

# SAS<sup>®</sup> System Programming Tips and Efficiencies: A Tutorial

Andrew H. Karp  
Sierra Information Services, Inc.

## Introduction

Most users of the SAS System want to know how to make their programs and applications be as efficient as possible. This tutorial sets out some general concepts for, and examples of how to implement, techniques which can improve the speed with which the SAS System will accomplish the desired tasks. It is not intended to be an exhaustive treatment of the topic; rather, users should be able, after mastering the concepts presented here, to identify other approaches to SAS System programming efficiency which are not presented here.

## General Concepts and Considerations

Achieving efficiencies in the SAS System can be approached from two perspectives: *user* efficiency and *machine* efficiency. Both are essential to developing SAS System programs and applications which:

- take the minimum amount of time to process
- consume fewer computing resources
- take up as little storage space as possible
- operate on variables and observations as efficiently as possible.

## *Machine Efficiency*

In this context, machine efficiency refers to developing SAS System programs and applications which use computing machine resources as efficiently as possible. These include issues addressing how the program or application requires:

- processing time
- input/output operations
- physical storage space for data sets.

## *User Efficiency*

This term refers to the SAS System user's ability to understand what the program at hand is doing and why it is doing it. It is not uncommon for a SAS System program to be written in a way that another user cannot comprehend the logic or approach taken

by the original author of the program. While the program at hand may in fact be very elegant and quite diligent in its approach to efficient use of *machine* resources, the time wasted by subsequent users to discern the logic employed by the program's original author or to attempt to modify it almost always exceeds the amount of machine efficiency (i.e., time) obtained from the original program.

User efficiency occurs when a user knows how the SAS System:

- processes observations in the Program Data Vector (PDV)
- controls the movement of observations in to the PDV
- controls the output of observations from the PDV to SAS data set(s)
- uses procedures (i.e., PROCs) on data sets.

## Fundamental Concepts

Two fundamental concepts govern the development of efficient SAS System programs and applications.

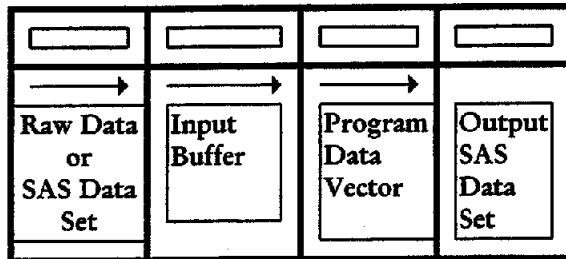
I) The Program Data Vector Operates on One Observation at a Time

II) The SAS System Executes Programming Statements in the Exact Order They Are Encountered in the Program

Figure One below presents a highly simplified depiction of how the SAS System creates a data set. In general, an observation from the "source file" (either raw data or an existing SAS data set) is moved in to the input buffer. From there it is read in to the Program Data Vector (PDV). SAS Programming Language statements in the Data Step are then performed on the observation while it is in the PDV. When all programming language statements are executed the observation is (usually) output to a SAS data set. This process repeats for each observation.

While many exceptions and details to this description of the SAS System's data set creation process are possible, it is sufficient to describe ways to improve SAS System programming efficiency.

**Figure One**  
How the SAS System Creates a SAS Data Set



Reference to Figure One makes it easy to identify points in the data set creation process where efficiencies can be implemented:

- reading raw data in to the input buffer
- reading observations from an existing SAS data set
- operations while the observation is in the Program Data Vector
- output of observations to new SAS data sets

Users who want to develop efficient SAS System programs and applications need to understand these core concepts underlying how the SAS System operates on observations and from these operations, creates data sets. It is therefore important that users understand how the programs they write: impact the movement of observations from source file (i.e., raw data or existing SAS data set) to the input buffer, what happens to the observation when it is in the PDV, and how the observation is output to the new SAS data set.

### **A General Rule**

A good rule to follow when developing SAS System programs and applications is to:

*maximize the number of operations performed on the minimum number of observations*

The remainder of this tutorial gives a number of suggestions on how to achieve both machine and user efficiency in the context of the concepts presented above.

