Paper 120-2018

Defensive programming techniques for ongoing clinical studies

Bharath Donthi, Statistics & Data Corporation, Tempe, AZ, 85281

Lingjiao Qi, Statistics & Data Corporation, Tempe, AZ, 85281

ABSTRACT

In clinical trial studies, statistical programming activities often start when data collection is still ongoing. It’s a common situation that we have variables or data sets with partial or completely missing data. Below we discuss defensive programming techniques which avoid re-programming when the database is updated. For missing SDTM supplemental qualifiers or variables, we discuss methods to defensively create these variables for continuing ADaM and table/listing programming. One method is to create a dummy SAS® data set with all needed variables with null values in a separate data step, then set it together with the current data set. The other method is to create required variables using array techniques before manipulating SDTM/SUPP data sets. Additionally, we also present a robust programming technique to dynamically generate table/listing outputs handling situations where the current data set does not have the required data for table/listing outputs. Implementing these useful defensive programming techniques minimize the risk of errors when processing ongoing clinical trial data, allow programming to start earlier in the life of a study, and reduce the need for rework as data change.

INTRODUCTION

In the clinical research industry, statistical programming activities start at the beginning stages of the project. Statistical programs are created with the expectation that they will be run multiple times in a variety of contexts. A program should be robust enough to handle differences in data cutoff dates, which results in differing subset groups for different deliverables. There are several challenges in setting up robust programs for analyzing clinical data sets. One challenge is that required variables, or even required data sets, do not exist during development of programs. Another challenge is that variables may not include all eventual possible values. Thus, unexpected results are often encountered when running existing programs with updated data sets. In this fast-paced industry, SAS code should be written to handle all possible scenarios, as well as any unexpected circumstances caused by database updates.

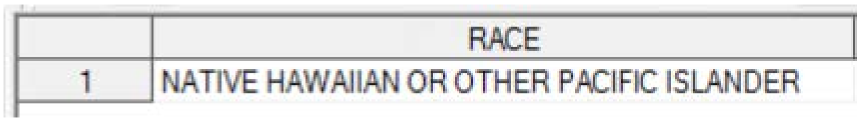
The aim of this paper is to describe several common situations in clinical trials programming where defensive program techniques could be applied to minimize program updates.

TOO MANY FREE TEXTS VARIABLES!

The clinical trial database contains all the data collected during the study, including a mixture of character variables. Accounting for all possibilities of character variables in free-text form can be particularly challenging since new values may be added each time the database updates. For example, if we need to use the RACE variable to create a sorting variable, RACEN, many programmers often write out the texts of available categories and assign a corresponding sorting value. If the database is still in an early stage of the study, there may only be one value for RACE (Display 1). Thus, only one line of code would be implemented to assign the sorting order (Display 2). While this straightforward statement is adequate for the current state of the data, if additional values of RACE are added at a later stage (Display 3) the code will no longer be adequate. The programmer must update the program with more lines of IF-THEN statements and write out every single category captured in the database for RACE (Display 4). In addition, as presented in Display 3, when the cases for different categories are not consistent (uppercase, lowercase, propcase), and when spelling mistakes are present in the database (e.g., “ALASA” should be “ALASKA”), writing out all of the free-texts is very error-prone; any typographical error in the IF statement will make the program inaccurate. While this method may work appropriately, repeatedly updating programs is time consuming and inefficient – especially in the fast-paced industry of clinical research.

To avoid the need of repeatedly updating programs, we introduce the UPCASE function and colon modifier (:) (Display 5). UPCASE changes the case of the argument to upper case, which ensures all values are consistently in the same case. In SAS, the colon can be used in conjunction with all comparison operators. When used together with the equal sign, the colon translates the meaning of the comparison operator to “starting with…”. Using the UPCASE function together with the colon modifier, we can account for all possible scenarios of free texts and avoid any potential typos. In addition, the amount of typing is reduced thus improving efficiency. Instead of the colon modifier, the INDEX function could also be utilized together with the UPCASE function to reduce typing and increase the robustness of the program. The INDEX function searches a character string (specified in the parenthesis), and if a match is found, returns the position of the first occurrence of the strings’ first character, which is larger than 0 (Display 6).

