Paper 836-2017

Automate Validation of CDISC SDTM with SAS®

Sy Truong, Pharmacylics, Inc.

ABSTRACT

There are many good validation tools for CDISC SDTM such as Pinnacle 21 Enterprise. However, the power and customizability of SAS® provides an effective tool for validating SDTM datasets used in clinical trials FDA submissions. This paper presents three distinct methods of using SAS to validate the transformation from EDC (Electronic Data Capture) data into CDISC SDTM format. This includes:

1. Duplicate Programming – An independent SAS program used to transform EDC data with PROC COMPARE.
2. Transformation Validation – SAS macro used to compare EDC data and SDTM using PROC FREQ to identify outliers.
3. Rules Checker – SAS program to verify a specific SDTM or regulatory rules applied to SDTM SAS datasets.

The three examples illustrate the diverse approaches which SAS programs can be applied to catch errors in data standard compliance or identify inconsistencies that would otherwise be missed by other general purpose utilities. The stakes are high when preparing for a FDA submission. Catching errors in SDTM during validation prior to a submission can mean the difference between success or failure for a drug or medical device.

DUPLICATE PROGRAMMING

A risk assessment is applied upon the SDTM domain for a particular study. This evaluates the complexity of the transformations performed along with other factors, such as if the transformation is performed by standard macros or newly developed custom SAS programs. In the event that the assessment determines that the SDTM transformation is at high risk of potential errors, duplicate programming is applied. This means that an independent SAS programmer develops a separate program to read the same source data to generate the SDTM domains. The duplicate program will use the same specifications; however, it will not share any of the same code logic that is used in the program which is then used to produce the "production" SDTM domain datasets. For example, if the principle programmer generates a program named DM.SAS to generate the demographic domain (DM.SAS7BDTAT); a separate and independent SAS program authored by another programmer named V_DM.SAS is developed. It will then generate another dataset named V_DM.SAS7BDAT. The validation program then performs a PROC COMPARE between the DM and V_DM dataset to identify differences. The following steps are taken to accomplish the duplicate programming validation process.

STEP 1 – Independent Programmer and Folder

The role of the project manager of this SDTM project is to assign an independent SAS programmer to perform validation, and another programmer to generate the SDTM dataset. The validation programmer will develop a program such as V_DM.SAS in a separate folder as compared to the SDTM programmer. He is given the all the same information such as the protocol, annotated CRF and SDTM specifications. The only difference is he will generate a dataset named V_DM rather than the DM.

STEP 2 – Libraries and Code List

The validation program needs to define a different set of libnames so that it will produce validation datasets in a separate location. It will also re-create the code lists as defined by the specification. The following illustrates an example code segment for this portion of the validation program.

```
*** Define the options and libraries for validation program ***;

OPTIONS YEARCUTOFF = 1920;

OPTIONS VALIDVARNAME = UPCASE;

OPTIONS LINESIZE = 256;


LIBNAME SRC "M:\STUDY-123\Development\Source Data";

LIBNAME VSDTM "M:\STUDY-123\Development\Validation\V_SDTM";
```

```
*** Define the code list used to generate standard SDTM ***;
PROC FORMAT;
   VALUE $SEX
   "Female" = "F"
   "Male"   = "M";

   VALUE $YN
   "Yes" = "Y"
   "No"  = "N";
run;
```

This may appear to be very similar to the program used to produce the production SDTM domain from the main programmer with only differences in style.

STEP 3 – Read Specification

The SDTM specifications are stored in a standard Excel format. The validation program must generate validation SDTM datasets from the information stored in the specifications. This is accomplished by programmatically reading the specifications into a dataset to automate the creation of the duplicate SDTM domain.

```
*** Define the location of the specification ***;
LIBNAME WORKBK1 'M:\STUDY-123\Development\SDTM\specs\sdtm_mapping_spec.xlsx';

*** Capture the Domain attributes stored in the specification ***;
DATA DOMAIN (KEEP = DOMAIN VARIABLE_ORDER CDISC_VARIABLE_NAME CDISC_VARIABLE_LABEL
TYPE LENGTH TRANSFORMATION_TYPE TRANSFORMATION_VARIABLES);
   LENGTH DOMAIN $8.;
   SET DM_SP;
   WHERE DOMAIN = "&DOMAIN" AND UPCASE(TRANSFORMATION_TYPE) ^= "DROP";
RUN;
```

There are multiple workbooks in the Excel file that stores different information for each domain. The program reads each workbook into separate temporary SAS datasets. This is then assigned to macro variables in order to automate the creation and validation.

