PharmaSUG 2017 - Paper DA03

Conversion of CDISC specifications to CDISC data – specifications driven SAS programming for CDISC data mapping

Yurong Dai, Jiangang Jameson Cai, Eli Lilly and Company

ABSTRACT

We'd like to introduce metadata driven approach utilizing sas programming techniques for SDTM and ADaM data mapping. Metadata extracted from specifications are converted into dataset's attributes, format, variable names and their order and sorting order for specification implementation in our reference code. It increases code's reusability, efficiency and consistency between data specifications and output data, and reduced re-work after data specification's update, during code development for SDTM mapping and ADaM datasets derivation.

INTRODUCTION

As CDISC data are required for NDA submission to FDA, it is very important to initiate an efficient way in SAS programming to keep high consistence between data specifications and output data for CDISC SDTM mapping and ADaM data derivation, in terms of speeding up submission and saving cost.

According to CDISC implementation guides, CDISC's data specs have to include 3 levels of metadata: data set, variable and value's metadata/definition. And the derived data set has to follow the specs. In order to avoid the repeat work in data derivation, our strategy was grabbing as most variable & value level's metadata as possible from specs into resulted data set. Detailed procedures are importing specification sheets to SAS data sets, and then automatically converting variables' attributes, variables' order, and variables used for sorting defined in specs into resulted data set, using our macros.

1. CONVERT THE SPECS FOR VARIABLE'S DEFINITION TO VARIABLE'S ATTRIBUTE

First we have variable level metadata or specification. An example can be as follows in Table 1:

VARIABLE	LABEL	ORDER	SASLENGTH	DISPLAY_FORMAT	SORTORDER	SASTYPE
STUDYID	Study Identifier	1	12		1	C
USUBJID	Unique Subject Identifier	2	22		2	C
AESEQ	Sequence Number	3	8		3	N
AEGRPID	Group ID	4	4			C
AEDECOD	Dictionary-Derived Term	5	200			C
AESOC	Primary System Organ Class	6	100			C
AETERM	Reported Term for the Adverse Event	7	200			C
AESEV	Severity/Intensity	8	8			C
AESEVN	Severity/Intensity (N)	9	8			N

ASEV	Analysis Severity/Intensity	10	10			C
ASEVN	Analysis Severity/Intensity (N)	11	8			N
AESER	Serious Event	12	1			C
AEREL	Causality	13	20			C
……	…….					
AESPID	Sponsor-Defined Identifier	19	4			C
ASTDT	Analysis Start Date	100	8	date9.		N
……	…….					

Table 1. Variable level metadata

1.1 A straight-forward way is to generate a macro as follows using the above metadata.

```
%macro specs;
 data all;
    attrib
    STUDYID label= 'Study Identifier' length = $12
    USUBJID label= 'Unique Subject Identifier' length = $22
    AESEQ label= 'Sequence Number'
    AEGRPID label= 'Group ID' length = $4
    AEDECOD label= 'Dictionary-Derived Term' length = $200
    AESOC label= 'Primary System Organ Class' length = $100
    AETERM label= 'Reported Term for the Adverse Event' length = $200
    AESER label= 'Serious Event' length = $1
    AEREL label= 'Causality' length = $20
    ……
    AESPID label= ' Sponsor-Defined Identifier' length = $4
    ASTDT label= 'Analysis Start Date' format = date9.
    ;
    set all;
run;
%mend specs;
```

Then call this macro at final programming step for generating this ADaM data set. So the specs for variable's definition become variable's attribute. This saves typing time if we copy the specs to sas code.

The following sas code creates the above sas macro %specs:

```
/*=====================================================================
the following macro is to generate a macro named as $specs, for adding
variable's label, format, length to variables defined in specs into
resulted final data
&specsdata: data set name for specs for variable's metadata
&dset: the data set name to be put on variable's attribute
&macrolib: the directory in which macro $specs to be output
=====================================================================*/
*FEXIST Function: Verifies the existence of an external file by its
 fileref;
*FDELETE Function: Deletes an external file or an empty directory;
```

```
%macro attrib(specsdata, dset);
   data specs; set &specsdata;
      length label_ $200; *this variable contains all attributes;
      if DISPLAY_FORMAT ^= " " and upcase(SASTYPE)^= "C" then
         label_ = "'"||strip(LABEL)||"' format = "||strip(DISPLAY_FORMAT);
       else if upcase(SASTYPE)= "C" and SASLENGTH >. then
         label_ = "'"||strip(LABEL)||"' length =
            $"||strip(put(SASLENGTH,best.));
       else label_ = "'"||strip(LABEL)||"'";
  run;
  filename specs "&macrolib.\specs.sas"; *Associates a SAS fileref with an
  external file;
  data _null_;
     set specs end=last;
     if fexist("specs") then rc=fdelete("specs");
     FILE "&macrolib.\specs.sas";
     if _n_=1 then do;
      put @1 "%nrstr(*This macro and file is output from macro %attrib;)";
      put @1 "%nrstr(*Generated or overridden when calling %attrib;)";
      put @1 "%nrstr(%macro specs;)";
      put @1 "data &dset;";
      put @5 "attrib";
     end;
    if label ^= ' ' then PUT @5  VARIABLE "label= " label_ ;
    if last then do;
      put @5 ";";
      put @5 "set &dset;";
      put @1 "run;";
      put @1 "%nrstr(%mend specs;)";
    end;
 run;
%mend attrib;
```

