# *SAS Components To Help You Survive A Deserted Island*

Stephen M. Noga
Rho, Inc.

**Abstract**
You are alone on a deserted island (your job) and can only use four components of the SAS® system to get your job done. What four components would they be, and why? Because of how expansive the SAS system is, sometimes programmers spend more time looking for a procedure to do a particular task than utilizing what they know to do the same thing. This paper will demonstrate that with a good working knowledge of only four components (all contained in Base SAS) out of the entire SAS system, roughly eighty to ninety percent of all tasks can be accomplished.

## Introduction

What are these four magic components that can make your job easier? The first component is the **DATA STEP**. In short, some of the many things you can do with this component are create or delete data, manipulate data and display data. Assume for the moment that you just created data with component #1. It is far easier using **PROC SORT** (a.k.a. component #2) to order the data than it is writing a bubble sort (think PL/1 or Pascal) in the data step.

You're halfway home in making your job easier. The next task is to compute a wide range of statistics on the variables created in the data step. **PROC UNIVARIATE** provides you with an impressive amount of descriptive statistics, and thus earns the title of component #3. Finally, you want to display the fruits of your labor. While nothing gives you the precision of a data _null**_, PROC REPORT** gives you almost as much display power with only a fraction of the effort.

The intent of this paper is not to teach the reader specifics of the SAS system. Instead, I would hope that the reader's eyes are opened to the fact that an enormous amount of work can be accomplished with just a few procedures.

## The Data Step

The data step is the foundation that this paper is built on. Everything that will be discussed in this paper can be done in the data step, not that you would necessarily want to do some of these things in the data step.

Let us begin by looking at some ways to create data within the data step.

- INPUT
- SET
- MERGE
- OUTPUT

INPUT allows the user a way to read data into a data step. This data can be in the form of an ASCII file, another data file format such as an Excel file, or CARDS data. SET and MERGE use at least one existing SAS data set to create or modify one or more data sets. OUTPUT is a statement that lets the user create observations within the data step.

Not only can you create data in the data step, but you can also delete data. When I talk about deleting data I am including sub-setting data sets because you are eliminating observations from the data set. There are different ways to delete observations from a data step. The WHERE option in a SET or MERGE statement will eliminate observations as will using the WHERE or IF statement in the data step. Of course, you could always use the DELETE statement to delete an observation. A note of caution, it is safer to subset a data set first and then create a new data set from it than it is to subset a data set and continue to perform operations within the same data step.

Let’s look at doing some operations in the data step instead of using a procedure to do the same thing. The first example gets the totals for two variables in a data set.

```
* Create Test Data *;
DATA test;
 do x=1 to 5;
  do y=1 to 3;
   output;
  end;
 end;
RUN;

* Get Totals of X and Y (procedure) *;
PROC summary data=test ;
 var x y;
 output out=way1 sum=sum1 sum2;
RUN;

* Get Totals of X and Y (data step) *;
DATA way2;
 set test end=last;
sum1+x;
sum2+y;
if last;
RUN;
```

Both way1 and way2 contain one observation with values of 45 for X and 30 for Y. Let’s look at the code that creates way2. The data step option END= is used to indicate the last observation in a data set. SUM1+ and SUM2+ are implied retain statements. However, if these statement were to be written like the following:

```
      SUM1 = SUM1 + X;
      SUM2 = SUM2 + Y;
```

then you would have to use:

```
      RETAIN SUM1 SUM2;
```

to achieve the same results. Finally, the IF statement deletes all observations except the last one as indicated by the variable LAST which was set by END=.