STEP 4 – Generate the Duplicate Domain

The duplicate validation program then generates the validation domain with the standard attributes as defined from the specifications.

```
*** Capture the attributes and create macro variables ***;
PROC SQL NOPRINT ;
   SELECT CDISC_VARIABLE_NAME INTO :&DOMAIN.VAR SEPARATED BY ' ' FROM ATTRIBUTE WHERE
UPCASE(DOMAIN)=%UPCASE("&DOMAIN");
   SELECT ATTRIB INTO :&DOMAIN.ATTRIB SEPARATED BY ' ' FROM ATTRIBUTE WHERE
UPCASE(DOMAIN)=%UPCASE("&DOMAIN");
   SELECT CDISC_VARIABLE_NAME INTO :SUPP&DOMAIN.VAR SEPARATED BY ' ' FROM ATTRIBUTE
WHERE UPCASE(DOMAIN)=%UPCASE("SUPP&DOMAIN");
   SELECT ATTRIB INTO :SUPP&DOMAIN.ATTRIB SEPARATED BY ' ' FROM ATTRIBUTE WHERE
UPCASE(DOMAIN)=%UPCASE("SUPP&DOMAIN");
```

```
QUIT;


*** Create the standard attributes ***;
DATA V_DM(LABEL="Demographics");
   RETAIN &DMVAR;
   KEEP   &DMVAR;
   ATTRIB &DMATTRIB ;
   SET DMALL;
RUN;
```

The actual program logic used to perform the transformation of the source data to SDTM domain can be elaborate and unique to each programmer and SDTM domain. The above example illustrates how the validation programmer used PROC SQL to manipulates datasets and macro variables. The main programmer who generated the target DM may have used DATA STEP or other macros instead of PROC SQL. This confirms that an independent approach may catch differences that may be missed otherwise.

STEP 5 – Perform Comparison with Duplicate

Once the duplicate validation dataset is created, it is sorted and compared with the target SDTM domain using PROC COMPARE.

```
*** Sort the data in preparation for comparisons ***;
PROC SORT DATA = SDTM.DM OUT = PROD_DM;
   BY USUBJID;
RUN;


PROC SORT DATA = VS_DM OUT= VSDTM.V_DM (LABEL="Demographics");
   BY USUBJID;
RUN;


*** Perform the comparisons to identify differences with validation ***;
PROC COMPARE BASE = PROD_DM COMP = VSDTM.V_DM LISTALL;
   ID USUBJID SUBJID;
RUN;
```

Once the duplicate validation dataset is created, it is sorted and then compared with the main SDTM datasets. The challenge with this step is for the validation programmer to research and understand the differences in order to determine if it is an error on the validation program or in the original SDTM production program. Once this has been identified, the validation programmer needs to then clearly communicate this finding to the original programmer and the project manager to have the error resolved. The project manager can function as the arbitrator in the event that the correct approach between the two programs is not obvious. This entire process does ensure for more accurate results, but does take much longer as compared to just generating the original program used to generate the production SDTM SAS dataset.

TRANSFORMATION VALIDATION

In this validation method, the goal is to catch discrepancies during the transformation from source data variables to SDTM. The user has defined the transformation in an Excel specification file. A SAS macro named %sdtm_freq is then used to perform a summary of the corresponding source variable and compare them against the SDTM variables using PROC FREQ as listed in the specification file. It can distinguish between categorical or continuous variables. In the event of continuous variables, it performs a PROC MEANS to identify outliers. The step by step logic is explained in the below.

