Paper BB-04

A SAS Programming Framework for Data Extraction Using Perl Regular Expression: The First Wave

Jiangtang Hu
d-Wise Technologies, Inc., Raleigh, NC

ABSTRACT

The Perl regular expression (PRX) is somehow easier to write than to read. This paper introduces bits of PRX by encouraging SAS users new to this field just to fill the programming building blocks presented step by step rather than to read the daunting regular expression (regex) syntax in their first learning stage.

Basically, it presents a simple while robust programming framework for SAS users to utilize PRX to retrieve data from multiple sources, structured or unstructured: 1) scratch the patterns using meta-characters with online tools, 2) validate the expression, 3) locate the patterns, and 4) extract the data needed with an explicit output statement. Such framework makes programmers more concentrated on the pattern needed to match based on their own domain knowledge (or Google Search of course), while left others to the almost same programming blocks. Expansibility wise, if users want to exact more data with different patterns, just repeat such programming blocks with different scratched meta-characters in step 1); also, even for the complicated pattern needed matched, it doesn't need any change for the building blocks but the increasing complication of the meta-characters accordingly.

Three real life examples are demonstrated using the same programming logic following a gradually design process: read titles and footnotes from Statistical Analysis Plan (SAP) shells, extract data from an XML file and validate a type of ISO 8601 format of date and time. Some useful supporting tools and websites are also introduced.

INTRODUCTION

Regular expression (regex [1]) is standard technique for text processing which is sequence of characters to match patterns in text. You may think regex is so weird and totally out of your life. But the truth is that: you may use them almost everyday! When you use asterisk wildcard "*.sas" to search all SAS files in a folder, you are using regex! One of its flavors, Perl regular expression (PRX), is considered the most popular one and SAS also implement PRX since SAS 9.0 by series of PRX functions.

Here is one of the most famous jokes about regex [2]:

> Some people, when confronted with a problem, think "I know, I'll use regular expression." Now they have two problems.

The original meaning of this joke may serve as caution sign for overuse of regex. But in SAS community, the situation is reversed: few programmers use regex so the itself may be an obstacle to be confronted. Some SAS programmers may feel uncomfortable when reading such regex

```
[\w\d!#$%&'*+-/=?^`{|}~]+(\.[\w\d!#$%&'*+-/=?^`{|}~]+)*
```

Don't be scared! Even the people who wrote it will also find it hard to get the meaning without reviewing the documentation. One of the key points of this paper is that regex itself is not suitable for reading but writing! You write it bit by bit, step by step, sequence by sequence, test it, fix it, and finally document it (and nerve just read it!). Furthermore, even the regex listed above is not the most difficult part: regex is so widely used that you can almost get the workable return for a regex to fit your needs from Google and various regex books, such as *Regular Expression Recipes* by Nathan A. Good [3], *Regular Expressions Cookbook* by Jan Goyvaerts and Steven Levithan [4]. The most difficult and important part is how to use it in you daily life to resolve problems and keep away of any fear. In SAS, then the question is how to implement it in SAS data steps as daily basis?

This paper presents a simple while robust programming framework for users new to this field through real life examples step by step:

1) scratch the patterns using meta-characters (with online supporting tools!),

2) validate the expression,

3) locate the patterns, and

4) extract the data needed with an explicit output statement for further processing.

Such framework makes programmers more concentrated on the pattern needed to match based on their own domain knowledge (or Google and other multiple sources), while left others to the almost same programming blocks. Expansibility wise, if users want to exact more data with different patterns, just repeat such programming blocks with different scratched meta-characters in step 1); also, even for the complicated pattern needed matched, it doesn't need any change for the building blocks but the increasing complication of the meta-characters accordingly.

Three real life examples are demonstrated using the same programming logic following a gradually design process: read titles and footnotes from Statistical Analysis Plan (SAP) shells, extract data from an XML file and validate a type of ISO 8601 format of date and time. Some useful supporting tools and websites are also introduced.