Display 1: Database at early stage with one possibility for RACE.

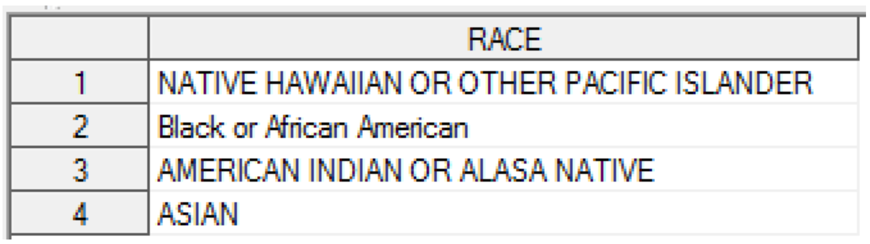


	RACE
1	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER

Display 2: IF – THEN statement to create RACEN.

```
if RACE = 'NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER' then RACEN =1;
```

Display 3: Database at later stage with four possibilities for RACE.



	RACE
1	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
2	Black or African American
3	AMERICAN INDIAN OR ALASA NATIVE
4	ASIAN

Display 4: IF – THEN statement to create RACEN.

```
if RACE = 'NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER' then RACEN =1;
else if RACE = 'Black or African American' then RACEN =2;
else if RACE = 'AMERICAN INDIAN OR ALASA NATIVE' then RACEN =3;
else if RACE = 'ASIAN' then RACEN =4;
```

Display 5: Upcase function and colon modifier with IF – THEN statement to create RACEN.

```
if upcase(RACE) =: 'NATIVE' then RACEN =1;
else if upcase(RACE) =: 'BLACK' then RACEN =2;
else if upcase(RACE) =: 'AMERICAN' then RACEN =3;
else if upcase(RACE) =: 'ASIAN' then RACEN =4;
```

Display 6: Upcase function and index function with IF – THEN statement to create RACEN.

```
if index(upcase(RACE),'NATIVE') then RACEN =1;
else if index(upcase(RACE),'BLACK') then RACEN =2;
else if index(upcase(RACE), 'AMERICAN') then RACEN =3;
else if index(upcase(RACE), 'ASIAN') then RACEN =4;
```

MISSING REQUIRED SUPPLEMENTAL QUALIFIERS OR EVEN THE ENTIRE DATA SET!

When CDISC standards are applied to the clinical study data, SDTM (study data tabulation model) data sets map and store source data from internal and external databases. ADaM (analysis data model) data sets are then derived from SDTM data sets and supplemental data sets to provide analysis-ready variables. Based on the CDISC SDTM implementation guide, only standard variables should be included in the SDTM data sets. Any non-standard variables should be mapped and included in the Supplemental Qualifiers special-purpose data sets. Supplemental data sets store additional information on a subject or event, and has a vertical structure in which each value is stored in a separate row. The fields QNAM, QLABEL, and QORIG store metadata (variable name, variable label, and origin, respectively) while the actual data value is stored in the field QVAL. QVAL should not be NULL; if there is no data value, no row is created in the corresponding supplemental qualifier data set. Records stored in supplemental data sets (if any) should be merged back to its parent domain by using the identifying variables.

Since programing activities start when data collection is still ongoing, some data sets in the database may be empty or incomplete (variables containing null values). Consequently, after SDTM mapping some SDTM supplemental data sets might not exist or might only contain partial data (caused by variables in the raw database containing null values). Thus, when the ADaM program is structured to merge supplemental data sets to the parent SDTM domain, errors occur. To avoid repeatedly checking the database and modifying the SDTM/ADaM program to adapt to any new data from database updates, we discuss several programming techniques to make the program more flexible and robust, with the aim of avoiding any potential issues caused by missing or partial supplemental data sets.

What if the SDTM Supplemental Data set Does Not Exist?