### **Reading Raw Data Selectively**

Using IF/THEN-ELSE logic in an INPUT statement can reduce the number of records of raw data which are read in to the input buffer. Appropriate use of testing observations to see if they meet the desired conditions *before* they are read in to the input buffer reduces the number of observations upon which a data step will subsequently operate. As such, "weeding out" unnecessary or undesired observations in a raw data file reduces the need for SAS System resources in subsequent steps in the program.

Consider the following two code fragments:

#### *Example 1*

```
Data Myfile1;
    INFILE 'C:\ANDREW\Rawdata';
INPUT @1 PATID @20 PROC @30
DATE:YYMMDD8. ;
IF YEAR(DATE) = 1996;
RUN;
```

#### *Example 2*

```
DATA MYFILE2;
    INFILE 'C:\ANDREW\RAWDATA';
INPUT @30 DATE:YYMMDD8.;
IF YEAR(DATE) = 1996 THEN DO;
    INPUT <more programming language statements>;
RUN;
```

While both code fragments would yield the same results, Example One requires that all observations, and all variables, in the raw data file be moved in to the PDV. The "subsetting if" statement then restricts output from the PDV to the new data set of those observations where YEAR = 1996.

The second example processes the raw data much more efficiently. The condition of interest (i.e., YEAR = 1996) is tested for each observation in the raw data file. Only when the condition is met is the rest of the observation moved in to the PDV for further operations.

In this example, use of IF-THEN/ELSE statement when reading the raw data file acts as a barrier, or filter, which restricts movement of observations in to the PDV of only those which are required in subsequent steps in the program.

## Use Nested DO-LOOPS When Reading Raw Data

More than one variable can be tested when reading raw data. When an observation in a raw data file must meet two or more conditions in order for it to be read in to the PDV, the most efficient approach is to set up a series of nested DO-LOOPS which will test each condition. The nesting conditions should be ordered from *most frequent* to *less frequent* in order to obtain maximum efficiency and eliminate unnecessary use of machine resources. Here is an example:

### Example Three

```
INPUT @1 GENDER;  
  IF GENDER = 1 THEN DO;  
    INPUT @10 SALEAMT;  
    IF SALEAMT > 1000 THEN DO;  
      INPUT <more programming statements>;  
    END;  
  END;  
  RUN;
```

In this example, two conditions are tested before the entire observation is moved from the input buffer to the PDV. Only those records of raw data where GENDER = 1 are considered for further processing. And, only those observations where the first condition (i.e., GENDER = 1) has already been met are available for testing the second condition (SALEAMT > 1000). If **both** conditions are met then the rest of the observation is read in to the PDV. If not, then a new observation is moved in to the input buffer and the process begins again.

## Working with Existing SAS Data Sets

Users can improve the efficiency of SAS System programs which use an existing SAS data set as either the source for a new data set or upon which a SAS Procedure (i.e., PROC) will operate. Among the easiest and most common efficiency technique available for these types of situations is to use SAS Programming Language *data set options*. These options can be used to:

- restrict the number of variables in the PDV
- restrict the observations moved in to the PDV to those which meet user-specified criteria.

### Restricting the Number of Variables in the PDV

By default, the length of the PDV is equal to the amount of space required for all the variables in the

data set. When only a few variables are required from a much larger (i.e., many variable) data set, it is not efficient to move the entire observation in to the PDV only to output a few variables to a new data set.

The KEEP = data set option is used to identify those variables which the user wants to have moved in to the PDV. Appropriate use of the KEEP data set option results in a smaller PDV, and reduced time required for the SAS System to create a new SAS data set.

Notice, however, that the word KEEP is used in the SAS programming language as both a data set option and as a programming language statement. Consider the following two examples:

### Example Four

```
DATA NEW;  
  SET OLD;  
  KEEP VAR1 VAR3 VAR4;  
RUN;
```

### Example Five

```
DATA NEW;  
  SET OLD(KEEP=VAR1 VAR2 VAR4);  
RUN;
```

The second program fragment (Example Five) is the most efficient. In Example Four, *all* variables in data set OLD will be moved in to the PDV, but only those in the KEEP statement will be output to data set NEW. Example Five, however, 'filters out' the unwanted variables *before* the observation moves in to the PDV. Depending on the size of data set OLD, and the platform upon which the SAS System is running, Example Five would result in much faster processing than would Example Four. But, as will be seen below, an even more efficient approach is available than that presented above.

