phuse·connect
connect·share·advance
March 8th–11th
Renaissance Orlando at SeaWorld Florida
US 2020
The Clinical Data Science Conference

Paper CT01

An automated code translation approach in SDTM specifications using %translate macro

Hrideep Antony, Syneos Health, Morrisville, North Carolina, USA
Aman Bahl, Syneos Health, Burlington, Ontario, Canada

ABSTRACT

This paper presents a standardized language specification writing approach that increases the overall efficiency of the SDTM programming. This SAS macro-based system invokes a %translate SAS macro which will scan through the SDTM specification document and identify all the variables for which the derivation can be translated to a SAS code and process those variables directly. It can be observed that around 50 percent of SDTM variable derivations involve directly reading raw data variables or a simple subset of data from the source. The %translate macro can also read the formats from control terminology and apply the formats to the data when applicable. The capabilities of this utility macro are currently being expanded with the ability to produce categorical and numerical summary statistics based on a one-line script which the users such as statisticians, medical writers, and content writers will be able to use to produce summaries effortlessly, without the need of any programming.

INTRODUCTION

The principal aspect of using a standardized language specification writing methodology is the vision to represent the variable derivations in a way that can be programmatically translated to an executable SAS code.
A more sophisticated means of achieving this could be by using machine learning models and artificial intelligence for language translation methods. However, this paper suggests a pragmatic SAS programming approach to reduce redundancy in SDTM programming by standardizing the specification documents.

The overall process flow of this utility macro can be found in the flow chart below (Figure 1).

The key process of the utility involves Identifying and converting the variable derivations that can be transformed into an executable SAS code. This paper will provide a few scenarios where such a conversion can be easily and practically done. This utility can be further expanded as more and more standardization is implemented in the specification document.

The functionality steps of this utility will be further discussed in this paper

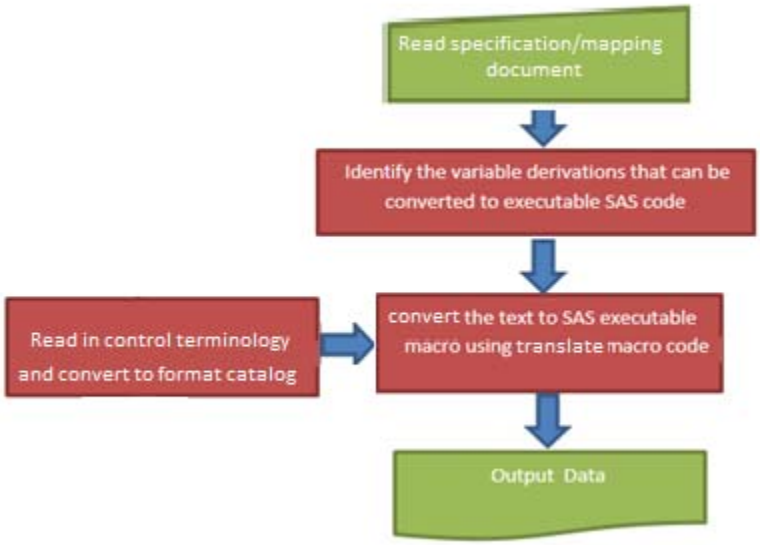

Figure1: Process flow chart

SPECIFICATION LANGUAGE STANDARDIZATION

The Specification language standardization is the key aspect to make the utility efficiently. With the introduction of CDISC , the database itself became more and more standardized. However, the language that is used in the specifications to illustrate derivations may not have reached that same level of standaridization that can be programmatically analysed.

For example, let us consider the derivation of the RFICDTC (Date/Time of Informed Consent) in the DM domain. Let us consider that this variable needs to be derived from the source variable 'SBJDTM" where INFTTYPE='ICDT' is located in the raw data 'SUB INFO'.

There are several ways to represent this in the specification document and each specification writer may have his or her own style of illustrating the derivation as shown in Table 1.

