PharmaSUG China 2016 - Paper 70

Make the Most Out of Your Data Set Specification

Thea Arianna Valerio, PPD, Manila, Philippines

ABSTRACT

A data set specification usually contains multiple data sets and variables that will be derived for a certain clinical study. Aside from ensuring the data set derivations are correct and robust, it is also important that the data set metadata is consistent with the specification. Manually verifying the data set and variable attributes against the specification is time consuming and might cause risk in ensuring quality. It is also inefficient typing all the variable names, labels, lengths and formats in the programs most especially when working on a dataset with many variables.

Through the macros presented in this paper, these steps can be automated and thus allow more time to review the variable derivations in the data set program. The specification is used as the input file to create macro variables that will hold the information on the attributes of all variables in a data set. This way, any changes in the specifications during the course of the study are automatically captured by the macro. Certain techniques are also presented to make the most out of your data set specification.

INTRODUCTION

In data set programming, there might be instances where a data set or variable attribute is overlooked. A significant variable needed for the analysis might be omitted from the data set output or an extra variable was not dropped. These issues will result to rework and might cause greater problems when discovered towards the middle or end of the study. Also imagine the unavoidable amount of specification changes during the course of the study; even a one variable label or length change in the specification will lead to a program update.

It is important to watch out for these seemingly minor but important components of data set programming. Specifications are created to ensure outputs accurately reflect the intended structure and metadata. However, when faced with multiple data sets and numerous variables in a study, manually verifying this will not guarantee high quality and will also constitute significant amount of time, which can be better devoted to reviewing the integrity of the data and ensuring accuracy of algorithms used.

We can automate this process and maximize the use of our specifications. The %GET_ATTRIB and %QC_FINALIZATION macros discussed in this paper offer different techniques to make the most out of your specification.

%GET_ATTRIB MACRO

The main purpose of this macro is to create global macro variables that will hold information such as data set name, data set label, data set sort, variable attributes and variable list. All these information will be extracted directly from the specification using the IMPORT procedure.

DATA SET SPECIFICATION

The macro is designed using the sample specification described below. At a minimum, there should be a 'Domains' and per data set worksheets. That is, each data set should have its own worksheet in the document and the worksheet name should be the same as the data set name.

Table 1 is an example of the 'Domains' worksheet which contains the list of all data sets to be completed in the study.

Dataset Name	Dataset Label	Key Variables	Common Dataset
ADSL	Subject-Level Analysis Dataset	STUDYID, USUBJID	Y
ADMH	Medical History Analysis Dataset	USUBJID, MHCAT, MHTERM, MHDTC	N
ADAE	Adverse Event Analysis Dataset	STUDYID, USUBJID, AETERM, AESPID	N
ADCM	Concomitant Medications Analysis Dataset	USUBJID, CMSTDTC, CMENDTC, CMTRT, CMCAT	N
ADVS	Vital Signs Analysis Dataset	USUBJID, AVISIT, ATPT, PARAMCD, PARAM	N
ADSV	Subject Visits Analysis Dataset	USUBJID, AVISIT	N

Table 1. Sample Data Set Metadata

Table 2 is an example of a data set worksheet containing all the variables to be derived therein.

ADSL - Subject-Level Analysis Dataset Programming Notes: <<insert notes here>>							
Variable Order	**Variable Name**	**Variable Label**	**Variable Type**	**Display Format**	**Possible Values**	**Source / Derivation**	**Common Variable**
1	STUDYID	Study Identifier	Char	20		DM.STUDYID	Y
2	USUBJID	Unique Subject Identifier	Char	20		DM.USUBJID	Y
3	SUBJID	Subject Identifier for the Study	Char	10		DM.SUBJID	Y
4	SITEID	Study Site Identifier	Char	200		DM.SITEID	Y
5	INVID	Investigator Identifier	Char	200		DM.INVID	Y
6	INVNAM	Investigator Name	Char	200		DM.INVNAM	Y
7	COUNTRY	Country	Char	200		DM.COUNTRY	Y
8	AGE	Age	Num	5.1		DM.AGE	Y
9	AGEU	Age Units	Char	10	YEARS	DM.AGEU	Y
11	SEX	Sex	Char	1	M, F	DM.SEX	Y
12	RACE	Race	Char	50	WHITE, BLACK OR AFRICAN AMERICAN, NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER, ASIAN, AMERICAN INDIAN OR ALASKAN NATIVE	DM.RACE	

