” or “ori_”. The macro variable num_pgms stores the total number of SAS programs that need to be updated. The macro variables indsas1, indsas2 … etc store each individual SAS program name to be processed later. The second DATA step initializes an empty data set which will later be used to store all of the updated details.

```
libname db "u:\temp";

filename indata pipe 'dir u:\temp/b';

data _null_;
   length fname $50;
   infile indata truncover;

   input fname $50.;
   lfname=length(fname);

   if substr(fname,lfname-3,4) eq ".sas" and substr(fname,1,4) ne "out_" and
     substr(fname,1,4) ne "ori_" then do;
     count+1;
     call symput('indsas'|| compress(put(count, 8.)), fname);
   end;
   else delete;

   call symput ('num_pgms',compress(put(count,8.)));
run;

data log;
   set _null_;
run;
```

PROCESS SAS PROGRAMS ONE AT A TIME INSIDE THE LOOP

The code uses a DO loop to process the SAS programs one at a time. The two FILENAME statements are file name references, one for the input file and the other for the output file. The DATA step reads a SAS program and stores each line of the program in the variable “line_ori”. It assumes the SAS program does not contain “!”, as it was used as the line delimiter. Please note that some blocks of code originally appearing after the DATA step have been omitted.. The omitted code used SAS PRX functions to update the SAS program, and will be introduced. Before the end of the loop, the X command was used to execute the Windows XP cmd command to rename the processed SAS program by adding the prefix “ori_” and rename the output SAS program by removing the prefix “out_”.

```
%macro processit;

   %do i=1 %to &num_pgms;

      filename myfile "u:\temp\&&indsas&i.";
      filename myout "u:\temp\out_&&indsas&i.";

      data indpgm;
         length line_ori $5000;
         infile myfile lrecl=5000 dsd missover dlm="!";
         input line_ori;
         row+1;
      run;

      …(omitted code for updating the SAS program – will be described later)…

      x "ren u:\temp\&&indsas&i.  ori_&&indsas&i." ;
      x "ren u:\temp\out_&&indsas&i.  &&indsas&i." ;

   %end;

%mend processit;
%processit;
```

USE SAS PRX FUNCTIONS TO UPDATE SAS PROGRAM SDD PARAMETERS

If the requested update involves a data reference, such as updating a data reference from QA to PRD for SDD SAS programs, instead of opening the programs to update the SDD parameter, SAS PRX functions can be used to make such an update.

The first step is to construct SAS Perl regular expressions using the function PRXPARSE. The code below constructs four SAS Perl regular expressions with the returned pattern IDs: pid1, pid2, spid1, and spid2.

- Pid1 is for the pattern that contains “displaypath” followed by zero or more white spaces, then a “=”, then zero or more white spaces, followed by a pair of double quotes containing the reference path. The reference path must contain “/qa/”.
- The patter for Pid2 is very similar to that of Pid1. The only difference is that the word “id”, rather than “displaypath”, must be in the pattern. Both pid1 and pid2 patterns use a pair of slashes “/…/” as the delimiters for the patterns.
- The syntax of the patterns for spid1 and spid2 are slightly different from that used in pid1 and pid2. Instead of a pair of slashes, they use 3 slashes and an “s” in the very front. This serves as a substitute operator – “s/…/…/”. The syntax is “s/original pattern/replacement text in SAS Perl regular expression/”. The “line_new” contains the updated original line or the original line when an update is not needed. Flagparm1 and flagparm2 are flags used to indicate the row(s) that were updated.

```
data temp1(drop=pid1 pid2 spid1 spid2);
   length line_new $5000;
   retain pid1 pid2 spid1 spid2;

   set indpgm;
   if _N_=1 then do;
      pid1=prxparse('/displaypath\s*=\s*\"[\d\w\/]+\/qa\/[\d\w\/]+\"/');
      pid2=prxparse('/id\s*=\s*\"[\d\w\/]+\/qa\/[\d\w\/]+\"/');
      spid1=prxparse('s/(displaypath\s*=\s*\"[\d\w\/]+)\/qa\/([\d\w\/]+\")/$1\/prd\/
         $2/');
      spid2=prxparse('s/(id\s*=\s*\"[\d\w\/]+)\/qa\/([\d\w\/]+\")/$1\/prd\/$2/');
   end;

   if prxmatch (pid1, line_ori) then do;
      flagparm1=1;
      line_new=prxchange(spid1, -1, line_ori);
```

```
      if prxmatch (pid2, line_new) then do;
         flagparm2=1;
         line_new=prxchange(spid2, -1, line_new);
      end;
   end;
   else if prxmatch (pid2, line_ori) then do;
      flagparm2=1;
      line_new=prxchange(spid2, -1, line_ori);
   end;

   if line_new = '' then line_new=line_ori;
run;
```

USE SAS PRX FUNCTIONS TO UPDATE SAS PROGRAM PATH AND RELATED INFORMATION

When promoting programs from QA to a production area, it is necessary to update not only the actual data reference, but also the program headers and footnotes to reflect the new path. If the title contains “Test Data – Test Mode”, it will be needed to be updated to “Production Data – Production Mode” at the time of promotion.