Index	ALGORITHM
1	Equal to SBJDTM in raw data SUBINFO where INFTTYPE value='ICDT'
2	Set to SBJDTM in raw data SUBINFO when INFTTYPE value='ICDT'
3	Set to SUBINFO. SBJDTM where INFTTYPE ='ICDT'

Table 1

It can be noticed that index 3 is a more standardized approach relative to the other two derivations as the representation is more direct with the usage of the period to separate the data and the variable and the subset condition uses a Where statement. Such a statement can easily be programmatically converted to an executable SAS code. The raw data to SDTM format output for this example can be seen in Figure 2.

Let's consider a few example scenarios as shown in Table 2 below where the %translate macro will identify and translate text from the specification file to executable SAS code.

DATASET	VARIABLE	LABEL	ALGORITHM	Transaleted SAS code language
DM	DOMAIN	Domain Abbreviation	Set to 'DM'	translate to : DOMAN="DM"
DM	RFXSTDTC	Date/Time of First Study Treatment	Set to RFSTDTC	translate to : RFXSTDTC=RFSTDTC
SUPPDM	ETHYSC5	Cuban	Set to DMx1009.ETHNICSCAT_35	translate to : data _trans_1; set DMx1009; ETHYSC5=ETHNICSCAT_35;run;
DM	DTHFL	Subject Death Flag	SET to 'Y' if DSx001.DSDECOD = 'DEATH'	translate to : data _trans_4; set DSx001; if DSDECOD ="DEATH" ; DTHFL='Y'; run;

Table 2

In the first scenario, the algorithm starts with a simple "Set to". The algorithm looks for the succeeding text after the "Set to" statement. If the text is within a quote, it gets translated to the variable="TEXT" as in the first example with the DOMAIN variable where it gets assigned to "DM".

In the second scenario, we are considering the algorithm that starts with "Set to" but has a text without quotes as in the example of RFXSTDTC variable. In this case, the value of the variable named RFSTDTC will be assigned to RFXSTDTC.

In the third scenario, the derivation algorithm has a "period" and the algorithm begins with a "Set to". Translate macro will convert this statement to a data step as shown in the translated SAS code language column in table 2 In this example, the ETHYSC5 variable gets assigned to the variable ETHNICSCAT_35 which is in the DMx1009 dataset.

In the fourth scenario, the DTHFL flag derivation begins with a “Set to” and uses a subset condition that is identified by “if” or “where” statements and this gets translated to corresponding SAS code as shown in figure 2. In this example, the subset condition if DSDECOD=”DEATH” is being used and the value ‘Y’ gets allocated to the DTHFL variable.

The overall programming process steps for the translate utility macro and the key programming components are explained in the five steps below.

Step 1: Read in the specification documents and identify the derivations that can be programmatically processed without needing additional programming efforts. In this step, the derivation algorithm will be split by words and each of the words is analyzed along with the sequences to identify the corresponding derivation programming statement as shown in the example code below.

```
%*=============================================================================================;
 *     Read in the specification document and convert the file to a SAS dataset         ;
 *%*=============================================================================================;

libname spec xlsx "&specpath" ;

data spec1;
set spec.&specname;;
 *%*=============================================================================================;
 *    read in the Derivation algorithm and process the data         ;
 *%*=============================================================================================;
STD_alg=trim(&algorithm);
rev_std_alg=compress(reverse(STD_alg));
*%*=============================================================================================;
 *  Split the text in the algorithm to multiple split variables for identifing the derivation logic.
    Note that the maximum allowable split words are 20 words in the cuurent version.
    The future versions of the utility will allow more texts to be processed ;
 *%*=============================================================================================;
array split (*)$50 split1-split20;
 do i = 1 to 20;
   split(i)= scan(STD_alg,i);
end;
```