STEP 1 – Named Parameters:
The first step is to capture user defined macro parameters which is clearly defined and explained through the comment header section.

```
*******************************************************************************
* SAS Macro used to verify SDTM datasets by PROC FREQ and PROC MEANS
*******************************************************************************
* Macro Parameters
*
*  Spec   C (200 chars)
*  This specifies the path and file name of the SDTM specification
*  file.  It is required that the Excel file follows a standard structure as *
defined by global specification template.
*
* Srcpath C (200 chars)
* The path location to where source SAS datasets are stored. These datasets
* are used to be verified against SDTM datasets.
*
* Sdtmpath C(200 chars)
* The path location to where SDTM SAS datasets are stored.  These datasets
* are used to be verified against source datasets.
*
* Output  C (200 chars optional)
* The name of the output validation report file.  If none is specified, the
* default file named: v_sdtm_freq.html will be generated at the same location * as the
input dataset specified by SDTMPATH.
*******************************************************************************;
%macro sdtm_freq (spec=, srcpath=, sdtmpath=, output= );
```

STEP 2 – Parameter Error Checking:
Before the macro algorithm is applied, a series of error checking is performed upon each parameter. This step verifies both missing and required variables including verifying invalid parameter values. If any of the error conditions are caught, the core algorithm of the macro will not be applied.

```
   *** Initialize validation error findings ***;
   %let anyerror=no;
   %let _err=no;

   *** Perform Error checking for missing parameters ***;
   data _null_;
      put "NOTE: You are currently running the macro sdtm_freq version 1.0.";

      *** Verify that all required variables are not missing ***;
      if compress("&spec") = '' then do;
         put " ";
         put "ERROR [sdtm_freq]: is missing or has an invalid parameter SPEC.";
         put "      This defines the path and Excel file name for the SDTM
specification.";
         put " ";
         call symput('_err','yes');
      end;
      if fileexist("&srcpath") = 0 then do;
         put " ";
         put "ERROR [sdtm_freq]: is missing or has an invalid parameter SRCPATH.";
         put "      This defines the path to the study source SAS datasets.";
         put " ";
         call symput('_err','yes');
      end;
   run;

   *** Skip the logic to the end if there was an error encountered ***;
   %if "&_err" = "yes" %then %goto endcheck;
```

In addition to displaying error messages in the log, a note is also generated in the log to inform the meaning of the parameter and the version number for the macro. This helps the user learn how to update the macro call and track the version of the macro being used.

STEP 3 – Import Transformation Specification:

The user has defined each SDTM variable and how it is to be transformed from source data through an Excel file. This lists out all source variables that are used for the creation of the each SDTM variable. There is a separate worksheet in the specification Excel file for each domain. The %sdtm_freq macro first imports the "TOC" worksheet which contains lists out all the domains.

```
*** Start by importing the domains ***;
proc import datafile = "&spec"
   OUT = work.domains
   replace;
   sheet="TOC";
   getnames=yes;
run;
```

It then stores the name of each domain in a series of macro variables with the "DAT" prefix. The macro then loops through each domain and imports the information for the variable specifications into separate SAS datasets.

```
*** Import the specification for dataset: &&dat&i ***;
proc import datafile = "&spec"
   OUT = work.&&dat&i
   replace;
   sheet="&&dat&i";
   TEXTSIZE=32767;
   getnames=yes;
run;
```

STEP 4 – Decipher Summary Type:

Each variable is identified as either "categorical" or "continuous". This will later be used to perform a PROC FREQ or PROC MEANS. The initial assignment for variable type is "categorical" which is assigned to all character variables.

```
*** Review all columns to see which are categorical vs continuous ***;
data variables;
   set sashelp.vcolumn;
   where upcase(libname)="SDTMLIB";

   *** Review Column Type to see which are automatic ***;
   attrib sumtype length=$20 label="Summary Variable Type";
   if type = 'char' then sumtype = 'categorical';

   *** Only keep datasets that are part of specs ***;
   if upcase(memname) in(
   %do i = 1 %to &datcnt;
      "&&dat&i"
      %if "&i" ne "&i" %then , ;
   %end;
   ) then output;
run;
```

It then evaluates numeric variables by counting distinct values to determine if the variable is categorical. If there are more than 20 distinct values, it will be identified as "continuous". Otherwise, it will assign the numeric variable to be of type categorical.

```
*** Assign to be categorical if it has less than 20 distinct values ***;
data nvariable3;
   set nvariable;
   attrib sumtype length=$20 label="Summary Variable Type";
   if rowcount <= 20 then sumtype = 'categorical';
   else sumtype = 'continuous';
run;
```

