PharmaSUG 2018 - Paper AD21

ADaM Define.xml v2.0 Validation – The Perl Way in SAS

Yuxin (Ellen) Jiang, Alkermes Inc.

ABSTRACT

In the latest version of the Study Data Technical Conformance Guide, the FDA indicated that Define-XML version 2.0 is the preferred version of the Define-XML format, which serves as a metadata guide to the data submitted for a clinical trial. There are various methods that can be used to validate the define.xml, such as Pinnacle 21 Community and SAS® Clinical Standard Toolkit. However, these tools do not cover all the checks for validating the contents of the define file. Additional SAS based methods have been proposed which involve a step to get define.xml contents using XMLMap software and some manipulation steps to transform the define file into SAS datasets in a desired format. The paper discusses using Perl Regular Expressions pattern matching functions to directly read the content from Define.xml to SAS macros or SAS datasets without using XMLMap, and then performing validation checks in SAS. By keeping all the programs in SAS, this approach can help streamline the validation process of Define.xml.

INTRODUCTION

Define.xml is the metadata file sent along with every clinical study in each regulatory submission, which specifies what datasets, variables, variable attributes, controlled terms, and other metadata were used in each study. Beginning in 2018, Define-XMLv2.0 is recommended by FDA as the standard for study data submissions. As shown below, it has well defined schema in terms of constraints on the structure and content of each type of elements including:

- Table of Contents: Dataset, Description, Structure, Purpose, Keys, Location
- Data Definition Tables: Provides the variable level attributes and definitions for each variable
- Controlled Terminology (Code List)
- Value Level Metadata (Value List)
- Computational Algorithms

Perl Regular Expressions (PRX) were added to SAS in Version 9. Based on the consistent schema pattern in Define.xml v2.0, utilizing joined power of PRX and SAS data steps, we can identify different types of metadata elements in Define.xml in compact solutions. The purpose of this paper is to show the key steps used to identify and extract different types of metadata elements via a set of PRX functions (eg. PRXPARSE, PRXMATCH, PRXPOSN) within SAS. Not depending on external software such as XMLMap or Pinnacle 21 to read the Define.xml at the initial steps, this approach keeps all the programs in SAS, which not only helps to streamline the define.xml validation process, but also makes some validation possible beyond what external software can provide.

Sample of Define.xml v2.0 structure:

```
<Description>
<TranslatedText xml:lang="en">Adverse Event Analysis Dataset</TranslatedText>
</Description>
<ItemRef ItemOID="IT.ADAE.STUDYID" OrderNumber="1" Mandatory="Yes" KeySequence="1"/>
<ItemRef ItemOID="IT.ADAE.USUBJID" OrderNumber="2" Mandatory="Yes" KeySequence="2"/>
<ItemRef ItemOID="IT.ADAE.SUBJID" OrderNumber="3" Mandatory="Yes"/>
<ItemRef ItemOID="IT.ADAE.SITEID" OrderNumber="4" Mandatory="Yes"/>
• • •
<ItemDef OID="IT.ADSL.STUDYID" Name="STUDYID" SASFieldName="STUDYID" DataType="text" Length="12">
<Description>
<TranslatedText xml:lang="en">Study Identifier</TranslatedText>
</Description>
• • •
<ItemDef OID="IT.ADAE.ADISCRFL" Name="ADISCRFL" SASFieldName="ADISCRFL" DataType="text" Length="1">
<Description>
<TranslatedText xml:lang="en">AE Leading to Study Discontinuation (R)</TranslatedText>
</Description>
<CodeListRef CodeListOID="CL.YNULL.NY"/>
<def:Origin Type="Derived"/>
</ItemDef>
• • •
</CodeList>
<CodeList OID="CL.ADAE.AEREL" Name="ADAE.AEREL" DataType="text">
<EnumeratedItem CodedValue="DEFINITELY NOT RELATED" OrderNumber="1"/>
<EnumeratedItem CodedValue="PROBABLY NOT RELATED" OrderNumber="2"/>
<EnumeratedItem CodedValue="POSSIBLY RELATED" OrderNumber="3"/>
<EnumeratedItem CodedValue="PROBABLY RELATED" OrderNumber="4"/>
<EnumeratedItem CodedValue="DEFINITELY RELATED" OrderNumber="5"/>
</CodeList>
• • •
<MethodDef OID="MT.ADSL.ENRLDT" Name="CM.ADSL.ENRLDT" Type="Computation">
<Description>
<TranslatedText xml:lang="en">Date part of DM.RFICDTC</TranslatedText>
</Description>
</MethodDef>
```