Step 2: A “Type” variable is assigned with a value of “Internal” if the logic does not involve external library references and a Value of “External” if the derivation logic involves external file references. The Type variable serves as an identifier for the future programming steps. In the current example, the “Set to” is considered as an assignment operator of corresponding SAS equivalent syntax.

```
*%*=============================================================================================;
 *  Here is an example of how words such as SET and TO can be proceesed to sas syntax ;
 *%*=============================================================================================;
if upcase(split1) ='SET' and upcase(split2)='TO'  then do;
   if  SPLIT3=:'"' and rev_std_alg=:'"'  then sas_code= trim(variable)!!'=' !! substr(STD_alg,index(upcase(STD_alg), '"') );
   type='Internal';
    lib=' ';
  end;
*%*=============================================================================================;
 * The External Type are those the needs data from the external library
 *%*=============================================================================================;
if upcase(split1) ='SET' and upcase(split2)='TO' and substr(SPLIT3,1,1 ) ne '"' and index(STD_ALG,'.') and count( STD_ALG ,'.')=2
   and split9= '' then do ;
 data_nm= reverse(scan(reverse(STD_ALG),2));
  lib=SPLIT3;
  sas_code=  trim(variable)!!'=' !!   reverse(scan(reverse(STD_ALG),1)) ;
  type='External';
 end;
```

```
*%*=============================================================================================================;
* The Internal Type derivations does not involve external data source and can be derived within the data step
*%*=============================================================================================================;

 if upcase(split1) ='SET' and upcase(split2)='TO' and substr(SPLIT3,1,1 ) ne '"' and index(STD_ALG,'.') and
count( STD_ALG ,'.')=1 and split9= '' then do ;
   data_nm= reverse(scan(reverse(STD_ALG),2));
   lib=' ';
   sas_code=  trim(variable)!!'=' !! reverse(scan(reverse(STD_ALG),2))!!'_'!! reverse(scan(reverse(STD_ALG),1)) ;
  type='Internal';
 end;
 if sas_code ne '';
run;
```

Step 3: In this step, additional macro variables are created for each external and internal file references as shown in the programming steps below. These macro variables serve will be called in step 5.

```
*%*=============================================================================================================;
* Assign macro variables with numbers to each of the External referenced variables
*%*=============================================================================================================;
data external/*(keep= type DATA_NM lib )*/;
set spec1 end=end;
count+1;
where type='External';
  call symputx('sasfile'||put(count,4.-l), compress(DATA_NM));
     call symputx('lib'||put(count,4.-l), compress(lib));
     call symputx('code'||put(count,4.-l), trim(SAS_CODE));
     keepvar=substr(SAS_CODE,1,index(SAS_CODE,'=')-1);
    call symputx('keepvar'||put(count,4.-l), trim(keepvar));
   if end then call symputx('maxc',count);

run;

 *%*=============================================================================================================;
* Assign macro variables with numbers to each of the Internal variables
*%*=============================================================================================================;

data Internal/*(keep= type DATA_NM lib )*/;
set spec1 end=end;
count+1;
where type='Internal';
       retain  code keepvari;
       if _n_=1 then code=sas_code;
        else if sas_code ne '' then code= trim(code)!!'; '!!trim(sas_code);
     call symputx('code' , trim(code));

     keepvar =substr(SAS_CODE,1,index(SAS_CODE,'=')-1);
       if _n_=1 then keepvari=keepvar;
       else if keepvar ne '' then keepvari= trim(keepvari)!!'; '!!trim(keepvar );
    call symputx('keepvari'||put(count,4.-l), trim(keepvari));
   if end then call symputx('maxi',count);

run;
```