If our specs data set for variable's attribute is named as adlb (it is from variable's metadata), then call %**attrib**(specs, adlb) first; %specs is generated. Then we call %specs after our derived data set (named all, which is from raw data) is created, and then variable's attribute is put in our final CDISC data set.

1.2 Another of our ways is to create an empty dataset from variable level metadata - a data set with variable names and attributes without any record. Here is the sas code:

```
/*=========================================================================
```

The following macro is to create an empty dataset from the variable definition of specs.

&input_meta: the variable definition SAS dataset.

&domain: the domain name.

&var_num: global macro variable containing all the numeric variable names in the output dataset.

&var_txt: global macro variable containing all the character variable names in the output dataset.

```
=========================================================================*

%macro zmshell(input_meta=, domain=);

  %global var_num var_txt;

  proc sort data= &input_meta out=meta__; by ORDER;
    where not missing(variable);
```

```
 run;
 %let dsid=%sysfunc(open(meta__,i));
 %let fmt_exist=%sysfunc(varnum(&dsid,DISPLAY_FORMAT));
 %if &dsid > 0 %then %let rc=%sysfunc(close(&dsid));
 data meta_; set meta__ end=eof;
    length var_def $50 var_def_final $5000 var_label $100 var_label_final
           var_fmt_final $30000 var_fmt $50 var_num var_txt $30000;
    retain var_def_final 'length ' var_label_final 'label '
           var_fmt_final 'format ' var_num var_txt ' ';
    if lowcase(SASTYPE)='c' then var_def=
        ' '||strip(VARIABLE)||'  $'||strip(put(SASLENGTH,best.));
       else if lowcase(SASTYPE)='n' then var_def=
        ' '||strip(VARIABLE)||'  '||strip(put(SASLENGTH,best.));
    %if &fmt_exist>0 %then %do;
       if not missing(DISPLAY_FORMAT) then var_fmt=
          ' '||strip(VARIABLE)||' '||strip(DISPLAY_FORMAT);
    %end;
%else %do;
   var_fmt='';
%end;
var_label=' '||strip(VARIABLE)||" = '"||strip(LABEL)||"'";
var_def_final=catx(' ', var_def_final, var_def);
var_label_final=catx(' ',var_label_final, var_label);
var_fmt_final=catx(' ',var_fmt_final, var_fmt);
if lowcase(SASTYPE)='n' then var_num=catx(' ',var_num, variable);
  else if lowcase(SASTYPE)='c' then var_txt=catx(' ',var_txt, variable);
if eof then do;
  call symput('var_def_final',strip(var_def_final)||';');
  call symput('var_label_final',strip(var_label_final)||';');
  call symput('var_fmt_final',strip(var_fmt_final)||';');
  call symput('var_num',strip(var_num));
  call symput('var_txt',strip(var_txt));
end;
run;

data &domain._shell;
  &var_def_final;
  %if %length(&var_num)>0 %then %do;
     array var_num{*} &var_num;
  %end;
  array var_txt{*} &var_txt;
  stop;
  &var_label_final;
  %if &fmt_exist>0 %then %do;
    &var_fmt_final;
  %end;
run;
%mend zmshell;
```

After calling %**zmshell,** for example, %**zmshell**(input_meta=adae_meta, domain=adae); the dataset with variable's attribute (adae_shell) without any record is generated.

The final CDISC dataset is created by appending the derived dataset from raw data to this empty dataset (for example, adae_shell), so the variable's attributes defined in the specs have been converted into our final CDISC data.

