

REPORTS BASED ON SAS OUTPUT: TAKING ADVANTAGE OF PROC PRINTTO, DATA STEPS AND PROC GPRINT

Lauren Haworth, Kaiser Permanente Center for Health Research, Portland, OR

ABSTRACT

The SAS® System is a powerful tool for data manipulation and analysis, but it often produces far more output than you want to read or report. Instead of scanning through a lengthy log printout looking for error and warning messages, or hand copying a few dozen figures into a table from a hundred-page output printout, let SAS do the work. This paper introduces a variety of time-saving techniques for producing concise summary reports.

The key is using the PRINTTO procedure to direct procedure output to log and print files. These files are then available to the DATA step for further refinement or analysis of the results. For example, by using a data step to review the SAS log, you can create self-monitoring production jobs that print error reports when warranted. By using a DATA step to review a lengthy output file, key results can be selected to report in summary tables or to use in advanced statistical computations. Finally, using the GPRINT procedure in SAS/GRAPH® you can add high-resolution graphics, titles and footnotes to the results.

INTRODUCTION

This paper quickly reviews the PRINTTO and GPRINT procedures, and then demonstrates how they can be used to produce concise customized reports that save time for the programmer as well as the end-user. The examples will show just a few of the many ways the SAS System can refine and summarize its results. The goal of this paper is to introduce the idea that SAS procedure output is not an end product, but an input to use in the process of creating custom reports.

QUICK REVIEW: PROC PRINTTO

The PRINTTO procedure allows you to specify destinations for SAS log and output files, instead of using the system defaults. The syntax is as follows:

```
proc printto log=fileref1 print=fileref2 new;  
run;  
  * SAS procedures go here ;  
proc printto;  
run;
```

In this example, the log file is rerouted to fileref1 and printed output is rerouted to fileref2. The second PROC PRINTTO closes the two output files and redirects output back to the default destinations.

The two files are now available to the DATA step, and can be read in as in the sample code below:

```
data temp;  
  infile fileref2;  
  input linetxt $90.;  
  * manipulate results here;  
run;
```

You can read it in line by line, or scan through the file for particular pieces of the output. In the example above, the file is read in a line at a time. The variable linetxt is formatted to match the current linesize setting. This paper will provide a number of examples of INPUT statements used to read in SAS output.

QUICK REVIEW: PROC GPRINT

The GPRINT procedure creates a graphics output file from a text input file. The syntax is as follows:

```
proc gprint fileref=fileref1;  
run;
```

In this example, the text file fileref1 is transformed into graphics output, and sent to the graphics device specified by the current GDEVICE setting. The output device may be the display monitor, or a printer or plotter.

PROC GPRINT is a handy way to print your text output file after it has been modified by a data step. This procedure can also be used to add high-resolution graphics to a text output file.

The following example – a macro used to create a self-monitoring production job – shows how PROC PRINTTO, the DATA step and PROC GPRINT can be used together.

EXAMPLE 1: ERROR REPORTS

When you are running a report that has been produced over and over many times without error, it is easy to get complacent and miss a new error caused by changes in the underlying data. The following sample program takes care of that problem. A DATA step is used to automatically check the SAS log file for a variety of errors each time you run the program. Then a macro is used to print either an error report, or, if no errors were found, to print the SAS procedure output.

The macro assumes PROC PRINTTO has been used to route the main program output to two files: fileref1 is the log and fileref2 is the output. A third file, fileref3, is used to hold the text for the error report (if needed). To run the macro, insert the code below at the end of the program:

```
%let founderr=0; * CHECK LOG FILE FOR ERRORS;
data _null_;
  infile fileref1 truncover;
  input linetxt $90.;
  if index(linetxt,"ERROR")>0 or
     index(linetxt,"uninitialized")>0 or
     index(linetxt,"repeats of BY values")>0
  then call symput("founderr", 1);
run;
%macro errcheck;
%if &founderr=1 %then %do;      %*PRINT ERROR;
  data _null_;                %*REPORT;
    file fileref3;
    put "PROGRAM: DEMO_A.SAS";
  run;
  title h=2 justify=left "Error: Check LOG file";
  proc gprint fileref=fileref3;
  run;
%end;
%else %do;      %* PRINT OUTPUT FILE;
  proc gprint fileref=fileref2;
  run;
%end;
%mend errcheck;
%errcheck;
```

The DATA step above reads in the log file, searching for several text strings. These include SAS error messages, as well as two other types of problems that SAS does not report as errors. This approach gives you the ability to search for a variety of problems in the log, whether or not SAS categorizes them as errors.

