PharmaSUG China 2019 - Paper DS-075

# Using SAS macro to automatically update study specific information in SDTM spec for generating Define.xml

Junkai Zhao, Paul Burmenko, Everest Clinical Research

## ABSTRACT

In this paper, a SAS macro will be introduced to help programmers to update some SDTM datasets' specific information (e.g., DATA TYPE, SIGNIFICANT DIGITS or FORMATS) automatically in the SDTM specification for generating Define.xml via Pinnacle 21.

An SDTM specification, which is generated based on study's SDTM XPT files, needed to be updated manually with some dataset specific information before it can be submitted via Pinnacle 21 Community to generate the Define.xml. The SAS macro can help programmers to update this information efficiently. In Variables sheet of the SDTM spec, the SAS macro will help programmers to check all variables in all SDTM datasets to understand every variable's Data Type, to derive significant digits if it is a numerical variable and to assign it a format. In ValueLevel sheet of the SDTM spec, the SAS macro will do the same things to check all values with their "Where Clause" and measure value's length of each variable as well. In Codelist sheet of the SDTM spec, the SAS macro will also help programmers to cross check from dataset's perspective if all value of SDTM controlled terminology (CT) is recorded correctly in the Codelists sheet of SDTM spec. For example, checking if the all necessary CTs from SDTM datasets are included and checking if all NCI codelist code value and decoded value are assigned correctly. The logic to determine whether a variable has a codelist will also be discussed in this paper.

Compared to conducting these assignments manually, efficiency will be improved in submission work and human errors will be minimized for publishing a correct Define.xml by using the SAS macro to automatically update this critical variable- and value-specific information.

## INTRODUCTION

When we try to generate a SDTM Define.xml file via Pinnacle 21, assuming we did not write SDTM specifications before programming began, an SDTM specification in an Excel file can be generated at first by Pinnacle 21 based on the SDTM datasets in the XPT files. Therefore, data specific information in the SDTM datasets could be extracted and recorded in the SDTM specification. Take the "Variables" sheet as an example— Pinnacle 21 will examine every variables' name, label, data type, length, significant digits and format of every SDTM domain and fill in the related columns of "Variables" sheet of the spec. However, there is still a lot of SDTM information to be manually checked and filled in by the user. For example, the entire "ValueLevel" sheet is blank and users have to manually update all the information, like 1) categorical variables (LBCAT), 2) original result of finding variable (LBORRES), and 3) data value variables of supplemental dataset (QVAL). The process is quite time-consuming for users to identify suitable variables, to write the "where clause" of these variables and to check these variables' data type, whether it is text, integer, float or datetime, display format, and length. In this paper, a SAS macro will be introduced to help programmers to automatically check which value-level variables and values are required to be extracted to the ValueLevel sheet of the SDTM spec and automatically assigned required information into the ValueLevel sheet.

## CODE STRUCTURE

The macro will firstly use PROC CONTENTS to read all variables information of all SDTM datasets. The program assumes that all datasets are structured according to the SDTM implementation guide. Then the macro will use the PRXMATCH function to scan expected variables for the ValueLevel sheet to obtain names of categorical variables, sub-categorical variables, original result of finding variables and data values of supplemental variables. The key word used to scan are the SDTM variable fragments "TSPARMCD", "DSDECOD", "CAT", "SCAT", "TEST", "ORRES" and "QNAM." The core part of this macro is to compile a loop PROC SQL procedure by CALL EXECUTE of inside a DATA STEP. Unfortunately, the SAS Editor syntax highlighting makes the code difficult to read, since SAS processes the quoted CALL EXECUTE parameters as character values rather than SAS code. While difficult to read, this method ensures that each desired variable will be checked by PROC SQL automatically and the user can use the code to conduct the following actions (Appendix 1):

- Find the longest value per category, sub-category, and test to derive the length information
- Identify and skip null values
- Identify float values if only a decimal point is left after removing all digits
- Identify integer values if nothing is left after removing all digits
- Identify datetime values if a valid datetime value is present.
- Identify text if no other types are assigned
- For float type's significant digits, count the digits after the decimal
- For float type's display format, count the digits before and after the decimal
- Process each unique value by category, sub-category, and test

Finally, this information will be summarized and output as an excel file whose structure is similar to the structure of the Pinnacle21 Define.xml input template for the user to update the SDTM specifications in this template. Manual review of the macro's output is required because the SDTM Define.xml must report all allowable values in the value level metadata and codelists sections. The user should carefully review the macro output against their understanding of the data. For example, if --ORRES or QNAM contains only the value "2004," then the macro will assign a data type of integer with a length of 4 to this result, when it may be a partial datetime value. Similarly, if --ORRES or QNAM contains only the value "2004-03," then the macro will assign a data type of text with a length of 7 instead of partial datetime. The user must always carefully review the data since full automation is not possible.

