

Paper TT24

Generating *N* Shell Programs for *N* Reports in a Clinical Trial

John R. Gerlach; Independent Consultant, Blue Bell, PA

Abstract

The Statistical Analysis Plan (SAP) in a clinical trial identifies all the reports for a protocol, often consisting of well over one hundred Tables, Graphs, and Listings (TGL's). Assuming that the analysis programmer writes one SAS program per report, there will be as many programs that contain, hopefully, several standard features, such as a program header. Rather than employ the usual cut-n-paste process of writing SAS programs, it behooves the SAS analyst to be more efficient, as well as consistent. This paper explains a method for generating all the SAS 'shell' programs for a clinical trial.

Introduction

Writing SAS programs for a large clinical trial poses a real challenge for the programmer / analyst, especially when one considers the numerous TGL's that are often required. Clearly, the usual method of cut-n-paste, along with arbitrary (and inconsistent) file naming conventions greatly compounds a demanding task that includes an ambitious timeline. Also, the related task of monitoring program development becomes almost impossible due to the varied styles and conventions used by a team of SAS analysts.

For example, imagine a set of SAS programs that are intended to generate eighty tables, ten graphs, and fifty listings. Now, imagine a programmer using his -own file-naming convention, such that program AE_01A produces Table 14.1.3, AE01A_G produces Graph 14.1.3.1, and program VIT03A_B produces Listings 16.5.1 and 16.5.2. Why did the programmer call his programs AE_01A, AE01A_G, and VIT03A_B, respectively? Usually, the name has a mnemonic significance, which is not always obvious. The point is that it's a naming convention used to identify programs, which is likely to be privy to that programmer, on par with a programming style.

The ensuing problems are obvious. Imagine that one colleague consults another in order to exploit a SAS solution for a related table, perhaps pertaining to adverse experiences. What's the name of the program? Where is it stored? Or, perhaps you are required to take over a colleague's program, which begs the question: "What's been done, thus far, with respect to a part of the project, such as demographics or eligibility criteria?"

Even worse, imagine that several programmers are assigned the task of developing the numerous reports, each employing very different naming conventions and programming styles. How will these programs look collectively? To say the least, these programs will be quite different, and difficult to maintain.

Of course, a Standard Operating Procedure (SOP) seems appropriate – a popular bureaucratic solution. But, then, who wants to read a boring SOP that does little more than burden the SAS professional with corporate nuances, amidst an ambitious mandate. There must be a better way. Certainly, the use of technology, rather than edicts and policy, will help the programmer / analyst. That is, the development of a SAS tool that will facilitate the mundane task of cut-n-paste, as well as ensure a consistent programming style, even amongst a team of programmers with diverse backgrounds. Finally, and vitally important, this same tool should not take away the fundamental joy of the SAS professional, that is, the creative process imbued with problem solving. The intent is to make life better, not restrictive and boring.

A Better Approach

Let's agree that there will be one SAS program for each report and that the program (file) name will identify the name of the report. Also, let's agree to have a similar 'look' for each program, which will be explained later. Given these two simple criteria, along with the analysis plan that identifies all the reports, we can implement a tool that generates all the SAS programs, that is, the so-called program shells. Consequently, the more time-consuming task of 'getting the program started' has been done;

and, even more importantly, the programs subscribe to a consistent naming convention in accordance with the SAP and have a standard layout.

The Analysis Plan

Typically, the biostatistician writes the analysis plan for a protocol in a clinical trial. Besides addressing important issues such as sample size and randomization, the plan enumerates all the TGL's, identifying each report with a number and specifying titles.

Let's assume that this information is stored in an Excel spreadsheet, including: the number of each report, the order number (to accommodate the collating sequence problem), and the text denoting the titles and footnotes, for which the latter information is not needed.

The Template 'Shell' Program

Assuming that the Derived Data Library exists and has been validated, we can focus on producing one of many reports. Now, consider a typical SAS program needed to produce a report, specifically a table or listing. *What does the program look like?* Certainly, each program is unique, that is, it produces a specific report stipulated by the Analysis Plan. However, there are several steps that seem to be standard for these report-generating programs, as follows:

- Indicate a Program Header
- Specify SAS options
- Define formats (optional)
- Create the analytical data set
- Perform the analysis (e.g., counts, P-values)
- Prepare the data for reporting
- Produce the report using PROC REPORT

Keep in mind that the template SAS program is a model (template) for creating all the report-generating programs for a given protocol, as stipulated by the analysis plan. This template 'shell' program is shown in Appendix A, and its components are briefly explained, below.

