How to QC your own programs
Kevin Lee, Therakos, Exton, PA

ABSTRACT
This paper is intended for the Statistical SAS® programmer who produces analysis SAS data sets, tables and listings. It will help the SAS programmer to minimize the possible errors in mapping and Clinical Study Report (CSR) programs by developing Quality Control (QC) procedures in his or her own programs. The QC procedures will be introduced as macros and simple methods in SAS code; both macros and methods are included in the programs.

INTRODUCTION
As a SAS programmer, you may have a hard time applying QC to your own programs. Since you have a bias toward your own programs, you tend to overlook your mistakes in the coding. Therefore, if you set up the standard QC procedures, you will be able to put aside personal feelings toward your own code. The following procedures will help you to QC your own programs.

EXPLANATION
In a clinical trial, as the Statistical SAS programmer, you derive the analysis SAS data sets from raw SAS data sets. The mapping programs that you prepare will convert the raw to analysis data sets. From the derived analysis data sets, you prepare CSR reports- tables, listings and sometimes graphs. The CSR programs will generate the CSR reports. So, in clinical trials, the Statistical Clinical SAS programmer mainly prepares two different types of SAS programs. One type are mapping programs and the other type are CSR programs. The following QC processes will involve both or one of types.

QC FOR MAPPING PROGRAM
We will discuss four methods to QC the mapping programs – checking with the Define Document, removing any duplicate records, comparison the modified data with old data, and checking SAS log files.

CHECKING WITH DEFINE DOCUMENT
Most importantly, the SAS mapping programs need to capture all the data with correct formats. In order to ensure capturing all the data, you need to compare the prepared data with the Define Document. The Define Document contains the information of data set, variable, its length, format and label. Below is a sample Define Document from Excel spread sheet.

<table>
<thead>
<tr>
<th>DSET</th>
<th>Name</th>
<th>Type</th>
<th>Length</th>
<th>Label</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEMO</td>
<td>STUDYID</td>
<td>char</td>
<td>8</td>
<td>Study ID</td>
<td>$8.</td>
</tr>
<tr>
<td>DEMO</td>
<td>USUBJID</td>
<td>char</td>
<td>20</td>
<td>Unique Subject ID</td>
<td>$20.</td>
</tr>
<tr>
<td>DEMO</td>
<td>AGE</td>
<td>num</td>
<td>8</td>
<td>Age in Years at Baseline</td>
<td>8.</td>
</tr>
<tr>
<td>DEMO</td>
<td>SEX</td>
<td>char</td>
<td>1</td>
<td>Sex</td>
<td>$1.</td>
</tr>
<tr>
<td>DEMO</td>
<td>RACE</td>
<td>char</td>
<td>20</td>
<td>Race</td>
<td>$20.</td>
</tr>
</tbody>
</table>

The code shown below will convert the above Excel spread sheet to a temporary SAS data set, _stds, and compare it with the derived demo data set, DEMO.
**** prepare data information from define doc;  
proc import dbms=excel2000 out=_stds  
datafile="C:\study000\document\dataspec_study000.xls" replace;  
run;  
data _stds2;  
  set _stds;  
  where dset = “DEMO”;  
run;  
**** obtain the data information from the derived data set;  
proc contents data=demo out=meta(keep=memname memlabel name  
type length label format);  
run;  
**** Compare define document and derived data sets;  
data name_not_data name_not_def dif_length ...,;  
  merge _stds2(in=a) meta(in=b);  
  by name;  
    if a and not b then output name_not_data;  
    if not a and b then output name_not_def;  
    if a and b then do;  
      ** difference in length of variables;  
      if length ne d_length then output dif_length;  
      ** difference in formats;  
    end;  
run;  
**** List of variable names that are not in the data sets, but in define  
documents;  
proc print data=name_not_data; run;  
**** List of variable names that are not in the define documents, but in  
data;  
proc print data=name_not_def; run;  
.....

The above code will print out the variable names that are in the define document, but not in demo data set and vice versa. You can add more listings such as variable format and length.

The above sample code could be included in a macro (ex, %comp_def(ds)).

REMOVING ANY DUPLICATE RECORDS IN THE ANALYSIS DATA SET
You also need to see if there are any duplicate records in the analysis data sets. For example, it is assumed, because it is true almost all the time, that DEMO data set should not contain two identical subjects. The following code will check whether there are any duplicate subjects in demo and if there are, such subjects will be provided as output.

proc sort data=demo; by subject; run;  
**** remove any duplicate subjects;  
proc sort data=demo out=outds nodupkey; by subject; run;
**** lst_dup is the listings of duplicate records;
proc sql;
    create table lst_dup as
    select * from demo
    except all
    select * from outds
quit;

**** List of duplicate records;
proc print data=lst_dup; run;

These codes will be better served in a macro like below.

%macro dup_lst(ds, sortvar);
    proc sort data=&ds; by &sortvar ; run;
    ....
%mend;

COMPARISON MODIFIED DATA WITH OLD DATA
You need to check whether you made the correct modification in the program. Before replacing old with new data set, you need to compare old and new data sets to make sure that there are no unwanted changes. Before writing demo in analysis data folder, you can compare the previous demo with the new demo data set.

