Paper CT09

No Compromise on QUALITY: Coding tips and Macros to improve Efficiency

Akhil Vijayan, Genpro Research, Thiruvananthapuram, India
Limna Salim, Genpro Research, Thiruvananthapuram, India
Roshan Stanly, Genpro Research, Thiruvananthapuram, India

ABSTRACT

When it comes to optimizing programming time, especially in ISE and ISS studies and in circumstances when there is a new data cut, programming reusability is favored by many organizations. But programmers still face issues while reusing the codes like truncations, handling variables with length more than 200, handling of QNAMs in SUPP domains, etc.

In this paper, we will discuss techniques and best practices for reducing the likelihood of subtle errors in software reuse scenarios like:

- Coding tips to handle supplementary variables efficiently.
- Methods to avoid truncation in data
- Importance and examples of custom checks that programmers can implement to improve efficiency.

The paper also discusses a few standard macros which support custom checks like

- Macro to ensure all variables with length more than 200 are successfully mapped in SDTM
- Macro for automating SAS formats from P21 specification template.

INTRODUCTION

Quality is obstinate in programming, and it is very disappointing that issues are not identified at an earlier stage even though people have followed standard programming practices. Experience is, of course, a key factor that will help predict the possibility of error. But the programming team does not always consist of experienced members.

When working on ISE/ISS deliverables or when handling many studies for the same customer, programmers may find it expedient to reuse the existing programs to generate the outputs. Below provided are some tips that programmers can use while creating a program.

PROGRAMMING APPROACHES

PRGM1: CREATING QNAMS FOR SUPPXX DATASETS

Consider an example below, a dummy AE dataset with supplementary variables AETERM1, AETERM2, AETERM3, AETERM4, AETERM5 ACTOTH. Where AETERM1- AETERM5 originated from the variable AETERM which has observation with length more than 1500.

```
data ae;
  array dummy $20 studyid domain usubjid aeterm1 aeterm2 aeterm3 aeterm4 aeterm5
actoth;
  do aeseq=1 to dim(dummy);
    dummy(aeseq)='X';
  end;
run;
```

and the programmers may write the SAS code as below for generating SUPPAE

```
proc transpose data=ae out=ae_t name=qnam label=qlabel;
  var aeterm1 aeterm2 aeterm3 aeterm4 aeterm5 actoth;
  by studyid domain usubjid aeseq;
run;
```

```
                  *OR;
proc transpose data=ae out=ae_t name=qnam label=qlabel;
  var aeterm1-aeterm5 actoth;
  by studyid domain usubjid aeseq;
run;
```

Possible issues: As supplementary variables AETERM1- AETERM5 depends on the length of the original variable AETERM and which can be changed based on the data.
1) There can be another variable AETERM6,
2) No existence of AETERMx variable based on the data.

```
data ae;
  array dummy $20 studyid domain usubjid aeterm1 aeterm2 aeterm3 actoth;
  do aeseq=1 to dim(dummy);
    dummy(aeseq)='X';
  end;
run;
```

Log warning if we used the above code to generate SUPPAE based on the above data.

```
31
32   proc transpose data=ae out=ae_t name=qnam label=qlabel;
33     var aeterm1 aeterm2 aeterm3 aeterm4 aeterm5 actoth;
ERROR: Variable AETERM4 not found.
ERROR: Variable AETERM5 not found.
34     by studyid domain usubjid aeseq;
35   run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.AE_T may be incomplete.  When this step was stopped there were 0 observations and 0 variables.
WARNING: Data set WORK.AE_T was not replaced because this step was stopped.
NOTE: PROCEDURE TRANSPOSE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

36                   *OR;


37   proc transpose data=ae out=ae_t name=qnam label=qlabel;
38     var aeterm1-aeterm5 actoth;
ERROR: Variable AETERM5 not found.
39     by studyid domain usubjid aeseq;
40   run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.AE_T may be incomplete.  When this step was stopped there were 0 observations and 0 variables.
WARNING: Data set WORK.AE_T was not replaced because this step was stopped.
NOTE: PROCEDURE TRANSPOSE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Suggested Method

Option1: When there are no metadata files available

```
*Identifying the QNAMs;
*Here AETERM is a standard variable in SDTM.AE and no need to consider the same in SUPP;
proc sql noprint;
  select distinct name into: qnams separated by ' '
  from sashelp.vcolumn
  where libname="WORK" and memname="AE" and
  (name in ("ACTOTH")or (find(name,'aeterm','i') and ~missing(compress(name,,'kd'))));
quit;
```