READ DEFINE.XML CONTENT INTO SAS VARIABLE

The first step here is to read in define.xml into a single SAS variable “xmltext”.

```
data xml;
   infile ‘&path\define.xml' dlm='<' truncover
   input xmltext $2000.;
run;
```

EXTRACTION KEY VARIABLE LIST

In Define.xml v2.0, the key variables are defined in the following syntax:

<ItemRef ItemOID="IT.**ADSL.USUBJID**" OrderNumber="2" Mandatory="Yes" **KeySequence**="**1**"/>

The following PRX functions lead SAS to extract the content in the hierarchical define.xml file into rows and columns in the rectangular SAS datasets.

- The **PRXPARSE** functions were used to compile PRX to identify the Key Variable element defined by patternID1 and patternID2:

```
patternID1 = prxparse('/IT.(AD\w\w+).(\w\w\w+)"/');
patternID2 = prxparse('/(KeySequence=\"\d)/');
```

/ The forward slashis the default Perl delimiters to define the PRX pattern
IT. The exact text string must be present at the specified position

(AD\w\w+) This is the first capture buffer. It would contain the text for the ADaM dataset name. The metacharacter string "\w\w+" matches any word string with at least two characters (upper- and lowercase letters, blank and underscore). The metacharacter plus sign means there should be at least one such character. This buffer would capture "ADSL" from ItemOID="IT.**ADSL**"

. The exact character dot must be present at the specified position

(\w\w\w+) This is the second capture buffer. It would contain the text for the key variable name. The metacharacter string "\w\w\w+" matches any word string with at least three more characters (upper- and lowercase letters, blank and underscore). This buffer would capture "USUBJID" from ItemOID="IT.**ADSL.USUBJID**"

KeySequence= The exact text string must be present at the specified position

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

\d Any single digit number, equivalent to [0-9].

- The **PRXMATCH** and **PRXPOSN** functions were used to locate the patterns and extract the XML text into SAS tables. **PRXMATCH** is used to locate the position in a string, where a PRX match is found. This function returns the first position in a string expression of the pattern described by the regular expression **PRXPARSE**. **PRXPOSN** returns the value from each capture buffer from the position in a string where each PRX match was found.

```
if prxmatch(patternID1, xmltext) then do;
   call prxposn(patternID1, 1, position, length);
   dsn = substr(xmltext, position, length);
   call prxposn(patternID1, 2, position, length);
   var = substr(xmltext, position, length);
end;
if prxmatch(patternID2, xmltext) then do;
   call prxposn(patternID2, 1, position, length);
   keyord =input(substr(substr(xmltext, position, length), length), best.);
end;
```

Output SAS dataset with extracted key information: Dataset name (dsn), Key variable name (var), and the key variable order (keyord) was shown below. In the ADAE dataset, the key list would contain USUBJID, AEDECOD, AETERM, ASTDT. This key list will be called into a macro %chkkey in a later section of this paper to validate if the key list identified unique records in the dataset.

xmltext	dsn	var	keyord
<ItemRef ItemOID="IT.ADAE.ASTDT" OrderNumber="44" Mandatory="No" KeySequence="4" MethodOID="MT.ADAE.ASTDT"/>	ADAE	ASTDT	4
<ItemRef ItemOID="IT.ADAE.AETERM" OrderNumber="52" Mandatory="Yes" KeySequence="3"/>	ADAE	AETERM	3
<ItemRef ItemOID="IT.ADAE.AEDECOD" OrderNumber="54" Mandatory="No" KeySequence="2"/>	ADAE	AEDECOD	2
<ItemRef ItemOID="IT.ADAE.USUBJID" OrderNumber="2" Mandatory="Yes" KeySequence="1"/>	ADAE	USUBJID	1