## Process Only the Observations You Need

The preceding example addressed issues where the user needs to restrict the number of *variables* which are placed in the PDV. Additional, and often substantial, efficiencies can be obtained when the user's program processes only the observations required to carry out the desired task.

Consider the following examples, where programs reading a library's circulation data base are to be written.

Suppose the programmer/analyst wanted to create a subset SAS data set from containing observations for books classified as NOVEL with a discharge date in 1995 or 1996. Example Six presents a perfectly acceptable approach to this problem.

#### *Example Six*

```
DATA NOVELS;  
  SET LIBRARY.CIRC;  
  IF TYPE = 'NOVEL';  
  IF YEAR(DISDATE) IN (1995, 1996);  
RUN;
```

This example of a 'subsetting IF' statements results in all variables in LIBRARY.CIRC being placed in to the PDV, and for all observations in the data to be evaluated in the PDV. Only those observations which meet the subsetting IF conditions (TYPE = 'NOVEL' and YEAR = 1995 or 1996) will be output to the new SAS data set. While the result obtained would be that which the programmer desired, it is clearly not the most efficient approach possible.

Remember that the SAS System operates on programming statements in the exact order they appear in the program. In Example Six, if an observation did not have a value of NOVEL for the variable TYPE, the rest of the data step (i.e., testing for year of discharge date) would not execute for that observation.

#### **Using the WHERE Clause**

A more efficient approach to this problem is to use the WHERE data set option to filter out, or restrict, movement in to the PDV of only those observations which meet the desired criteria. The result is that fewer observations will be operated upon in the PDV, thus reducing program execution time. Example Seven shows a more efficient approach to the problem first addressed in Example Six.

#### *Example Seven*

```
DATA NOVELS;
```

```
  SET LIBRARY.CIRC ( WHERE =  
    (TYPE='NOVEL' AND YEAR(DISDATE)  
    IN(1995,1996)) );  
RUN;
```

Using the WHERE clause as a data set option in Example Seven is a much more efficient approach than that presented in Example Six. The WHERE clause acts as a "traffic cop," deciding which observations (those which mean the conditions stated in the clause) enter the PDV for further operations (if any) and which do not. This is a good example of the "general rule" presented above. In Example Seven only the *minimum number of observations* required to accomplish the desired result will enter the PDV.

#### **Working in the Program Data Vector**

Up to this point we have addressed methods by which to efficiently limit the number of observations which enter in to the PDV for processing. This section of the tutorial describes ways to efficiently process observations once they are in the PDV.

#### **Testing Multiple Conditions**

Users frequently need to test the value of a *single variable* to determine if it meets one or more conditions of interest. A medical economist, for example, may want to test a variable representing medical procedure code to extract from a larger data base only those records containing information on procedures of interest.

Alternatively, SAS System users may need to test the values of *multiple variables* to ascertain if the *entire observation* should be used in subsequent parts of the program, analysis or application under development. The medical economist may, for example, want to only look at observations with information about a certain series of medical procedures performed only on female patients.

There are several methods to efficiently test multiple conditions for both single variables and for several variables while they are in the PDV.

#### **Using the IN operator vs. logical OR operators**

Using the IN operator is a more efficient approach to testing a single variable for multiple conditions of interest. When the SAS System evaluates an

expression containing multiple or operators, it evaluates the entire expression even if one comparison has already made the expression true. When the IN operator is used, however, the SAS System stops evaluating the expression as soon as a comparison makes the expression true. Also, it is easier to write and maintain code containing use of the IN operator than a series of OR operators.