The next example demonstrates how the data step can be used instead of PROC TRANSPOSE to transpose a data set.

```
* Create Test Data *;
DATA test;
 do x=1 to 5;
  do y=1 to 3;
   z = x * y;
   output;
  end;
 end;
RUN;

* Transpose Data Set (procedure) *;
PROC transpose data=test out=way1
(drop=_name_);
 by x;
 var z;
RUN;

* Transpose Data Set (data step) *;
DATA way2 (drop=y z);
 retain col1 col2 col3;
  set test;
  by x;
array cols{3} col1 col2 col3;
cols{y}=z;
if last.x;
RUN;
```

The printout of data sets way1 and way2 would look like the following:

```
OBS  COL1  COL2  COL3   X
 1    1     2     3     1
 2    2     4     6     2
 3    3     6     9     3
 4    4     8     12    4
 5    5     10    15    5
```

In the way2 data set, the use of RETAIN, ARRAY, and LAST. make it possible to replicate the results from the PROC TRANSPOSE. The ARRAY statement is another way of saying:

```
      if y=1 then col1 = z;
      else if y=2 then col2 = z;
      else if y=3 then col3 = z;
```

When a BY statement is used, SAS provides two automatic data set variables FIRST. and LAST. At the first occurrence of a by variable value (ie x=1, _n_=1), FIRST.X = 1 and LAST.X = 0. At the last occurrence of a by variable value (ie x=1, _n_=5), FIRST.X = 0 and LAST.X = 1. Any observations in between the first and last occurrence will have values of FIRST.X = 0 and LAST.X = 0. By using these automatic variables, I can output only one observation per unique value of X, with this observation being the last occurrence of a unique value.

Let's now add just a few functions to the way2 data step. By implementing these functions (MEAN, MIN, MAX, SUM, N, NMISS) and by placing these functions after the LAST.X, the resulting data set is the same as if a class statement was used with PROC MEANS or a by statement with PROC UNIVARIATE.

```
DATA way2 (drop=y z);
 retain col1 col2 col3;
  set test;
  by x;
array cols{3} col1 col2 col3;
cols{y}=z;
if last.x;
mean = MEAN(of col1-col3);
min = MIN(of col1-col3);
max = MAX(of col1-col3);
sum = SUM(of col1-col3);
nmiss = NMISS(of col1-col3);
n = N(of col1-col3);
RUN;
```

As the reader can see from these few examples, the DATA STEP can be used to replicate the results of a number of procedures. I barely demonstrated the flexibility of the data step but I think the reader can infer that a SAS programmer who is comfortable with the data step can accomplish much. The next component to help you through your job, PROC SORT, can be done with a data step, but in almost all cases, PROC SORT is much easier to use.

## PROC SORT

In some of the previous examples with the data step, the BY statement was used to read sorted data into a data step. PROC SORT was not necessary since the test data set was sorted when it was created. This usually is not the case in the real world. PROC SORT is the SAS component that accomplishes this task without much effort.

The SAS Procedures Guide contains a complete description of the SORT procedure. This paper will just review some of the basics. The bare-bones syntax for this procedure is:

```
      PROC SORT data=in_ds;
        BY var1 . . . var_n;
      RUN;
```

In this example, the input data set IN_DS is sorted by variables var1 through var_n and then written out as IN_DS. All data set options, such as WHERE or KEEP, are available in the SORT procedure. As was just noted, the input data set was over-written by the sorted data set. If the WHERE option was used, then only those observations that met the criteria would be kept in IN_DS. To avoid that, use the OUT= option like the following:

```
PROC SORT data=in_ds (where=(x > 5))
out=out_ds;
```

The default sort order for the by variables is ascending, but the user has the option for specifying a descending sort on each sort variable. In the following example, only VAR3 will be in descending order:

```
BY var1 var2 descending var3 var4;
```

Two other options that come in handy are NODUPKEY and NODUPLICATES. NODUPKEY allows the user to specify that only the first observation with the same by variables should be kept. NODUPLICATES compares all variables in an observation with the variables in the next sorted observation and deletes one of the observations if all variables are equal.