EXTRACT VARIABLE ATTRIBUTES

As shown in the sample Define.xml below, the block with multiple lines from <ItemDef ItemOID … > to </ ItemDef > always contains key data attributes including Variable Name, DataType, Length, SignificantDigits, SASFieldName, Label, Origin, Comment, DisplayFormat.

```
data attrib_xml;
   infile datalines dsd truncover;
   length xmltext $500;
   input xmltext $ 1-500;
   datalines;

   <ItemDef OID="IT.ADSL.AGEGR2" Name="AGEGR2" SASFieldName="AGEGR2" DataType="text" Length="5">
   <Description>
   <TranslatedText xml:lang="en">Pooled Age Group 1</TranslatedText>
   </Description>
   <CodeListRef CodeListOID="CL.AGEGR2"/>
   <def:Origin Type="Derived"/>
   </ItemDef>
   ....
   <ItemDef OID="IT.ADSL.BMIBL" Name="BMIBL" SASFieldName="BMIBL" DataType="float" SignificantDigits="0" Length="8">
   <Description>
   <TranslatedText xml:lang="en">Basln Bdy Mass Index (kg/m2)</TranslatedText>
   </ItemDef>
   ...
   <ItemDef OID="IT.ADSL.RFICDT" Name="RFICDT" SASFieldName="RFICDT" DataType="integer" Length="8" def:DisplayFormat="YYMMDD10.">
   <Description>
   <TranslatedText xml:lang="en">Date of Informed Consent</TranslatedText>
   </Description>
   <def:Origin Type="Derived"/>
   </ItemDef>
run;
```

Below is the code sample using similar PRX function combined with the SAS RETAIN step to extract data attributes from Define.xml and retain attributes information which embedded in different lines of Define.xml into SAS regular tables.

```
data attr(keep=xmltext dsn var type lengh format) label(keep=dsn var label);
   set attrib_xml;
   retain dsn var;

   pattern2 =  prxparse('/\<ItemDef OID="IT.(AD\w\w+).(\w\w\w+)\" (.+) DataType="(\w\w\w+)"/');
   pattern3 =  prxparse('/\<ItemDef OID="IT.(.+)" Length\="(\d+)"/');
   pattern4 =  prxparse('/\<ItemDef OID="IT.(.+)" def:DisplayFormat="(.+)"/');
   pattern5 =  prxparse('/\<ItemDef OID="IT.(.+)" SignificantDigits="(\d+)"/' );
   pattern6 =  prxparse('/\<TranslatedText xml:lang="en"\>(.+)\<\/TranslatedText\>/');

   rx2=prxmatch(pattern2,xmltext);
   rx3=prxmatch(pattern3,xmltext);
   rx4=prxmatch(pattern4,xmltext);
   rx5=prxmatch(pattern5,xmltext);
   rx6=prxmatch(pattern6,xmltext);
       if rx2 = 1 then do;
       dsn=prxposn(pattern2, 1, xmltext);
       var=prxposn(pattern2, 2, xmltext);
       type=prxposn(pattern2, 4, xmltext);
       end;
       if rx3 then do;
       lengh=prxposn(pattern3, 2, xmltext);
       end;
       if rx4 then do;
           format=prxposn(pattern4, 2, xmltext);
       end;
       if rx5 then do;
           dec=prxposn(pattern5, 2, xmltext);
       end;
       if rx6 then do;
           label=prxposn(pattern6, 1, xmltext);
       end;
   if ^rx4 then format=catx('.', lengh, dec);
   if label>'' then output label;
   else if type>'' then output attr;
run;
```

(.+) Captures any character string with different length into the buffer. The metacharacter "." means "any character" including space.