```
proc transpose data=ae out=ae_t name=qnam label=qlabel;
  var &qnams;
  by studyid domain usubjid aeseq;
run;
```

Option2: When there is a metadata available for domain (recommended approach)

```
*Comparing variables when there is a metadata library available with list of all variables in standard domain(SDTM.AE);
proc sql noprint;
  select distinct name into: qnams separated by ' '
  from (select distinct name from sashelp.vcolumn where libname="WORK" and memname="AE")
  where name not in (select distinct name from sashelp.vcolumn where libname="METADATA" and memname="AE");
quit;
```

```
proc transpose data=ae out=ae_t name=qnam label=qlabel;
  var &qnams;
  by studyid domain usubjid aeseq;
run;
```

PRGM1A:

The above approach can be used in concatenating variables. For example, if we need to present the adverse event in listing for the above case, we need to concatenate AETERM, AETERM1, AETERM1, etc. As AETERM is a required variable in SDTM problems may occur.

1) Existence of AETERMxx, or

2) Reusing the program for another study. So, suggesting to not write the code like catx(' ',of AETERM, AETERM1).

```
*Identifying the available TERMS;
proc sql noprint;
  select distinct name into:aeterms separated by ', '
  from sashelp.vcolumn
  where libname="WORK" and memname="ADAE" and find(name,'aeterm','i');
  select distinct max(200,sum(length)) into:aelen
  from sashelp.vcolumn
  where libname="WORK" and memname="ADAE" and find(name,'aeterm','i');
quit;
```

```
data adael;
  length aeterm_ $&aelen.;
  set adae;
  aeterm_=catx(" ",of &aeterms);
run;
```

PRGM2: AVOIDING TRUNCATION WHILE CONCATENATING VARIABLES

Concatenating two or more variables is a common occurrence in programming and programmers are exploring various ways to get outputs. It is the duty of the programmer to ensure that no truncation has occurred in programming. Without any custom check, programmers can ensure the quality with the use of cat function. The code given below is the two different concatenation methods, but only the catx shows an alert when truncation occurs:

```
data example1;
  length conc $10;
  set sashelp.cars (obs=2);
  conc=strip(model)||"-"||strip(type);
run;

data example2;
  length conc $10;
  set sashelp.cars (obs=2);
  conc=catx("-",of model,type);
run;
```

Log: Log report of trial one and trial 2

```
374  data example1;
375    length conc $10;
376    set sashelp.cars (obs=2);
377    conc=strip(model)||"-"||strip(type);
378  run;

NOTE: There were 2 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.EXAMPLE1 has 2 observations and 16 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds


379
380  data example2;
381    length conc $10;
382    set sashelp.cars (obs=2);
383    conc=catx("-",of model,type);
384  run;

WARNING: In a call to the CATX function, the buffer allocated for the result was not long enough to contain the
         concatenation of all the arguments. The correct result would contain 20 characters, but the actual result might
         either be truncated to 10 character(s) or be completely blank, depending on the calling environment. The following
         note indicates the left-most argument that caused truncation.
NOTE: Argument 2 to function CATX('-',' RSX Type S '[12 of 40 characters shown],'Sedan    ') at line 383 column 8 is invalid.
CONC=  MAKE=Acura MODEL=RSX Type S 2dr TYPE=Sedan ORIGIN=Asia DRIVETRAIN=Front MSRP=$23,820 INVOICE=$21,761 ENGINESIZE=2
CYLINDERS=4 HORSEPOWER=200 MPG_CITY=24 MPG_HIGHWAY=31 WEIGHT=2778 WHEELBASE=101 LENGTH=172 _ERROR_=1 _N_=2
NOTE: There were 2 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.EXAMPLE2 has 2 observations and 16 variables.
NOTE: DATA statement used (Total process time):
      real time           0.05 seconds
      cpu time            0.00 seconds
```

PRGM3: MACROTISE THE FORMATS VALUE

This approach will aid in enhancing the quality of TFL outputs, and the developer must adhere to the shell in order to progress with the programming. Consider a scenario where one parameter is presented for one treatment and no records for others and as per shell, we need to present it as 0 count. This can be implemented with the help proc format and procedure like proc means, proc tabulate etc. In proc format we need to provide the formats for parameter and treatment, while copying the code to another study the treatment group values can be different. An example of code that a programmer can use to generate formats from data is shown in the following screenshot.