Table 2. Sample Individual Data Set Metadata

PROCESS FLOW

Let us look into the details of how the %GET_ATTRIB macro works and also identify its macro parameters:

Parameter Name	Description	
SPEC_LOCNAME	The location and name of the data set specification, including the file extension (in XLS format).	Required
DSET_NAME	The name of the data set for which attributes will be extracted. Note that this should also correspond to the worksheet name.	Required

Table 3. Macro Parameters of %GET_ATTRIB macro

1. The macro reads the 'Domains' and specified data set name worksheet using PROC IMPORT. Shown below is the accompanying code for this step:

```
%macro import_xls (rangeval=, outdsname=);
 proc import out = &outdsname
   datafile = "&speclocname" dbms = excel replace;
   sheet="&rangeval.";
   mixed=yes;
   getnames=yes;
 run;
%mend import_xls;

**ALL DOMAINS METADATA**;
%import_xls (rangeval=%str(Domains$), outdsname=_all_domains);
**VARIABLES METADATA**;
%import_xls (rangeval=%str(&dsetname.$A2:H1000), outdsname=_var_attrib);
```

2. Then it processes the resulting data sets and creates the following global macro variables using CALL SYMPUT routine in DATA step and the 'INTO:' clause of the SQL procedure:

   - DSETLABEL - contains the data set label
   - DSETSORT - contains the data set key variables
   - VARATTRIB - contains the ATTRIB statement for all variables in the data set
   - VARLIST - contains the list of all variables (in order when specified) in the data set

3. Lastly, it performs the following checks which are helpful in ensuring quality in the specification. When any of these criteria is satisfied, a message will be printed in the log. The data set programmer can then raise these issues to the specification writer for resolution as soon as it is encountered.

    - multiple variable or data set entries in the specification
    - data set and variable name length greater than 8 characters
    - data set and variable label length greater than 40 characters
    - character variable length greater than 200 characters

    Presented below are the supporting codes for steps 2 and 3:

```
%global DSETLABEL DSETSORT VARATTRIB VARLIST DSETNAME SPECLOCNAME;
%let SPECLOCNAME = &spec_locname;
%let DSETNAME = &dset_name;

%macro QC_check (condition=, message=,value=);
 if &condition. then do;
  put "ALERT: " &value "%qcmpres(&message)";
 end;
%mend QC_check;

**--PROCESS DATASET ATTRIBUTES--**;
**Create DSETLABEL and DSETSORT macro variables**;
proc sort data_all_domains (where=(upcase(dataset_name) ="%upcase(&dsetname)"))
          out=_all_domains_s;
 by dataset_name;
run;

data _domain_attrib;
 set _all_domains_s end=eof;
 by dataset_name;

 if eof then do;
   call symput("DSETLABEL",strip(dataset_label));
   call symput("DSETSORT",tranwrd(key_variables,',',' '));

  **Quality checks**;
  %QC_check (condition=%str(not eof)
            , message=%str(Dataset has two or more entry in spec.));
  %QC_check (condition=%str(length(strip(dataset_name))>8)
            , message=%str(Dataset name is greater than 8.));
  %QC_check (condition=%str(length(strip(dataset_label))>40)
            , message=%str(Dataset label is greater than 40.));
 end;
run;

**--PROCESS VARIABLE ATTRIBUTES--**;
data _all_atribs;
 set _var_attrib (where=(~missing(variable_name)));

**Order of entries in the spec will be used in case variable_order is all blank**;
 ord=_n_;
run;

proc sort data=_all_atribs out=_all_attribs_s;
 by variable_name variable_order ord;
run;
data _all_attribs_final;
 set _all_attribs_s;
 by variable_name variable_order ord;

 **Quality checks**;
```

```
 %QC_check
 (condition=%str(length(strip(variable_name))>8)
 ,message=%str(Variable name is greater than 8.),value=variable_name);
 %QC_check
 (condition=%str(length(strip(variable_label))>40)
 ,message=%str(Variable label is greater than 40.),value=variable_name);
 %QC_check
 (condition=%str(upcase(variable_type)='CHAR' and input(display_format,best.)>200)
 ,message=%str(Variable length is greater than 200.) ,value=variable_name);
 %QC_check
 (condition=%str(last.variable_name and not first.variable_name)
 ,message=%str(has multiple entries, please check.) ,value=variable_name);

 **Keep the last entry if there are multiple entries in the spec**;
 if last.variable_name;
run;

**Resort based on the desired display order of variables**;
proc sort data=_all_attribs_final;
 by variable_order ord;
run;

**Set up for attribute statement**;
data _null_ ;
 length _format _length $50.;
 set _all_attribs_final end=eof;

 if upcase(variable_type)='CHAR' then do;
  if ~missing(display_format) then _length='length=$'||strip(display_format);
 end;
 if upcase(variable_type)='NUM' then do;
  if anyalpha(display_format) then _format='format='||strip(display_format);
  else if ~missing(display_format) then _length='length='||strip(display_format);
 end;

 call symput('attribln'||strip(put(_n_, best.)),strip(variable_name)||'
label="'||strip(variable_label)||'" '||strip(_length)||""||compress(_format));

 if eof then call symput('total', strip(put(_n_, best.)));
run;

**Create VARATTRIB and VARLIST macro variables**;
data _null_;
 length attrline $32000.;
 retain attrline;
 attrline='attrib ';

 %do order=1 %to &total.;
  attrline=strip(attrline)||" "||strip(symget("attribln&order."));
 %end;

 call symput("varattrib", strip(attrline));
run;

proc sql noprint;
 select variable_name into :varlist separated by ' '
 from _all_attribs_final;
quit;
```