A. PREPARATION

To best clone the real life scenarios, all example files (source data) used for this paper are put on my own Github page:

https://github.com/Jiangtang/SESUG2012

and it can be easily reached by SAS with internet connections by submitting the following statement:

```
filename SESUG URL "https://raw.github.com/Jiangtang/SESUG2012/master/";
```

For the SAP shell (*SESUG2012_SAP_Shell_dummy_0.doc*), three actions need take your consideration:

1. Delete all Table of Contents (TOC) if available.

   In Microsoft Office Word, on **View** menu, select **Outlining** option; in **Outlining** toolbar, click **Go to TOC** button image, then the TOC is selected, just **Delete** it (then we get *SESUG2012_SAP_Shell_dummy.doc*).

   This action is optional but still highly recommended. If don't, you should programmatically delete all the duplicate titles both in TOC and body text.

2. Delete all table frames

   The table shells will be converted to numerous trivial special symbols into a text file. Again, it can be programmatically deleted in the later process, it is best to delete in advance.

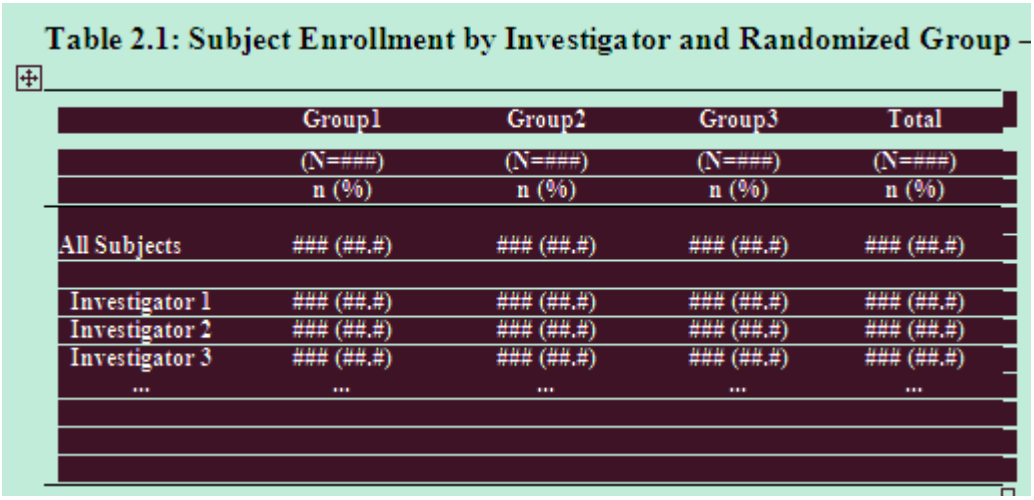


Table 2.1: Subject Enrollment by Investigator and Randomized Group –

	Group1	Group2	Group3	Total
	(N=###)	(N=###)	(N=###)	(N=###)
	n (%)	n (%)	n (%)	n (%)
All Subjects	### (##.#)	### (##.#)	### (##.#)	### (##.#)
Investigator 1	### (##.#)	### (##.#)	### (##.#)	### (##.#)
Investigator 2	### (##.#)	### (##.#)	### (##.#)	### (##.#)
Investigator 3	### (##.#)	### (##.#)	### (##.#)	### (##.#)
...	...	...	...	...

Piece of VBA codes is available in my Github (*VBA_delete_table_frame.txt* supplied by Shuai Han) to help you delete even hundreds of table frames in seconds.

3. Save the Word version of SAP shell to a text file

This is the much-do action since text files are easily to process for SAS. So finally, *SESUG2012_SAP_Shell_dummy.txt* will serve as input file for the following demo instead of the Word version itself.

XML file *xml_ex1.xml* will be used in example of XML processing.

B. READ TITLES AND FOOTNOTES FROM SAP SHELL

In pharmaceutical section, clinical SAS programmers produce Tables, Figures, and Listings (TLFs) according to Statistical Analysis Plan (SAP) where TLFs shells with titles and footnotes are imbedded. Usually, such titles and footnotes (should be considered as metadata for the TLFs submission [5]) change more or less during the clinical analysis life cycle, so it is not the practical and smart way just to change the titles and footnotes in TLFs programs according to the change in SAP.

Xianchan Cui (and etc. 2009[6]) presents a automatic process to create and update titles and footnotes from SAP directly while not in details. Daniel Huang and Lois Lynn([7]) supply the full code to read these titles and footnotes but some key tag words such as "TOC", "SHELL", "END" needed to insert to SAP manually for program detection. In this section, a PRX approach will be presented while no specific tags needed in the original SAP file.