If any of the text strings are found, the macro variable founderr is set to 1. After scanning the entire log file, the macro checks founderr to see if any errors were discovered.

If there are errors, then the macro uses PROC GPRINT to print a report with an error message, the name of the program and a prompt to the user to check the log file. PROC GPRINT allows a large-font banner title to be used to draw attention to the error message. If there are no errors, then the macro prints the output file from the main program.

In the following example (see Figure 1), the error-checking code identifies two subtle errors in the program. These errors did not trigger a SAS error message, or even a warning message, but were identified and reported by the error-checking code (see Figure 2).

Figure 1: Sample program log file

```
158      data both;
159          merge demo.testdata demo.testdat2;
160          by id;
161          X=Y/2;
162      run;

NOTE: Variable Y is uninitialized.
NOTE: MERGE statement has more than one data set with
repeats of BY values.
NOTE: Missing values were generated as a result of
performing an operation on missing values. Each place is
given by: (Number of times) at (Line):(Column).
      1211 at 161:20
NOTE: The data set WORK.BOTH has 1211 observations and 9
variables.
NOTE: The DATA statement used 2.2 seconds.

163      proc means data=both;
164          var age x;
165      run;
```

Figure 2: Sample program output

Error in this Program: Check LOG

PROGRAM: DEMO_A.SAS

EXAMPLE 2: PRINTING A SUMMARY REPORT WITH SELECTED RESULTS

The previous example showed how to use PROC PRINTTO to identify errors in your programs. This next example shows how to use the procedure to produce a summary table.

The code below produces a lengthy output file, reads it back in to select a few key statistics, and prints a concise summary report. The original output file is not printed, but remains available if the user later needs to review the detailed results.

In the example, PROC GLM was used to compute adjusted standard deviations for a number of population subgroups. The adjusted standard deviations are reported as the "Root MSE" in the resulting output file (See Figure 3).

Figure 3: PROC GLM output

COHORT=1 TREATMENT=1					
General Linear Models Procedure					
Dependent Variable: WEIGHT					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	39	60765.31910275	1558.08510535	1.39	0.1982
Error	24	26898.80415514	1120.79350646		
Corrected Total	63	87664.12326389			
R-Square		C.V.	Root MSE	WEIGHT Mean	
0.693161		30.26697	33.47810488	110.60937500	

However, each PROC GLM creates three pages of output, and in the above example the original output file ended up being 54 pages long. This code reduces the output file to a single-page summary table:

```
proc printto log=fileref1 print=fileref2 new;
run;
title1 "3-way ANOVA: Gender, Age and BP";
title2 " ";
proc glm data=demo.testdata;
  by cohort tx;
  class gender age bp;
  model weight=gender age bp;
run;
proc printto;
run;
data temp;
  infile fileref2;
  input @"COHORT=" cohort $1.
        @"TREATMENT=" tx $1.
        @"Root MSE" // junk1 junk2 adj_sd;
  keep cohort tx adj_sd;
run;
proc sort data=temp nodupkey;
  by cohort tx;
run;
proc print data=temp label;
  by cohort;
  id tx;
  var adj_sd;
  format adj_sd 5.1;
  title j=c "PNWSUG Demonstration Report";
  title2 j=c "Std Dev of Weight, Adjusted for Gender,
Age and BP";
run;
```

To capture the information needed to produce the summary table, the output file is read back in using a DATA step. Instead of reading it in line by line, as in the previous example, the @ and / pointer controls were used to scan through the file for the required information.