The generated macro variables can then be used in the program to ensure a complete match between the data set output and its corresponding specification. There is no need to type the dataset and all the variables' attributes as these values are already stored in the macro variables. In case there are attribute updates in the specification

throughout the course of the study, only a program re-run is needed to capture these changes. Since these macro variables will most likely be used towards the end of the program, caution should be exercised in setting variable lengths in the intermediate parts of the program to avoid value truncations. It is also important to note that it is still part of the programmer's responsibility to regularly check the logs and provide feedback on any issues prompted to the specification writer when needed. It will be shown later on how these macro variables can be efficiently used.

%QC_FINALIZATION MACRO

This next macro is for finalizing and performing checks on the created data set. It performs a couple of things like merging common variables, setting the appropriate dataset and variable attributes and order, checking if data set keys specified will yield distinct records, determining NULL variables and checking for variable values not listed in the 'Possible Value' column of the specification. It also goes hand in hand with the %GET_ATTRIB macro as the latter's macro parameters: SPEC_LOCNAME and DSET_NAME will be used here as well. These are the macro parameters of this macro:

Parameter Name	Description	
DSET_LIB	Name of output library for the data sets. This is where the final data set will be outputted and the common data set will be read (if prompted for merging).	Required
PREFINAL_DSET	Name of the latest data set prior to finalization.	Required
MERGE_COMMON_VARS	Indicator whether to merge common variables into the data set of concern. Possible values are Y or N.	Optional
MERGE_ID_VAR	Variable used for merging the common variables into the data set of concern (e.g. USUBJID). Note that this is a required parameter when MERGE_COMMON_VARS is set to Y.	Conditional
POSSIBLE_VAL_DELIM	Delimiter used to enumerate the possible values of a variable in the specification. Default value is "," (comma).	Required

