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 briefly 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 necessarily an **end product**, but can also be 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:

```
filename logfile 'c:\demo.log';
filename lisfile 'c:\demo.lis';
proc printto log=logfile print=lisfile new;
run;
   * SAS procedures go here ;
proc printto;
run;
```

In this example, the log file is rerouted to the fileref called logfile and printed output is rerouted to the fileref called lisfile. Be aware that filename statements are highly system-dependent; the above example works for SAS operating under Windows™ but may not work for your operating system.

The NEW parameter causes PROC PRINTTO to start a new file, otherwise the procedure appends the new information to the bottom of the existing file.

PROC PRINTTO acts much like a switch. First, you turn it on, and then you have to turn it off again. In the above example, the first PROC PRINTTO reroutes output to the two filerefs, and the second PROC PRINTTO closes the two output files and redirects output back to the default destinations. The two output files are then available for DATA step processing.

QUICK REVIEW: INFILE AND INPUT STATEMENTS

The sample code below shows how to read in the output file created by PROC PRINTTO:

```
data temp;
   infile lisfile truncover;
   input linetxt $90.;
   * manipulate your results here, then output;
run;
```

In the example above, the file is read in a line at a time. Each line of output is stored in a single variable, linetxt, so it can be manipulated later in the DATA step. The variable linetxt is formatted to a length of 90 to match the current linesize setting. The TRUNCOVER[1] option on the INFILE statement is used to truncate and retain any lines that are less than 90 characters wide. 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:

```
goptions \*set graphics options here*\;
proc gprint fileref=lisfile;
run;
```

1 For more information on TRUNCOVER and other options related to INFILE and INPUT statements, see "Using the DATA Step to Read Varying Length Record Files on ASCII Devices" (1995), *SAS Communications,* 21:1, 29-32.

In this example, the text file called lisfile 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. GPRINT settings are highly system-dependent, and will not be covered in this paper.

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 the SAS procedure output.

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

```
%let founderr=0;    * CHECK LOG FILE FOR ERRORS;
data _null_;
   infile logfile truncover;
   input linetxt $90.;
   if index(linetxt,"ERROR")>0 or
        index(linetxt,"uninitialized")>0 or
        index(linetxt,"repeats of BY values")>0
        then do;
             call symput("founderr", 1);
             file errfile;
             put "PROGRAM: DEMO_A.SAS";
             put linetxt;
        end;
run;
%macro errcheck;
%if &founderr=1 %then %do;          %*PRINT ERROR;
   title h=2 justify=left "Error: Check LOG File";
   proc gprint fileref=errfile;
   run;
%end;
%else %do;                    %* PRINT OUTPUT FILE;
   proc gprint fileref=lisfile;
   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. The name of the program and the line containing the error are then output to an error file.

After scanning the entire log file, the macro checks founderr to see if any errors were discovered. If there are no errors, then the macro prints the output file from the main program. 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

In the following example, the error-checking code identifies two subtle errors in the program.

```
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;
```

These errors did not trigger a SAS error message, or even a warning message, but were identified and reported by the error-checking code.

Error: Check LOG File

PROGRAM: DEMO_A.SAS

NOTE: Variable Y is uninitialized.

PROGRAM: DEMO_A.SAS

repeats of BY values.

EXAMPLE 2: SUMMARY REPORT WITH SELECTED RESULTS - PROC GLM

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.

```
--------------------- COHORT=1 TREATMENT=1 --------------------
                General Linear Models Procedure
Dependent Variable: WEIGHT

Source          DF  Sum of Squares    Mean Square F Value  Pr > F

Model           39  60765.31910875  1558.08510535    1.39  0.1982

Error           24  26898.80415514  1120.78350646

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 was more than 50 pages. This code reduces the output file to a single-page summary table:

```
proc printto log=logfile print=lisfile 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 lisfile;
   input @"COHORT=" cohort $1.
         @"TREATMENT=" tx $1.
         @"Root MSE" / / junk1 junk2 adj_sd;
   keep cohort tx adj_sd;
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.

The @"character-string" control locates the specified series of characters in the input file and moves the pointer to the first column after the character string. The / control moves the pointer to the beginning of the next line of data.

In this example, the input file is scanned for the text strings "COHORT=" and "TREATMENT=" to read in the values of the BY variables. Then the file is scanned for the text string "Root MSE" to find the adjusted standard deviation. Because the statistic is below the label, two / controls are used to move down two lines. Then, because the pointer has moved to the beginning of the line, the program has to skip past the two preceding statistics. These are assigned to variables junk1 and junk2 and later discarded. Finally, the adjusted standard deviation is read in as variable adj_sd.

You can use PROC PRINT to output your summary table.