Examples Eight and Nine demonstrate these concepts;

#### *Example Eight*

```
DATA NEW;  
  SET HOSPITAL.MAINFILE;  
  IF CODE = 'X102' OR CODE = 'X387'  
    OR CODE = 'C453';  
RUN;
```

#### *Example Nine*

```
DATA NEW;  
  SET HOSPITAL.MAINFILE;  
  IF CODE IN('X102','X387','C453');  
RUN;
```

In Example Eight, the SAS System will continue to process the IF statements even if the observation in the PDV has a value of the variable CODE equal to X102. In Example Nine, however, if the variable CODE has a value equal to 102, then the expression is not evaluated further. Use of the IN operator, rather than a series of logical OR operators, can therefore lead to significant reductions in processing time.

### **Using Mutually Exclusive Conditions in Order of Descending Probability**

#### **IF-THEN/ELSE or SELECT Groups vs. IF-THEN Statements**

SAS Data Steps oft require testing the values of one or more variables for condition(s) of interest, and taking actions conditionally (that is, depending upon) the value of the variables being tested. Significant reductions in processing time are possible if the user applies IF-THEN/ELSE logic and orders the testing conditions in order of descending probability.

If the user writes a series of IF-THEN statements the SAS System checks each condition for every

observation which is processed in the Program Data Vector. This can be very time consuming especially if a large data set is to be processed and there are multiple conditions to be tested.

If, however, a series of IF-THEN/ELSE or SELECT group statements are used, the SAS System stops checking the statements as soon as the condition(s) is/are true for that observation and then skips to the bottom of the series to execute additional statements (if any) in the data step.

Where feasible, users should write the IF-THEN/ELSE conditions in order of descending probability. In other words, the most frequently occurring condition should be placed first, the second most second, and so forth. Examples Ten and Eleven demonstrate these concepts. In these examples, the prices of books will be increased based on whether they are classified as NOVEL, COOKBOOK or TRAVEL. Notice use of the WHERE clause as a data set option to restrict entry in to the PDV of only those observations whose values of the variable TYPE meets the condition of interest.

For these examples we will assume that NOVEL is the most frequently occurring TYPE of book, followed by TRAVEL. COOKBOOK occurs least frequently.

#### *Example Ten*

```
DATA PRICES;  
  SET LIBRARY.CIRC(WHERE=(TYPE  
  IN('COOKBOOK','TRAVEL','NOVEL')));  
  IF TYPE = 'COOKBOOK' THEN  
    PRICE=PRICE * 1.05;  
  IF TYPE = 'NOVEL' THEN  
    PRICE=PRICE*1.23;  
  IF TYPE = 'TRAVEL' THEN  
    PRICE = PRICE * 1.17;  
RUN;
```

Even though the result of Example Ten will give the user their desired results, this is a most inefficient method by which to approach the problem. Use of a series of IF-THEN statements results in each observation being tested three times, one for each unique value of the variable TYPE. Also, COOKBOOK is (according to the assumption set out above) the least frequently occurring value of the variable TYPE, but is the first condition in the IF-THEN.

Now consider Example 11, which uses IF-THEN/ELSE statements.

#### Example 11

```
DATA PRICES;
  SET LIBRARY.CIRC(WHERE=(TYPE
IN('COOKBOOK','TRAVEL','NOVEL')));
IF TYPE = 'NOVEL' THEN
  PRICE=PRICE*1.23;
ELSE IF TYPE = 'TRAVEL' THEN
  PRICE = PRICE * 1.17;
ELSE IF TYPE = 'COOKBOOK' THEN
  PRICE=PRICE * 1.05;
RUN;
```

This data step will execute more efficiently than that presented in Example 10. Why? First, use of IF-THEN/ELSE statements reduce the amount of time that each observation spends in the Program Data Vector. If an observation with a value of the variable TYPE equal to NOVEL enters the PDV, the value of the variable PRICE is changed and the observation then moves to the new SAS data set. The rest of the conditions are not tested.

Also, since the testing conditions are ordered in descending probability of occurrence, those observations with a value of TYPE equal to NOVEL (the most frequently occurring value) will not 'wait' until all the conditions in the IF-THEN/ELSE statements are tested before moving to the new SAS data set.

Ordering testing conditions in descending order of probability is often well worth the small amount of time it may take for the SAS System program to find the frequency (e.g., by using PROC FREQ) with which the conditions occur *before* writing the program or application.