READ ALL DATA INTO SAS

First, we just pull all the data from the SAP file:

```
filename SESUG URL " https://raw.github.com/Jiangtang/SESUG2012/master/";
data sap;
      infile SESUG(SESUG2012_SAP_Shell_dummy.txt);
      input;
      line=left(_infile_);
      if line = '' then delete;
run;
```

STEP 1: SCRATCH PATTERN

Then we start with step 1, scratch the pattern needed. Just open the file *SESUG2012_SAP_Shell_dummy_0.doc* and could note that all table titles begin with "Table 2.1:", "Table 2.2:" and the like, and figures and listing the same. To practice how to get familiar with the meta-characters, an online site for regex testing is strongly recommended:

http://osteele.com/tools/rework/

You can take a look at the following picture for the usage, simple and straightforward:

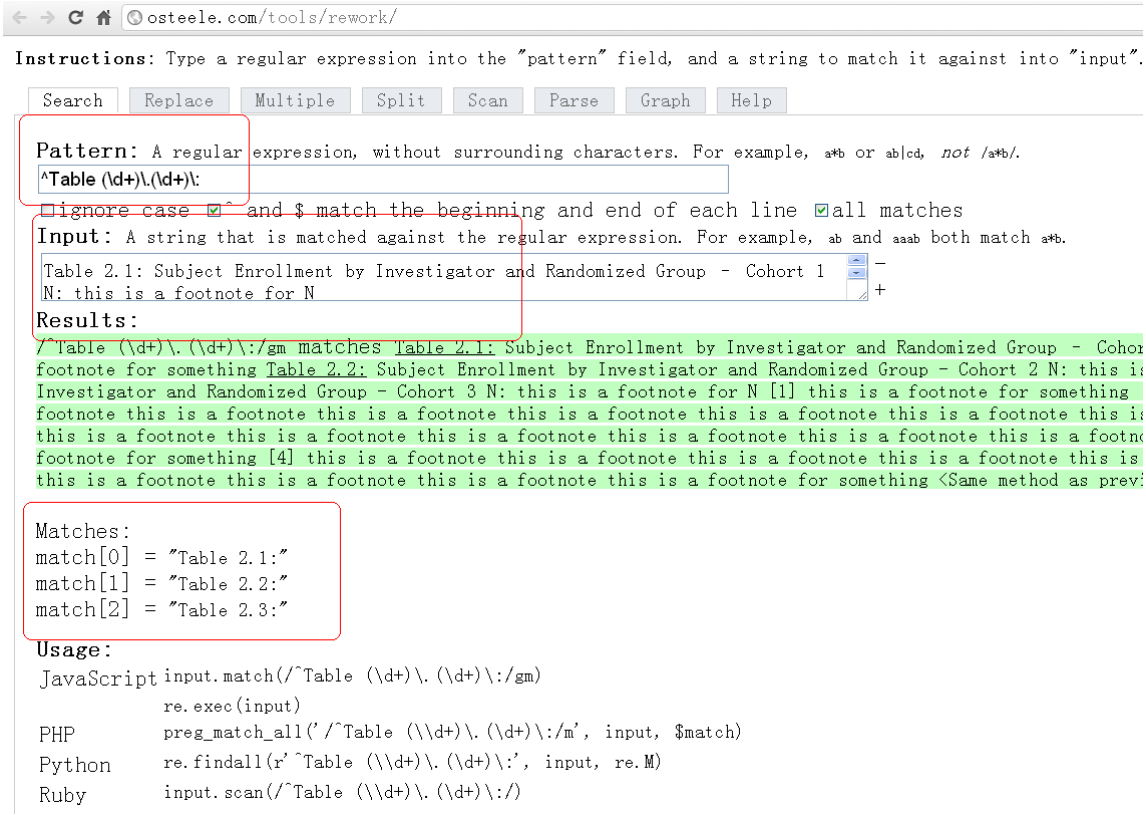
1) type the regex in the "Pattern" form; if it is not a valid regex, an error will pop up.
2) copy and paste a sample source file in the "Input" form
3) then you can see the returned matched patterns and some examples on how to imbed it in some programming languages.

