PharmaSUG 2020 - Paper AP240

Next Level Programming-Reusability and Importance of Custom Checks

Akhil Vijayan, Genpro Research, Thiruvananthapuram, India;
Limna Salim, Genpro Research, Thiruvananthapuram, India;
Anoop Ambika, Genpro Research, Thiruvananthapuram, India;

ABSTRACT

SDTM-domain structures and relationships are similar across studies under a therapeutic area which leads to code standardization and reusability especially within ISS/ISE submissions. Interim data transfers also come with changes in data leading to rerun of existing programs with minor updates. The possibility of errors in such scenarios are large with truncation in data, new data issues being unidentified, attribute changes etc. This paper details the importance of using standard macros and alternative programming approaches like enabling custom checks/warnings that makes reusability of programs a much smoother process.

Not all Data Issues are identified at the initial stage of Source Data Validation, but they tend to surface during the development of CDISC datasets. In addition, certain data issues identified at the initial run might not always be necessarily resolved in the next data transfer. This is where custom errors / warnings play a significant role. Similarly, in case of Statistical programming, a standard code might be replicated for various TFLs with changes only to the parameters considered. In such cases, specific custom checks based on parameters also comes into importance.

This paper discusses various situations with examples where user defined errors/warnings can be implemented, like

- Validation of subject included alongside DM data
- Verifying the length of source variable (ensuring data after 200 characters are successfully mapped to SUPP)
- Verifying the baseline flags populated after derivation

The paper also discusses the need for standard macros which support custom checks like, macros for

- Source data variable length check
- Automating formats
- Attribute generation.

INTRODUCTION

Quality and Time are the two major factors that define efficient clinical research. Development of high-quality Study Data Tabulation Model (SDTM)/ Analysis Data Model (ADaM)/ Tables, Listings, and Figures (TLF) datasets can be time consuming while implementing the QC checks at different phases of the development. Industries are focusing on the automation process, and the importance of custom checks in such cases are even greater. This paper assumes the readers to have a basic understanding about SAS, SDTM, ADaM and TLF Programming.

Quality is obstinate in programming, and it is very disappointing that issues are not identified at an earlier stage even though people have followed standard programming practices. Experience is, of course, a key factor that will help predict the possibility of error. But the programming team does not always consist of experienced members. Well, who's an experienced programmer? Some of the factors that influence 'experience' include programming skills, therapeutic area knowledge, SDTM / ADAM / TLF work experience, etc.

This paper focuses on and presents some ideas that programmers can use to reach utmost quality such as detailing the importance with examples of custom errors / warnings to identify data issues and programming issues. Also included are some basic macros that are required for smart programming and details on the same.

IMPORTANCE & EXAMPLES OF USER DEFINED ERRORS/WARNINGS

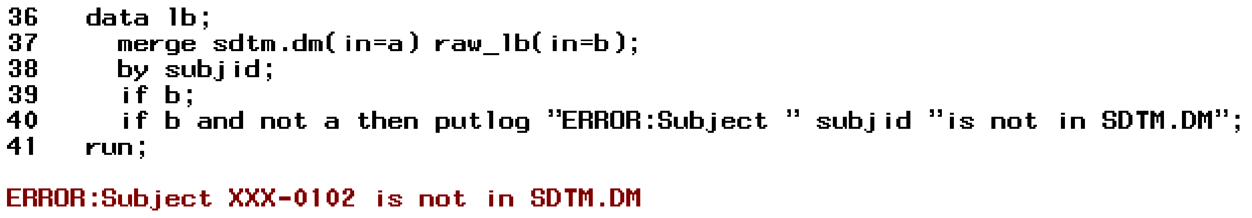
'User-defined errors / warnings' used by programmers for various programming purposes helps to ensure reliability if used in a standardized manner. The question is how these checks can help us to ensure the quality of programming. There are common (standard) and study-based checks that the programmer can carry out in their respective programs some of which specific to SDTM may include ensuring that all subjects presented in the study are presented in SDTM.DM, identifying study-based approaches in the program, ensuring that all column values exceeding 200 are mapped to SUPP

SCENARIO 1: SUBJECTS IN ANY RAW DATASET MUST BE PRESENT WITHIN SDTM.DM

Programmers need to merge SDTM.DM dataset for the development of all other SDTM dataset and there are chances that subjects in external data are not in DM:

```
data lb;
  merge sdtm.dm(in=a) raw_lb(in=b);
  by subjid;
  if b;
  if b and not a then putlog "ERROR: Subject " subjid "is not in SDTM.DM";
run;
```