To recap, PROC SORT is an easy to use procedure that accomplishes much with little effort. In addition, sorting a data set allows the user to take advantage of the automatic data set variables FIRST. and LAST. when doing subgroup processing in the data step

## PROC UNIVARIATE

We have seen how to manipulate data within a data step and also by using the sort procedure. Data manipulation is an everyday need in the work environment. Another everyday need is to get descriptive statistics about a data set. PROC UNIVARIATE provides these statistics. As with PROC SORT, the SAS Procedures Manual is the place to get a complete description of the UNIVARIATE procedure. This paper will give a brief overview of the procedure and why it is preferred over similar procedures.

```
* Test Data *;
DATA test;
 do age=5 to 20 by 5;
  do y=1 to age*2;
   score1 = age * y;
   score2 = (age * y)**2;
   score3 = (age * y)/.5;
   output;
  end;
 end;
RUN;

* Basic Univariate *;
PROC univariate data=test ;
 by age;
 var score1-score3;
RUN;
```

On the following page is one page from the default UNIVARIATE print. Everything you see on the printout under the headings MOMENTS and QUANTILES can be written to an output data set:

```
* Basic Univariate (with output data
set) *;
PROC univariate data=test ;
 by age;
 var score1-score3;
 output out=stats
 n=n1-n3
 median=med1-med3
 std=std1-std3;
RUN;
```

In addition to these statistics, by using the PLOT option, you could produce a stem-and-leaf plot, a box plot, and a normal probability plot (see next page). One other option that deserves mention is the FREQ option. This option will produce a frequency table similar to one that would be produced by a PROC FREQ.

As this brief look at PROC UNIVARIATE has demonstrated, this procedure gives the user a lot of bang for the buck. In fact, sometimes a user may complain that there is too much information being displayed in the printout. However, if you could only select four components of the SAS system (remember the premise of this paper), select UNIVARIATE as the procedure for calculating descriptive statistics.

## Default UNIVARIATE Print

```
--------------------------------------- AGE=5 ---------------------------------------
                                Univariate Procedure
Variable=SCORE1
                 Moments                                  Quantiles(Def=5)

 N                10  Sum Wgts          10      100% Max         50        99%        50
 Mean           27.5  Sum              275       75% Q3          40        95%        50
 Std Dev    15.13825  Variance    229.1667       50% Med       27.5        90%      47.5
 Skewness          0  Kurtosis        -1.2       25% Q1          15        10%       7.5
 USS            9625  CSS           2062.5        0% Min          5         5%         5
 CV         55.04819  Std Mean    4.787136                                  1%         5
 T:Mean=0   5.744563  Pr>|T|        0.0003      Range            45
 Num ^= 0         10  Num > 0           10      Q3-Q1            25
 M(Sign)           5  Pr>=|M|       0.0020      Mode              5
 Sgn Rank       27.5  Pr>=|S|       0.0020

                                      Extremes
                         Lowest    Obs     Highest    Obs
                              5(       1)       30(       6)
                             10(       2)       35(       7)
                             15(       3)       40(       8)
                             20(       4)       45(       9)
                             25(       5)       50(      10)
```

## UNIVARIATE Print with plots

```
-------------------------------------- AGE=10 ---------------------------------------

                  Stem Leaf                     #             Boxplot
                     4 0                        1                |
                     3 6                        1                |
                     3 2                        1                |
                     2 69                       2                |
                     2 02                       2             +-----+
                     1 7                        1             |     |
                     1 024                      3             *--+--*
                     0 568                      3             |     |
                     0 001224                   6             +-----+
                       ----+----+----+----+
                   Multiply Stem.Leaf by 10**+4

                                  Normal Probability Plot
               42500+                                                  * +++
                    |                                             *  ++++
                    |                                          * ++++
                    |                                      * *+++
               22500+                                    +*++
                    |                                ++**
                    |                            ++*+**
                    |                        +++++**
                2500+      *    *  *+*+** *
                     +----+----+----+----+----+----+----+----+----+----+
                         -2        -1         0        +1        +2
```

UNIVARIATE Print with a Frequency Table

```
                 Percents                      Percents
 Value  Count  Cell   Cum      Value  Count  Cell   Cum
   100      1   5.0   5.0      12100      1   5.0  55.0
   400      1   5.0  10.0      14400      1   5.0  60.0
   900      1   5.0  15.0      16900      1   5.0  65.0
  1600      1   5.0  20.0      19600      1   5.0  70.0
  2500      1   5.0  25.0      22500      1   5.0  75.0
  3600      1   5.0  30.0      25600      1   5.0  80.0
  4900      1   5.0  35.0      28900      1   5.0  85.0
  6400      1   5.0  40.0      32400      1   5.0  90.0
  8100      1   5.0  45.0      36100      1   5.0  95.0
 10000      1   5.0  50.0      40000      1   5.0 100.0
```