Table 4. Macro Parameters of %QC_FINALIZATION macro

PROCESS FLOW

- If prompted (or MERGE_COMMON_VARS is set to Y), the macro merges common variables into the data set of interest by the specified ID variable. The common dataset and common variables will be determined from the 'Common Dataset' and 'Common Variables' columns of the specification. These are the codes for this step:

```
%if %upcase(&merge_common_vars)=Y %then %do;
  **Determine the common dataset**;
  data _common_dataset;
   set _all_domains (where=(strip(upcase(common_dataset))="Y")) end=eof;
   if eof then call symput("COMMON_DATASET",compress(dataset_name));

   **Quality check**;
   %QC_check (condition=%str(not eof)
              ,message=%str(Two or more common dataset defined.));
  run;

  **Determine the common variables**;
  %import_xls (rangeval=%str(&COMMON_DATASET.$A2:H1000), outdsname=_common);
  data _common_vars;
   set _common (where=(strip(upcase(common_variable))="Y"));
  run;
  proc sql noprint;
   select distinct variable_name into :common_vars separated by " "
   from _common_vars;
  quit;

  **Check prefinal dataset if containing common vars.
    If yes, then drop those variables**;
  proc contents data=&prefinal_dset out=_prefinal_dset_chk noprint;
  run;
  proc sql noprint;
   select distinct a.name into :prefinal_drop separated by ' '
   from _prefinal_dset_chk as a inner join
       _common_vars (where=(upcase(variable_name) ne "%upcase(&merge_id_var)")) as b
```

```
   on a.name = b.variable_name;
  quit;
  proc sort data=&dset_lib..&COMMON_DATASET out=&COMMON_DATASET;
   by &merge_id_var;
  run;
  proc sort data=&prefinal_dset;
   by &merge_id_var;
  run;
  data _&dsetname;
   merge &dset_lib..&COMMON_DATASET (keep=&common_vars)
         &prefinal_dset (in=main drop=&prefinal_drop);
   by &merge_id_var;
   if main;
  run;
  %let final_var_list = &common_vars &varlist;
%end;
%if %upcase(&merge_common_vars) ne Y %then %do;
  %let final_var_list = &varlist;
  data _&dsetname;
   set &prefinal_dset;
  run;
%end;
```

- After the common variables were merged, the macro then finalizes the data set by keeping only the needed variables and maintaining the variable order according to the specification. It also assigns the variable attributes, dataset name and label and sorts the data set according to the key variables.

```
data &dsetname;
 &varattrib;
 retain &final_var_list;
 set _&dsetname (keep=&final_var_list);
run;
**Assign dataset attributes and output in the specified library**;
proc sort data=&dsetname out=&dset_lib..&dsetname (label=&dsetlabel);
 by &dsetsort;
run;
```

- Once the data set is finalized, the macro first checks if the key variables specified would yield distinct records. If not, a log message will be sent to the log. The programmer can then raise this issue to the specification writer for confirmation whether additional key variables are needed to make records unique.

```
proc sort data=&dset_lib..&dsetname out=_sortchk nodupkey dupout=_sortchkdup;
 by &DSETSORT;
run;
**Check if the duplicate output dataset has contents**;
data _null_;
 set _sortchkdup end=eof;
 if eof then do;
  put "ALERT: Sort specified in the spec does not produce unique records. Please
confirm.";
 end;
run;
```

- Then the macro checks for NULL variables. This is a significant check since there might be instances where a variable is initialized (making it present in the data set) but it was not actually derived or the intended derivation was not performed properly. The list of variables with all blank values will be printed as a log message. The programmer can then ascertain if this issue is expected or makes appropriate action if needed.

```
**Determine the number of variables in the data set**;
proc contents data=&dset_lib..&dsetname. out=output_&dsetname. noprint; run;
data _null_;
 set output_&dsetname. end=eof;

 call symput('VAR_' || compress(put(_n_,8.)), compress(name));
```

```
 if eof then call symput('VARCNT',compress(put(_n_,8.)));
run;

**Determine the total number of observations of the data set**;
proc sql noprint;
 select count(*) into :nobs
 from &dset_lib..&dsetname.;
quit;
**Count the number of missing observations in a variable.
  If that is equal to the number of observations, then send a message to the log**;
proc sql noprint;
  %do i=1 %to &varcnt;
   select count(*) into :varcntmiss&i
   from &dset_lib..&dsetname. (keep=&&var_&i. where=(missing(&&var_&i..)));

   %if &&varcntmiss&i..=&nobs %then %do;
    %put ALERT: Variable &&var_&i.. has all values missing. Please check.;
   %end;
  %end;
quit;
```