In this case, the pattern we need for tables' titles is

```
^Table (\d+)\.(\d+)\:
```

This pattern can be read as

begin (^) with Table then space then digits(\d+) then dot (\.) then digits then colon(\:), like **Table 2.1:**



And all the same for figures, listing and some special symbols. In SAS, a regex should be put in paired "//" and we just assign variables and parse them (and these codes will be in parts of data steps):

```
Table="/^Table (\d+)\.(\d+)\:/";
Figure="/^Figure (\d+)\.(\d+)\:/";
List="/^Listing (\d+)\.(\d+)\:/";
Symbol="/^\<|\>$/";

queTable  = prxparse(Table);
queFigure = prxparse(Figure);
queList   = prxparse(List);
queSymbol = prxparse(Symbol);
```

PRXparse is the first PRX function we need to know (and nothing special, right?).

STEP 2: VALIDATE THE REGEX

Actually the regex we scratched above already took some validation by the online parser. Here we just routinely add a SAS validation process:

```
if missing(queTable) then do;
   putlog "ERROR: Invalid regexp" Table;
   stop;
end;
```

And all the same for others.

STEP 3: LOCATE THE PATTERNS

To locate the position of the pattern matched, we need a function like index(). In SAS PRX function series, the corresponding one is prxmatch(), which returns the position at which the pattern is found:

```
queTableN  = prxmatch(queTable ,line);
```

STEP 4: PUT THEM TOGOTHER: EXTRACT THE DATA NEEDED WITH AN EXPLICIT OUTPUT STATEMENT

The last step is just to put all the programming blocks together and then use a explicit output statement to get all the data needed for the later processing:

```
data hf0 (keep = line newline Table_Type tag tagN) ;
      set sap;

      if _n_=1 then do;
            retain queTable;

            Table="/^Table (\d+)\.(\d+)\:/";

            queTable  = prxparse(Table);

            if missing(queTable) then do;
               putlog "ERROR: Invalid regexp" Table;
               stop;
            end;
      end;

      queTableN  = prxmatch(queTable ,line);

      if queTableN > 0  then do ;
            call PRXsubstr (queTable, line, position, length);
            newline = substr(line, position, length-1);
            Table_Type = 'T';
            tag="title_id";
            tagN=1;
            output;

            newline = substr(line, position+length+1);
            Table_Type = '';
            tag="title";
            tagN=2;
            output;
      end;
run;
```

For the full code, you can find in

http://jiangtanghu.com/docs/en/Orlando/read_SAP.sas

and you can run it by

```
filename readSAP url " https://raw.github.com/Jiangtang/SESUG2012/master/
/read_SAP.sas";
%include readSAP;
```

C. READ XML FILE

Since the introduction of define.xml as replacement of define.pdf for clinical submitting, the industry standard XML file also come into clinical SAS programmers' toolbox. Here just a simple example to use PRX to processing XML file.

READ ALL DATA INTO SAS

again, pull all the data from the XML file:

```
data xml;
      infile SESUG(xml_ex1.xml);
      input;

      text=left(_infile_);
      if text = '' then delete;
run;
```

FIRST ROUND

Just follow the procedures above, try the following codes:

```
filename readXML url "https://raw.github.com/Jiangtang/SESUG2012/master/
/read_XML_1.sas";
%include readXML;
```

```
 1 data import2 (keep = text );
 2       set xml;
 3
 4        if _n_=1 then do;
 5              retain quename ;
 6              data="/^\<text\> /";
 7
 8              quename=prxparse(data);
 9
10              if missing(quename) then do;
11                  putlog "ERROR: Invalid regexp" data;
12                  stop;
13              end;
14        end;
15
16        quenamen=prxmatch(quename,text);
17
18        if quenamen > 0  then do ;
19              output;
20        end;
21 run;
```

And you can see in the output file, XML tags still need to remove.