Libname old 'C:\study000\analysisdata':
proc compare base=old.demo compare=demo;
id subject;
run;

The above sample code will compare the old permanent SAS data set in libref old, DEMO, with new SAS data set, DEMO, and provide the results of the comparison. This code could be prepared in a macro (ex, %comp_old(ds)).

CHECKING THE SAS LOG FILES
After writing SAS code and successfully running the code, you need to check the log. You should not only look for error and warning messages, but also for uninitialized variables and notes. You could check the log right after running each program.

In addition, you can check the logs of all the mapping programs. The first step is to save each log to subfolder. The following code will be included in the end of each program.

dm log "print file = 'C:\study000\mapping\loglist\demo.log' replace";

This code is prepared in a macro(ex. %logsave(ds)). It will save the log of demo to the subfolder of loglist.

The second step is to write an error check program that contains the following code.

**** Find error and warning message;
data f_error f_warning;
    attrib fname f_name format=$200.;
infile " C:\study000\mapping\loglist\*.log " filename=fname;
f_name = fname;
    input txt $200.;
    if index(upcase(txt),'ERROR') > 0 then output f_error;
    if index(upcase(txt),'WARNING') > 0 then output f_warning;
    ....
run;
This error check code reads all the log files in c:\study000\mapping\loglist. Then, it finds the lines that contain the word ‘error’ and ‘warning’. You can also include other categories such as ‘warning’, ‘uninitialized’ and so on.

The example of mapping program that uses the four macros explained above is included in appendix A. To help the understanding, all four macros are highlighted.

** QC FOR CSR PROGRAM **

We will discuss three methods to QC the CSR programs – checking SAS log files, checking the spelling errors in the CSR reports, and checking the modifications in the program generate the intended reports.

** CHECKING THE SAS LOG FILES **
First of all, just like in the mapping programs, you need to check the logs in the CSR programs. You can prepare the CSR programs the same way as the mapping programs.

** CHECKING THE SPELLING ERRORS IN THE CSR REPORTS **
You need to check any spelling and grammar errors in the output reports-summary tables and listings. Usually those spelling errors occur in titles, footnotes, column headers and so on. So, if you are able to assemble all the reports into one report, you will have much easier time to check the spelling errors.

In order to put all the tables and listings in one report, you should prepare two steps. The first step is to save each table and listing to the subfolder of out. The following codes will explain the first step.

```sas
filename printlst "c:\study000\programs\out\ldemo.lst";
proc printto print=printlst;
run;

title1 ;
title2 ;
proc report data=final;
column ..........;
run;
footnote1 ;
footnote2 ;
proc printto; run;
```

You will put this SAS code in a macro (ex, `%prt_lst(ds);`). The second step is to prepare the program that collects all the tables and listings. Since some listings such as lab data can be very long, it is encouraged to collect the first or second page of each table and listing. Below is the sample code that collect all the reports together.

```sas
data all_report;
   attrib fname f_name format=$200.;
   retain f_name line;
infile " C:\study000\programs\out\*.lst " filename=fname;
if f_name ne fname then do;
   f_name = fname;
```
line = 0;
end;
line + 1;
input txt $200.;
** collect only the first 20 lines of report; 
if line < 20 then output;
......
run;

The final step is to check the spellings in this report. This procedure will help you to check the spelling in all tables and listings without opening all of them.

CHECKING THE MODIFICATIONS IN SAS PROGRAM GENERATE THE INTENDED CSR REPORT

You need to check whether any modifications in CSR program correctly modify the reports. You need to check the new report with the old one. However, unlike the mapping programs, the CSR programs generate the final output in report format, not as the data set. So in order to compare properly, you need to prepare a SAS data set and perform the comparison. The first step is to convert the report into a SAS data set. The following code will convert the report output in proc report to a SAS data set.

```sas
proc report data=final out=dtable;
    column .......;
run;
```

```sas
libname datatable "c:\study00\programs\datatable";
data datatable.table_14_2_1;
set dtable;
run;
```

The above sample code could be saved in a macro (ex, %dtable(ds, t_name)). It saves the table_14_2_1 data set into the subfolder of datatable. If there are any changes made in CSR programs, you can compare the permanent table_14_2_1 data set with the new dtable data set. This comparison will ensure that all the changes made in CSR program will reflect the intended changes in the report.

The example CSR program that uses above two macros, which are highlighted, is included in appendix B.

QC BY YOUR OWN EYES

Another useful way of QC is using your own eyes. This method is not useful if you have to QC the output in detail, but if you just want to look at the general format of the output, this method is very useful. You all have an idea of how output should look. You expect certain formats or numbers. If it is a report, you print out the report and take a look at it. And if it is a SAS data set, you can open it with SAS viewer and check if anything is extraordinary.

CONCLUSION

As explained above, the QC of your own programs is not a fun thing to do, but it is necessary. If you set up the process or system (mainly macro) beforehand, it will be much easier and more efficient to perform the QC. Remember that SAS programmer always tries the best to provide error-free deliverables.

CONTACT INFORMATION

Your comments and questions are valued and welcomed. Please contact the author at
ACKNOWLEDGEMENT

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.