Log: If a subject is not presented in SDTM.DM:



```
36   data lb;
37     merge sdtm.dm(in=a) raw_lb(in=b);
38     by subjid;
39     if b;
40     if b and not a then putlog "ERROR:Subject " subjid "is not in SDTM.DM";
41   run;

ERROR:Subject XXX-0102 is not in SDTM.DM
```

SCENARIO 2: TO ENSURE THAT THE RESULT WILL BE POPULATED FOR THE TEST WHEN THE PERFORMANCE STATUS IS 'YES'

In certain scenarios it is possible that the data from EDC can be collected in such a manner that there can be 2 datasets for a single object, one containing the performance status of a particular examination and second dataset containing the respective results of the tests. Programmers can use the above approach to ensure that all the subject with performance status “Yes” has a corresponding value in the external data.

SCENARIO 3: ENSURING THE AUTHENCITY OF THE DERIVED BASELINE FLAG

Derivation of the baseline flag depends on how it's described in the protocol. Usually, this might be the last non-missing measurement prior to first administration of study drug. So, programmers could use the DATA or PROC SQL step to identify this record and merge it back with the original record. It is necessary for the programmer to ensure that there are no multiple baseline flags for a test within a subject populated. The code provided below checks the same. For this, first we need to derive count of the baseline based on the dependent variables:

```
proc sql;
  create table basechk as
  select distinct usubjid,lbcat,lbscat,lbtestcd,lborres,lbdtc,count(lbblfl)
as cnt
  from lb5
  where ~missing (lbblfl)
  group by usubjid,lbcat,lbscat,lbtestcd;
```

```
quit;


data null;
  set basechk;
  if cnt gt 1 then put "ERROR: Multiple baseline populated for Subject="
usubjid "and TEST=" lbtestcd;
run;
```

SCENARIO 4: ADDRESSING TEMPORARY APPROACHES USED WITHIN PROGRAMS PRIOR TO DB LOCK.

There may be situations where the programmer may be stuck or unable to move to the next step due to data issues or missing data sets. In such scenarios, the team can either stop processing or move forward with a temporary approach. Consider a case where multiple records are populated under a visit for an LBTEST with different sponsor defined identifier and the team needs confirmation that the record should be used for analysis. The programmer can select the maximum / minimum, first / last, etc. values for each category as an analysis record, and the temporary approach was to consider the last value for each visit:

```
data lb2;
  set sdtm_lb end=ls;
  by usubjid visit lbcat lbscat lbtest;
  if last.lbtest;
  if ls then put "WARNING: Temporary - Duplicates records not considered
for analysis";
run;
```

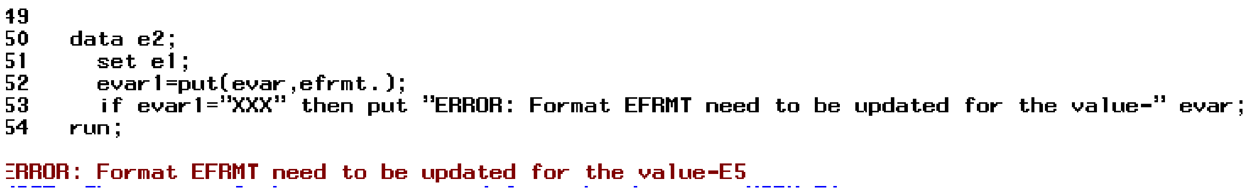
If the team has a standard log check macro that also works with hint words, then the check can also be provided as put "TEMPORARY -xxx".

SCENARIO 5: CHECK THAT ALL DATA VALUES ARE CONSIDERED IN THE PROGRAMMING FORMATS.

