# Your Program Works, But is it Efficient?

Sheree H. Andersen


## Abstract

Usually errors are thought to be logical and syntactical in nature because they prevent a program from successful completion. However, a well-trained user community also recognizes inefficient processing as a type of error even though inefficient programs can execute successfully and lead to accurate results. For most applications, it is possible to develop more than one method of programming a task. Some methods utilize more resources than others. Being aware of the relative efficiency of alternate methods is the first step toward quality SAS programming.


## Introduction

This paper is intended for beginning- to intermediate-level SAS users. It covers topics relating to efficiency and gives recommendations on how optimize resource utilization.

As an instructor of the SAS System and an Information Center Support Programmer I've seen many different styles of programming. These various styles include everything from how data sets are named to how final results are obtained. It didn't take long to recognize that "all roads lead [eventually] to Rome" but some take a long time getting there. I soon adopted the philosophy of considering efficiency as important as logical design and syntactical accuracy.

## Terminology

Logical design consists of how a program is structured to obtain the desired results. It includes consideration of all possible circumstances and provides for their appropriate management. Logical errors typically occur when assumptions do not reflect reality. A logical error might be something as simple as not considering the effect of missing data values in a process. A program that contains logical errors may execute successfully but the information processed could be spurious.

Syntactical accuracy is the correctness of the SAS code that is submitted for processing. Although the SAS compiler is very liberal, as far as position and placement of statements on a line are concerned it has little tolerance for inaccurate or misplaced statements. A program with syntactical errors will not execute. The SAS compiler does its best to highlight syntactical errors by issuing error messages to the log. Sometimes these messages seem cryptic and deciphering them is an exercise not unlike translating hieroglyphs.

Efficiency consideration is the cognizant effort to select processing methods that utilize the least amount of resources for a given task. The motivation for it is to reduce quality waste.

Quality waste is the unnecessary consumption of material or human resources. It is a timely concept since streamlining is a popular catchword for the nineties.

## Generalizations

You can maximize DATA step efficiency by being aware of these conditions:

. Formatted input using the pointer control and format specification is more efficient than either list or column input.

. Reading a particular field to test record selection criteria while holding the record with a trailing @ is more efficient than reading a complete record which does not satisfy processing criteria.

. Direct access, that is, SET with the POINT= option is more efficient when the sampling is less than 25% of the data set.

. The OUTPUT statement is the most efficient method for subsetting a data set (example follows).

. Sorting is a CPU intensive task.

. The efficiency of formats compared with IF and IF/ELSE constructs used for data translations depends upon the size of the format and the number of observations processed.

. PROC APPEND is more efficient than a SET which vertically concatenates two similar data sets.

. Data stored in SAS data sets is accessed more efficiently than data stored in flat files.

In general, it is more efficient to handle as much of the data preparation needed in a program, as possible, within a DATA or PROC step. It has been estimated that as much as 90% of CPU is consumed in I/O. With that in mind selecting a method that limits the number of transitions between DATA and PROC steps is best because each transition requires that data be read from or written to some computer device.

## Recommendations

. Omit unnecessary PROC or DATA steps.

. Restrict variables to those needed in the process by using DROP or KEEP statements or their respective data set options. Avoid carrying excess baggage from step to step.

. Keep programs modular, that is, self-contained instruction sets for related processing but integrate them within DATA or PROC steps.

. Try to order your data with as few sorts as possible. If a data set with many variables needs to be sorted by a relatively few variables consider using the TAGSORT option.

. Use benchmarking techniques to identify the most efficient methods.

Since benchmarking is so important it is considered separately in this paper.

## Benchmarking

If there is a question about which method is most efficient, it is worthwhile to do an application-specific benchmark run, particularly if the application will process large amounts of data or will be run on a frequent basis.

A benchmark run is performed by coding simple representations of the proposed process using various methods, then submitting them to the SAS System for execution. A large enough data sample should be used to discern true variability between the methods. The benchmark run is complete once a comparison of CPU utilization for the various methods is made. Documenting the version of SAS software and the platform used when a benchmark run is done is important, since dramatically different results may occur under different circumstances. Two benchmark examples follow.

### Relative Efficiency of Subsetting Methods:

A benchmark run demonstrates the relative efficiency of the various subsetting methods. The process that was evaluated selects only specific zip codes in a geographic region.

```
/* A format designating selected Zip codes is created */
        proc format;
          value $zip
          '84001' = 'Y'
          '84005' = 'Y'
          '84010' = 'Y'
          '84013' = 'Y'
          '84050' = 'Y'
          '84070' = 'Y'
          '84088' = 'Y'
          '84100' = 'Y';

/* Generate data set with incremental zip values */
        data zips;
          do j=1 to 100;
            do i=1 to 100;
              zip=put(84000 + i, $5.);
              output;
              end;
            end;

/* Traditional subsetting IF w/o format */
        data sub_if1;
          set zips;
          if (zip='84001' | zip='84005' | ...);

/* Subsetting IF statement with format. */
        data sub_if2;
          set zips;
          if (put(zip,$zip.)='Y');

/* Subset with the WHERE data set option. */
        data wheredso;
          set zips(where=(put(zip,$zip.)='Y'));

/* Subset with the DELETE statement. */
        data deletes;
          set zips;
          if (put(zip,$zip.)^='Y') then delete;

/* Subset with the OUTPUT statement. */
        data outputs;
          set zips;
          if (put(zip,$zip.)='Y') then output;
```

The results obtained are shown in Table 3.

