PharmaSUG 2016 - Paper AD08

The Power of Perl Regular Expressions: Processing Dataset-XML documents back to SAS® Data Sets

Joseph Hinson, inVentiv Health, Princeton, NJ, USA

ABSTRACT

The seemingly intimidating syntax notwithstanding, Perl Regular Expressions (PRE) are so powerful they can overcome and parse the most complex non-uniform textual data. A "regular expression" is a string of characters that define a particular pattern of data, and is used for matching, searching, and replacing text. In SAS®, PRE is implemented via the PRX functions such as PRXPARSE, PRXMATCH, PRXCHANGE, and PRXPOSN. Consider a situation where a date has to be extracted from data and the date can be present in a wide variety of forms: "Jan 1, 1960", "January 1st, 1960", "1st January, 1960", "1/1/60", "01:01:1960". With PRE, all the above forms of dates can be deciphered with the same, single PRXPARSE code, the way the human eyes can quickly glance through all those date formats and instantly know they are always referring to the same "first day of the first month of the year 1960". Thus it comes as no surprise that the XML data format, with its disparate forms of tags and elements, can easily be processed using PRE techniques. With PRE, all the extraneous non-SDTM text can be "ignored" as the records of XML data are read. SAS® 5 XPT file format is scheduled to be replaced, according to FDA, by the new CDISC Dataset-XML standard, and various techniques are currently being developed for processing the XML data structure. The present paper will show how to easily convert an SDTM data in XML format back to a regular SAS® data set, using the PRE technique.

INTRODUCTION

Programmers are often faced with the daunting task of extracting uniform data from a set of non-uniform text in free-text fields of documents. In clinical trials, subjects might not easily recall historical dates for starting certain concomitant medication or when particular symptoms first appeared, as often is the case with Medical History and Concomitant Medication data. Thus one would obtain from such subjects vaguely described recollection of partial information in non-standard formats. In other situations, a desired bit of information might be buried within an irregular set of text containing mostly irrelevant information. The regular expression technique was designed for all such situations. A "regular expression" is a code made up of a string of characters used for identifying patterns in text strings especially in unstructured data. That code is designed to recognize a variety of features, either by specificity or by generalization. For instance, a regular expression can recognize a US telephone number, whether written as nnn-nnn-nnnn, (nnn)-nnn-nnnn, nnn.nnn.nnnn, or nnnnnnnnnn. Furthermore, it can recognize those telephone numbers regardless of where they occur within an irregular text. In many instances, one needs to extract specific pieces of data within undefined text.

The present paper will focus on the extensible markup language, XML, which presents information as a series of "elements". An XML element is made up of pieces of data and their attributes enclosed by special tags:

```
<ItemData Value="Psychiatric disorders" ItemOID="IT.AEBODSYS"/>
```

In the XML element above, one might want to extract just the two pieces of information:

"AEBODSYS" and **"Psychiatric disorders"**

A regular expression code can be designed to ignore all the characters preceding **Psychiatric** and those between **disorders"** and **AEBODSYS**, as well as the closing segment **"/>** .

The US Food and Drug Administration (FDA) is planning to have the SAS® XPORT files replaced with CDISC's Dataset-XML documents. This means there has to be a way to translate the XML format back to SAS® data set records for data analysis and reporting. As this paper will show, Perl Regular Expressions are very suitable for such conversion tasks.

A TYPICAL REGULAR EXPRESSION CODE:

At first glance, a regular expression code can be quite intimidating:

```
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
```

But once the individual commands ("metacharacters") are known, the expression becomes quite straightforward:

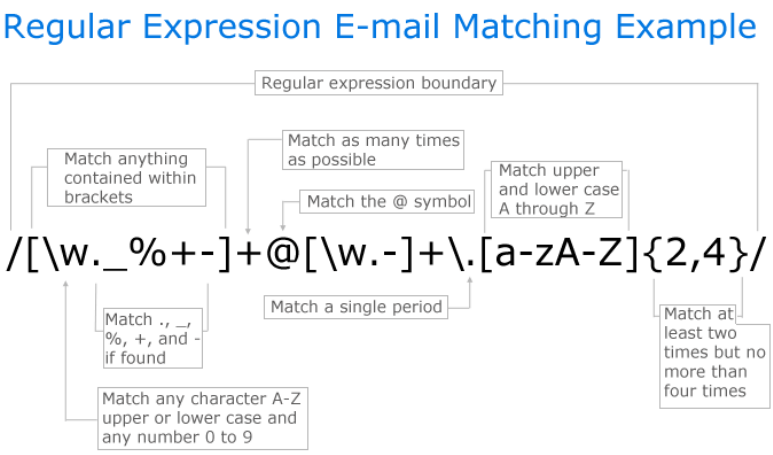

Figure 1. PRE Code for Identifying An E-mail Adddress

PERL REGULAR EXPRESSION METACHARACTERS

Metacharacter	Description	Examples
*	Matches the previous subexpression zero or more times	cat* matches "cat", "cats", "catanddog" c(at)* matches "c", "cat", and "catatat"
+	Matches the previous subexpression one or more times	\d+ matches one or more digits
?	Matches the previous subexpression zero or one times	hello? matches "hell" and "hello"
. (period)	Matches exactly one character	r.n matches "ron", "run", and "ran"
\d	Matches a digit 0 to 9	\d\d\d matches any three digit number
\D	Matches a non-digit	\D\D matches "xx", "ab" and "%%"
^	Matches the beginning of the string	^cat matches "cat" and "cats" but not "the cat"
$	Matches the end of a string	cat$ matches "the cat" but not "cat in the hat"
[xyz]	Matches any one of the characters in the square brackets	ca[tr] matches "cat" and "car"
[a-e]	Matches the letters a to e	[a-e]\D+ matches "adam", "edam" and "car"
[a-eA-E]	Matches the letter a to e or A to E	[a-eA-E]\w+ matches "Adam", "edam" and "B13"
{n}	Matches the previous subexpression n times	\d{5} matches any 5-digit number and is equivalent to \d\d\d\d\d
{n,}	Matches the previous subexpression n or more times	\w{3,} matches "cat" "_NULL_" and is equivalent to \w\w\w+
{n,m}	Matches the previous subexpression n or more times, but no more than m	\w{3,5} matches "abc" "abcd" and "abcde"
[^abcxyz]	Matches any characters except abcxyz	[^8]\d\d matches "123" and "999" but not "800"
x|y	Matches x or y	c(a|o)t matches "cat" and "cot"
\s	Matches a white space character, including a space or a tab,	\d+\s+\d+ matches one or more digits followed by one or more spaces, followed by one or more digits such as "123•••4" Note: •=space
\w	Matches any word character (upper- and lowercase letters, blank and underscore)	\w\w\w matches any three word characters
\(	Matches the character (	\(\d\d\d\) matches three digits in parentheses such as "(123)"
\)	Matches the character)	\(\d\d\d\) matches three digits in parentheses such as "(123)"
\\	Matches the character \	\D•\\•|D matches "the \ character" Note: •=space
\1	Matches the previous capture buffer and is called a back reference.	(\d\D\d)\1 matches "9a99a9" but not "9a97b7" (.)\1 matches any two repeated characters

Table 1. PRE Metacharacters with Descriptions and Examples

THE SAS® VERSION OF PERL REGULAR EXPRESSIONS

In SAS®, a Perl Regular Expression code containing metacharacters is created with the PRXPARSE function. Other PRX functions available for regular expression processing are typically PRXMATCH, PRXCHANGE, PRXPOSN, PRXPAREN, PRXNEXT, and PRXSUBSTR.

SYNTAX:

(a) The forward slashes (/……../) are the default Perl delimiters for the PRXPARSE code.

(b) The single or double quote within the parentheses (“/……../”) are valid SAS® syntax.

(c) The back slash, (\) is used to distinguish special meaning of characters (also called “metacharacters”) from their regular use.
For example, “\d” denotes “any digit”, rather than the letter “d”.

(d) Within the PRXPARSE code, certain matched text to be extracted are enclose in parenthesis.
For example, to extract the 4-digit year "2004" from a text containing a date:
"Please note that [2004/04/12] is the date of this article."
the regular expression code would be:
PRXPARSE("/.+(\d\d\d\d).+/")
The quotes and forward slashes surround the regular expression: "/_____/"
The metacharacters ".+" mean "any character including whitespace" " repeated one or more times".
The parentheses (_____) isolate the pattern to be extracted: \d\d\d\d

OTHER EXAMPLES:

`PRXPARSE("/\d\d\d/")`	matches any three digits in a row
`PRXPARSE("/\d+/")`	matches one or more digits
`PRXPARSE("/\w\w\w* /")`	matches any word with two or more characters followed by a space
`PRXPARSE("/\w\w? +/")`	matches one or two word characters such as x, xy, or _X followed by one or more spaces
`PRXPARSE("/(\w\w) +(\d) +/")`	matches two word characters, followed by one or more spaces, followed by a single digit, followed by one or more spaces. Note that the expression for the two word characters (\w\w) is placed in parentheses. Using the parentheses in this way creates what is called a capture buffer. The second set of parentheses (around the \d) represent the second capture buffer. Several of the Perl regular expression functions can make use of these capture buffers to extract and/or replace specific portions of a string. For example, the location of the two word characters or the single digit can be obtained using the PRXPOSN function.