Programmers use user-defined formats for the generation of SDTM or ADAM that helps in reusability of programs for example - in case of an extension study. Yet there may be a risk that there might be new values added as part of the formats which can be missed out during updates. Suppose that the value "E5" available within Study 2 is not included in the formats because no such value has been recorded or listed in Study1. In such cases where the chances of missing these records are high, the below approach can be taken.

```
*Formats used for the example program;
proc format;
  value $efrmt
    "E1"="Example1"
    "E2"="Example2"
    "E3"="Example3"
    "E4"="Example4"
    other="XXX";
run;


data e2;
  set e1;
  evar1=put(evar,efrmt.);
  if evar1="XXX" then put "ERROR: Format EFRMT need to be updated for the
value-" evar;
run;
```

Log: Log report when used for study 2



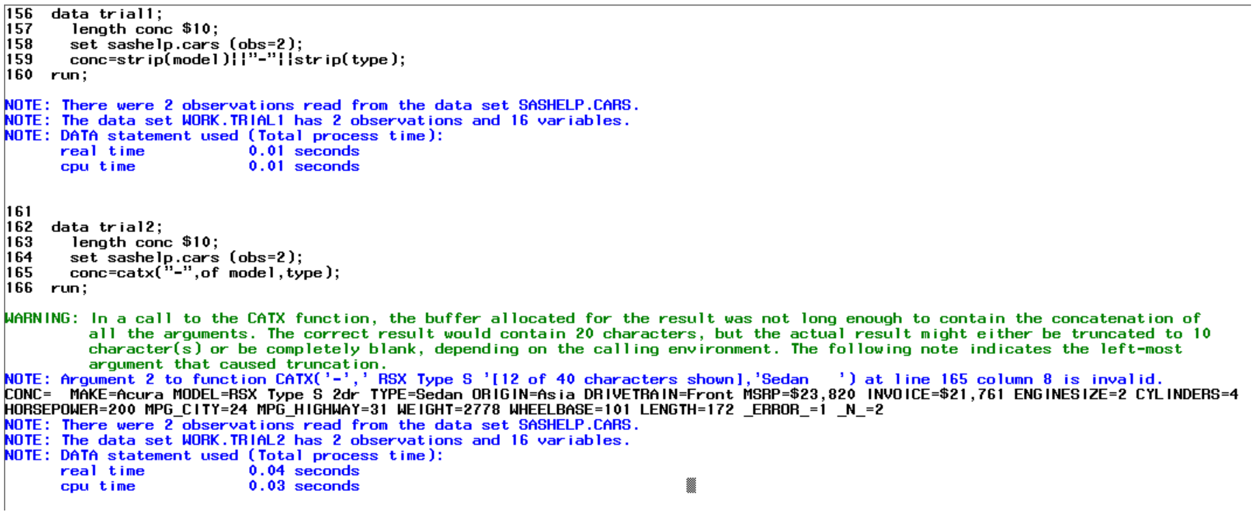
```
49
50    data e2;
51      set e1;
52      evar1=put(evar,efrmt.);
53      if evar1="XXX" then put "ERROR: Format EFRMT need to be updated for the value-" evar;
54    run;

ERROR: Format EFRMT need to be updated for the value-E5
```

SCENARIO 6: ENSURE THAT NO TRUNCATION OCCURS WHILE CONCATENATION

Concatenating two or more variables is a common occurrence in programming and programmers are exploring various ways to get outputs. It is the duty of the programmer to ensure that no truncation has occurred in programming. Without any custom check, programmer can ensure the quality with the use of cat function. The code given below is the two separate concatenation methods, but only the catx shows an alert when truncation occurs:

```
data trail1;
  length conc $10;
  set sashelp.cars (obs=2);
  conc=strip(model)||"-"||strip(type);
run;

data trail2;
  length conc $10;
  set sashelp.cars (obs=2);
  conc=catx("-",of model,type);
run;
```