Output SAS dataset with variable attributes merged in one row for each variable including Dataset Name (dsn), Variable name (dsn), Variable type (type, Variable length (length), Variable format (format) and Variable label (label).

dsn	var	type	lengh	format	label
ADSL	AGEGR2	text	5	5	Pooled Age Group 1
ADSL	BMIBL	float	8	8.0	Basln Bdy Mass Index (kg/m2)
ADSL	RFICDT	integer	8	YYMMDD10.	Date of Informed Consent

EXTRACT CODELIST

In the sample Define.xml below, Code list is defined by <CodeList OID=…> and </CodeList> block.

```
data xmlcl;
    infile datalines dsd truncover;
    length xmltext $300;
    input xmltext $ 1-300;
    datalines;
    <CodeList OID="CL.ADCM.CMTGR1" Name="ADCM.CMTGR1" DataType="text">
    <EnumeratedItem CodedValue="PRI" OrderNumber="1"/>
    <EnumeratedItem CodedValue="CON" OrderNumber="2"/>
    <EnumeratedItem CodedValue="PRI, CON" OrderNumber="3"/>
    <EnumeratedItem CodedValue="POST" OrderNumber="4"/>
    </CodeList>
    ...
    <CodeList OID="CL.DCSREAS" Name="DCSREAS" DataType="text">
    <EnumeratedItem CodedValue="Adverse Event" OrderNumber="1"/>
    <EnumeratedItem CodedValue="Lost to Follow-up" OrderNumber="2"/>
    <EnumeratedItem CodedValue="Other" OrderNumber="3"/>
    <EnumeratedItem CodedValue="Pregnancy" OrderNumber="4"/>
    <EnumeratedItem CodedValue="Protocol Deviation" OrderNumber="5"/>
    <EnumeratedItem CodedValue="Study Terminated by Sponsor" OrderNumber="6"/>
    <EnumeratedItem CodedValue="Withdrawal by Subject" OrderNumber="7"/>
    <EnumeratedItem CodedValue="Non-Compliance with Study Drug" OrderNumber="8"/>
    </CodeList>
run;
```

As shown in the coded example below, similar PRX functions and SAS RETAIN statements were used to extract Codelist values for each dataset and variable.

- PRX PATTERN1 '/"CL.(.+)" Name=/' to capture the Code list name after the "CL." such as codelist "CMTGR1" from ADCM dataset or a global code list named as "DCSREAS".
- PRX PATTERN2 '/CodedValue="(.+)" OrderNumber=\"(\d+)\"/'to parse CodedValue in any character "(.+)" into first buffer, and parse numerical OrderNumber into the second buffer captured by \"(\d+)\"/.

```
data codelist;
    set xmlcl;
    length clname $40;
    retain clname;
        pattern1 = prxparse('/"CL.(.+)" Name=/');
        pattern2 = prxparse('/CodedValue="(.+)" OrderNumber=\"(\d+)\"/');

        rx1=prxmatch(pattern1,xmltext);
        rx2=prxmatch(pattern2,xmltext);
        if rx1 then do;
            clname=prxposn(pattern1, 1, xmltext);
        end;
        if rx2 then do;
            cl=prxposn(pattern2, 1, xmltext);
            clord=prxposn(pattern2, 2, xmltext);
        end;
    if cl>'';
run;
```

Output of dataset includes Codelist Name (clname), Codelist Value (cl) and Codelist Value order (clord). This dataset can be very useful when comparing with the data frequency tables to validate the Codelist.

clname	cl	clord
ADCM.CMTGR1	PRI	1
ADCM.CMTGR1	CON	2
ADCM.CMTGR1	PRI, CON	3
ADCM.CMTGR1	POST	4
DCSREAS	Adverse Event	1
DCSREAS	Lost to Follow-up	2
DCSREAS	Other	3
DCSREAS	Pregnancy	4
DCSREAS	Protocol Deviation	5
DCSREAS	Study Terminated by Sponsor	6
DCSREAS	Withdrawal by Subject	7
DCSREAS	Non-Compliance with Study Drug	8