```
proc print data=temp;
   by cohort;
   id tx;
   var adj_sd;
run;
```

```
                  SUGI 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: Summary Report With Selected Results - PROC TTEST

Another way to produce a summary report is to read in selected statistics and, based on their values, pick which results to print in a summary report.

The TTEST procedure is one of many procedures that produces more than one set of results. Each t-test yields two values of T, one that is appropriate if the variances of the two groups are equal, and one that is appropriate if the variances of the two groups are not equal. In order to pick the correct result, you have to examine the F' test at the bottom of the output.

```
Variable: Q24

GENDER    N   Mean  Std Dev  Std Error  Minimum  Maximum
----------------------------------------------------------
FEMALE   200  6.67    4.50      0.31      1.24     35.24
MALE     236  7.88    8.35      0.54      1.02    112.02

Variances        T       DF     Prob>|T|
----------------------------------------
Unequal     -1.9190    372.1     0.0557
Equal       -1.8336    434.0     0.0674

For H0: Variances are equal,F'=3.44 DF=(235,199)  Prob>F'=0.00
```

If the F' value is less than .05, then you reject the null hypothesis that variances are equal, and use the results from the row labeled "Unequal." Otherwise, you use the results from the row labeled "Equal."

If you have just a few variables, you can easily scan through your output to find the results. But if you are doing hundreds of t-tests, this can get very time-consuming.

The code below shows how to get SAS to do this work for you. In this example, the goal is to find which of the 87 questionnaire items yield different results based on gender. All of the necessary information is in the t-test output, it's just a matter of extracting the information and creating a user-friendly summary report.

```
options ls=64;
proc printto print=lisfile new; run;
   proc ttest data=demo.testdat2;
        class gender;
        var q1-q87;
   run;
proc printto; run;

data temp;
   infile lisfile;
   input @"Variable: " varname $8.
        @"FEMALE " junk1 meanF stdF
        @"MALE " junk2 meanM stdM
        @"Unequal " junk3 junk4 pval_un
        @"Equal " junk5 junk6 pval_eq
        @"Prob>F' = " fval;
   if fval<=.05 then pval=pval_un; else pval=pval_eq;
   drop junk1-junk6;
   label      meanF='Females*Mean'
              stdF='Std'
              meanM='Males*Mean'
              stdM='Std';
run;

proc print data=temp noobs label split='*';
   where pval<=.05;
   id varname;
   var meanF stdF meanM stdM pval;
   title 'SIGNIFICANT (<=.05) T-TEST RESULTS';
run;
proc print data=temp noobs label split='*';
   id varname;
   var meanF stdF meanM stdM pval;
   title 'ALL T-TEST RESULTS';
run;
```

The DATA step is used to read in the name of the analysis variable, and pick up the means and standard deviations for females and males.

Next, the program reads in the p values for the two t-tests. Only one of these results is appropriate for the data, so the program reads in the p value from the F' test to determine the correct result. The key to the whole program is the IF-THEN statement that selects the correct t-test result

Finally, the results are reported in two separate tables. The first PROC PRINT selects only those results where the t-test showed significant differences. This table provides a quick summary of the results.

```
SIGNIFICANT (<=.05) T-TEST RESULTS

          Females                  Males
VARNAME     Mean        Std         Mean        Std      PVAL

  Q19      292.36     169.74      342.26     215.69    0.0072
  Q28     4396.97    3179.82     5090.65    4140.79    0.0488
  Q30     4581.58    4381.18     5903.12    5929.32    0.0079
  Q49      154.15     109.71      178.77     148.96    0.0481
  Q50     4590.81    4382.32     5911.37    5931.04    0.0079
  Q55       34.14       6.89       37.30       7.39    0.0000
  Q56       13.68       2.70       15.03       3.23    0.0001
  Q57       43.40       7.13       46.13       8.71    0.0004
  Q60       34.68       6.96       37.88       7.57    0.0000
...
```

The second PROC PRINT displays all of the results, whether or not significant differences were found, in case you want to review the rest of the results.

```
ALL T-TEST RESULTS

          Females                  Males
VARNAME     Mean        Std         Mean        Std      PVAL

  Q1      2125.29     925.44     2232.00     900.88    0.2242
  Q2        76.82      34.34       82.68      34.36    0.0766
  Q3        88.76      45.84       94.22      46.11    0.2171
  Q4       247.45     111.41      253.70     103.78    0.5454
  Q5       813.69     452.97      856.54     430.11    0.3123
  Q6      1291.81     586.00     1358.79     566.18    0.2264
  Q7        14.93       7.43       15.75       7.03    0.2337
  Q8      3818.01    1870.43     4191.98    2090.50    0.0515
  Q9      3144.51    1399.04     3258.63    1308.53    0.3799
  Q10     9550.33    5786.27    10587.55    6624.34    0.0817
...
```