In the code below, “i” was used after the ending slash of all of the patterns defined by PRXPARSE. This indicates that a case insensitive match is to be performed. For example:

- The pattern for PIDQA defines a pattern that contains “/qa/”, “/QA/”, “/qA/”, or “/Qa/”.
- The pattern for PIDREQ can match “Requirements Location:”, “Requirements:”, or “Requirement :”, et al. In our case, the path for the requirements documentation does not need any updates.
- The pattern for PIDQA1 can match “%let status=qa;”, “%let area = QA;”, et al.
- The patterns for SPIDQA, SPIDTEST and SPIDQA1 contain substitution text.
- The final updates are stored in line_new2.

```
data temp2(drop=pidqa pidreq pidtest pidqa1 spidqa spidtest spidqa1);
   retain pidqa pidreq pidtest pidqa1 spidqa spidtest spidqa1;
   length line_new2 $5000;
   set temp1;

   if _N_=1 then do;
      pidqa=prxparse('/.*\/qa\//i');
      pidreq=prxparse('/requirements?\s*(location)?\s*:/i');
      pidtest=prxparse('/TEST DATA\s*-\s*TEST MODE/i');
      pidqa1=prxparse('/%let\s+\w+\s*=\s*qa\s*/i');

      spidqa=prxparse('s/(.*)\/qa\//$1\/prd\//i');
      spidtest=prxparse('s/TEST DATA(\s*-\s*)
         TEST MODE/PRODUCTION DATA$1TEST MODE/i');
      spidqa1=prxparse('s/(%let\s+\w+\s*=\s*)qa(\s*)/$1prd$2/i');
   end;

   **-- There is no need to update the requirements path indicated in the --**
   **-- program header. Other than that, all qa need to be updated to prd --**;

   if ^prxmatch (pidreq, line_new) then do;
      if prxmatch (pidqa, line_new) then do;
         flagqa=1;
         line_new2=prxchange(spidqa, -1, line_new);
      end;
   end;

   if prxmatch (pidtest, line_new) then do;
      flagtest=1;
      line_new2=prxchange(spidtest, -1, line_new);
   end;

   **-- Update qa into prd for macro variable --**;
```

```
   if prxmatch (pidqa1, line_new) then do;
      flagqa1=1;
      line_new2=prxchange(spidqa1, -1, line_new);
   end;

   if line_new2 = '' then line_new2=line_new;
run;
```

PROGRAM OUTPUT

The output of our case study will be

- updated SAS programs – the contents of line_new2 was output to a new SAS file line by line;
- SAS data set log – the data set logtemp, in the code below, was used to store the updates from each program

Finally, the updates were appended to the data set log. Output 1 shows the contents of the log data set in our case study.

Program Name	Row	Original Line	Updated Line
demog.sas	1011	/* <sourceContainer system="SDD" source="DOMAIN" displaypath="/root/qa/study1/data/shared/pgx" displayname="modified" id="/root/qa/study1/data/shared/pgx/modified" itemtype="Container"*/	/* <sourceContainer system="SDD" source="DOMAIN" displaypath="/root/prd/study1/data/shared/pgx" displayname="modified" id="/root/prd/study1/data/shared/pgx/modified" itemtype="Container"*/
dsbas.sas	2	CODE NAME : /root/qa/study1/programs_stat/dsbas.sas	CODE NAME : /root/prd/study1/programs_stat/dsbas.sas
dsbas.sas	8	OUTPUT : /root/qa/study1/data/custom/bas.sas7bdat	OUTPUT : /root/prd/study1/data/custom/bas.sas7bdat
kmnomrk.sas	2	CODE NAME : Home/root/qa/study1/programs_stat/kmnomrk.sas	CODE NAME : Home/root/prd/study1/programs_stat/kmnomrk.sas
kmnomrk.sas	10	OUTPUT : Home/root/qa/study1/programs_stat/tfl_output/kmnomrk.ps	OUTPUT : Home/root/prd/study1/programs_stat/tfl_output/kmnomrk.ps
kmnomrk.sas	30	%let title1 =TEST DATA - TEST MODE;	%let title1 =PRODUCTION DATA - TEST MODE;
kmnomrk.sas	31	%let status=qa;	%let status=prd;
kmnomrk.sas	273	/* <sourceContainer system="SDD" source="DOMAIN" displaypath="/root/qa/study1/data/shared/pgx" displayname="modified" id="/root/qa/study1/data/shared/pgx/modified" itemtype="Container"*/	/* <sourceContainer system="SDD" source="DOMAIN" displaypath="/root/prd/study1/data/shared/pgx" displayname="modified" id="/root/prd/study1/data/shared/pgx/modified" itemtype="Container"*/

Output 1. Output Data Set Log

```
data _null_;
   set temp2;
   file myout ;
```

```
   put line_new2;
run;

**-- Extract all updated lines into a data set for validation purpose --**;
data logtemp;
   length pgmname $50;
   set temp2;
   where flagqa=1 or flagqa1=1 or flagtest=1 or flagparm1=1 or flagparm2=1;
   pgmname="&&indsas&i.";
run;

data log(keep = pgmname row line_ori line_new2);
   set log logtemp;
   label pgmname="Program Name" line_ori="Original Line" line_new2="Updated Line";
run;
```

CONCLUSION

Using SAS Perl regular expressions for pattern matching enables the search for and the extract of multiple matching patterns in one DATA step. The same result could be achieved by SAS programs without SAS Perl regular expressions. However multiple programs, due to different coding styles, would require complicated coding logic to handle each string for each program. SAS Perl regular expressions can be used to summarize similar strings to a pattern and efficiently process updates to multiple programs. This paper used a case study to show their usage in batch processing clinical trial programs.

REFERENCES

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

CONTACT INFORMATION

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

Zhong Yan
PharmaNet/i3
4745 Haven Point Blvd,
Indianapolis, IN 46280
(317) 564-2858
ZYan@Pharmanet-i3.com

Kimberly Jones
PharmaNet/i3
1250 South Capital of Texas Hwy, Bldg. 1, Suite 250
Austin, TX 78746
(512) 347-2671
KJones@Pharmanet-i3.com