```
proc sql noprint;
  select distinct cats('"',trt01p,'"="',trt01pn,'"') into: trtmac separated by ' '
  from adsl;
quit;

proc format;
  value $trtfmt
    &trtmac;
run;
```

USER-DEFINED ERRORS / WARNINGS

User-defined checks are very important or helpful in programming scenarios to ensure the quality of outputs when there are issues in the data or programmer followed any temporary approaches.

CHECK1: STUDY SPECIFIC/ TEMPORARY APPROACHES

Consider a scenario where programmer follows an approach in one study based on sponsor confirmation and later the program was reused for another study for the same sponsor. In this case there is a chance that the approach can be followed in another study without the confirmation of a sponsor. The code below is an example that helps the programmer to ensure that no study specific approaches are copied from another study.

```
*When there is a global macro available for study, no need to use the below code;
data adsl;
  set adsl end=ls;
  if ls then call symput ('studyid',studyid);
run;

data adae4;
  set adae3; *Where ADAE4 is a dataset which has TFL related transformations;
  *Based on sponsor confirmation, when AAFL=Y and BBFL is missing then set BBFL to Y;
  if "&studyid"='XYZ-001' then do;
    if cmiss(bbfl) and aafl="Y" then bbfl="Y";
  end;
    else put "WAR" "NING: Study specific approach is copied from &studyid study";
run;
```

Log: Log report of the code when used in another study

```
15
16   data adae4;
17     set adae3; *Where ADAE4 is a dataset which has TFL related transformations;
18     *Based on sponsor confirmation, when AAFL=Y and BBFL is missing then set BBFL to Y;
19     if "&studyid"='XYZ-001' then do;
20       if cmiss(bbfl) and aafl="Y" then bbfl="Y";
21     end;
22       else put "WAR" "NING: Study specific approach is copied from &studyid study";
23   run;

WARNING: Study specific approach is copied from ABC-001 study
NOTE: There were 1 observations read from the data set WORK.ADAE3.
NOTE: The data set WORK.ADAE4 has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

CHECK2: WHILE USING USER-DEFINED FORMATS

Formats are commonly using pieces of code in programming and can be a cause of issues while reusing program in other projects. For example, consider a scenario where a programmer used a format to display the values as mentioned in the specification and later reused the program in another study. In this case, the issue can be occur with values which are not presented in the dataset of project 1 but presented in the dataset of project 2.

```
*Formats used for the example program;
proc format;
  value $efrmt
  "E1"="Example1"
  "E2"="Example2"
  "E3"="Example3"
  "E4"="Example4"
  other="XXX";
run;
```

```
data study1;
  set e1;
  ex_fmt=put(ex,efrmt.);
  if evarl="XXX" then put "WAR" "NING: Format EFRMT need to be updated for the value-" ex;
run;
```

```
data study2;
  set e2;
  ex_fmt=put(ex,efrmt.);
  if ex_fmt="XXX" then put "WAR" "NING: Format EFRMT need to be updated for the value-" ex;
run;
```

Log: Log report of the same code in used study 1 and study 2

```
39   data study1;
40     set e1;
41     ex_fmt=put(ex,efrmt.);
42     if ex_fmt="Data Check" then put "WAR" "NING: Format EFRMT need to be updated for the value-" ex;
43   run;

NOTE: There were 5 observations read from the data set WORK.E1.
NOTE: The data set WORK.STUDY1 has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

```
45   data study2;
46     set e2;
47     ex_fmt=put(ex,efrmt.);
48     if ex_fmt="Data Check" then put "WAR" "NING: Format EFRMT need to be updated for the value-" ex;
49   run;

WARNING: Format EFRMT need to be updated for the value-E5
NOTE: There were 5 observations read from the data set WORK.E2.
NOTE: The data set WORK.STUDY2 has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

To avoid unnoticed issues, programmers can create user-defined errors or issue warnings in such cases. In SDTM, to ensure only one baseline value is populated for a subject in a test, temporary check to ensure that data issue is resolved, and so on are some examples of where a programmer can apply these checks.