EXAMPLE 4: CUSTOM STATISTICS

The above program is easily adapted to seek out, evaluate 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. The formula for intra-class correlations is:

$$ic = \frac{(MS_m - MS_t)}{MS_m + ((k-1) * MS_t)}$$

$$where\ k = \frac{df_t + 1}{df_m - 1}$$

All of the values used in the formula are found in the output from PROC GLM. You can extract the values and compute the intra-class correlation by using the following code:

```
proc printto log=logfile print=lisfile new;
run;
   proc glm data=demo.testdata;
         by 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 lisfile;
   input @"- " byvar1 $ byvar2 $
         @"Dependent Variable:" depvar $
         @"Model" dfm junk1 msm
         @"Error" junk2 junk3 mst
         @"Corrected Total" dft;
   drop junk1-junk3;
   k=(dft+1)/(dfm+1);
   ic=(msm-mst)/(msm+((k-1)*mst));
   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 "SUGI 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.

```
--------------- COHORT=1 TREATMENT=1 ----------------

General Linear Models Procedure

Dependent Variable: AGE   AGE

Source           DF  Sum of Squares   Mean Square F Value  Pr > F

Model             1     14.19968782   14.19968782    0.04  0.8375

Error            62  20744.40968718  334.58725302

Corrected Total  63  20758.60937500
```

First the DATA step scans for a dash followed by a space to find the BY group header. Then it reads in the two BY variables. Next, it looks for the text label for the dependent variable and reads in the name of the variable. Then it scans for the degrees of freedom and mean square for the model and the total. These values are used to calculate the intra-class correlation, using the formula above. After calculating the correlation, the results are reported in the following summary table.

```
                   SUGI 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 5: 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:

```
options nonumber;
proc printto log=logfile print=lisfile new;
run;
   title1 "SUGI Demonstration Report";
   title2 "Long Output File with Customized Page
   Numbering";
   title4 "Page ### (of ###)";
   proc univariate data=demo.testdata;
        by cohort tx gender;
        var weight;
   run;
proc printto;
run;
```

The only change made in the main body of the program was to add the "Page ### (of ###)" title, turn off the standard page numbers, and use PROC PRINTTO to route the output to a file. The pound signs in the page number title are used as placeholders for the new page numbers. At this point, the output file looks like this:

```
                    SUGI Demonstration Report
          Long Output File with Customized Page Numbering

                        page ### (of ###)

 ----------- 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
```

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.

```
%let ls=90;
data _null_;  * COUNT PAGES IN DOCUMENT ;
   retain pgcnt 0;
   infile lisfile lrecl=&ls end=eof missover pad;
   input line $&ls..;
   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 lisfile lrecl=&ls end=eof missover pad;
   input line $&ls..;
   file lisfile2 lrecl=&ls;
   if index(line,"###")>0 then do;
        pgcnt+1;
        substr(line,floor(&ls/2)-3,3)=put(pgcnt,z3.);
        substr(line,floor(&ls/2)+5,3)=put(&totpage,z3.);
   end;
   put line $&ls..;
run;
title; footnote;
proc gprint fileref=lisfile2;
run;
```

The macro variable LS is used to pass the value of the LINESIZE setting, since it is used in numerous parts of the program.

This example uses a different type of INFILE statement than the previous examples. Instead of using the TRUNCOVER option to truncate shorter lines of data, this program uses the LRECL, MISSOVER and PAD options to read in each line of data into text strings of equal length. Lines that are shorter than the LRECL setting are padded with blanks at the end to make them longer.

These options are needed to preserve all of the blank spaces in the file. The blanks are needed to keep the lines of output correctly centered and the columns of data in the output correctly aligned.

```
                    SUGI Demonstration Report
          Long Output File with Customized Page Numbering

                        page 001 (of 036)

 ----------- 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
```

PROC GPRINT is used to print the resulting output file. This example assumes the program will produce no more than 999 pages of output. For longer output files, adjust the placeholder title and the two substring statements.

Example 6: 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 becomes available.