2. CONVERT VALUE LEVEL METADATA TO CDISC VARIABLE'S VALUES

Per CDISC standard, variable's controlled terms have to be clearly documented in the specs. Some controlled terms (CTs) have 1 to 1 value matched from one variable to another/or multiple variable(s). Examples are PARAMCD to PARAM and PARAMN, AVISIT to AVISITN in ADaM; LBTESTCD to LBTEST, VISITNUM to VISIT in SDTM, etc. or vice versa.

An example for controlled terms defined in specs for value level metadata is in table 2. Here we want to use AVISIT's value to get AVISITN's value atomically.

DATASET	VARIABLE	SUBMISSION_VALUE	DECODE
ADLB	AVISITN	1	SCREENING
ADLB	AVISITN	2	WEEK0
ADLB	AVISITN	3	WEEK2
ADLB	AVISITN	4	WEEK4
ADLB	AVISITN	5	WEEK8
ADLB	AVISITN	6	WEEK12
ADLB	AVISITN	7	WEEK16
ADLB	AVISITN	801	FOLLOW UP1
ADLB	AVISITN	802	FOLLOW UP2
ADLB	AVISITN	803	FOLLOW UP3
ADLB	AVISITN	5001	BASELINE LAST
ADLB	AVISITN	5002	BASELINE MIN
ADLB	AVISITN	5003	BASELINE MAX
ADLB	AVISITN	6001	POST BASELINE LAST
ADLB	AVISITN	6002	POST BASELINE MIN
ADLB	AVISITN	6003	POST BASELINE MAX

Table 2. Value level metadata

What we need are: 1. to derive one set of variable' values from raw data, 2. To use sas proc format, to output format names for all 1 to 1 matched controlled terms, 3. To apply format names using put sas function.

2.1 An intuitive way is to convert the controlled terms by variable names into a macro as follows:

```
%macro getfmt;
proc format  cntlout = cntlout;

  VALUE $AVISITN
     'BASELINE LAST' ='5001'
     'BASELINE MAX' ='5003'
     'BASELINE MIN' ='5002'
     'FOLLOW UP1' ='801'
     'FOLLOW UP2' ='802'
     'FOLLOW UP3' ='803'
     'POST BASELINE LAST' ='6001'
     'POST BASELINE MAX' ='6003'
     'POST BASELINE MIN' ='6002'
     'SCREENING' ='1'
     'WEEK0' ='2'
     'WEEK12' ='6'
     'WEEK16' ='7'
     'WEEK2' ='3'
     'WEEK4' ='4'
```

```
      'WEEK8' ='5'
    ;
 VALUE $ANRINDN
      'ABNORMAL' ='4'
      'HIGH' ='3'
      'LOW' ='1'
      'NORMAL' ='2'
      'NOT APPLICABLE' ='5'
      'NOT EVALUABLE' ='6'

 run;
%mend getfmt;
```

From this created macro, we may check whether all wanted formats are generated and ready to use. AS we showed in above example, we can use format $AVISITN, to derive AVISITN's values from AVISIT's value, using AVISITN = input(put(AVISIT, $AVISITN.), best.) in a data step.

Similar to creating the macro %specs for variable's attribute, we used the following code to write the above macro %getfmt using controlled terms defined in specs:

```
/*=========================================================================

The following macro will generate macro %getfmt for converting CTs from a
CT file into format names
&macrolib: the directory where the %getfmt file is written to
&indata: the controlled terms' data set name(from specs
&fromvar: its value on the left side in format definition, its values
Exist in derived data and controlled terminology. Will be used to derive
another variable's value by 1-1 matching in format
&tovar: its values on the right side in format definition
=========================================================================*/
%macro zvintofmt(indata, fromvar, tovar);
filename getfmt "&macrolib.\getfmt.sas"; /*Associates a SAS fileref with
an external file;*/
data _null_;
   set &indata end=last;
   by variable;
   if fexist("getfmt") then rc=fdelete("getfmt");
   FILE "&macrolib.\getfmt.sas";
   if _n_=1 then do;
   PUT @1 "%nrstr(*This macro and file is output from macro %zvintofmt;)";
   PUT @1 "%nrstr(*It is generated/overriden whencalling% zvintofmt;)";
   PUT @1 "%nrstr(%macro getfmt;)";
   PUT @1 "%nrstr(proc format /*fmtlib*/ cntlout = cntlout;)";
   end;
   if first.variable then
   PUT @5  "VALUE $" variable;
   PUT @9 &fromvar "=" &tovar;
   if last.variable then
   put @5 ";";

   if last then do;
   PUT @1 "run;";
   PUT @1 "%nrstr(%mend getfmt;)";
   end;
run;
%mend zvintofmt;
```

After call %zvintofmt(ctdata, DECODE, SUBMISSION_VALUE); where ctdata is the CTs data set defined

in specs, formats will be generated in %getfmt. After we call macro %getfmt, the formats are ready to use.