We often need to merge the parent SDTM data set with the supplemental data set to get information stored in the supplemental data set. If the program attempts to merge two data sets when the supplemental data set does not exist, an error occurs. The SYSFUNC and EXIST functions can be used in this situation to check the existence of a particular data set before attempting a merge. The example code below first checks the existence of the supplemental data set. If the supplemental data set exists, the macro variable SCNT is set to a value of 99; otherwise, the macro variable SCNT is set to a value of 0. Next, based on the value of the macro variable SCNT, the program conditionally produces data set DM1. If SUPPDM exists and the value of the macro variable SCNT is 99, the SUPPDM data set will be transposed and merged with the parent data set DM to create a new data set, DM1. If SUPPDM does not exist and the value of the macro variable SCNT is 0, DM1 is set to be the same as DM.

```
%macro supp;
/****check if SUPPDM exists*****/
 %if %sysfunc(exist(SDTM.SUPPDM)) %then %let SCNT = 99;
      %else %let SCNT = 0;


/****if SUPPDM exists then merge it with parent domain****/
%if &SCNT. gt 0 %then %do;
   proc sort data=SDTM.SUPPDM out=SUPPDM(keep=USUBJID QNAM QVAL); by USUBJID;
run;
   proc transpose data=SUPPDM out=SUPPDMT(drop=_:);
      by USUBJID;
      var QVAL;
      id QNAM;
   run;
   data DM1;
      merge DM  SUPPDMT;
      by USUBJID;
   run;
%end;
```

```
/****if SUPPDM does not exist then DM1 is set with DM data set****/
%else %do;
   data DM1;
      set DM;
   run;
%end;
%mend supp;
%supp;
```

What if a Required Variable in the Supplemental Data set Does Not Exist?

ADaM data sets often contain variables which originate from the QNAM variable in SDTM supplemental data sets. If the required supplemental data set is empty (as in the example above) or does not yet include the required QNAM, creating the required variables for ADaM programing could cause a perplexing problem for programmers. Ideally, a dummy variable with NULL values should be created as a place holder for this missing, yet required, variable to enable programming to continue; however, this dummy variable should be automatically replaced with data values once updated and present in the database.

Continuing the example in the previous section, after creating DM1 we can use the methods below to create dummy place holders for required variables (LANG, RESIDE, RACEOTH, HERITAGE) without any hard-coding.

Method 1 first creates a dummy data set with all the required variables having null values and then appends this dummy data set together with DM1. Method 2 utilizes the ARRAY statement to create these expected variables after reading in DM1. Both methods create the required variables whether or not the SUPP data set is complete. The dummy variable place holders will be automatically replaced with any real values when the program is rerun with the updated data. Thus, there is no need to check the database and update the program for each deliverable.

Method 1:

```
data DUMMY;
   length LANG RESIDE RACEOTH HERITAGE $200;
   LANG = " ";
   RESIDE= " ";
   RACEOTH=" ";
   HERITAGE=" ";
   Label LANG = "Native Language"
         RESIDE = "Ever Lived in US?";
         RACEOTH="Race, Other";
         HERITAGE="Parents Heritage";
run;

data DM2;
   set DM2 DM1;
run;
```

Method 2:

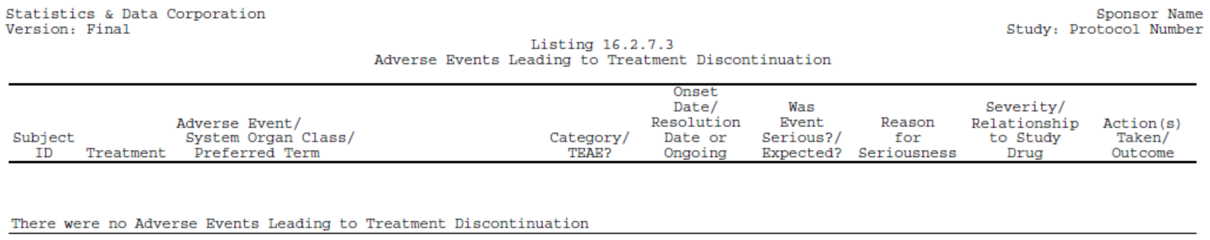
```
data DM2;
   set DM1;
   array temp $ LANG RESIDE RACEOTH HERITAGE;
run;
```

NO OBSERVATIONS MEETING CRITERIA FOR REPORTING!