By using CALL EXECUTE as the macro's core, not only the ValueLevel could be updated automatically, but also other user-defined metadata could take this macro as reference to mimic the way how to extract info from datasets automatically and precisely.

In WhereClauses sheet, value of all columns (ID, Datasets, Variables, Comparator and Value) of this sheet will be assigned based on the WHERE CLAUSE column of the ValueLevel Sheet. For example, if the value of "Where Clause" in ValueLevel sheet is "EG.EGTESTCD.EQ.QRS", the macro will assign "EG.EGTESTCD.EQ.QRS" to ID, "EG" to Dataset column, "EGTESTCD" to Variable column, "EQ" to Comparator column and "QRS" to Value column in WhereClauses sheet.

In Codelist sheet of the SDTM spec, the SAS macro will also help users to cross check from the dataset's perspective if all values of SDTM controlled terminology (CT) are recorded correctly in the Codelists sheet of SDTM spec. The macro will help users to take SDTM implementation guide as reference to check if a variable needs a codelist. The macro will firstly output a CSV file with all variables in all SDTM datasets.

Then users can give a flag to those variables that should have a codelist, outputting a CSV file. The macro will read the modified CSV file to select these variables to further extract their distinct values. (Appendix 2) Therefore, even though there is always an element of checking the CRF and other source metadata to include the completed codelist of a variable, users don't need to manually check the distinct value of that variable that exist in the SDTM datasets and manually write them in the codelist sheet. Take AESEV as example, the macro will pick "MILD" and "MODERRATE" correctly and automatically from SDTM.AE and user still need to check Case Report Form to include "SEVERE" into the codelist even if "SEVERE" is not recorded in SDTM.AE. In addition, the macro will also merge SDTM terminology file to help user to check if all NCI codelist code values and decode values are assigned correctly to the related codelist value.

In the Methods sheet, value of ID, Name and Type of this sheet could be assigned based on the method column of Variables Sheet. For example, the value of the "Method" column in Variables Sheet of USUBJID in SDTM.AE is "AE.USUBJID". Then the macro will assign "AE.USUBJID" to the ID column of Method sheet, assign "Algorithm to derive AE.USUBJID" to the Name column of Method sheet and assign "Computation" to the Type column of Method sheet.

## CONCLUSION

Compared to conducting these assignments manually, efficiency will be improved in submission work and human errors will be minimized for publishing a correct Define.xml by using the SAS macro to automatically update this critical variable- and value-specific information.

## REFERENCES

H. Ian Whitlock. "CALL EXECUTE: How and Why" Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi22/CODERS/PAPER70.PDF.

Bob Virgile."MAGIC WITH CALL EXECUTE" Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi22/CODERS/PAPER86.PDF.