STEP 5 – Perform Summary on Target Variables:
Once each variable has been identified as either categorical or continuous, macro variables are assigned to keep track of each variable type. It will then loop through each variable and perform either a PROC FREQ or PROC MEANS upon each target SDTM variable.

```
*** Generate either PROC FREQ or PROC MEANS depending on summary type ***;
%do i = 1 %to &varcnt;
   ods listing close;
   ods html body="&subfolder\&&dat&i.._&&var&i...html";
   title1 "Dataset = &&dat&i";
   title2 "Variable = &&var&i";

   %if "&&typ&i" = "CONTINUOUS" %then %do;
      proc means data = sdtmlib.&&dat&i (keep=&&var&i);
      run;
   %end;
   %if "&&typ&i" = "CATEGORICAL" %then %do;
      proc freq data = sdtmlib.&&dat&i (keep=&&var&i);
      run;
   %end;
   ods html close;
   ods listing;
%end;
```

STEP 6 – Perform Summary on Source Variables:
For each target SDTM variable, there are one or more source variables used in the transformation. This is listed in the Excel specification file. Now that the Excel is converted into a SAS dataset, a data step is used to identify source variables corresponding to each target SDTM variable. A similar PROC FREQ and MEANS are performed upon these source variables. The variable type which was assigned by the target variables will have a corresponding source variable applied with the same summary statistics.

```
*** Generate either PROC FREQ or PROC MEANS depending on summary type ***;
%do i = 1 %to &varcnt;
   ods listing close;
   ods html body="&subfolder\source_&&dat&i.._&&var&i...html";
   title1 "Source Dataset = &&dat&i";
   title2 "Variable = &&var&i";

   %if "&&typ&i" = "CONTINUOUS" %then %do;
      proc means data = srclib.&&s_dat&i (keep=&&s_var&i);
      run;
   %end;
   %if "&&typ&i" = "CATEGORICAL" %then %do;
      proc freq data = srclib.&&s_dat&i (keep=&&s_var&i);
      run;
   %end;

   ods html close;
   ods listing;
%end;
```

The output HTML report file name will retain a similar name to the target variable report with the prefix “source_” to distinguish the two sets of reports. For simplicity, the above reports are only performed if there is a one to one correspondence between source and SDTM variables. It leaves the one to many or many to many relationship to other forms of validation.