PROC PRINT is used to output the results, labeled with the appropriate BY group values (see Figure 4).

Figure 4: Sample program output

PNWSUG Demonstration Report		
Std Dev of Weight, Adjusted for Gender, Age and BP		
COHORT=1		
TX	ADJ_SD	
1	33.5	
2	38.4	
3	38.9	
4	32.9	
5	35.6	
6	39.2	
7	38.8	
8	40.2	
9	36.2	
COHORT=2		
TX	ADJ_SD	
1	35.0	
2	41.3	
3	39.0	
4	31.6	
5	42.2	
6	36.2	
7	35.2	
8	30.9	
9	43.3	

EXAMPLE 3: CUSTOM STATISTICS

The above program is easily adapted to seek out and report other statistics. It also comes in handy when you need to calculate statistics not available from SAS. For example, intra-class correlations are not readily available from any SAS/STAT® procedure, but it is a simple matter to scan through PROC GLM output to pick up the required inputs and calculate the statistic yourself using the following code:

```
proc printto log=fileref1 print=fileref2 new;
run;
proc glm data=demo.testdata;
  cohort tx;
  model age=bp;
run;
proc glm data=demo.testdata;
  by cohort tx;
  model weight=bp;
run;
proc printto;
run;
data temp;
  format byvar1 byvar2 $12.;
  infile fileref2;
  input @"- " byvar1 $ byvar2 $
        @"Dependent Variable:" depvar $
        @"Model" dfm junk1 msb
        @"Error" junk2 junk3 msw
        @"Corrected Total" dft;
  drop junk1-junk3;
  k=(dft+1)/(dfm+1);
  ic=(msb-msw)/(msb+((k-1)*msw));
  label ic="INTRA-CLASS CORRELATION";
  format ic 8.3;
run;
proc print data=temp;
  by depvar byvar1;
  id depvar byvar1;
  var byvar2 ic;
  title j=c "PNWSUG Demonstration Report";
  title2 j=c "Intra-class Correlation";
run;
```

The DATA step is used to seek out the dependent variable, the two by-group variables, and the statistics needed to calculate intra-class correlations. After calculating the correlation, the results are reported in the following summary table (see Figure 5).

Figure 5: Sample program output

PNWSUG Demonstration Report Intra-class Correlation			
DEPVAR	BYVAR1	BYVAR2	IC
AGE	COHORT=1	TREATMENT=1	-0.031
		TREATMENT=2	0.156
		TREATMENT=3	-0.022
		TREATMENT=4	-0.006
AGE	COHORT=2	TREATMENT=1	-0.029
		TREATMENT=2	0.017
		TREATMENT=3	0.008
		TREATMENT=4	0.071
WEIGHT	COHORT=1	TREATMENT=1	-0.000
		TREATMENT=2	-0.022
		TREATMENT=3	0.079
		TREATMENT=4	0.021
WEIGHT	COHORT=2	TREATMENT=1	-0.027
		TREATMENT=2	-0.029
		TREATMENT=3	-0.022
		TREATMENT=4	-0.028

EXAMPLE 4: CUSTOM PAGE NUMBERS

PROC PRINTTO can also be used to make changes to the format of your output. For example, you can create custom page numbers.