Program Header – This part of the program alone will afford much to the general *look* of all the SAS programs. Already, the utility begins to modify the template in this section of the program, specific to a report. For example, the token (See Appendix A) "Rep_ProgShell.sas" that identifies the program, is replaced with the actual name of the file. Also, as with most information in the header of a program, the analyst is not burdened with the tedious task of typing repetitive information, such as the name of the clinical trial and copyright information. Also, standard macros that are used in virtually every program can be part of the program header. Even the dates indicate a pre-determined format (*ddmmyyy*), which the programmer replaces, accordingly. This attention to detail, which is accomplished upfront by the utility, creates an informative and attractive program header.

SAS Options, etc. – In order to ensure consistency across reports, several tasks are done prior to producing every report. Typically, a *%include* statement is used to execute SAS code pertinent to a protocol, specifying SAS options, defining SAS libraries, generating macro variables that denote sub-populations, and other sundry tasks.

Immediately following the *%include* statement, there's a *%let* statement that creates a macro variable denoting the particular report. The utility supplants the initial text "T/G/L/_00_0_0", as shown in Appendix A, with the actual report identifier (e.g., T_14_3_1). Notice that the original table identifier (14.3.1) has been altered using the underscore character.

Define formats – User-defined SAS formats are often created in a single program; hence, this component of the template program may be optional. On the other hand, besides the collection of formats that apply to the whole study, it might be feasible to define a format that is unique to that report, and even contribute to self-documentation of the program.

Create the analytical data set – Given the Derived Data Library, it is necessary still to create an analytical data set specific for that report, which represents the pertinent population (e.g., Intent to Treat patients) and contains the several variables required to produce the report.

Perform the analysis – Once you have the appropriate analytical data set, it is often necessary to perform some kind of analysis or imputation. Common tasks for this component of the template program includes: computing counts and percentages, generating descriptive statistics, and determining P-values. Perhaps, instead, the report may require a rather complicated imputation (which should have been done when creating the Derived Data Library). Although this portion is vital for generating tables, it might be not be relevant for listings and graphs.

Prepare the data for reporting – Depending on the layout of the report, which often defies the textbook meaning of a report, it may be necessary to build a data set in order to accommodate the Report procedure. For example, there are reports that include counts and percentages along with other descriptive statistics (e.g., minimum / maximum, mean) *in the same column*, that is, representing a treatment arm, as well as the TOTAL column. Moreover, these statistics require their own formatting, such as having percentages in parentheses juxtaposed with counts. Typically, this portion of the template program plays an important role for tables, but not for listings and graphs.

Produce the report – The REPORT procedure is used widely, nowadays, rather than the Data _null_, however, either method could be stipulated in the template. Also, in this section of the program template, there is a very important macro %hdrftr that defines the header and footer of a report, using other pertinent information, outside the scope of this paper. Because the Report procedure is not used to generate graphs, the utility ought to be able to discern when to ignore the REPORT procedure. For this paper, the utility assumes that there are no graphs.

Thus, given several simple conventions, as well as a standard template program, it is possible to generate numerous SAS ‘shell’ programs that will have a consistent look and feel, rather than the myriad styles often found in a team effort.

The Utility

The SAS utility requires two data sources: a document (e.g., Excel spreadsheet) that indicates the reports, as stipulated by the analysis plan and the SAS ‘shell’ program, or template. The information identifying the reports has been obtained and stored in a SAS data set called REPORTS_ALL. For our immediate purpose, it contains two important pieces of information, the report identifier, for example: Table 14.1.2 and Listing 16.2.4, as stipulated by the SAP, and the order. Notice that these identifiers have been slightly altered (Table 14.1.2 becomes T_14_1_2) in order to comply with syntax rules for file identifiers on a PC platform.

The following SQL step performs three tasks. First, it creates a SAS data set, aptly called REPORTS, which contains an ordered collection of report identifiers. The next statement creates a macro variable, called &nprogs., which denotes the number of programs that the utility will generate. The third task is to create n macro variables (&prog1. - &&prog&n.), that represent the complete set of report identifiers. Thus, for example, the first macro variable (&prog1.) denotes the first report, Listing 16.2.1 (L_16_2_1), while the i th macro variable denotes the last report, Table 5.1.3 (T_5_1_3).

```
proc sql noprint;
  create table reports as
  select order, prog
  from meta.reports_all
  order by order;

  select left(put(count(distinct prog),3.))
  into :nprogs
  from reports;

  select distinct prog into
  :prog1 - :prog&nprogs.
  from reports
  order by order;
quit;
```

It is necessary to modify the template shell program so that it contains code pertinent to a particular report. By reading the template program so that each observation represents a line of code, stored in a character variable called RECORD, the utility can modify specific records (lines of code in the template) as it generates the shell program. Thus, the following Data step reads the template program only once, an appropriate SAS program for the study.

```
filename temp "Rep_ProgShell.sas";

data template;
  infile temp length=len;
  input record $varying120. len;
run;
```

Once the report (program) identifiers are known and the template SAS program is stored in a SAS data set, the following macro `%progs` readily generates all the shell programs.

```
%macro progs;

  %do i = 1 %to &nprogs.;

    /* Check existence of program ;

    /* Generate program, if appropriate ;

    /* Modify template, accordingly ;

    /* Generate the shell program ;

    end;

%mend progs;
```

Several tasks are accomplished for each iteration of the `%do` loop. As a safeguard, prior to generating a shell program, it is a good idea to check whether the program exists already, which is easily accomplished by using the FILEEXIST function in a Data `_null_` step. Assuming that the program does not exist already, the data set TEMPLATE is modified accordingly, that is, specific to that report; thus, the program is generated straight away.