THE SAS® REGULAR EXPRESSIONS FOR READING DATASET-XML ELEMENTS

(1) Converting data:ItemGroupDataSeq to AESEQ value from the element

```
- <ItemGroupData ItemGroupOID="IG.AE" data:ItemGroupDataSeq="1">
```

Only the number 1 in `"1"` will be captured.
The regular expression to do the capturing is:

```
prxparse ('/(.+)data:ItemGroupDataSeq\=\"(\d*)/');
```

INTERPRETATION:

(.+) The period symbol is a metacharacter for "Any character including spaces". The plus sign is a metacharacter for "must be present at least once". So, at least one character must begin the text to parse.

data:ItemGroupDataSeq This exact text must follow the above. That is our "landmark" for locating the desired item to extract ("capture buffer").

\= The equal sign must follow the above. The back slash preceding "=" implies the equal sign is not a metacharacter.

\" A double quote must follow the above. The back slash means it is a real double-quote symbol.

(\d*) This is the capture buffer that contains the item we want to extract. The back slash

preceding the letter 'd' makes it a metacharacter, meaning "any digit". The asterisk symbol is a metacharacter meaning we must have 0 or more digits. The value would be put into AESEQ.

Any other text or characters that do not match all the above would be ignored.

(2) Converting ItemOID="IT.AETERM" Value="AGITATED" into SDTM variable name and value

The XML element to process is: `<ItemData ItemOID="IT.AETERM" Value="AGITATED" />`

Two items will be captured: `AETERM` and `AGITATED` :

The regular expression to do the capturing is:

```
prxparse ('/\<ItemData ItemOID\=\"IT\.(.+)\" Value\=\"(.+|)\" \/\>/')
```

INTERPRETATION:

`\<` The back slash before "<" means the symbol is actually "less than" and not a metacharacter. It would be present as an XML opening tag.

`ItemData ItemOID` The exact text must be present and in the indicated position.

`\=` The back slash before "=" indicates it is the true equal symbol and not a metacharacter.

`\"` The back slash before the double quote means it is indeed a double quote symbol and not a metacharacter.

`IT` The exact string of letters must be present in the specified position.

`\.` The back slash before the period symbol make it a true period.

`(.+)` This is the first capture buffer. It would contain the text for the SDTM variable name. The metacharacter "." Means "any character" including space. The metacharacter plus sign means there should be at least one such character. This buffer would capture **AETERM** from `ItemOID="IT.AETERM"`

`Value` This exact text must be present in the specified position.

`(.+|)` This is the second capture buffer. It would contain the text for the SDTM variable value. The metacharacter "." Means any character including space. The metacharacter "+" means there should be at least one such character. The metacharacter "|" means 'OR'. So, "| " means "or a space". This buffer would capture **AGITATED** from `Value="AGITATED.`

`\/` The back slash before the "/" means it is a real forward slash and not a metacharacter. It would be part of the XML close tag.

`\>` The back slash before the ">" means it is a real greater-than symbol and not a metacharacter. It should be part of the XML end tag.

CONVERTING DATASET-XML TO SDTM SAS® DATA SET

(3) An entire line of XML element from the dataset-xml document is read into a single variable, *element*.