Step 4: Ensure that the external datasets that have been referenced exist using the %fileok SAS macro. SAS Pipe option is used with the filename statement to check the legitimacy of the file path as shown in the code below. The Pipe option along with "dir" and the file path will return a value ' dir: cannot access. <reference file path> ' if the dataset cannot be accessed in the specified location.

```
%put &maxc;
%let byvar= SUBJID;
%let getdata= ;
%let indata=;
 *%*=============================================================================================================;
 * The Fetchdata macro will fetch the data from the corresponding library
 *%*=============================================================================================================;
%macro fileok;
filename ckfile  pipe "dir &&read&i ";
data ckfile ;
   infile ckfile lrecl=200 truncover;
   input file_name_compare $100.;
run;
 %*=============================================================================================================;
  *If the corresponding dataset is not available in the library then stop the derivation     ;
 %*=============================================================================================================;
data ckfile;
set ckfile;
 if compress(lowcase(file_name_compare))=:"dir:cannotaccess" then fileok='N';
 else fileok='Y';
 CALL SYMPUT('Fileok', fileok);
run;
%put &fileok;
%mend fileok;
%macro fetchdata;
```

Step 5: In this final step the %fetchdata macro is been called. The macro is looped based on the number of variables that need to be derived. With each iteration for each variable. The &code&i macro variable will have the corresponding executable syntax using the new variables that are now derived and merged back with the original input data.

```
%macro fetchdata;
 %*=============================================================================================================;
  *The Fetchdata macro will be looped based on the number of variable derivations    ;
 * Each variable will be assigned with a unique macro variable
 %*=============================================================================================================;
 %do i=1  %to &maxc;
 %fileok;
  %if &fileok=Y %then %do;
  data &&sasfile&i(keep= &byvar &&keepvar&i);
   set &&lib&i...&&sasfile&i ;
   &&code&i;
%let getdata= &&sasfile&i &getdata;
   run;
 proc sort data= &&sasfile&i;
  by &byvar;
 run;
%end;
%end;
data &outdata;
merge &indata &getdata;
  by &byvar;
  &code;
 run;
%mend;
%fetchdata;
```

With reference to the original example in table 1, the variable RFICDTC gets derived using the %translate macro from the SBJDTM variable in the SUBINFO raw-data as shown in the figure 2.

SUBINFO Raw data

SUBJID	SBJDTM	INFTTYPE
6503	2020-01-08	ICDT
5672	2020-01-07	ICDT
5877	2020-01-07	ICDT
5593	2020-01-08	ICDT
5588	2020-01-07	RAND
6139	2020-01-07	RAND
5845	2019-12-17	RAND
5812	2019-12-06	RAND
5771	2019-12-05	RAND

Output dataset

SUBJID	RFICDTC
6503	2020-01-08
5672	2020-01-07
5877	2020-01-07
5593	2020-01-08

Figure2: Raw data to SDTM format output

CONCLUSION

By using the standardized language approach, the overall efficiency of SDTM database programming can be improved considerably. The future capabilities of the %translate macro will also include the ability to produce categorical and numerical summary statistics based on a one-liner script using which the users such as statisticians, medical writers, content writers etc. will be able to use to produce summaries effortlessly without requiring prior knowledge of SAS.

REFERENCES

Using Unnamed Pipes : SAS(R) 9.3 Companion for Windows, Available at URL:
support.sas.com/documentation/cdl/en/.../n16puwsro9pakqn1jamy1vwyaqx6.html

CDISC SDTM - An Automated Approach, Available at URL:
https://www.lexjansen.com/phuse/2018/si/SI11.pdf

ACKNOWLEDGMENTS

Sincere thanks to Steve Benjamin, Director Statistical Programming, Biostatistics and Global Contracts, for his vision, great leadership, persistent support, and encouragement throughout and for his valuable assistance in reviewing this paper.

CONTACT INFORMATION

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

Hrideep Antony
Principal Statistical Programmer, Clinical Division, Syneos Health
Work Phone: +1- 984 459 4785
Email: hrideep.antony@syneoshealth.com
Web: http://www.syneoshealth.com

Aman Bahl
Associate Director, Statistical Programming, Clinical Division, Syneos Health
Phone: +1-905 399 6715
E-mail: Aman.Bahl@syneoshealth.com
Web: http://www.syneoshealth.com