MACROS

Macros offer an excellent way to automate the process. Standard macros are macros that have clear definitions and are checked for all potential causes that the standard macros have preferences over the macros generated by the programmer within the programs. So, it is recommended to use such macros or tools to ensure the quality of the data. A few examples of macros and codes are noted below

MACRO 1: TO ENSURE THAT ALL VARIABLE LENGTHS OVER 200 ARE PROPERLY MAPPED

As per SDTM standard, a user can populate 200 characters in free text variables, the rest will be mapped into SUPP. SDTM programming might begin before the database lock and there is a high risk of modifying the data from the previous extract. As part of multiple data extracts, it is the responsibility of the programmer to ensure that all the variable values are successfully mapped to the SDTM domain. How does the programmer ensure a variable with a length of 200 or more is successfully mapped? We recommended the creation of a macro to ensure that the values are correctly mapped. Currently no macros are available for the same, the method provided below is an option for ensuring the quality in such a situation

The 'VARIDENTIF' is a variable ID macro to indicate whether the variable in a library is longer than the provided length.

- The macro will identify the variable with a length longer than the specified length and output to the specified location (provided in the macro) in the name 'Variables List,'
- The programmer will then have to check the variables against the specification,
    - Populate value 'Y' in SDTMVAR column if the variable is correctly mapped.
    - If the variable is [NOT SUBMITTED], populate value 'Y' in RMVDVAR column.
- For the next run onwards, the program will read the value from the existing file and populate a warning when a new variable is identified or missing in both SDTMVAR and RMVDVAR.

The input required for the macro are divided into 3, DIR-is location of Variables List.csv. LIB- input library used for the process; default value is ‘raw’. LEN is the value used for comparison: default value is 200:

```
%macro varidentif(dir=, lib=raw,len=200);
  %*Step 1.Identifying the variables with length more than 200;
  data varlen;
    set sashelp.vcolumn;
    %*Subsetting the data based on the input from macro;
    where libname=upcase("&lib") and type="char" and length gt &len.;
    %*Removing unwanted files, programmer can update this section based on the data;
    if find(name,"_raw","i") then delete;
    %*Flag used to ensure the variable mapping;
    sdtmvar='';
    rmvdvar='';
    keep memname name sdtmvar rmvdvar;
  run;

  %*2.Outputing this in a external location, if already existed step 1a will work else
step 2;
  %*Step 1a - Checking whether the required file exists or not;
  %if %sysfunc(fileexist(&dir\Variables List.csv)) %then %do;
    %put File already exist;
    proc import file="&dir\Variables List.csv"  dbms=csv out=varlen_old replace;
```

```
      getnames=yes;
      guessingrows=100000;
    run;

    proc sort data=varlen;
      by memname name;
    run;

    proc sort data=varlen_old;
      by memname name;
    run;

    data varlen1;
      merge varlen(in=a) varlen_old(in=b);
      by memname name;
      if a and not b or cmiss(sdtmvar,rmvdvar)=2 then put "WARNING: variable- " name
"need to check against specification";
    run;

      %*Replacing the external file with latest information;
      proc export data=varlen1 dbms=csv outfile="&dir\Variables List.csv" replace;
      run;
  %end;
    %*Step 2;
    %else %do;
      proc export data=varlen dbms=csv outfile="&dir\Variables List.csv" replace;
      run;
    %end;
%mend varidentif;
```