SECOND ROUND

Then we just add some regex to remove the tags while keep the programming blocks consistent:

```
filename readXML url "https://raw.github.com/Jiangtang/SESUG2012/master/
/read_XML_2.sas";
%include readXML;
```

Note “s” in "s/<.*?>//" indicates a substitute. In this case, it changes all the tags to null (aka, remove all the tags). A SAS PRX call routine **call PRXchange** performs this pattern-matching replacement.

```
data import3 (keep = text );
     set xml;

      if _n_=1 then do;
           retain quename rx1;
           data="/^\<text\> /";
           tag="s/<.*?>//";

           quename=prxparse(data);
           rx1=prxparse(tag);

           if missing(quename) then do;
               putlog "ERROR: Invalid regexp" data;
               stop;
           end;

           if missing(rx1) then do;
               putlog "ERROR: Invalid regexp" tag;
               stop;
           end;
      end;

      quenamen=prxmatch(quename,text);

      if quenamen > 0  then do ;
           call prxchange(rx1,99,text);
           output;
      end;
run;
```

D. VALIDATE ISO 8601 DATE AND TIME

The following piece of codes validates a kind of ISO 8601 formats: calendar date and time, with optional microseconds and time zone (such as `2008-08-30T01:45:36.123Z`):

```
data ISO ;
      input time $40.;

    if _n_=1 then do;
      retain ISO_re;

        ISO="/^(-?(?:[1-9][0-9]*)?[0-9]{4})-(1[0-2]|0[1-9])-(3[0-
1]|0[1-9]|[1-2][0-9])T(2[0-3]|[0-1][0-9]):([0-5][0-9]):([0-5][0-
9])(\.[0-9]+)?(Z|[+-](?:2[0-3]|[0-1][0-9]):[0-5][0-9])?$/";
      ISO_re  = prxparse(ISO);
```

```
      if missing(ISO_re) then do;
         putlog "ERROR: Invalid regexp" time;
         stop;
      end;
   end;

   ISO_re_n  = prxmatch(ISO_re ,compress(time));

   if ISO_re_n > 0  then output;
datalines;
2008-08-30T01:45:36.123Z
2008-08-30T01:45:36
2008-08-30T01
2009-05-19 14:39:22+0600
;
```

You may still focus on this same programing blocks and don't scared of this long PRX(you can get it online). Actually it is only a combination of few simple expressions:

- First part, `^(-?(?:[1-9][0-9]*)?[0-9]{4})-(1[0-2]|0[1-9])-(3[0-1]|0[1-9]|[1-2][0-9])`, the calendar date, like 2008-09-21
- Second part, `T(2[0-3]|[0-1][0-9]):([0-5][0-9]):([0-5][0-9])(\.[0-9]+)?,` the minute
- Third part, `(Z|[+-](?:2[0-3]|[0-1][0-9]):[0-5][0-9])?$,` the time zone

CONCLUSION

Perl regular expression (PRX) is extremely powerful tool to process text files while its syntax is extremely complicated at first glance for programmers new to this field. This paper encourages SAS users utilize some useful tool to scratch regex step by step and then use it to solve real life problems. For quick dive into regex for relative newcomer, a programming framework also supplied with simple programming building blocks which SAS users can fill in. More complicated examples and techniques will present in sequential posts.

REFERENCES

[1] Regular Expression, http://en.wikipedia.org/wiki/Regular_expression

[2] Jeffrey Friedl. Source of the famous "Now you have two problems" quote.

http://regex.info/blog/2006-09-15/247

[3] Nathan A. Good. *Regular Expression Recipes*. Apress, 2005

[4] Jan Goyvaerts and Steven Levithan.*Regular Expressions Cookbook*. O'Reilly, 2009

[5] Stephen Hunt and Brian Fairfield-Carter. *Using TFL Metadata to Populate Titles & Footnotes in SAS Programs for Clinical Trials.*

[6] Xiangchen (Bob) Cui, Shan Chen, and Mei Wu. *Automate the Process of Creating and Updating Titles and Footnotes of TLG for a Clinical Study Report from a Table Shell Document*.

[7] Daniel Huang and Lois Lynn. *Read Titles and Footnotes from a Table Shell into a SAS Dataset*.

ACKNOWLEDGMENTS

Very appreciate the nice discussion with Dr. Huiling Xiong of sanofi pasteur Biostatistics platform on how to process SAP files.

CONTACT INFORMATION

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

Jiangtang Hu
Life Sciences Consultant
d-Wise Technologies, Inc.
4020 Westchase Blvd, Raleigh, NC, 27607
www.d-Wise.com

(O) 919-334-6092
(C) 919-801-9659
(F): 888-563-0931
(O) jhu@d-Wise.com
(P) jiangtanghu@gmail.com
http://jiangtanghu.com/