#### Evaluating More Than One Condition at a Time

Many SAS System programs applications require that the values of two or more variables meet certain conditions in order for some action to occur. Appropriate use of DO-groups and IF-THEN/ELSE statements often leads to efficient processing of observations in the Program Data Vector.

Examples 12 and 13 demonstrate efficient methods by which to evaluate two conditions (i.e., the values of two variables) when performing conditional

operations in the Program Data Vector. In these examples, the value of the variable PRICE will change depending on both the value of the variable TYPE and a second variable, PUBCODE (publisher code). We will again assume that NOVEL is the most frequently occurring TYPE of book, followed by TRAVEL and COOKBOOK.

#### Example 12

```
DATA PRICES;
  SET LIBRARY.CIRC(WHERE=(TYPE
IN('COOKBOOK','TRAVEL','NOVEL')));
IF TYPE = 'NOVEL' and PUBCODE = 1 THEN
  PRICE = PRICE * 1.22;
IF TYPE = 'NOVEL' and PUBCODE = 2 THEN
  PRICE = PRICE * 1.09;
IF TYPE = 'COOKBOOK' and PUBCODE = 22
THEN PRICE = PRICE * 1.09;
  < more programming statements >
RUN;
```

As with several of the previous examples, while this data step will execute without error and provide the desired results, it is not the most efficient way to approach the task at hand. At a minimum, replacing the series of IF-ELSE statements with IF-THEN/ELSE statements will speed data step processing. Additional efficiency is available, however, if a series of DO groups are created which test for one condition. If that condition is true, then the second condition will be tested and appropriate action taken on the variable. Example 13 demonstrates this concept.

#### Example 13

```
DATA PRICES;
  SET LIBRARY.CIRC(WHERE=(TYPE
IN('COOKBOOK','TRAVEL','NOVEL')));
IF TYPE = 'NOVEL' THEN DO;
  IF PUBCODE = 1 THEN PRICE = PRICE *
1.22
  ELSE IF PUBCODE = 2 THEN PRICE =
  PRICE * 1.09;
END;
ELSE IF TYPE = 'COOKBOOK' THEN DO;
  IF PUBCODE = 22 THEN PRICE =
  PRICE * 1.09;
  < more programming statements >
RUN;
```

This example shows how a combination of IF-THEN/ELSE and DO-group processing can lead to

potentially significant reductions in program execution time. By ordering conditions in descending probability of occurrence, as well as taking secondary actions based on the value of the first condition, the amount of time each observation will stay in the PDV is reduced to the minimum required to accomplish the desired tasks.

### SELECT Expressions

Although not presented here, SAS System users may want to investigate use of SELECT expressions as an alternative to IF-THEN/ELSE statements. The SELECT expression is documented in the SAS Language: Reference text. Some users prefer writing programs and applications using the SELECT expression approach as it is often easier to write and maintain than IF-THEN/ELSE statements.

### Creating Multiple SAS Data Sets

An often overlooked aspect of SAS System data step programming features is the ability to create multiple SAS data sets in a single data step. Using appropriate programming syntax to create more than one data set in a single data step demonstrates how it is possible to *maximize the number of operations performed on the minimum number of observations* required to accomplish the desired task. Consider Examples 14 and 15. In these examples, the desired outcome is to create two SAS data sets, one containing observations where the value of TYPE is equal to NOVEL and a second where the value of TYPE is equal to COOKBOOK.

#### *Example 14*

```
DATA NOVEL;  
SET LIBRARY.CIRC  
    (WHERE=(TYPE='NOVEL'));  
RUN;  
  
DATA COOKBOOK;  
SET LIBRARY.CIRC  
    (WHERE=(TYPE='COOKBOOK'));  
RUN;
```

In Example 14 two (temporary) SAS data sets are created from an existing (permanent) SAS data set. In order to this, however, all observations in the 'source' data set need to be read *twice*, which leads to an approximate doubling of the time required to accomplish the desired outcome. Example 15 below shows how to create both data sets in *one* data step.