The header elements are included, although the second header element has been shortened for clarity. The full second header element should be:

```
- <ODM xmlns="http://www.cdisc.org/ns/odm/v1.3"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:data="http://www.cdisc.org/ns/Dataset-XML/v1.0" FileType="Snapshot"
ODMVersion="1.3.2" data:DatasetXMLVersion="1.0.0" FileOID="www.cdisc.org.Studycdisc01-
Define-XML_2.0.0(IG.AE)" PriorFileOID="www.cdisc.org.Studycdisc01-Define-XML_2.0.0"
Originator="SAS Programmer John Doe" CreationDateTime="2015-12-01T16:05:35">
```

In any case, whether full or truncated, PRE will ignore this second header element as well, like all elements in the document with non-SDTM data.

```
options nocardimage;
data xmlsource;
       infile datalines dsd truncover;
       length element $300;
       input element $ 1-300;
datalines;
<?xml version="1.0" encoding="UTF-8" ?>
- <ODM xmlns="http://www.cdisc.org/ns/odm/v1.3" CreationDateTime="2015-12-01T16:05:35">
- <ClinicalData>
- <ItemGroupData ItemGroupOID="IG.AE" data:ItemGroupDataSeq="1">
  <ItemData ItemOID="IT.STUDYID" Value="CDISC01" />
  <ItemData ItemOID="IT.DOMAIN" Value="AE" />
  <ItemData ItemOID="IT.USUBJID" Value="CDISC01.100008" />
  <ItemData ItemOID="IT.AESEQ" Value="1" />
  <ItemData ItemOID="IT.AESPID" Value="1" />
  <ItemData ItemOID="IT.AETERM" Value="AGITATED" />
  <ItemData ItemOID="IT.AEMODIFY" Value="AGITATION" />
  <ItemData ItemOID="IT.AEDECOD" Value="Agitation" />
  <ItemData ItemOID="IT.AEBODSYS" Value="Psychiatric disorders" />
  <ItemData ItemOID="IT.AESEV" Value="MILD" />
  <ItemData ItemOID="IT.AESER" Value="N" />
  <ItemData ItemOID="IT.AEACN" Value="DOSE NOT CHANGED" />
  <ItemData ItemOID="IT.AEREL" Value="POSSIBLY RELATED" />
  <ItemData ItemOID="IT.AESTDTC" Value="2003-05" />
  <ItemData ItemOID="IT.AEENDTC" Value="" />
  <ItemData ItemOID="IT.AESTDY" Value="3" />
  <ItemData ItemOID="IT.AEENDY" Value="." />
  <ItemData ItemOID="IT.AEENRF" Value="AFTER" />
  </ItemGroupData>
- <ItemGroupData ItemGroupOID="IG.AE" data:ItemGroupDataSeq="2">
  <ItemData ItemOID="IT.STUDYID" Value="CDISC01" />
  <ItemData ItemOID="IT.DOMAIN" Value="AE" />
  <ItemData ItemOID="IT.USUBJID" Value="CDISC01.100008" />
  <ItemData ItemOID="IT.AESEQ" Value="2" />
  <ItemData ItemOID="IT.AESPID" Value="2" />
  <ItemData ItemOID="IT.AETERM" Value="ANXIETY" />
  <ItemData ItemOID="IT.AEMODIFY" Value="" />
  <ItemData ItemOID="IT.AEDECOD" Value="Anxiety" />
  <ItemData ItemOID="IT.AEBODSYS" Value="Psychiatric disorders" />
  <ItemData ItemOID="IT.AESEV" Value="MODERATE" />
  <ItemData ItemOID="IT.AESER" Value="N" />
  <ItemData ItemOID="IT.AEACN" Value="DOSE NOT CHANGED" />
  <ItemData ItemOID="IT.AEREL" Value="POSSIBLY RELATED" />
  <ItemData ItemOID="IT.AESTDTC" Value="2003-05-13" />
  <ItemData ItemOID="IT.AEENDTC" Value="" />
  <ItemData ItemOID="IT.AESTDY" Value="15" />
  <ItemData ItemOID="IT.AEENDY" Value="." />
  <ItemData ItemOID="IT.AEENRF" Value="AFTER" />
  </ItemGroupData>
;
run;
```

(4) Using PRE, the elements are parsed into SDTM variable-value pairs.

```
data xmlextract(keep=vtext vdata seq);
        length vtext $8 vdata $100;
        retain seq;
        set xmlsource;
        test=index(element,'ItemGroupDataSeq')>0;
        if test eq 1 then do;
        regex1=prxparse ('/(.+)data:ItemGroupDataSeq\=\"(\d*)/');
        rx1=prxmatch(regex1,element);
        seq=prxposn(regex1, 2, element); end;
        regex2=prxparse ('/\<ItemData ItemOID\=\"IT\.(.+)\" Value\=\"(.+|)\" \/\>/') ;
        rx2=prxmatch(regex2,element);
        vtext=prxposn(regex2,1, element);
        vdata=prxposn(regex2,2, element);
        if not missing(vtext) then output;
run;
```