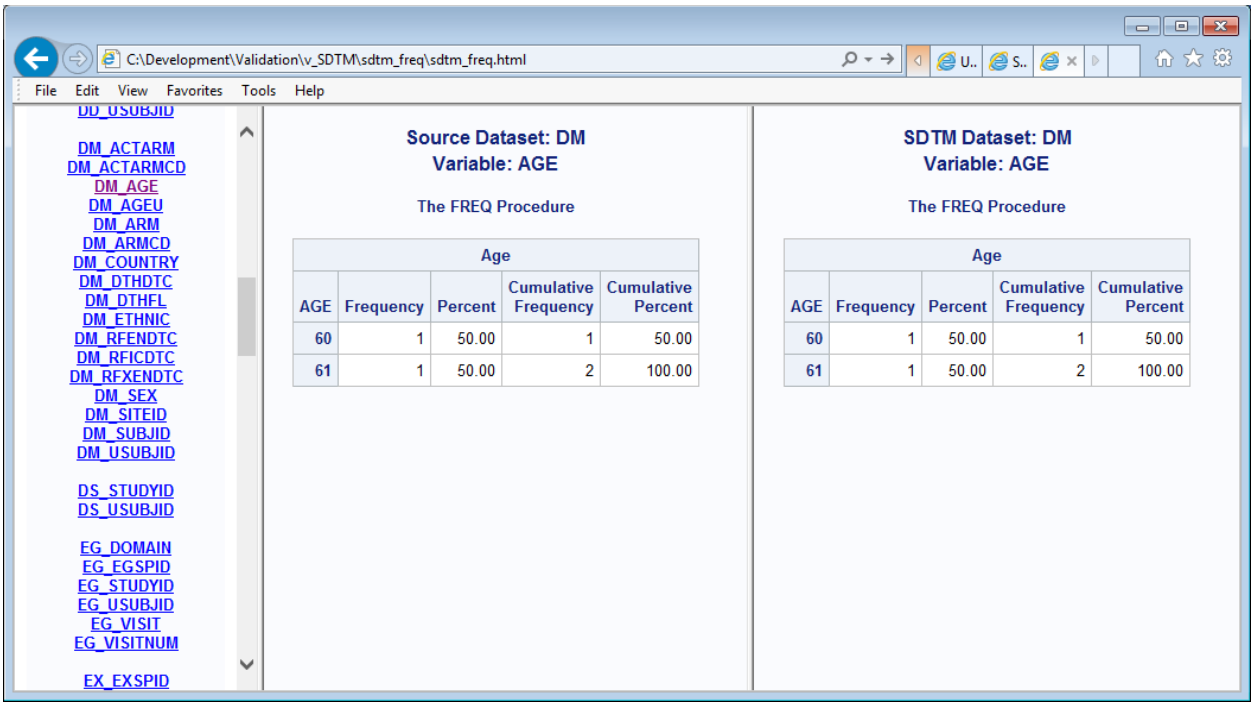
STEP 7 – Generate HTML Frames:
The final report is generated as HTML frames displaying the source and destination data summaries side by side. The HTML components of the HTML frames are generated with a data step. A portion of one component is shown here, with the same style as the other ODS stylesheet:

```
   *** Generate the sdtm_freq.html define the frame set of three panels ***;
   filename framea "&output";

   *** Read in template and create new version ***;
   data _null_;
      file framea lrecl=400;

      put "<!DOCTYPE html>";
      put "<html>";
      put "<head>";
      put "<title>SDTM_FREQ - Study &studynum</title>";
      put "</head>";
      put "<frameset cols='20%,40%,40%'>";
      put "   <frame name='left' src='&subfolder\left_index.html' />";
      curout = "&s_out1";
      slashpos =1;
      do i = 1 to length(curout);
         curchar = substr(curout,i,1);
         if curchar in("/","\") then slashpos = i;
      end;
      curout2 = substr(curout,slashpos+1);
      curline = "   <frame name='center' src='" || strip("&subfolder") || "\" ||
strip(curout2) || "'/>";
      put "   " curline;

      put "   <frame name='right' src='&subfolder\&dat1._&var1..html' />";
      put "   <noframes>";
      put "   <body>";
      put "      Your browser does not support frames.";
      put "   </body>";
      put "   </noframes>";
      put "</frameset>";
      put "</html>";
   run;
```

The frame has three panels with the left most panel only taking 20% of the window. This is because the left panel will display the list of datasets and variables. It will function as a navigational panel. If the user clicks on any of the variables on this left panel, the two panels on the right of it will be refreshed with the corresponding summary reports corresponding to the selected variable.

STEP 8 – Review Report:

Once the program has been successfully executed, it will generate many HTML files. Because of this, all the HTML files are stored under a subfolder. The key index that displays the frames is named sdtm_freq.html as shown here:



In this example, the user has selected on a numeric variable AGE. In this small dataset, there were only two distinct values including ages 60 and 61. Thus, the report has determined that this is a categorical variable and that the values are consistent between source and SDTM. The user can scroll and click on any of the other variables and the left pane. This will refresh the two panels to the right with the corresponding selected summary report comparison. This allows the user to identify outliers and potential deviations during transformation.

RULES CHECKER

The implementation guides provided from CDISC presents many suggestions as guidelines. Each one of these guidelines can be interpreted as a rule that can be applied. In these cases, a SAS validation program can function as a checker to verify and confirm if the rule being applied is correct and consistent. This is not limited to the implementation guide for SDTM. For example, the "CDISC Define-XML Specification Version 2.0" guideline has the following rule for the ORIGIN column:

> **Predecessor**: Data that is copied from a variable in another dataset. For example, predecessor is used to link ADaM data back to SDTM variables to establish traceability.