Often, investigators in clinical studies are interested in reports with a subset of subjects who meet certain criteria. If there are no subjects meeting these criteria, programmers should make it clear on the report so that the reviewers will not question if the programming activity was not completed. In addition, in an ongoing study, reporting programs should be flexible enough to handle two scenarios: first, a scenario in which no subjects meet the defined criteria, and second, a scenario in which subjects meeting the defined criteria exist after data has been updated.

We provide an example of macro code, which dynamically presents a subject listing based on the number of subjects meeting the defined criteria. When there are no subjects meeting the required criteria, the pre-defined shell template is still produced and text is included to state that there were no observations meeting the criteria (Display 7). This statement could also be customized to reflect the title of the subject listings. Later in the study, if there are records satisfying these criteria in the database, this program will generate a listing that includes those records meeting the defined criteria (Display 8). Details about the macro code are included in the "code work-through" section below.

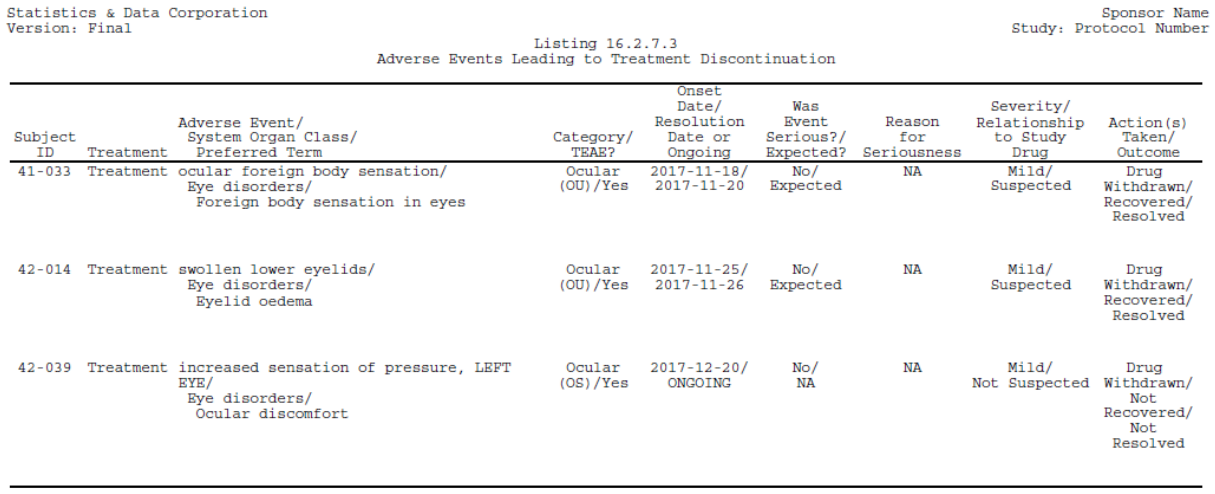
Display 7: Reporting when there are no observations meeting the criteria.



```
Statistics & Data Corporation                                                                    Sponsor Name
Version: Final                                                                          Study: Protocol Number
                                                 Listing 16.2.7.3
                                     Adverse Events Leading to Treatment Discontinuation
______________________________________________________________________________________________________________
                                                                 Onset
                                                                 Date/       Was                   Severity/
                     Adverse Event/                            Resolution   Event       Reason    Relationship   Action(s)
Subject               System Organ Class/           Category/   Date or   Serious?/      for       to Study      Taken/
  ID    Treatment      Preferred Term                 TEAE?     Ongoing   Expected?  Seriousness     Drug        Outcome
______________________________________________________________________________________________________________


There were no Adverse Events Leading to Treatment Discontinuation
______________________________________________________________________________________________________________
```