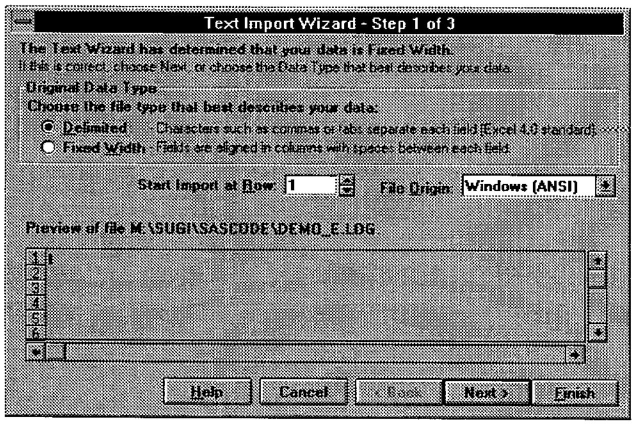
Although ODBC and PROC ACCESS are handy ways to pass data to a spreadsheet application, they are used to transfer data, not procedure output. And if you have ever tried to import SAS output files into a spreadsheet, you know how hard it can be to get all the columns to line up.

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=500;
proc printto print=lisfile;
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 large LINESIZE setting is used to keep wide tables from wrapping onto a second page. The maximum setting allowed is 254. The long PAGESIZE setting is used to prevent carriage controls being sent for new pages. You can set the PAGESIZE as high as 32,767 for a very large table. The FORMCHAR setting is a comma followed by 10 spaces, which puts commas between each of your columns of data and eliminates all other outlines and dividers. NOSEPS is used to prevent the addition of row separators.

As an example, to import such a file into Microsoft Excel®, you would take the following steps. First, select File and then Open from the main menu bar. Then select your SAS output file. Excel will detect that the file is a text file, and bring up the Text Import Wizard.



Text Import Wizard - Step 1 of 3

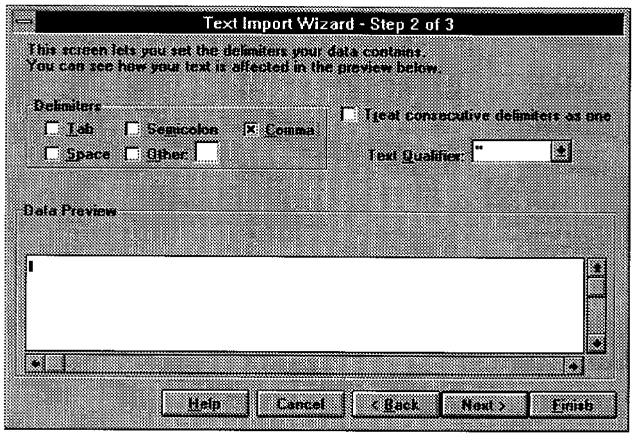
The Text Wizard has determined that your data is Fixed Width.
If this is correct, choose Next, or choose the Data Type that best describes your data.

Original Data Type
Choose the file type that best describes your data:
(●) Delimited - Characters such as commas or tabs separate each field [Excel 4.0 standard]
( ) Fixed Width - Fields are aligned in columns with spaces between each field

Start Import at Row: 1   File Origin: Windows (ANSI)

Preview of file M:\SUGI\SASCODE\DEMO_E.LOG

On the first screen, the wizard will ask whether the file is delimited or fixed-width. Select delimited. On the second screen, the wizard asks for the character used as a delimiter. Select comma. Make sure the "treat consecutive delimiters as one" box is not checked. Then select Finish.



Text Import Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains.
You can see how your text is affected in the preview below.

Delimiters
[ ] Tab [ ] Semicolon [✓ Comma]
[ ] Space [ ] Other:

[ ] Treat consecutive delimiters as one

Text Qualifier: "

Excel then imports the data, and the resulting spreadsheet file is shown below. If your columns of data or your column or row headings are particularly wide or narrow, you may need to adjust column widths slightly, but the rows and columns of data will always be perfectly aligned.

	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	114.63	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	110.59	109.96	
17		ALL	109.54	112.84	110.96	
18						

The only limitation to this approach is that complex tables with stacked column headings may need to have the headings adjusted. All of the data columns will line up, but the heading for a group of columns will appear over the left-most column. All you have to do is center the heading across the group of columns. Stacked row headings do not cause this problem.

While this example used Excel, other spreadsheet programs have similar options for importing comma-delimited text files.