In this case, a variable used in the ADaM data model is directly copied from a SDTM variable. This presents an opportunity for a program to verify if the two corresponding variables between SDTM and ADaM which has defined with "Origin" as being "Predecessor". For these variables, the predecessor SDTM variable must have the same variable attributes as the corresponding ADaM variable. The following steps are taken to perform this rule check.

STEP 1 – Capture ADaM Predecessor Variables:

The define.xml in this example was originally defined in an Excel specification file. That Excel file was then converted into a SAS dataset named: _DEFINE. The following SAS data step will process capture only predecessor variables.

```
*** Capture all predecessor variable except ADSL ***;
data _define (keep=datname variable _label _type _length _format memname name) ;
   set deflib._define;
   where upcase(compress(origins)) = "PREDECESSOR" and datname ne "ADSL";

   *** Define SDTM dataset as MEMNAME and SDTM variable as NAME ***;
   length memname name $32;
   if length(datname) = 4 then do;
      memname = upcase(substr(datname,3));
      name = variable;
```

```
      *** Rename type, length and format ***;
      attrib _type length=$4 label="Define.xml Type";
      attrib _length length=8 label="Define.xml Length";
      attrib _label length=$256 label="Define.xml Label";
      attrib _format length=$49 label="Define.xml Format";

      _type=type;
      _length=length;
      _label=label;
      _format=format;

      output;
   end;
run;
```

Since the ADSL dataset contain variables that are duplicated across other ADaM domains, it is not included in this check. The attributes are renamed with variables with an underscore prefrix to distinguish them from SDTM variables.

STEP 2 – Capture SDTM Variables:

The SDTM variables are stored as SAS datasets. The following step captures the metadata similar to a PROC CONTENTS.

```
*** Capture all the variables attributes from SDTM data ***;
libname sdtmlib "&sdtmpath";
data _sdtmvars (keep=memname name type length label format);
   set sashelp.vcolumn;
   where libname = "SDTMLIB";
run;
```

STEP 3 – Identify Predecessors not in SDTM:

The first check will verify if the predecessor is missing or not as defined and matching with the corresponding SDTM variable. This is identified through a DATA STEP merge.

```
*** Identify Define.XML Predecessor not in SDTM ***;
data XMLnotSDTM;
   merge _define (in=A)
         _sdtmvars (in=B);
   by memname name;

   if (A) and not(B) then output;
run;
```

STEP 4 – Identify Differences in Attributes:

For the other predecessor variables that does exist between ADaM and SDTM, a comparison between the attributes are performed. Any differences will be captures and documented in the REASON variable.

```
*** Identify Attribute differences between XML and SDTM ***;
data AttribDiff;
   merge _define (in=A)
         _sdtmvars (in=B);
   by memname name;

   attrib reason length=$200 label="Findings";
   reason = "Different:";
   if (A) and (B) then do;
      if upcase(type) ne upcase(_type) or
         length ne _length or
         label ne _label or
         format ne _format then do;

         if upcase(type) ne upcase(_type) then reason = trim(reason) || " type";
         if length ne _length then reason = trim(reason) || " length";
         if label ne _label  then reason = trim(reason) || " label";
         if format ne _format then reason = trim(reason) || " format";
```

```
            output;
          end;
      end;
   run;
```

There can be multiple attribute differences. Thus, the program concatenates all the attribute findings into one variable REASON.