When you number pages using the format "Page # of ##", the end-user of a report can be sure he/she received all the pages. However, SAS does not produce page numbers in this format. The following code uses PROC PRINTTO and a DATA step to count pages and then builds an output file with custom-formatted page numbers:

```
proc printto log=fileref1 print=fileref2 new;
run;
title1 "PNWSUG Demonstration Report";
title2 "Long Output File with Customized Page
Numbering";
title4 "#####"; * PLACE HOLDER FOR PAGE #;
proc univariate data=demo.testdata;
  by cohort tx gender;
  var weight;
run;
proc printto;
run;

%let ls=90;
data _null_; * COUNT PAGES IN DOCUMENT ;
  retain pgcnt 0;
  infile fileref2 lrecl=ls end=eof missover;
  input line $;
  if index(line,"#####")>0 then pgcnt+1;
  if eof then call symput("totpage",left(put(pgcnt,3)));
run;
```

```
data _null_; * ADD NEW PAGE NUMBERS ;
  retain pgcnt 0;
  infile fileref2 lrecl=ls end=eof missover pad;
  input line $charls.;
  file fileref3 lrecl=ls;
  if index(line,"#####")>0 then do;
    pgcnt+1;
    datepage="page "||trim(left(put(pgcnt,3)))||
      " (of "||left(trim("&totpage"))||")";
    startcol=floor((ls/2)-(length(datepage)/2));
    substr(line,startcol)=datepage;
  end;
  put line $charls.;
run;
title; footnote;
proc gprint fileref=fileref3;
run;
```

The only change made in the main body of the program was to add a series of pound signs to the title, turn off the standard page numbers, and use PROC PRINTTO to route the output to a file. The pound signs are used as placeholders for the new page numbers.

Following the main body of the program, a new section is added to read in the output file to count the pages, and read in the output file again to add the new page numbers. PROC GPRINT is used to print the resulting file (see Figure 6).

Figure 6: Sample program output

PNWSUG Demonstration Report			
Long Output File with Customized Page Numbering			
page 1 (of 36)			
----- COHORT=1 TREATMENT=1 GENDER=1 -----			
Univariate Procedure			
Variable=WEIGHT		WEIGHT	
Moments			
N	43	Sum Wgts	43
Mean	106.4109	Sum	4575.667
Std Dev	37.9522	Variance	1440.37
Skewness	0.416217	Kurtosis	-1.18809
USS	547396.1	CSS	60495.52
CV	35.66572	Std Mean	5.787656
T:Mean=0	18.38583	Pr> T	0.0001
Num ^= 0	43	Num > 0	43
M(Sign)	21.5	Pr>= M	0.0001
Sgn Rank	473	Pr>= S	0.0001

EXAMPLE 5: EXPORTING PROC TABULATE TABLES TO A SPREADSHEET

Sometimes you want to do even more reformatting of your SAS output. One way to produce elegant tables is to take the SAS output and move it into a spreadsheet program, where a wide array of formatting and graphics tools become available.

Although ODBC and PROC ACCESS are handy ways to pass data to a spreadsheet application, they are used to transfer data, not procedure output. And if you have tried to import SAS output files into a spreadsheet, you know how hard it can be to get all the columns to line up.

The following example uses PROC PRINTTO to pass the results of PROC TABULATE to a comma-delimited file that is easily imported into most spreadsheet programs:

```
options nodate nonumber ls=254 ps=32767;
proc printto print=fileref1;
run;
proc tabulate data=demo.testdata formchar=","
noseps;
  class tx gender;
  var weight;
  table mean*weight, tx all, gender all;
run;
proc printto;
run;
```

The long PAGESIZE setting is used to prevent carriage controls being sent for new pages. The setting of 32,767 in the example is the maximum allowed. You can speed up the importing process by reducing this value to the length needed to fit your output on a single page. The FORMCHAR setting is a comma followed by 10 spaces, which puts commas between each of your columns of data. NOSEPS is used to prevent the addition of row separators. The resulting spreadsheet file is shown in Figure 7.

Figure 7: Spreadsheet produced after importing data from sample program

	A	B	C	D	E	F
1	MEAN OF WEIGHT					
2						
3			GENDER			
4						
5			1	2	ALL	
6						
7		TREATMENT				
8		1	106.3	115.23	110.14	
9		2	111.21	109.29	110.31	
10		3	107.61	113.42	109.98	
11		4	110.25	114.83	112.23	
12		5	117.38	109.95	114.96	
13		6	109.13	116.82	112.69	
14		7	105.09	115.95	110.36	
15		8	107.95	108.17	108.03	
16		9	109.37	110.89	109.96	
17		ALL	109.64	112.84	110.95	
18						

EXAMPLE 6: TAKING ADVANTAGE OF GRAPHICS IN A REPORT

The previous examples have used PROC GPRINT to output simple text files, but the procedure is far more powerful. When PROC PRINTTO is used to pass your results to PROC GPRINT, instead of being limited to basic output styles produced by SAS procedures, a whole new world of alternative type faces, sizes and colors is opened up.