- Lastly, this macro checks for variable values in the data set not listed in the 'Possible Values' column of the specification. There might be instances where the programmer misspelled a value in the program or the specification failed to account for all expected values in the data set. This macro aims to cover for these cases by going through the distinct variable values in the data set, comparing it against the enumerated values in the specification and printing the list of issues in the SAS® output.

```
**Determine variables with exhaustive list of possible values**;
data possible_values (where=(~missing(name)));
 length name $200;
 set _all_attribs_final (keep=variable_name possible_values variable_type
                         where=(~missing(possible_values)));
 _num=count(possible_values,"&possible_val_delim");
 do cnt=1 to _num+1;
  name=strip(scan(possible_values,cnt,"&possible_val_delim"));
  output;
 end;
run;
**Pass the variable name/type and total number of variables in macro variables**;
data _null_;
 set possible_values end=eof;

 call symput('VARPV_' || compress(put(_n_,8.)), compress(variable_name));
 call symput('VARPVTYPE_' || compress(put(_n_,8.)), compress(variable_type));
 if eof then call symput('VARPVCNT',compress(put(_n_,8.)));
run;
**Loop through the variables in the data set and get list of all its values**;
**Loop through the specification to get list of possible values specified**;
**Merge the two list and if a data set value is not in the spec, prompt the user**;
  %do i=1 %to &VARPVCNT;
    proc sql;
     create table &&VARPV_&i.._dset as
     %if %upcase(&&VARPVTYPE_&i)=CHAR %then %do;
       select distinct &&VARPV_&i.. as value length=200
     %end;
     %if %upcase(&&VARPVTYPE_&i)=NUM %then %do;e
       select distinct strip(put(&&VARPV_&i..,best.)) as value length=200
     %end;
     from &dset_lib..&dsetname. (keep=&&VARPV_&i.. where=(~missing(&&VARPV_&i..)))
     order by value;

     create table &&VARPV_&i.._spec as
     select distinct strip(name) as value length=200
```

```
     from possible_values (where=(upcase(variable_name)="&&VARPV_&i"))
     order by value;
    quit;
     data chk_&&VARPV_&i;
     length variable $8;
     merge &&VARPV_&i.._dset (in=indset) &&VARPV_&i.._spec (in=inspec);
     by value;
     variable="&&VARPV_&i";
     if indset and not inspec then output;
    run;
  %end;

  data _chk_possiblevalues;
    set chk_:;
  run;
  title2 "List of variables and their values not specified in possible values
column.";
  proc print data=_chk_possiblevalues;
  run;
```

CONCLUSION

We can definitely maximize the use of our data set specification. Aside from using it to document the intended data set structure and derivation rules in the study, we can make it become the driver of certain parts of our data set program. We can also add checks that will help ensure quality in both data set outputs and specification.

The techniques presented in this paper offer ways to ensure consistency between the data set metadata and specification while still saving time and effort. This gained efficiency can then be devoted to ensuring the integrity of the data set created. Moreover, the checks presented in this paper also enable the programmer to validate his or her own work and provide feedback to the specification writer should there are possible issues with the document.

REFERENCES

- Rittman, Misha. 2010. “Automating the Link between Metadata and Analysis Datasets”. Proceedings of the Pharmaceutical Industry SAS® Users Group 2010 Conference. Lex Jansen’s Homepage. Available at http://www.lexjansen.com/pharmasug/2010/ad/ad16.pdf. Accessed 05 August 2016.
- Danner, Bradford. 2008. “Exploitation of Metadata for Restructuring Datasets and Code Generation”. Proceedings of the Pharmaceutical Industry SAS® Users Group 2008 Conference. Lex Jansen’s Homepage. Available at http://www.lexjansen.com/pharmasug/2008/cc/CC21.pdf. Accessed 05 August 2016.

ACKNOWLEDGMENTS

The author would like to thank Ping-Chung Chang, Jason Ralph Isturis, Mari Penelope Mendoza and PPD Biostatistics and Programming Manila team who provided valuable inputs to improve this paper.

CONTACT INFORMATION

Your comments and questions are highly valued and encouraged. Please contact the author at:

Thea Arianna Valerio
PPD
Taguig City, Philippines
E-mail: Thea.Valerio@ppdi.com