Log: Log report of trial one and trial 2



```
156  data trial1;
157    length conc $10;
158    set sashelp.cars (obs=2);
159    conc=strip(model)||"-"||strip(type);
160  run;

NOTE: There were 2 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.TRIAL1 has 2 observations and 16 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


161
162  data trial2;
163    length conc $10;
164    set sashelp.cars (obs=2);
165    conc=catx("-",of model,type);
166  run;

WARNING: In a call to the CATX function, the buffer allocated for the result was not long enough to contain the concatenation of
         all the arguments. The correct result would contain 20 characters, but the actual result might either be truncated to 10
         character(s) or be completely blank, depending on the calling environment. The following note indicates the left-most
         argument that caused truncation.
NOTE: Argument 2 to function CATX('-',' RSX Type S '[12 of 40 characters shown],'Sedan   ') at line 165 column 8 is invalid.
CONC=  MAKE=Acura MODEL=RSX Type S 2dr TYPE=Sedan ORIGIN=Asia DRIVETRAIN=Front MSRP=$23,820 INVOICE=$21,761 ENGINESIZE=2 CYLINDERS=4
HORSEPOWER=200 MPG_CITY=24 MPG_HIGHWAY=31 WEIGHT=2778 WHEELBASE=101 LENGTH=172 _ERROR_=1 _N_=2
NOTE: There were 2 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.TRIAL2 has 2 observations and 16 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

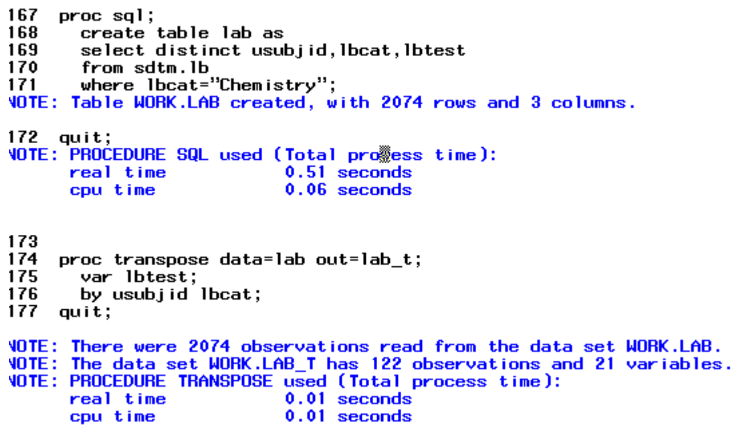
SCENARIO 7: TO ENSURE THAT ALL THE VARIABLES RESULTING FROM PROC TRANSPOSE ARE CONSIDERED IN FURTHER PHASES.

Concatenating variables resulting from the process of transposing are also common in programming and are most frequently used in the generation of patient narratives and listings. If there are unique Identifier Parameters (IDs) for transposing, there would be no problem. If there is no Identifier, the resulting variables will be in the form col1, col2, etc. and, in these situations, the programmer must typically manually define the last variable and construct the program to use up to that variable. So, here's a way to define the last variable and construct a program based on it.

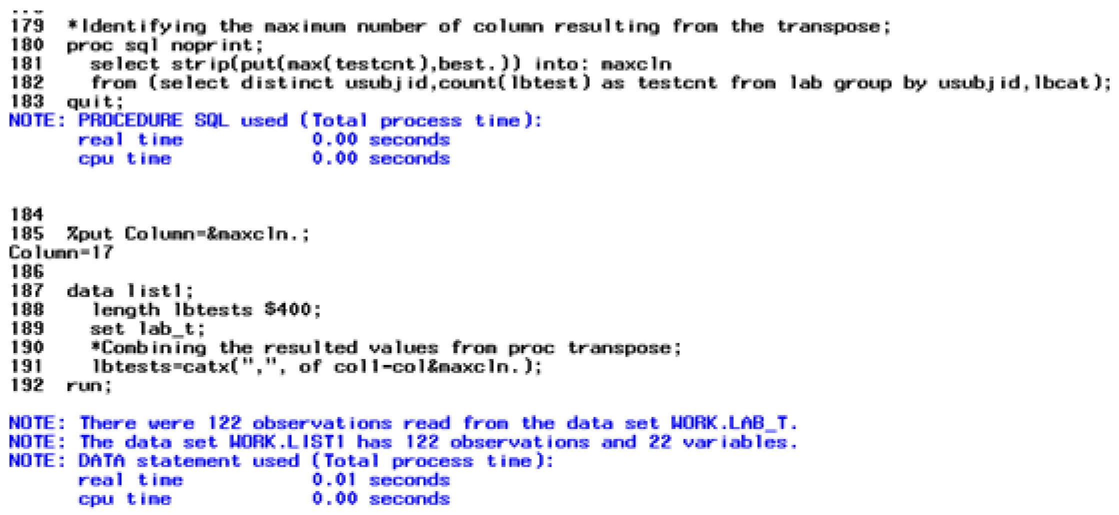
The example given below is to illustrate the specific laboratory test presented for the subject under the lab category ‘Chemistry’ in the column called LBTESTS. The laboratory measurements are separated by a comma:

```
*Identifying the distinct lab test populated for each subject under the
category Chemistry;
proc sql;
  create table lab as
  select distinct usubjid,lbcat,lbtest
  from sdtm.lb
  where lbcat="Chemistry";
quit;

proc transpose data=lab out=lab_t;
  var lbtest;
  by usubjid lbcat;
quit;

