PharmaSUG 2017 - Paper TT14

Defensive Programming -- Tips and Techniques for Producing that Dataset, Table, Listing or Figure First Time

By David Franklin, QuintilesIMS

ABSTRACT

If you have ever bought one of those 'put together furniture kits', it is not usually an easy process. There is the setting up of the work area, taking out of the box, reading the instructions, getting the tools needed that were not included, construction, and finally look it over and check what you have done and that it is going to do as intended.

A similar process is used for writing programs – we should first do some planning, construct the program, and check it over to see that it producing what was asked. This paper takes a brief and lighthearted look at each of these stages, providing a few tips for avoiding some of the many pitfalls and gives a few pieces of SAS® code that are useful in the development of your program, hopefully avoiding having to say that dreaded phrase, "Oh Clanger".

INTRODUCTION

When beginning to put a kit together there are three main stages; getting equipment and materials together, and scheduling; building; and then checking everything is as expected. In a similar fashion, there are three basic stages to writing our SAS code:

- planning
- writing the program(s)
- testing and validating

This presentation will present some useful ideas in these three stages, hopefully to avoid any programming issues, and having to say "Oh Clanger".

PLANNING

There are four key documents that you must have access to before programming. These are:

- Study Protocol
- Study SAP
- Study Output Shells
- CRF (would be nice if this was annotated)

Also helpful is to do a PROC CONTENTS of any datasets. A useful macro is listed in Appendix A that produces a Directory listing of datasets, a contents listing and printing of the first 20 observations of each dataset, and a listing of formats. An example of the first output is shown in Output 1, below:

Dataset Directory

Member Name	Data Set Label	Date Created	Number of Physical Observations	Number of Variables
AE	Adverse Events	30MAR17:09:31:00	646	62
CM	Concomitant Medications	30MAR17:09:32:00	2964	60
DM	Demographics	30MAR17:09:33:00	79	29
DS	Disposition	30MAR17:09:34:00	362	35
EG	ECG Test Results	30MAR17:09:35:00	69	41
EX	Treatment Exposure	30MAR17:09:36:00	1474	55
FA	Findings About	30MAR17:09:37:00	748	31
IE	Inclusion/Exclusion Criteria Not Met	30MAR17:09:38:00	62	18
LB	Laboratory Test Results	30MAR17:09:39:00	22567	58
MH	Medical History	30MAR17:09:40:00	321	41
PE	Physical Examination	30MAR17:09:41:00	3687	36
QS	Questionnaires	30MAR17:09:42:00	21632	44
SC	Subject Characteristics	30MAR17:09:43:00	41	20
SV	Subject Visits	30MAR17:09:44:00	2876	27
VS	Vital Signs	30MAR17:09:45:00	6412	38

Output 1: Directory Listing of Datasets

When planning, look for similar outputs where a population and/or a variable cut is the only difference, then consider a setup where a program calls a macro. A good example of this is the AE tables where they are the same layout and use the same code for counting but only differ in the variable used to select the Adverse Events for the count and/or a population. Another example is if the Demographics table is generated for Safety and ITT populations – the code is the same, the only difference is in the population selection. The advantages of using macros are:

- only one set of code needs to be written
- change is the difference in one or more parameter values
- only one set of code needs to be maintained
- only one set of code needs to be QC'ed

WRITING THE PROGRAM(S)

NO HARDCODES

The title says it all. A hardcode maybe obvious, for example:

```
if subjid='001001' then height=178;
```

or less obvious

```
if stop_y>year("&sysdate9"d) then stop_y=year("&sysdate9"d);
```

If hardcodes are requested try to talk the requester out of it as it creates an integrity issue with the data. If hardcodes are necessary then this should be indicated in the log with a very clear message. One good way of doing a hardcode is given in the following example:

```
if subjid='001001' then do;
    if birthdt='13MAY2017'd then do;
        birthdt='13MAY1970'd;
        put 'WAR' 'NING: Hardcode for patient 001001 to correct birth date issue.';
    end;
    else put 'WAR' 'NING: Hardcode for patient 001001 should be reviewed as DOB '
             ' in DM database has changed';
End;
```

Occasionally programmers will use a “time limited hardcode”, as the following example shows:

```
if subjid='001001' and date()<="31JAN2017"d then
  birthdt='25MAY1970'd;
```

Unfortunately this type of hardcode is used far too often and does not allow for any data point change to reflect in the database, for example, what if the BIRTHDT value was corrected to 25MAY1971 before a run date of 31JAN2017?

No hardcodes should exist in the final version of a table, listing or figure. A hardcode is different from a conversion as is done in vital signs and lab data, just to name a few.

PROGRAM TO THE eCRF

The structure of the CRF is useful when writing the code. There are three types of variable – database, monitoring and data variables. Although eCRF has helped in trapping database errors don't assume that it works every time. Try to use data variables wherever possible. An example of a database variable is

```
Record ID
```

a monitoring variable is

```
Are there any Adverse Events? (Y/N)
```

and a data variable is

```
Adverse Event Description
```

ASSUME INVALID AND DUPLICATE DATA

Invalid or Duplicate data is something that should be considered when writing SAS code. If there are multiple records possible then discuss with a statistician which record to take for analysis and make a note of it in the program as a comment. The following code fragment shows the use for a check of duplicates and invalid data:

```
if n(hghtval) then do;
  select(hghtunt);
    when('CM') hghtsi=hghtval;
    when('IN') hghtsi=hghtval*2.54;
    otherwise
      put 'WAR' 'NING: Unexpected or unknown '
      'height unit: ' subjid= hghtval= hghtunt=;
  end;
  end;
  if sum(first.subjid,last.subjid)<2 then
    put 'WAR' 'NING: Unexpected multiple result: '
    subjid= hghtval= hghtunt=;
end;
```

If the variable being analyzed is a numeric but the value is stored as a character it is best to do any comparisons using numeric values. This can be best shown in the following example:

```
if ^missing(labvalue) and labvalue^='0';
```

This line of code is okay when zero means '0' but during the life of a database a value of '0.00' may included that will not be caught.