## PROC REPORT

The fourth and final component is PROC REPORT. If I want total control over the data display, then the PUT statement within a DATA _NULL _ is the way to go. Since the data step is one of the four components that I selected, I can do this. However, displaying data with this technique takes some time, and therefore won't be discussed here. My three options to display data are PROC PRINT, PROC TABULATE and PROC REPORT. PROC PRINT is the easier of the three to master, but it is also the more limited, while PROC TABULATE is the more inflexible. It is for those reasons that PROC PRINT and PROC TABULATE are not selected.

PROC REPORT comes with its own manual. In fact, it comes with two manuals, one for interactive programming and one for batch programming. Suffice it to say that I will not attempt to explain the hows and whys of this procedure. What I will do is demonstrate some of the power and flexibility of PROC REPORT in hopes that the reader will be interested enough in learning more about it. The following example uses PROC REPORT to replicate the output that a PROC FREQ would produce.

```
** Create Test Data **;
DATA tst;
 retain string 'a b c d e f g' seed 3765092;
do i = 1 to 7;
 x = scan(string,i);
 do j = 1 to sum(3,(ranuni(seed) > .3),
(ranuni (seed) > .5));
   output;
 end;
end;
RUN;

** EASY way **;
PROC freq data=tst;
 tables x;
RUN;

** PROC REPORT way **;
PROC report data=tst nowd headline split='|';
 column x n pctn sumn cpct;
 define x    / group  width=1  'X';
 define n    / width=9  'Frequency';
 define pctn / width=7 format=5.1  'Percent';
 define sumn /  width=10  format=6.
'Cumulative|Frequency';
 define cpct /  width=10  format=5.1
'Cumulative|Percent';

compute before;
 hold=0;
 cumn=0;
endcomp;
compute pctn;
  pctn=pctn*100;
endcomp;
compute cpct;
  cpct=sum(pctn,hold);
endcomp;
compute sumn;
  sumn = sum(n,cumn);
endcomp;
compute after x;
  cumn = sum(cumn,n);
  hold = sum(hold,pctn);
endcomp;
RUN;
```

Both the PROC FREQ and the PROC REPORT produce the following output. Of course, PROC FREQ is the easy and sensible way to produce this table. The point of the example, however, was to demonstrate that PROC REPORT could do something that PROC PRINT could not even pretend to do.
PROC REPORT can also produce simple listings just like PROC PRINT, but unlike PROC PRINT, the data does not need to be sorted beforehand in order for the procedure to produce a sorted listing.

```
                           Cumulative  Cumulative
X   Frequency  Percent  Frequency   Percent
------------------------------------------
a           3     11.5          3      11.5
b           4     15.4          7      26.9
c           4     15.4         11      42.3
d           4     15.4         15      57.7
e           3     11.5         18      69.2
f           5     19.2         23      88.5
g           3     11.5         26     100.0
```

## CONCLUSION
I hope that you, the reader, are beginning to understand that with the SAS system, there is more than one way to do things. Granted, sometimes using a procedure makes more logical sense than doing the same task in the data step. However, the main theme of this paper is that a new SAS user would be better served by learning a limited number of SAS components, and learning them well, than by trying to learn, understand, or use the multitude of choices presented in the SAS system.

## REFERENCES
SAS Institute Inc *SAS Procedures Guide*, Version 6, Third Edition
SAS Institute Inc *SAS Guide to the REPORT Procedure, Usage and Reference,* Version 6, Third Edition
SAS Institute Inc *SAS Technical Report P-258, Using the REPORT Procedure in a Nonwindowing Environment*, Release 6.07

## AUTHOR CONTACT
Stephen M. Noga
Rho
121 S Estes Drive
Chapel Hill, NC 27514
snoga@rhoworld.com