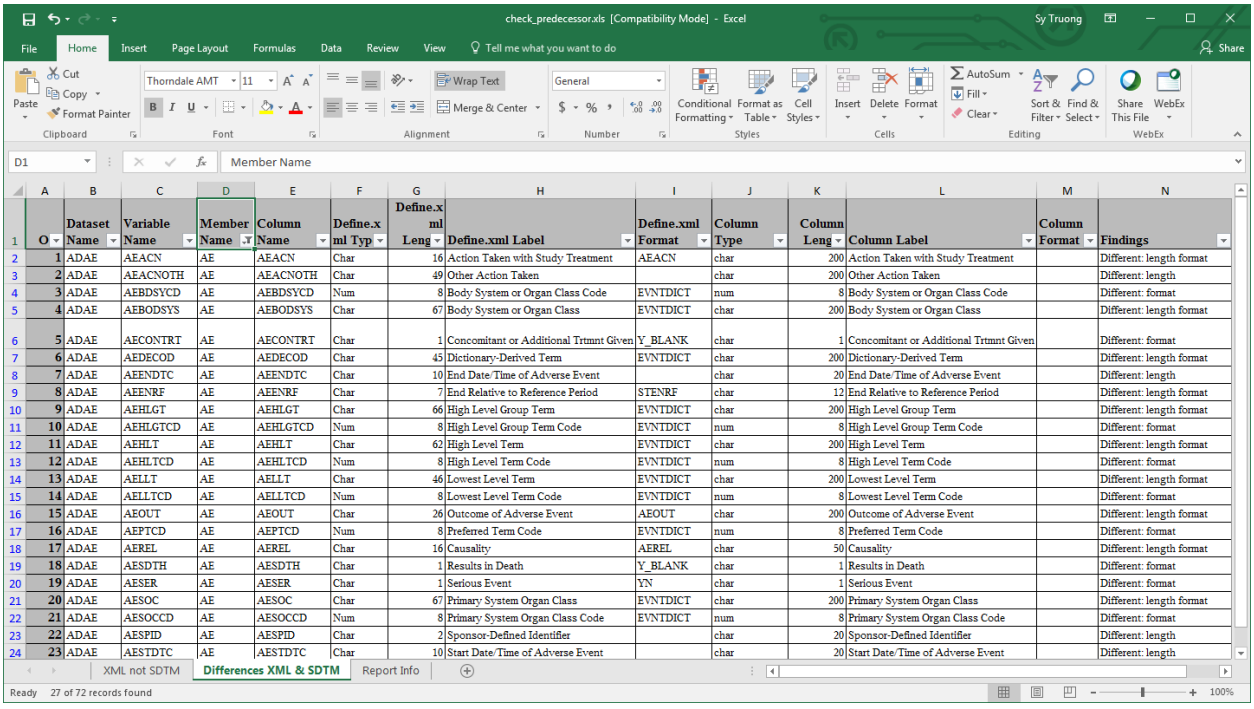
STEP 5 – Generate Excel Report:

A report using ODS is used to generate as an Excel file. This allows the user to apply filters to easily review findings pertaining to any specific domain or variable.

```
   *** Generate the output findings in Excel file ***;
   ods _all_ close;
   ods tagsets.ExcelXP path="&outpath" file="&outfile2"
   style=Printer;

   ods tagsets.ExcelXP options(sheet_name='XML not SDTM'
                               absolute_column_width='4,6,8,6,8,6, 6,20,8'
                               autofit_height='yes');

   proc print data = xmlnotsdtm label;
      var datname variable memname name _type _length _label _format;
   run;

   ods tagsets.ExcelXP options(sheet_name='Differences XML & SDTM'
                               absolute_column_width='4,6,8,6,8,6,6,20, 8,8, 8,20, 8,
14'
                               autofit_height='yes');

   proc print data = attribdiff label;
   run;
```

STEP 6 – Review Report:

Each data set is represented as a separate workbook in the final Excel. A filter can be applied as it is here for dataset "AE".