MACRO 2: MACROS TO AUTOMATE SDTM/ ADAM FORMATS FROM SPECIFICATION

Standardizing the variable values as per controlled terminology is a process that the programmer needs to follow while developing SDTM/ADaM. If the organization is following a P21 specification template for SDTM/ADaM then there will be a tab called 'codelist' containing the CDISC submission values or custom values when the codelist is extensible. The issue here can arise with the custom values or codelist used in a study. For example, the person who write the specification used some custom codelist values in specification and later the values changed due to some discussion and missed to inform the programmer about the updates. This will be identified only at the stage of P21 validation. So, we suggest using a macro for automating the formats from specification. In the 'codelist' tab an additional column needs to be included for the macro creation and let's call it as 'Original Value' (contains the values in the raw datasets against that coded value). So as per the P21 template we have the information like variables ID, decoded value, and original value. Now instead of creating the format manually, we can create a SAS macro for formats based on the decoded value and the original value directly from the specification.

```
%macro autoformats(dir=,infile=,rown=50000,flen=10000,cchk=XXX,nchk=99);

  *calling the input file from the specified location;

  options noxwait noxsync validvarname=upcase;

  *Opening the specification;

  %sysexec "&dir\&infile";


  *Referencing the codelist tab, rows 2 to 8 from the specification;

  filename forfile dde "Excel|Codelists!R1C1:R&rown.c10";


  *Base datasets for the macro;

  data autof_base;

    infile forfile missover dsd firstobs=2 notab dlm='09'x ;
```

```
    input id  :$50.name  :$200. nci_code:$50. otype :$20. order :8. term :$200.
nci_term :$50. decod :$200. rtype :$20. rvall :$1000.;
  run;


  *Closing the infput excel file;
  filename forfile dde 'excel|system' ;


  data _null_;
    file forfile ;
    put '[close("&dir\&infile")]';
    put '[Quit]';
  run;


  *Identifying uniq IDs, number of ID's used for working the dop loop, maximum number
of values across the IDs;
  proc sql;
    create table autof_uid as
    select distinct *,monotonic () as idnum,strip(put(count(id),best.)) as idcnt
    from (select distinct id,strip(put(max(order)+1,best.)) as maxfval from
autof_base);
  quit;


  proc sort data=autof_base;
    by id;
  run;


  data autof_model;
    merge autof_base autof_uid end=ls;
    by id;
*Combining formats values for do loop working;
if lowcase(rtype) in ("text" "char") and lowcase(otype) in ("text" "char") then
format='"'||strip(rvall)||'"'||'='||'"'||strip(term)||'"';
  else if lowcase(rtype) in ("text" "char") then
format='"'||strip(rvall)||'"'||'='||strip(term);
  else if lowcase(rtype) in ("integer" "float") and lowcase(otype) in ("integer"
"float") then format=strip(rvall)||"="||strip(term);
  else if lowcase(rtype) in ("integer" "float") then
format=strip(rvall)||'='||'"'||strip(term)||'"';
*Macro IDs for programming purpose;
macid_type='rtpe'||strip(put(idnum,best.));
macid_fname='fname'||strip(put(idnum,best.));
    if last.id then do;
      call symput(macid_type,rtype);
      call symput(macid_fname,id);
```

```
    end;
*For Do loop;
if ls then call symput('datcnt',idcnt);
    *For transpose;
if ls then call symput('valcnt',maxfval);
  run;

  proc transpose data=autof_model out=autof_model_t;
    var format;
by id idnum otype;
  run;

  data autof_model1;
    length format $&flen.;
    set autof_model_t;
if lowcase(otype) in ("text" "char") then col&valcnt.="other='"||"&cchk"||"'";
  else col&valcnt.="other=&nchk";
format=catx(' ',of col1-col&valcnt.);
macid_fval='fval'||strip(put(idnum,best.));
    call symput(macid_fval,format);
  run;

  %do i=1 %to &datcnt.;
    %if %sysfunc(lowcase(&&rtpe&i))=text %then %do;
      proc format;
        value $&&fname&i. &&fval&i.;
      run;
    %end;
      %else %do;
        proc format;
          value &&fname&i. &&fval&i.;
        run;
    %end;
  %end;

  *Deleting datasets created in the macro;
  proc datasets lib=work nolist;
    delete autof_:;
  run;
%mend autoformats;
```

CONCLUSION

Experience is the core factor that can used to improve the quality of the outputs. Expect the possible issues while developing your code and your code should be handled to avoid such possible issues. Data issues can be identified at the initial stage of programming and that might be documented as part of the project documentation. In addition, certain data issues identified at the initial run might not always be necessarily resolved in the next data transfer. This is where custom errors / warnings play a significant role. For such cases the programmer can apply the user defined errors/warning to add an action flag in programmer. Given the dire consequences of poor quality, it is strongly advised that these frequently used defined checks be implemented in automation programmers.

ACKNOWLEDGMENTS

The content, ideas and recommendations presented in this paper are all developed from experiences in our career. These experiences come through previous companies, various industry leaders, colleagues, mentors, conferences, and direct experience.

CONTACT INFORMATION

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

Author Name : Akhil Vijayan
Company : Genpro Research PVT LTD
Address : Second Floor, Nila Building Technopark, Campus
City / Postcode : Trivandrum, Kerala, India
Work Phone : +91 9061208074
Email : akhil.vijayan@genproresearch.com
Web : www.genproresearch.com

Co-author Name : Limna Salim
Company : Genpro Research PVT LTD
Address : Second Floor, Nila Building Technopark, Campus
City / Postcode : Trivandrum, Kerala, India
Work Phone : +91 8086272364
Email : limna.salim@genproresearch.com
Web : www.genproresearch.com

Co-author Name : Roshan Stanly
Company : Genpro Research PVT LTD
Address : Second Floor, Nila Building Technopark, Campus
City / Postcode : Trivandrum, Kerala, India
Work Phone : +91 9745266416
Email : roshan.stanly@genproresearch.com
Web : www.genproresearch.com