By using GOPTIONS settings, and modifying titles and footnotes, you can change the font to a different typeface, point size or color. (One warning: stick to the uniform fonts, or columns and tables will not line up correctly the resulting output. Uniform fonts generally have names ending in "U".)

The following example illustrates how to dress up a simple SAS table. The table is taken from example 2 above (see Figure 4 for comparison), and assumes that the results are stored in output file fileref3.

```
goptions fby=swissu hby=1.5 ftext=swissu htext=1;
title j=c f=brush h=4 "PNWSUG Demonstration
Report";
title2 j=c f=swissb h=1.5
"Standard Deviation of Weight";
title3 j=c f=swissi h=1
"(Adjusted for Gender, Age and BP)";
title4 " ";
footnote j=r f=swiss h=.5 "DEMO_F.SAS 6/28/96";
proc gprint fileref=fileref3;
run;
```

This example, shown in Figure 8 below, uses variations in font (BRUSH, SWISSB, SWISSI, SWISSU) and variations in font height (which ranges from .5 for the footnote to 4 for the top title) to create a more appealing. A large title draws attention to the report; a small font for footnotes allows the addition of unobtrusive labels for the program name and run date.

Figure 8: Sample program output

<i>PNWSUG Demonstration Report</i>	
Standard Deviation of Weight	
<i>(Adjusted for Gender, Age and BP)</i>	
COHORT=1	
TX	ADJ_SD
1	33.5
2	38.4
3	38.9
4	32.9
COHORT=2	
TX	ADJ_SD
1	35.0
2	41.3
3	39.0
4	31.8

DEMO_F.SAS 6/28/96

Other possibilities for dressing up the output include changing the font colors or adding ANNOTATE graphics.

LIMITATIONS

The examples above are powerful tools to use in reporting your SAS results, but keep in mind important limitations. First, it is very easy to make a mistake that creates a summary table containing incorrect data. It is important to check all summary tables against the original SAS output. Second, this type of reporting is somewhat vulnerable to changes in SAS versions. A new SAS release might change the format of a particular procedure's output, causing the summary program to fail. Finally, it takes time to set up a summary report. For a simple, one-time run, it may be easier to create a report by hand.

CONCLUSION

This paper reviewed a number of techniques and tricks for producing customized summary reports from SAS output. But the main point is to introduce a new way of looking at SAS procedural output: as an **input**, not an **end product**. For SAS procedures with long and complex output, or for standard reports that will be run repeatedly, it is well worth the time to set up automated error checking, summary reporting and custom output formatting. This paper has presented a few ideas to try, but there are many other potential uses for this approach.

REFERENCES

- SAS Institute Inc. (1990), *SAS Procedures Guide*, Cary, N.C.: SAS Institute Inc.
- SAS Institute Inc. (1990), *SAS/Graph Software*, Cary, N.C.: SAS Institute Inc.
- SAS Institute Inc. (1990), *SAS/STAT Software*, Cary, N.C.: SAS Institute Inc.

ACKNOWLEDGMENTS

The page-numbering code was written by Larry Parker of Parker & Brooks Associates.

The code for exporting PROC TABULATE output to a spreadsheet was borrowed with permission from an internet posting located at the following URL: <http://is.rice.edu/~radam/procs/listsas/formchar.html>.

SAS, SAS/GRAPH and SAS/STAT are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

CONTACTING THE AUTHOR

Please direct any questions or feedback to the author at:

Kaiser Permanente Center for Health Research
3800 N. Kaiser Center Drive
Portland, OR, 97227-1098

or by e-mail to: haworthla@chr.mts.kpnw.org