Notice that the data set TEMPLATE is not altered, permanently. Instead, the observation is modified as required, then written to a text file, line by line of code. Consider the two data steps that perform these tasks.

```
filename prog&i. "&rep.&&prog&i...sas";

data _null_;
  retain fname "&rep.&&prog&i...SAS";
  call symput
    ('rc',put(fileexist(fname),2.));
run;
```

```

data _null_;
  file prog&i.;
  set template;
  if substr(record,4,7) eq 'Program'
    then do;
      record = tranwrd(record,
        'Rep_ProgShell.sas  ','');
      record = tranwrd(record,
        'L/T/F_00_0_0.SAS',
        "&&prog&i...SAS");
    end;
  if substr(record,4,4) eq '%let'
    then do;
      record = tranwrd(record,
        'L/T/F_00_0_0',
        "&&prog&i..");
    end;
  put record $char.;
run;

```

Caveats

There is one major drawback that stems from any project in the real world – in a word, *modifications*. What if the statistical analysis plan is revised? What if more tables are added to the plan? Perhaps the most dreadful change would be to change the numbering (Table 14.1.3 becomes 14.1.4) of all the reports, which would make the names of the existing SAS programs obsolete, as well as some of the code therein.

On the other hand, in all fairness, the use of this utility should be done *after* the SAP has been signed-off. But, then, regardless of whether this utility is used in lieu of the traditional method of copying existing files or segments of code, major modifications to the SAP poses a nasty clerical chore, to say the least.

Possible Enhancement

This utility could be enhanced so that the program header indicates the actual title of the report. Thus, there might be additional lines in the header, as shown below, where the utility would supplant the ‘Line’ items with the full title of the report. Certainly, this added feature would greatly enhance the internal documentation of the SAS programs, as shown below.

```

/* =====
Program      : Rep_ProgShell.sas

Author       : John R. Gerlach
Date        : ddmonyyyy
Validated   : ddmonyyyy by

Study       : A Clinical Trial ...
Report    : Title Line1
             Title Line2
             Title Line3

Purpose     : Produce report.

Input      : COHORT
Output     : Report

Macros     : %hdrftr - Generate ...

Notes      : N/A
===== */

```

Conclusion

A clinical trial offers many problems for the programmer / analyst, which includes the production of many reports. Usually, SAS programs can vary greatly with respect to naming conventions and programming style. A utility that generates shell programs for producing all the reports in a clinical trial alleviates much of the clerical work of getting the program started. Moreover, this approach does not take away from the creative aspect of the programmer's duties. In fact, it diminishes the more mundane task of replicating SAS code in order to get to the real task of problem solving and producing the actual deliverable, accurately and on time.

Author Information

John R. Gerlach
Independent Consultant
Blue Bell, PA
215.284.2176

SAS® is a registered trademark of SAS Institute.

Appendix A: SAS 'Shell' Program.

```
/* =====
Program      : Rep_ProgShell.sas

Author       : John R. Gerlach
Date        : ddmonyyy
Validated    : ddmonyyy by

Study        : A Clinical Trial / Protocol ABC001.

Purpose      : Produce report.

Input        : COHORT

Output       : Report

Macros       : %hdrftr - Generate the header / footer of a report.

Notes        : N/A

-----

Copyright © 2004, Your Company
All Rights Reserved.
===== */

* %include "<Path>\Root.sas";

%let report = T/G/L_00_0_0;

/* -----
Create format.
----- */

/* -----
Create Analysis Data Set
----- */

* data adset;
*   merge derived.cohort ...;
*     by pid;
*   if rand eq 'Y' and ...;
*   keep ...;
* run;

/* -----
Create reporting data set.
----- */

/* -----
Produce report.
----- */

proc printto print="&rout.&report..lst" new;
run;
```

```
proc report data=rep nowindows headline headskip spacing=2
    missing ls=145 ps=&psize. Split='|';

*   columns ('---' page_ group_ . . .);

*   define page_    / order noprint;
*   define group_  / order;
*   break after group_ / skip;
*   break after page_ / page;
*   %hdrftr(&report., path=&rout.);
run;
```