SDTM VARIABLE-VALUE PAIRS:

	vtext	vdata	seq
1	STUDYID	CDISC01	1
2	DOMAIN	AE	1
3	USUBJID	CDISC01.100008	1
4	AESEQ	1	1
5	AESPID	1	1
6	AETERM	AGITATED	1
7	AEMODIFY	AGITATION	1
8	AEDECOD	Agitation	1
9	AEBODSYS	Psychiatric disorders	1
10	AESEV	MILD	1
11	AESER	N	1
12	AEACN	DOSE NOT CHANGED	1
13	AEREL	POSSIBLY RELATED	1
14	AESTDTC	2003-05	1
15	AEENDTC		1
16	AESTDY	3	1
17	AEENDY	.	1
18	AEENRF	AFTER	1
19	STUDYID	CDISC01	2
20	DOMAIN	AE	2
21	USUBJID	CDISC01.100008	2
22	AESEQ	2	2
23	AESPID	2	2
24	AETERM	ANXIETY	2
25	AEMODIFY		2
26	AEDECOD	Anxiety	2
27	AEBODSYS	Psychiatric disorders	2
28	AESEV	MODERATE	2
29	AESER	N	2
30	AEACN	DOSE NOT CHANGED	2
31	AEREL	POSSIBLY RELATED	2
32	AESTDTC	2003-05-13	2
33	AEENDTC		2
34	AESTDY	15	2
35	AEENDY	.	2
36	AEENRF	AFTER	2

Output 1. Output from Parsing Dataset-XML Elements with PRE

(1) The variable-value pairs are transposed into the wide SDTM table format:

```
libname dsout "C:\Users\admin\Desktop\SASoutputs\";

proc transpose data=xmlextract out=dsout.aefromxml(drop=seq _NAME_);
        by seq;
        id vtext;
        var vdata;
run;
```

SDTM OUTPUT

The two observations embedded in the dataset-XML are recomposed into an SDTM table AE, as shown below:

	STUDYID	DOMAIN	USUBJID	AESEQ	AESPID	AETERM	AEMODIFY	AEDECOD	AEBODSYS
1	CDISC01	AE	CDISC01.100008	1	1	AGITATED	AGITATION	Agitation	Psychiatric disorders
2	CDISC01	AE	CDISC01.100008	2	2	ANXIETY		Anxiety	Psychiatric disorders

AESEV	AESER	AEACN	AEREL	AESTDTC	AEENDTC	AESTDY	AEENDY	AEENRF
MILD	N	DOSE NOT CHANGED	POSSIBLY RELATED	2003-05		3	.	AFTER
MODERATE	N	DOSE NOT CHANGED	POSSIBLY RELATED	2003-05-13		15	.	AFTER

Output 2. Output from transposing SDTM Variable-Value Pairs

CONCLUSION

As shown in this paper, the Perl Regular Expression technique can be a powerful tool for converting Dataset-XML elements into SAS® data set records. This is in addition to its application in other challenging clinical programming tasks such as extracting start dates from Medical History, Concomitant Medication, and Adverse Event data.

ACKNOWLEDGMENTS

This author is very grateful to his manager, mentor, and teacher Brian Shilling, for his constant guidance and ready assistance.

RECOMMENDED READING

PharmaSUG 2013 Paper BB02 “The Baker Street Irregulars Investigate: Perl Regular Expressions and CDISC” by Peter Eberhardt and Wei Liu, SAS Research and Development, Beijing, China

http://www.lexjansen.com/pharmasug/2013/BB/PharmaSUG-2013-BB02.pdf

SUGI 29 Paper 265-29 “An Introduction to Perl Regular Expressions in SAS® 9”
by Ron Cody, Robert Wood Johnson Medical School, Piscataway, NJ.

http://www2.sas.com/proceedings/sugi29/265-29.pdf

SUGI 29 Paper 043-29 “An Introduction to Regular Expressions with Examples from Clinical Data”
by Richard Pless, Ovation Research Group, Highland Park, IL

http://www2.sas.com/proceedings/sugi29/043-29.pdf

SAS Perl Regular Expressions Tip Sheet

http://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

CONTACT INFORMATION

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

Joseph W. Hinson, PhD
inVentiv Health
202 Carnegie Center, Suite 200
Princeton, NJ, 08540
1-609-282-1615
joehinson@outlook.com