2.2 Another of our format creation from our CTs dataset is using the CNTLIN= option in PROC FORMAT. We rearrange or rename our CT data from specs into the required variables and names, i.e. we converted CTs in the following data format, and here we name this data set as ct_data:

Table 3. Example for rearranged Controlled terms, named as ct_data, ready for proc format to output format names

START	END	LABEL	FMTNAME	TYPE
SCREENING	SCREENING	1	AVISITN_FMT	C
WEEK0	WEEK0	2	AVISITN_FMT	C
WEEK2	WEEK2	3	AVISITN_FMT	C
WEEK4	WEEK4	4	AVISITN_FMT	C
WEEK8	WEEK8	5	AVISITN_FMT	C
WEEK12	WEEK12	6	AVISITN_FMT	C
WEEK16	WEEK16	7	AVISITN_FMT	C
FOLLOW UP1	FOLLOW UP1	801	AVISITN_FMT	C
FOLLOW UP2	FOLLOW UP2	802	AVISITN_FMT	C
FOLLOW UP3	FOLLOW UP3	803	AVISITN_FMT	C
BASELINE LAST	BASELINE LAST	5001	AVISITN_FMT	C
BASELINE MIN	BASELINE MIN	5002	AVISITN_FMT	C
BASELINE MAX	BASELINE MAX	5003	AVISITN_FMT	C
POST BASELINE LAST	POST BASELINE LAST	6001	AVISITN_FMT	C
POST BASELINE MIN	POST BASELINE MIN	6002	AVISITN_FMT	C
POST BASELINE MAX	POST BASELINE MAX	6003	AVISITN_FMT	C

After applying the following code:

```
proc format cntlin=ct_data;
run;
```

Then the formats will be ready to use.

3. AUTOMATION FOR KEEPING THE VARIABLES ONLY IN THE SPECS AND APPLYING DATA SORTING IN FINAL DERIVED CDISC DATA

In data derivation step, we usually keep much more variables than the specs defined in the intermediate data step. At final data step, we keep the variables defined in specs only. Copying and pasting these variables into sas code is time consuming. In addition, whenever specs change, the sas code has to be changed accordingly. We like a way in that our code can take care of it automatically. Whenever specs change, what we do is re-running our code only.

In order to achieve automation, we read these variables from the specs into sas data set in order, and then put them into a macro variable, we name it &VARLIST.

In the same way, we put the variables for sorting order in a macro variable too, we name it &SORTORDER.

Here is the code for defining macro variables &VARLIST and &SORTORDER:

```
*=========================================================================
 &specsdata: the variable level metata, from specs, as in Table 1
 &varlist: it contains all variables in specs and in the specified order
 &sortorder: it contains the sort variables in the specified order
 =========================================================================*/

 %macro varlist(specsdata);
     %global varlist sortorder
     proc sort data = &specsdata out=specs; by order; run;
     data null;
       retain var;
       set specs end = last;
       by order;
       length var $2000;
       if _n_ = 1 then var = strip(VARIABLE);
         else var = strip(var)||", "||strip(VARIABLE);
       if last then call symput('VARLIST',strip(VAR));
     run;
     data sort; set specs;
       where SORTORDER ^in (".", " ");
       keep variable SORTORDER;
     run;
     proc sort data = sort;
       by SORTORDER;
     run;
     data null;
       retain var;
       set sort end = last;
       length var $2000;
       if _n_ = 1 then var = strip(VARIABLE);
         else var = strip(var)||", "||strip(VARIABLE);
       if last then call symput('SORTORDER',strip(VAR));
       run;
 %mend varlist;
```

In one step of sql as follows, and the final CDISC data will include the specified variables only and in specified sorting order:

```
 proc sql;
   create table adae(label = 'Adverse Event Analysis) as
   select &VARLIST
   from allae
   order by  &SORTORDER;
 quit;
```

CONCLUSION

In summary, the derived CDISC data sets are inherently consistent with the metadata (specification) by this metadata driven programming approach, because the metadata automatically come from specs. It not only increases programming efficiency and our code reusability, but also keeps highly consistence between data set's specifications and final output data.

ACKNOWLEDGMENTS

Thank Xiangdong Liu for coordinating our work and Thank Mei Zhao for reviewing this paper

CONTACT INFORMATION

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

Yurong Dai
Eli Lilly and Company
myydai@yahoo.com

Jiangang Jameson Cai
Eli Lilly and Company
jamesoncai@yahoo.com