	O	Dataset Name	Variable Name	Member Name	Column Name	Define.xml Typ	Define.xml Leng	Define.xml Label	Define.xml Format	Column Type	Column Leng	Column Label	Column Format	Findings
2	1	ADAE	AEACN	AE	AEACN	Char	16	Action Taken with Study Treatment	AEACN	char	200	Action Taken with Study Treatment		Different: length format
3	2	ADAE	AEACNOTH	AE	AEACNOTH	Char	49	Other Action Taken		char	200	Other Action Taken		Different: length
4	3	ADAE	AEBDSYCD	AE	AEBDSYCD	Num	8	Body System or Organ Class Code	EVNTDICT	num	8	Body System or Organ Class Code		Different: format
5	4	ADAE	AEBODSYS	AE	AEBODSYS	Char	67	Body System or Organ Class	EVNTDICT	char	200	Body System or Organ Class		Different: length format
6	5	ADAE	AECONTRT	AE	AECONTRT	Char	1	Concomitant or Additional Trtmnt Given	Y_BLANK	char	1	Concomitant or Additional Trtmnt Given		Different: format
7	6	ADAE	AEDECOD	AE	AEDECOD	Char	45	Dictionary-Derived Term	EVNTDICT	char	200	Dictionary-Derived Term		Different: length format
8	7	ADAE	AEENDTC	AE	AEENDTC	Char	10	End Date/Time of Adverse Event		char	20	End Date/Time of Adverse Event		Different: length
9	8	ADAE	AEENRF	AE	AEENRF	Char	7	End Relative to Reference Period	STENRF	char	12	End Relative to Reference Period		Different: length format
10	9	ADAE	AEHLGT	AE	AEHLGT	Char	66	High Level Group Term	EVNTDICT	char	200	High Level Group Term		Different: length format
11	10	ADAE	AEHLGTCD	AE	AEHLGTCD	Num	8	High Level Group Term Code	EVNTDICT	num	8	High Level Group Term Code		Different: format
12	11	ADAE	AEHLT	AE	AEHLT	Char	62	High Level Term	EVNTDICT	char	200	High Level Term		Different: length format
13	12	ADAE	AEHLTCD	AE	AEHLTCD	Num	8	High Level Term Code	EVNTDICT	num	8	High Level Term Code		Different: format
14	13	ADAE	AELLT	AE	AELLT	Char	46	Lowest Level Term	EVNTDICT	char	200	Lowest Level Term		Different: length format
15	14	ADAE	AELLTCD	AE	AELLTCD	Num	8	Lowest Level Term Code	EVNTDICT	num	8	Lowest Level Term Code		Different: format
16	15	ADAE	AEOUT	AE	AEOUT	Char	26	Outcome of Adverse Event	AEOUT	char	200	Outcome of Adverse Event		Different: length format
17	16	ADAE	AEPTCD	AE	AEPTCD	Num	8	Preferred Term Code	EVNTDICT	num	8	Preferred Term Code		Different: format
18	17	ADAE	AEREL	AE	AEREL	Char	16	Causality	AEREL	char	50	Causality		Different: length format
19	18	ADAE	AESDTH	AE	AESDTH	Char	1	Results in Death	Y_BLANK	char	1	Results in Death		Different: format
20	19	ADAE	AESER	AE	AESER	Char	1	Serious Event	YN	char	1	Serious Event		Different: format
21	20	ADAE	AESOC	AE	AESOC	Char	67	Primary System Organ Class	EVNTDICT	char	200	Primary System Organ Class		Different: length format
22	21	ADAE	AESOCCD	AE	AESOCCD	Num	8	Primary System Organ Class Code	EVNTDICT	num	8	Primary System Organ Class Code		Different: format
23	22	ADAE	AESPID	AE	AESPID	Char	2	Sponsor-Defined Identifier		char	20	Sponsor-Defined Identifier		Different: length
24	23	ADAE	AESTDTC	AE	AESTDTC	Char	10	Start Date/Time of Adverse Event		char	20	Start Date/Time of Adverse Event		Different: length

The last column on the right describes how variable length and formats are not consistent between the ADaM predecessor variable and some of its corresponding SDTM variables.

CONCLUSION

A comprehensive validation of SDTM data is essential in a successful FDA submission. The SDTM data model is constantly being enhanced with more domains added and growing complexity. The need for an automated validation approach for the data as it is transformed from EDC source data into SDTM is very specific to each organization. There are many unique interdependent components such as the data structure of the EDC source data, transformation SDTM specification and the interpretation on how each SDTM variable is to be transform will be vary for each organization. Thus, the need for a customized, flexible, yet powerful set tools are required to perform validation unique to each SDTM implementation. SAS fulfills these requirements since users can develop custom SAS scripts and macros to import EDC data and parse user specific Excel specifications. The user's interpretation of the Implementation Guide is then transcribed into SAS programs to automate the validation of SDTM. The use of SAS programs is not intended to completely replace tools such as Pinnacle 21 Enterprise, but rather it augments and compliment these standardized tools. By combining the custom SAS scripts with existing tools, users can obtain a more comprehensive validation approach, ensuring the greatest data integrity for their successful submission.

CONTACT INFORMATION

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

Name: Sy Truong
Company: Pharmacyclics, Inc.
Work Phone: 669-224-1107
E-mail: struong@pcyc.com