**Table 3:** Seconds of CPU Time for Various Subsetting Methods

| Operating System..... | Windows | MVS | UNIX |
|---|---|---|---|
| Version of SAS..... | 6.08 | 6.08 | 6.09 |
| Subsetting IF w/o fmt | 5.85 | na | na |
| Subsetting IF w/format | 2.08 | .21 | .43 |
| WHERE data set option | 57.95 | .63 | .60 |
| DELETE statement | 2.08 | .21 | .39 |
| OUTPUT statement | 1.37 | .21 | .36 |

Table 3 shows that very different results occur according to different platforms and versions of SAS software. This underscores the importance of making the benchmark specific to the site and circumstances under which the application will be run.

In general, the WHERE data set option is less efficient than the other methods analyzed. These results should not dissuade you from using WHERE in either its statement or data set option forms. Used appropriately, it can also provide efficiency. If a procedure is to be executed for only a subset of data, it is more efficient to execute it in conjunction with WHERE than to first subset the data

in a DATA step.

### Combining Statistics and Percents

A common report writing request places statistics and percent-of-total information side by side. Method 1 relies on several SAS procedures and can produce the desired results. It uses the following procedures:

. PROC MEANS, to calculate the statistics
. PROC FREQ, to calculate the percents
. PROC TRANSPOSE, to change the output data set orientation of PROC FREQ so that it can be merged with the PROC MEANS output

However, the same results can be obtained with Method 2, which uses only PROC MEANS followed by a DATA step that is programmed to calculate the percents.

A benchmark run under Windows with SAS version 6.08 shows the relative efficiency of the two methods. It also demonstrates that Method 2 offers processing benefits when percents for additional attribute variables are needed.

```
/* Combining Statistics & Percents Efficiently */
        data report;
          set info;

/* Prepare raw data for processing Method 1 */
/* Set up categorical variable: LEVEL */
        if (opinion=9.5) then level='Com sat ';
         else if (opinion=7.5) then level='Very sat ';
         else if (opinion=5.5) then level='Satisfied';
         else if (opinion=3.5) then level='Dis satis';
         else if (opinion=1.5) then level='Very dis';

/* Prepare raw data for processing Method 2*/
/* Set up counters for opinion categories */
        if (opinion=9.5) then csat=1;
         else if (opinion=7.5) then vsat=1;
         else if (opinion=5.5) then sat=1;
         else if (opinion=3.5) then dsat=1;
         else if (opinion=1.5) then vdsat=1;

/* Method 1 */
        proc summary data=report nway;
          class seg;
          var age;
          output out=method1 mean(age)=;
        proc sort data=report nway;
          by seg;
        proc freq data=report noprint;
          by seg;
          tables level / out=list;
        proc transpose data=list out=tranlist;
          by seg;
        data method1;
          merge method1 tranlist;
          by seg;
        proc print data=method1(drop=_label _name_);
```

```
/* Method 2*/
        proc summary data=report nway;
          class seg;
          var age csat--vdsat;
          output out=method2 mean(age)=
            sum(csat--vdsat)=;
        data method2;
          set method2;
          array pct csat--vdsat;
          do over pct;
            pct = (100 * pct) / _freq_;
            end;
        proc print data=method2;
```

The results are shown in Table 2:

Table 2: Combining Statistics and Percents

| Method | CPU Utilization |
|---|---|
| 1 using PROCS | 180.2 seconds |
| 2 PROC MEANS and DATA step | 50.5 seconds |

Method 2 uses less than a third of the CPU required by Method 1. When greater complexity is added to the program and other factors are considered the savings are even more pronounced. For example, consider the number of lines of code and the number of steps required when percents for two additional attribute variables are needed. Table 3 shows the results of the benchmark run that calculates the mean age and percents for 3 attribute variables.

Table 3: CPU Comparison

| Method | CPU | # lines of code | # PROC or DATA steps |
|---|---|---|---|
| 1 | 3.49 min | 28 | 7 |
| 2 | 1.12 min | 11 | 2 |

### Conclusion

Quality SAS programming begins with an understanding about the relative efficiency of alternative processing methods. When several methods are available for programming a specific process it is worthwhile to set up a benchmark run to evaluate processing efficiency. Usually limiting the number of transitions between PROC and DATA steps is the best method since it reduces the CPU intensive I/O task.