BUILD CHECKS USING EXTRACTED METADATA

Now that we have define.xml metadata (Key variable list, Variable attributes and Codelist) in SAS data sets, we have the power of SAS to develop home grown validation checks. For example, a macro ***%chkkey*** below was created to validate if the key variable list extracted from the "Extract Key Variable List" section above can uniquely define each record for each submission dataset. FDA has previously reported that most regulatory submissions have problems with duplicate records. They represent potentially contradictory information and make it difficult to summarize results [4]. We were able to identify the insufficient key list for these duplicate records that could have been missed if we had just relied on Pinnacle 21. The name list of all datasets can be captured into a macro variable &dslist using PROC SQL code below. The code sample from %chkkey macro is below.

```
proc sql noprint;
select distinct dsn into :dslist separated by '|'
from dsin;
quit;
```

SAS macro ***%chkkey*** validates whether the key list can identify the unique records in each dataset.

```
%macro chkkey;
      %let i=1;
      %do %while (%scan(&dslist,&i,|)^= );
        %let ds=%scan(&dslist,&i,|);
        proc sql noprint;
            select distinct var, keyord, var into :keylist separated by ' '
            from keylist
            where dsn="&ds"
            order by keyord;
        quit;

        proc sql noprint;
            select distinct var into :lvar
            from lstvar
            where dsn="&ds";
        quit;

        proc sort data=test.&ds out=&ds ;
         by &keylist;
        run;

        data &ds._chk;
         set &ds;
         by &keylist;
         if ^first.&lvar. or ^last.&lvar.;
        run;
        %let i=%eval(&i+1);
        %put Keylist=&keylist;

        %LET dsid=%SYSFUNC(OPEN(&ds._chk));
        %LET nobs=%SYSFUNC(ATTRN(&dsid.,nobs));
        %LET rc=%SYSFUNC(CLOSE(&dsid.));

        %if &nobs. ^= 0 %then %do;
            %put "!!!!!!!!!!! Key list not sufficient to identify unique records !!!!!!!!!";
        %end;
        %else %do;
            %put "&ds.  Passed Keylist check";
        %end;
      %end;

%mend;
%chkkey;
```

CONCLUSION

As shown in this paper, the Perl Regular Expression in SAS can be a powerful tool for directly converting metadata from Define.xml into SAS data set records. This approach eliminates the extra step to use XMLmap or other external software in the Define.xml validation process and the following manual steps for formatting the metadata dataset. By keeping all the code in SAS, this approach makes an automated process possible from beginning to the end. Another advantage of this approach is that you have the ability and flexibility to extract different parts of Define.xml selectively based upon your validation needs, and build in-house Define.xml validation beyond what external software can provide.

We developed the code in SAS 9.4 for extracting metadata from ADaM define.xml version 2.0. The approach and Perl Regular Expression Pattern matching technique presented in this paper should work for SDTM prepared using version 2.0 as well.

ACKNOWLEDGMENTS

This author is very grateful to Gretchen Murphy and Sondra Smyrnios for their editing and review.

REFERENCES

[1] Ron Cody, Robert Wood , 2004. SUGI 29 Paper 265-29 “An Introduction to Perl Regular Expressions in SAS® 9” Robert Wood Johnson Medical School, Piscataway, NJ.
http://www2.sas.com/proceedings/sugi29/265-29.pdf

[2] Richard Pless. 2004. SUGI 29 Paper 043-29 “An Introduction to Regular Expressions with Examples from Clinical Data”, Ovation Research Group, Highland Park, IL
http://www2.sas.com/proceedings/sugi29/043-29.pdf

[3] SAS Perl Regular Expressions Tip Sheet
https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

[4] Doi, Mary. 2016. “*How Good is Your SDTM Data? Perspectives from JumpStart*”. PhUSE CSS.
http://www.phusewiki.org/docs/CSS%202016%20Presentations/SDTM%20Mary%20Doi.pptx

CONTACT INFORMATION

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

Yuxin (Ellen) Jiang
Principal Programmer, Clinical Operation
Alkermes, Inc.
850 Winter St., Waltham, MA
Email: Yuxin.jiang@Alkermes.com