*Identifying the maximum number of columns resulting from the transpose;
proc sql noprint;
  select strip(put(max(testcnt),best.)) into: maxcln
  from (select distinct usubjid,count(lbtest) as testcnt from lab group by
usubjid,lbcat);
quit;

%put Column=&maxcln.;

data list1;
  length lbtests $400; *Length used for avoiding truncation;
  set lab_t;
  *Combining the resulted values from proc transpose;
  lbtests=catx(",", of col1-col&maxcln.);
run;
```

Log:



```
167  proc sql;
168    create table lab as
169    select distinct usubjid,lbcat,lbtest
170    from sdtm.lb
171    where lbcat="Chemistry";
NOTE: Table WORK.LAB created, with 2074 rows and 3 columns.

172  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.51 seconds
      cpu time            0.06 seconds


173
174  proc transpose data=lab out=lab_t;
175    var lbtest;
176    by usubjid lbcat;
177  quit;

NOTE: There were 2074 observations read from the data set WORK.LAB.
NOTE: The data set WORK.LAB_T has 122 observations and 21 variables.
NOTE: PROCEDURE TRANSPOSE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```



```
...
179   *Identifying the maximum number of column resulting from the transpose;
180   proc sql noprint;
181     select strip(put(max(testcnt),best.)) into: maxcln
182     from (select distinct usubjid,count(lbtest) as testcnt from lab group by usubjid,lbcat);
183   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


184
185   %put Column=&maxcln.;
Column=17
186
187   data list1;
188     length lbtests $400;
189     set lab_t;
190     *Combining the resulted values from proc transpose;
191     lbtests=catx(",", of col1-col&maxcln.);
192   run;

NOTE: There were 122 observations read from the data set WORK.LAB_T.
NOTE: The data set WORK.LIST1 has 122 observations and 22 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

There are some common scenarios in which programmers can make mistakes, and there are a variety of other situations in which programming precautions are required.

IMPORTANCE OF STANDARD MACROS

Macros offer an excellent way to automate the process. Most companies typically use macros for their programming operations, which will certainly help to minimize time and boost reliability. Standard macros are macros that have clear definitions and are checked for all potential causes that the standard macros have preferences over the macros generated by the programmer within the programs. The need for standard macros in programming cannot be avoided. Some of the Standard macros used within programming for easiness and quality check are the log check macro, attributes macro, reporting macros etc. Below are described in detail, some of these standard macros that can be applied on a day to day basis.

MACRO 1: VALIDATION OF THE PROGRAMMING LOG

To ensure that the program runs as expected, log files should be checked for errors, warnings and other messages. Most programmers check the log interactively during programming, in order to obtain clean, error-free, code. However, if data changes or programs are executed in a batch environment, it is particularly important to check all log files after a data sets production process or a report. It is also a difficult and time-consuming task to check every program log for a custom error or warning. So, the necessity of log checks macro for log validation is unavoidable, and with the help of online references, the programmer can easily create it. We recommend using the reference which also allows to check log by keyword.

MACRO 2: TO ENSURE THAT ALL VARIABLE LENGTHS OVER 200 ARE PROPERLY MAPPED

As per SDTM standard user can populate 200 characters in free text variables, rest will be mapped into SUPP. SDTM programming might begin before the database lock and there is a high risk of modifying the data from the previous extract. As part of multiple data extract, it is the responsibility of the programmer to ensure that all the variable values are successfully mapped to SDTM domain. How does the programmer ensure a variable with a length of 200 or more is successfully mapped? We recommended the creation of a macro to ensure that the values are correctly mapped. Currently no macros are available for the same, the method provided below is an option for ensuring the quality in such situation.

The 'VARIDENTIF' is a variable ID macro to indicate whether the variable in a library is longer than the provided length.

- The macro will identify the variable with a length longer than the specified length and output to the specified location (provided in the macro) in the name 'Variables List,'

- The programmer will then have to check the variables against the specification,
    - Populate value 'Y' in SDTMVAR column, if the variable is correctly mapped.
    - If the variable is [NOT SUBMITTED], populate value 'Y' in RMVDVAR column.
- For the next run onwards, the program will read the value from the existing file and populate a warning when a new variable is identified or missing in both SDTMVAR and RMVDVAR.

The input required for the macro are divided into 3, DIR-is location of Variables List.csv. LIB- input library used for the process; default value is ‘raw’. LEN is the value used for comparison: default value is 200:

```
%macro varidentif(dir=, lib=raw,len=200);
  %*Step 1.Identifying the variables with length more than 200;
  data varlen;
    set sashelp.vcolumn;
    %*Subsetting the data based on the input from macro;
    where libname=upcase("&lib") and type="char" and length gt &len.;
    %*Removing unwanted files, programmer can update this section based on 
the data;
    if find(name,"_raw","i") then delete;
    %*Flag used to ensure the variable mapping;
    sdtmvar='';
    rmvdvar='';
    keep memname name sdtmvar rmvdvar;
  run;

  %*2.Outputing this in a external location, if already existed step 1a 
will work else step 2;
  %*Step 1a - Checking whether the required file exists or not;
  %if %sysfunc(fileexist(&dir\Variables List.csv)) %then %do;
    %put File already exist;
    proc import file="&dir\Variables List.csv"  dbms=csv out=varlen_old 
replace;
      getnames=yes;
      guessingrows=100000;
    run;

    proc sort data=varlen;
      by memname name;
    run;

    proc sort data=varlen_old;
      by memname name;
    run;

    data varlen1;
      merge varlen(in=a) varlen_old(in=b);
      by memname name;
      if a and not b or cmiss(sdtmvar,rmvdvar)=2 then put "WARNING: 
variable- " name “need to check against specification”;
    run;

    %*Replacing the external file with latest information;
    proc export data=varlen1 dbms=csv outfile="&dir\Variables List.csv" 
replace;
    run;
  %end;
    %*Step 2;
```

```
      %else %do;
        proc export data=varlen dbms=csv outfile="&dir\Variables List.csv"
replace;
        run;
     %end;
%mend varidentif;
```

MACRO 3: ATTRIBUTE GENERATION FROM SPECIFICATION FOR SDTM/ADAM

The generation of variables attributes with the specification plays an important role in the SDTM and ADaM development. There is a hidden probability that programmers may skip the final stage changes in the specification or there is a possibility of losing changes from the specification in the case of programming reusability. Online reference for macro attributes is also available. The programmer can create a set of dummy databases with zero observations using the attributes macro. To check that all variables are present in the final datasets are only from the specification it is advisable to construct an additional macro to compare data sets from the attribute macro and final SDTM / ADaM data sets.

MACRO 4: MACROS FOR AUTOMATING FORMATS

User defined formats are widely used for the generation of SDTM/ADaM variables and most commonly programmer will create formats manually. This has also some higher probability of hidden risk. In SDTM Development, for a few variables team can include custom codelist when code for that in the controlled terminology is extensible. So, there's a chance of updating the codelist in the specification at a later stage. Thus, the importance for a macro for automating the formats is high. The first task of the programmer is to add an additional variable called 'Original Value' (contains the values in the raw datasets against that coded value) to the specification code list tab. Now the codelist tab contains the information like variables ID, decoded value and original value. Now instead of creating the format manually, we can create a SAS macro for formats based on the decoded value and the original value directly from the specification

CONCLUSION

Not all Data Issues are identified at the initial stage of Source Data Validation, but they tend to surface during the development of CDISC datasets. In addition, certain data issues identified at the initial run might not always be necessarily resolved in the next data transfer. This is where custom errors / warnings play a significant role. Similarly, in case of Statistical programming, a standard code might be replicated for various TFLs with changes only to the parameters considered. In such cases, specific custom checks based on parameters also comes into importance. It is highly recommended that these kinds of custom checks be carried out in automation programs because, if quality fails, everything fails.

ACKNOWLEDGMENTS

The content, ideas and recommendations presented in this paper are all developed from experiences in our career. These experiences come through previous companies, various industry leaders, colleagues, mentors, conferences, and direct experience. Any brand and product names are trademarks of their respective companies.

CONTACT INFORMATION

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

Name	: Akhil Vijayan
Enterprise	: Genpro Research PVT LTD
Phone	: 781-373-8455
E-mail	: akhil.vijayan@genproresearch.com
Web	: www.genproresearch.com

Name	: Limna Salim
Enterprise	: Genpro Research PVT LTD
Phone	: 781-373-8455
E-mail	: limna.salim@genproresearch.com
Web	: www.genproresearch.com

Name	: Anoop Ambika
Enterprise	: Genpro Research PVT LTD
Phone	: 781-373-8455
E-mail	: anoop.ambika@genproresearch.com
Web	: www.genproresearch.com