#### *Example 15*

```
DATA NOVEL COOKBOOK;  
SET LIBRARY.CIRC(WHERE = (TYPE IN  
'NOVEL','COOKBOOK'));  
IF TYPE = 'NOVEL' THEN OUTPUT NOVEL;  
ELSE OUTPUT COOKBOOK;  
RUN;
```

In Example 15 only a single pass through the 'source' data set is required. Use of the WHERE clause restricts entry in the Program Data Vector of *only* those observations from which the new data sets will be created.

Since, however, two data set names are placed in the DATA statement, the OUTPUT statement is required to 'direct' observations to their 'proper' data set.

### The OUTPUT Statement: Pointers and Pitfalls

Appropriate use of the OUTPUT statement demonstrates the second 'Fundamental Concept' presented above: The SAS System Executes Programming Statements in the Exact Order They are Encountered in the Program. Since the OUTPUT statement is an example of an "executable statement" in the SAS programming language, it is executed as soon as it is encountered in the program. Users therefore need to be careful as to how they place it in their programs.

As a general rule, SAS System programming language statements operating on the observations need to be placed in the data step *before* an OUTPUT statement directs the observation(s) to one or more SAS data sets. Otherwise, the programming language statements will execute *after* the observation has already been placed in the new SAS data set. This is a particularly easy mistake to make since the SASLOG will not indicate that an error has occurred.

### Avoiding the Data Step Entirely

Several BASE SAS procedures are available which may obviate the need for Data Step operations. As a general rule, using a "housekeeping" PROC such as PROC DATASETS or PROC SORT is more efficient than using a Data Step.

### Using PROC DATASETS

This procedure manages SAS System data sets and also operates upon the descriptor portion of a SAS Data Set *without* accessing the data set's observations and variables. It is therefore a better alternative to a Data Step when performing housekeeping operations such as:

- copying SAS Data Sets
- renaming SAS Data Sets
- appending one Data Set to another
- renaming variables
- adding LABELS and/or FORMATS

The key benefit to using PROC DATASETS, rather than a Data Step, is that no Program Data Vector is created and, consequently, observations are not processed in the same manner as they would be in the Data Step.

### Using PROC SORT

This procedure will 're-arrange' the order of observations in a SAS Data Set. Use of the frequently overlooked NODUPKEY and NODUPPLICATES options available in this procedure may permit the user to bypass an otherwise required Data Step to remove duplicate observations from a SAS System Data Set.

### Reducing the Size of SAS System Data Sets by Using the COMPRESS Data Set Option

This option, new to Releases 6.06 and above of the SAS System, will reduce the size of a Data Set by compressing:

two or more blanks

two or more binary zeros

three or more occurrences of any other character.

This option is therefore best suited to data sets with long character variables. The amount of storage space saved will depend on the extent to which repeating values appear in the observations. As a word of caution, if the COMPRESS option is used on a data set containing only numeric variables, it is likely that the resulting data set will be *larger* than the original data set.

### Conclusions

This tutorial presents some of the ways SAS System users can improve the efficiency of their programs and applications.

The key to achieving efficiency in use of the SAS system is to understand how data sets are created, and how, during that process, to maximize the number of operations performed on the minimum number of observations required to accomplish the desired results.

When appropriate, using a SAS Procedure may be more efficient than using a Data Step.

SAS System programming efficiency increases with the skills and abilities of the SAS user. Achieving both user and machine efficiency at the same time is often difficult. When left with a choice between the two, user efficiency is always better than machine efficiency.

### Note:

SAS is a trademark of SAS Institute, Inc. in the United States and other countries. ® indicates USA registration.

The author can be contacted at:  
Sierra Information Services, Inc.  
1489 Webster Street #1308  
San Francisco, CA 94115  
415/441-0702  
SFBAY0001 @ AOL.COM

### References:

SAS Institute, Inc., *SAS® Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 1990.

SAS Institute, Inc., *SAS® Programming Tips: A Guide to Efficient SAS Processing*, Cary, NC: SAS Institute, Inc., 1990

SAS Institute, Inc., *Advanced SAS® Programming Efficiencies Course Notes*, Cary, NC: SAS Institute, Inc., 1992.

SAS Institute, Inc., *SAS® Technical Report P-222, Changes and Enhancements to Base SAS® Software, Release 6.07*, Cary, NC: SAS Institute, Inc., 1991