EXAMPLE 7: TAKING ADVANTAGE OF GRAPHICS IN A REPORT

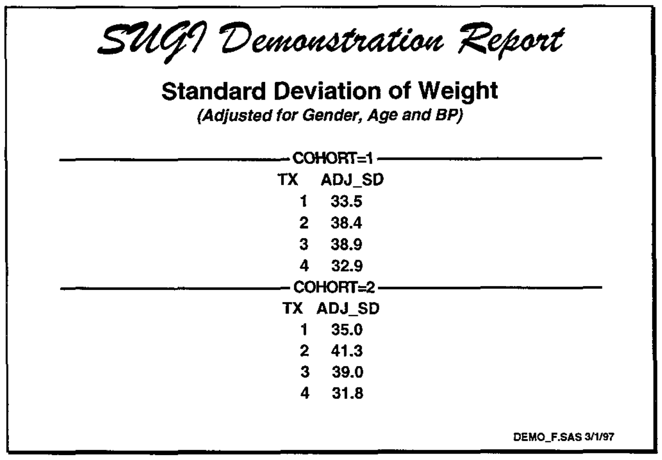
The previous examples have used PROC GPRINT to output simple text files, but the procedure is far more powerful. When PROC PRINTTO is used to pass your results to PROC GPRINT, instead of being limited to basic output styles produced by SAS procedures, a whole new world of alternative type faces, sizes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

By using GOPTIONS settings to modify 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 in 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, and assumes that the results are stored in an output file called lisfile2.

```
goptions fby=swissu hby=1.5 cby=blue
   ftext=swissu htext=1 ctext=purple;
title j=c f=brush h=4 c=red
   "SUGI Demonstration Report";
title2 j=c f=swissb h=1.5 c=green
   "Standard Deviation of Weight";
title3 j=c f=swissi h=1 c=green
   "(Adjusted for Gender, Age and BP)";
title4 " ";
footnote j=r f=swiss h=.5 "DEMO_F.SAS 3/1/97";
proc gprint fileref=lisfile2;
run;
```

This example, shown 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) and color (which you can't see in this black & white format) to create a more appealing report. 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.



SUGI Demonstration Report

Standard Deviation 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
----------------------COHORT=2----------------------
                     TX   ADJ_SD
                      1   35.0
                      2   41.3
                      3   39.0
                      4   31.8

                                        DEMO_F.SAS 3/1/97
```

The following table outlines some common GOPTIONS parameters and their settings.

FBY=fontname	Sets the font used for BY group headings.[1]
HBY=n	Sets the height of the text for BY group headings. Try a setting of 1.5 to make the headings somewhat larger than the rest of the text.
CBY=color	Sets the color of the text for BY group headings.[2]
FTEXT=fontname	Sets the default font used for text (not titles or BY group headings).
HTEXT=n	Sets the default height of text (not titles or BY group headings). Try .9 or 1. If this setting is too large, the output will no longer fit on the page.
CTEXT=color	Sets the default color for text (not titles or BY group headings).
BORDER	Puts a border around your output.
DEVICE=	Selects an output device (monitor, laser printer, plotter, etc.).

To set the font, height, color and justification for titles, it is easier to do so in the individual TITLE statements. This allows you to select different options for each title, instead of setting overall defaults. The most common parameters are:[3]

FONT=fontname	Sets the font for each title. Some interesting fonts for titles include BRUSH, SCRIPT, SWISSXBU and ZAPFBU.
HEIGHT=n	Sets the text size for each title. For your first title, try height settings of 3 to 5. Additional titles should be smaller, try settings of 1.5 to 2.
COLOR=color	Sets the color of the title text.
JUSTIFY=L|C|R	Aligns the title to the left, center or right.

[1] For a list of available software fonts, see Chapter 6 of *SAS/GRAPH Software: Reference, Version 6, First Edition.*

[2] For a list of predefined colors, see Chapter 7 of *SAS/GRAPH Software: Reference.*

[3] For additional TITLE statement options, including boxed and angled titles, see Chapter 17 of *SAS/GRAPH Software: Reference.*

LIMITATIONS

The examples above are powerful tools to use in reporting your SAS results, but keep in mind some 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. Third, if you change the LINESIZE setting, the same output format problem may occur. Many SAS procedures modify the output format and content based on line size. 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

Hatcher, Larry & Stepanski, Edward J. (1994), *A Step-by-Step Approach to Using the SAS System for Univariate and Multivariate Statistics*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Procedures Guide*, Cary, N.C.: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS/GRAPH Software: Reference, Version 6, First Edition*, Cary, N.C.: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS/STAT Software, Version 6, Fourth Edition*, Cary, N.C.: SAS Institute Inc.

"Using the DATA Step to Read Varying Length Record Files on ASCII Devices", (1995), *SAS Communications,* 21:1, 29-32.

ACKNOWLEDGMENTS

The page-numbering program is based on a program 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.

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

E-mail: haworthla@chr.mts.kpnw.org