Display 8: Reporting when there are observations meeting the criteria.



```
Statistics & Data Corporation                                                                    Sponsor Name
Version: Final                                                                          Study: Protocol Number
                                                 Listing 16.2.7.3
                                     Adverse Events Leading to Treatment Discontinuation
______________________________________________________________________________________________________________
                                                                 Onset
                                                                 Date/       Was                   Severity/
                     Adverse Event/                            Resolution   Event       Reason    Relationship   Action(s)
Subject               System Organ Class/           Category/   Date or   Serious?/      for       to Study      Taken/
  ID    Treatment      Preferred Term                 TEAE?     Ongoing   Expected?  Seriousness     Drug        Outcome
______________________________________________________________________________________________________________
 41-033  Treatment ocular foreign body sensation/     Ocular   2017-11-18/   No/          NA         Mild/         Drug
                    Eye disorders/                   (OU)/Yes  2017-11-20  Expected               Suspected    Withdrawn/
                     Foreign body sensation in eyes                                                            Recovered/
                                                                                                                Resolved


 42-014  Treatment swollen lower eyelids/             Ocular   2017-11-25/   No/          NA         Mild/         Drug
                    Eye disorders/                   (OU)/Yes  2017-11-26  Expected               Suspected    Withdrawn/
                     Eyelid oedema                                                                             Recovered/
                                                                                                                Resolved


 42-039  Treatment increased sensation of pressure, LEFT  Ocular   2017-12-20/  No/        NA         Mild/         Drug
                   EYE/                              (OS)/Yes   ONGOING      NA                 Not Suspected  Withdrawn/
                    Eye disorders/                                                                                Not
                     Ocular discomfort                                                                         Recovered/
                                                                                                                  Not
                                                                                                                Resolved

______________________________________________________________________________________________________________
```

Example Code Walk-through:

1) The first step in the code below reads in data and subsets it based on the required criteria for the report. This step also evaluates the number of observations in the subset data set and stores the count information in a macro variable &OBS. If there are no observations in the subset data set, we assign &OBS to 0. If there are observations in the subset data set, the CALL SYMPUT routine assigns the actual counts of observations to &OBS.

2) Next, we write conditional code based on the resolved value of &OBS. If &OBS = 0, meaning there are no observations in the input data set, step 3 will be executed to create several macro variables for the PROC REPORT section, which will produce a report stating that no subjects meet the criteria for this table/listing report. In step 3, the PROC SQL step creates a data set PREP which has a structure similar to IDNATA (subset data set). The data step first creates a

variable NONE, which has a value of &TEXT. &TEXT is a character string that contains the statement clarifying that no observations met the required criteria (which could be modified to reflect the title of output). The CALL SYMPUT routine creates macro variables &NOBS_VAR, &NOBS_COL, &NOBS_MSG for PROC REPORT to be used later.

3) If the conditional check &OBS ^= 0 is true, then step 4 will be executed. The REPORT data set will include all contents in the INDATA data set.

4) This step dynamically writes the PROC REPORT code to output the table/listing, regardless the origin of the data set used (the REPORT data set can originate from STEP 3 or STEP 4). If there are no observations meeting the criteria, PROC REPORT outputs a blank shell with variables listed in the COLUMN statement, with &NOBS_VAR being resolved to defined texts (&TEXT). If there are observations meeting the criteria, values for variables listed in the COLUMN statement will be reported in the output. In this case, all macro variables &NOBS_VAR, &NOBS_COL, &NOBS_MSG are resolved to NULL.