## CONTACT INFORMATION

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

Junkai Zhao
Everest Clinical Research
junkai.zhao@ecrscorp.com

## APPENDIX 1

```
%macro valuelevel(inlib=sdtm);

*Obtain all dataset and variable metadata from input library;
proc contents data=&inlib.._all_ out=cont noprint;
run;

*Program assumes that dataset is structured according to Implementation Guide;
*With Category Sub-category Test Result variables appearing in exactly this order;
proc sort data=cont;
       by memname varnum;
run;

*Identify category, sub-category, test, and result variables for vertical structures;
data cont;
       length cat scat testcd test value $200;
       retain cat scat testcd test value "'";
       set cont;
       by memname varnum;
       where prxmatch("m/CAT|SCAT|TEST|QNAM|ORRES|QVAL/oi",name) > 0 and
prxmatch("m/ORRESU/oi",name) = 0;

  *Initialize variables - missing should appear with redundant quotes for Call Execute facility;
       if first.memname then do;
              cat="'"; scat="'"; test="'"; value="'";
       end;

       *Assign category, sub-category, and result variables;
       if prxmatch("m/CAT/oi",upcase(name)) then cat=name;
              else if prxmatch("m/SCAT/oi",upcase(name)) then scat=name;
              else if prxmatch("m/TESTCD|QNAM/oi",upcase(name)) then testcd=name;
              else if prxmatch("m/TEST|QNAM/oi",upcase(name)) then test=name;
              else if prxmatch("m/ORRES|QVAL/oi",upcase(name))     then do;
                     value=name;
                     output;
              end;

run;

%mend;

%valuelevel(inlib=sdtm);

data _null_;
       length crit: type: bygrp $100 dataset catvar scatvar testvar resultvar $20;
       set cont;
       *Assign variable names and values and special CRIT and TYPE logic variables for Call
Execute facility;
       dataset='"'||strip(memname)||'"';
       catvar='"'||strip(cat)||'"';
       scatvar='"'||strip(scat)||'"';
       testvar='"'||strip(testcd)||'"';
       resultvar='"'||strip(value)||'"';
       *Numeric values have digits;
       crit_nums="'1234567890'";
       *Float values have a decimal point left when digits are removed;
       crit_float="'.'";
       *Integer values have nothing left when digits are removed;
       crit_integer="'";
       *Check for values that are all missing;
       *Assign XML types of text, datetime, integer, and float;
       type_miss="'all values are missing - check!'";
       type_text="'1.text'";
       type_datetime="'2.datetime'";
       type_float="'3.float'";
       type_integer="'4.integer'";
```

```
        *Each dataset will be processed by category, sub-category, and test;
    bygrp=catx(",",cat, scat, testcd, test);

    *Loop through each input library dataset to create value level metadata;
         call execute(
                'proc sql noprint;
                 create table test'||strip(put(_n_,best.))||' as
                 select distinct
                 '||strip(testcd)||' as testcd length=8, '
                  ||strip(dataset)||' as dataset length=8,'
                  ||strip(catvar)||' as catvar length=8,'
                  ||strip(scatvar)||' as scatvar length=8,'
                  ||strip(testvar)||' as testvar length=8,'
                  ||strip(resultvar)||' as resultvar length=8,'
                  ||strip(cat)||' as cat length=200,'
                  ||strip(scat)||' as scat length=200,'
                  ||strip(test)||' as test length=40,

/*find the longest value per category, sub-category, and test*/
        max(length(strip('||strip(value)||'))) as length,

/*identify values that are all null and check them*/
        case when max(compress('||strip(value)||')) eq '||crit_integer
        ||' then '||strip(type_miss)||'

/*identify float values if only a decimal point is left after removing all digits*/
        when compress('||strip(value)||','||strip(crit_nums)||') eq '||strip(crit_float)
        ||' then '||strip(type_float)||'

/*identify integer values if nothing is left after removing all digits*/
        when compress('||strip(value)||','||strip(crit_nums)||') eq '||strip(crit_integer)
        ||' then '||strip(type_integer)||'

/*identify datetime values if a valid datetime value is present -- experimental and may need
edits*/
        when input('||strip(value)||',anydtdte12.) ne . then '||strip(type_datetime)||'

/*identify text if no other types are assigned*/
        else '||strip(type_text)||' end as type,

/*for float type significant digits, count the digits after the decimal*/
        case when calculated type='||strip(type_float)
        ||' then length(scan(strip('||strip(value)||'),2,'||crit_float||'))
        else . end as sigdig,

/*for float type format, count the digits before and after the decimal*/
        case when calculated type='||strip(type_float)
        ||' then (strip(put(length('||strip(value)||'),best.))
        ||"."||strip(put(calculated sigdig,best.)),best.)
        else . end as format

         from &inlib..'||strip(memname)||'
        /*process each unique value by category, sub-category, and test*/
        group by '||strip(bygrp)||';
        quit;
        ');
run;
```

## APPENDIX 2

```
proc contents data=&inlib.._all_ out=cont noprint;
run;

data codepick;
        set cont;
        keep memname name codelist;
        codelist="";
```

```
run;

proc export data=codepick
        outfile=' output_area/codepick.csv'
        dbms=csv
        replace;
run;

proc import
        out=codepick
        datafile='output_area/codepick.csv'
        dbms=csv
        replace;
        getnames=yes;
run;

data _null_;
        set codepick(where=(codelist="Y"));
        length dataset $20;
        dataset='"'||strip(memname)||'"';
        vname='"'||strip(name)||'"';
        call execute('
                                proc sql noprint;
                                  create table test'||strip(put(_n_,best.))||' as
                                  select distinct
                                  '||strip(dataset)||' as dataset length=8,
                                  '||strip(vname)||' as vname length=8,
                                  '||strip(name)||' as value length=200
                                  from &inlib..'||strip(memname)||';
                                quit;
                        ');
run;
```