Do not be afraid to use the log or PRINT procedure to output data considered dubious, the latter usually going to an LST file with the same program name that will not go to Medical Writing.

DATES

This is a special category of its own. If a full date is expected for a particular variable, e.g. Date of Birth, Treatment Date or Concomitant Medication date, then treat it as a full date and put out an exception to a log or LST file if a partial date is given. Only try to write code to deal with partial dates if partial dates are expected.

AVOID OVERWRITING DATABASE VARIABLES

It is not good programming practice to write over data in database variables, even within a program where a temporary (work) dataset is being used. It makes it harder to track the flow of data when data from the stored dataset is being overwritten in the work area.

INITIALIZE VARIABLES

At the very least it is good programming practice to define the type and length of a new variable in a data step. This avoids the issue where SAS will assume a type and length of a newly created variable while the programmer may want another type and/or length.

USE COMMENTS

This makes the program easier to understand when someone else other than yourself is looking at the program – that program may be looked at again in five years time!

USE A PROGRAM HEADER

It is good practice to use the program header. From an informational view the two most important fields are “WHY” and “NOTES” as the former say why the program was written the latter gives any information that shows any particular oddities that may be needed by anyone looking at the program in the future. Even if you are just changing a line of code for whatever reason, keep the NOTES field up to date.

DON’T BE AFRAID TO USE THE HELP BUTTON

If you are not sure of a SAS statement, function or procedure do click the help button or ask someone. To use an example

```
select trtcd, count(*) as count
```

is different to

```
select trtcd, count(age) as count
```

The first line will count the number of observations in the dataset, but the second will count the number of observations with non-missing values.

TESTING AND VALIDATING

CHECK THE LOG

This cannot be emphasized enough. Attached in Appendix B is an example of a small QC macro that goes through the SAS Logs specified to check if there are programming issues that need resolution. As a general rule nothing should be found, however there may be instances where this may not be possible but these should be explained somewhere.

REVIEW THE OUTPUT

Just have a quick look at the output and check if the N's are as expected (n should always be less than or equal to N), that page breaks are okay and see if there are results for each parameter are around what is anticipated. Maybe consider doing a quick PROC TABULATE to check that nothing was lost in the code to make the data useful for the REPORT procedure call. It is good practice to spell check the titles and footnotes, and make sure that footnote references actually align with references in the output.

CONCLUSION

While it is not possible to eliminate all the “Oh Clanger” when writing programs for clinical output, this paper has presented some ideas for avoiding at least the majority of them.

CONTACT

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

David Franklin
QuintilesIMS
david.franklin1@quintilesims.com
quintilesims.com

APPENDIX A: PRTCONT MACRO -- DESCRIBE YOUR DATA

```
%macro prtcont(libdir= /*Libname where data is*/);
   title1 "Dataset Directory";
   proc print data=sashelp.vtable noobs label;
      var memname memlabel crdate nobs nvar;
      where libname="%upcase(&libdir)";
   proc sql noprint;
      select memname into :dslist separated by ' '
      from sashelp.vmember
      where libname in("%upcase(&libdir)");
      quit;
   run;
   %let i=1;
   %do %while(%scan(&dslist,&i) ne );
      title1 "Dataset: %scan(&dslist,&i)";
      proc contents data=%upcase(&libdir).%scan(&dslist,&i) nodetails varnum;
      proc print data=%upcase(&libdir).%scan(&dslist,&i) (obs=20);
      run;
      %let i=%eval(&i+1);
   %end;
   proc format fmtlib;
      title1 "Listing of Formats Set";
   run;
%mend;
```

APPENDIX B: QC MACRO -- CHECK YOUR SAS LOG(S) FOR ISSUES

```
*Define location of LOG Files to process;
%let logdir=%str(C:\MyStudy\Tables);

*Bring in ALL SAS LOG(s);
data _alllogs0;
   attrib _txt length=$256 informat=$char256. format=$char256.
          _fn length=$256
          _myinfile length=$256
          _ln length=8
          _myinline length=8;
   infile "&logdir.\*.log";
          lrecl=256 filename=_myinfile line=_myinline length=len;
   input _txt $varying256. len;
   _fn=scan(strip(_myinfile),-1,'\');
   _ln=_n_;
proc sort data=_alllogs0;
   by _fn _ln;
run;

*Output findings;
title1 "QC of SAS LOG(s)";
data _null_;
   retain _k 0; *Internal counter;
   file PRINT;
   set _alllogs0 end=eof;
   by _fn _ln;
   if first._fn then do;
      put // @1 '***** LOG FILE: ' _fn /;
      _k=0;
   end;
   if index(_txt,'ERROR:') or
      index(_txt,'WARNING:') then do;
      _k+1;
      put _txt;
   end;
   if last._fn and _k=0 then put @1 '** NO ISSUES FOUND **';
   if eof then put / '/*EOF*/';
run;
```