5) This step calls and evokes the macro %lis.

```
%macro lis;
%let OBS=0;

data INDATA;
   set AE &WHERE nobs=obs;
   call symput("OBS",put(OBS,best.));                                1
run;

%let NOBS_VAR =;
%let NOBS_COL =;
%let NOBS_MSG =;

%if &OBS. =0 %then %do;
      proc sql;
         create table PREP like INDATA;
      quit;

      data REPORT;
         NONE=&TEXT.;                                                2
         output;
         call symput('NOBS_VAR' , 'NONE');
         call symput('NOBS_COL' , 'define NONE  / order noprint;');
         call symput('NOBS_MSG','compute after NONE; line @1 &TEXT.;
                     endcomp; ');
         set PREP;
      run;
%end;

%if &OBS.^=0 %then %do;
data REPORT;
   set INDATA;                                                       3
run;
%end;


ods listing close;
ods pdf file="AE_listing.pdf" notoc ;
```

```
proc report data=REPORT missing nowd headskip split="~" list ;
   column SUBJID &NOBS_VAR AESEQ ARM TERM LOC DATE SER REASON SEV ACN;
   define SUBJID /"Subject~ID" order
                  style(column)=[just=c asis=on cellwidth=0.6 in];
   define AESEQ  /order noprint;
   define ARM    /"Treatment" order
                  style(column)=[just=c asis=on cellwidth=0.8 in];
   define TERM   /"Adverse Event/$-2n System Organ Class/$-2n Preferred
                  Term" orderstyle(column)=[just=c asis=on cellwidth=3
                  in];
   define LOC    /"Category/~TEAE?" order
                  style(column)=[just=c asis=on cellwidth=0.9 in];
   define DATE   /"Onset~Date/~Resolution~Date or~Ongoing" order
                 style(column)=[just=c asis=on cellwidth=0.9 in];
   define SER    /"Was~Event~Serious?/~Expected?"
                 style(column)=[just=c asis=on cellwidth=0.9 in];
   define REASON /"Reason~for~Seriousness"
                 style(column)=[just=c asis=on cellwidth=0.9 in];
   define SEV    /"Severity/~Relationship~to Study~Drug"
                 style(column)=[just=c asis=on  cellwidth=1.1 in];
   define ACN/ "Action(s)~Taken/$-2nOutcome"
                 style(column)=[just=c asis=on    cellwidth=0.9 in];


   compute after SUBJID;
   line @1 '';
   endcomp;

   compute after TERM;
   line @1 '';
   endcomp;

   &NOBS_COL.
   &NOBS_MSG.

run;

ods pdf close;
ods listing;

%mend;
```

4

```
%let WHERE= (where=(AEACN="DRUG WITHDRAWN" and TRT01A^=:'Screen' ));
%let TEXT="There Were No Adverse Events Leading to Treatment
Discontinuation";
%lis;
```

5

CONCLUSION

In this paper we have presented several defensive techniques for statistical programming in clinical trials. These techniques help ensure high quality of deliverables while minimizing the impact of potential database updates.

REFERENCES

John Woods, Jennie McGuirk, “Defensive programming techniques” available at: https://www.lexjansen.com/phuse/2006/po/PO08.pdf

W. Jodi Auyuen, “Reporting Tips for No Observations”, SAS Global Forum 2013, available at: http://support.sas.com/resources/papers/proceedings13/320-2013.pdf

Suwen Li, Sai Ma, Bob Lan, Regan Li, 2013 “From SDTM to ADaM”, SAS Global Forum 2013, available at: http://support.sas.com/resources/papers/proceedings13/199-2013.pdf

Nelson, Gregory S. and Zhou, Jay. 2011.”Good Programming Practices in Healthcare: Creating Robust Programs”. Proceedings of the 2011 Pharmaceutical Industry SAS Users Group (PharmaSUG) Annual Conference. Available at http://www.lexjansen.com/pharmasug/2011/tt/pharmasug-2011-tt05.pdf.

Nancy Brucken, Donna E. Levy, “Defensive Coding by Example: Kick the Tires, Pump the Brakes, Check Your Blind Spots, and Merge Ahead!” MWSUG 2016, Available at https://www.mwsug.org/proceedings/2016/TT/MWSUG-2016-TT12.pdf

Mary F. O. Rosenbloom, Kirk Paul Lafler, “Best Practices: Use Coding Shortcuts to Avoid Typsos and Add Flexibility”, Available at: https://www.lexjansen.com/wuss/2012/83.pdf

Sudhir Singh, “Using SAS Colon Effectively”, PharmaSUG 2012, Available at: https://www.pharmasug.org/proceedings/2012/TA/PharmaSUG-2012-TA05.pdf

ACKNOWLEDGMENTS

The authors thank Kevin Uchimura, Deanna Driss, Lot Slade, Kirk Bateman, Faith Kolb, and Melanie Ciotti at Statistics & Data Corporation for their thoughtful review of this manuscript.

CONTACT INFORMATION

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

Name: Bharath Donthi
Enterprise: Statistics & Data Corporation
Address: 21 E 6th St #110, Tempe, AZ 85281
E-mail: bdonthi@sdcclinical.com
Web: https://www.sdcclinical.com/

Name: Lingjiao Qi
Enterprise: Statistics & Data Corporation
Address: 21 E 6th St #110, Tempe, AZ 85281
E-mail: lqi@sdcclinical.com
Web: https://www.sdcclinical.com/