

How to Write SAS Code Which is Efficient and Easy to Maintain

Thomas J. Winn, Jr.

Texas State Comptroller's Office, Austin, TX

Introduction

Most beginning SAS programmers would be content just to have their programs execute successfully. Typically, novice programmers are satisfied if their programming code generates the intended results without generating any *fatal* messages from the SAS System. It was while I was at this "minimal sufficiency" stage in my professional development that I learned to hate the much-dreaded message:

NOTE: The SAS System stopped processing this step because of errors.

NOTE: SAS set option OBS=0 and will continue to check statements.

This may cause NOTE: No observations in data set.

I still dislike this message, even though I don't encounter it very often by myself these days.

However, our goal should be higher than mere avoidance of mistakes which cause our programs to fail. It is possible to assemble executable SAS code which is inelegant, inefficient, and difficult to understand. I know this for a fact. I do it myself sometimes. Moreover, even though we may think that we understand what our code is attempting to accomplish *at the time we write it*, chances are our own programs would be unintelligible to us even a few months afterward, in the absence of at least a few critical explanatory remarks. And if even we have difficulty readily understanding our own code, what chance would someone else have in figuring it out?

Eventually, novice SAS users are transformed into experienced SAS programmers. Sooner or later, just getting your program to run isn't enough anymore. At this later stage, we expect that our programming code should run with a minimum of computer resource utilization, and without requiring much human intervention. There exist principles whose implementation will result in less computer time, memory and data storage used, and in much less grief for the programmer. The purpose of this paper is to present a few of these principles.

Improving Human Efficiency of SAS Code

Let's begin from the perspective of improving the *human efficiency* of the SAS programs we write. This involves the total time spent by the programmer, both in developing and in maintaining the computer program. Improving human efficiency mostly is a matter of coding in the clearest possible manner, and using a consistent approach. Programs which are clearly and consistently written require less programmer time for debugging, updating, or modifying. Here are some recommendations:

- a) Use program headers to describe what the program was intended to accomplish, and also to highlight any features which may need to be modified from time to time.

Here is an example of a program header -- it would be placed very near to the beginning of the program:

```
*-----*
THIS PROGRAM PRODUCES A REPORT WHICH SUMMARIZES THE NUMBER OF
SALES TAX PAYERS AND THE NUMBER OF SALES TAX AUDITS, BY REPORTED
AMOUNT OF SALES TAX, FOR THE MOST RECENTLY-COMPLETED YEAR. THE
PROGRAM USES DATA FROM THE AUDIT FAIS FILE AND THE CLASS FILE. IT
WAS WRITTEN DURING MARCH, 1993, IN RESPONSE TO A REQUEST RECEIVED
FROM XXXXX XXXX.
*-----*
```

Here is another example of a program header:

```
*****;
*** THIS SAS PROGRAM GENERATES A REPORT OF ALL COMPLETED SALES TAX ***;
*** AUDITS CONDUCTED ON TAXPAYERS HAVING PARTICULAR INDUSTRY CLAS- ***;
*** SIFICATION CODES. IT WAS WRITTEN IN RESPONSE TO A REQUEST ***;
*** RECEIVED AUGUST 11, 1995 FROM XXXX XXXXXXXX (FOR XXXX XXXXXXXX, ***;
*** OFFICE OF ATTORNEY GENERAL), PERTAINING TO "XXXXXX COURT CASE".***;
*** THE PROGRAM-ID IS DSN=USER.W30.APD.TWIN504.WINNLIB(AUDITS). ***;
***
*** ANY QUESTIONS ABOUT THIS PROGRAM, OR THE REPORT IT PRODUCES, ***;
*** SHOULD BE REFERRED TO TOM WINN, PHONE 463-4907 ***;
*****;
*** MODIFIED AUGUST 15, 1995 TO SELECT ONLY CERTAIN AUDIT PERIODS ***;
*** -- DELETE RECORD IF ENDING PERIOD IS 8603 OR EARLIER, OR IF ***;
*** BEGINNING AUDIT PERIOD IS 8903 OR AFTERWARD. (TJW) ***;
*****;
```

- b) Comments should also be used to present module descriptions, and explanations of subsetting or other conditional logic.

Here is an example which illustrates the two different types of comments which may be used in a SAS job (** comment* ; and */* comment */*):

```
*** CHECK TO SEE IF AUDIT PERIODS ARE MONTH-TYPE OR QUARTER-TYPE, ***;
*** THEN ELIMINATE UNWANTED OBSERVATIONS ACCORDINGLY. ***;
LEN1 = LENGTH(BEGPER);
LEN2 = LENGTH(ENDPER);
IF LEN1 = 4 /* BEGPER IS MONTH-TYPE */
    AND BEGPER GE 8908 THEN DELETE;
ELSE IF LEN1 = 3 /* BEGPER IS QUARTER-TYPE */
    AND BEGPER GE 893 THEN DELETE;
IF LEN2 = 4 /* ENDPER IS MONTH-TYPE */
    AND ENDPER LE 8603 THEN DELETE;
ELSE IF LEN2 = 3 /* ENDPER IS QUARTER-TYPE */
    AND ENDPER LE 861 THEN DELETE;
```

- c) Use meaningful names for variables, data sets, and formats. Do provide names on DATA steps, and PROC steps that create data sets. Specify data set names as inputs to SAS procedures.

Well-chosen names furnish important information, which can help the reader to grasp the meaning and intent of the code in which they are used. Generic names like X1, X2, X3, etc. are meaningless, and provide the reader with no information.

The SAS System will automatically assign the names DATA1, DATA2, DATA3, etc. to each successive data set upon creation, but it is much more preferable for the programmer to be intentional about naming SAS data sets. Also, if the DATA= option is omitted in a PROC statement, the SAS procedure will automatically process the most recently created data set. It's much better to be explicit about identifying the data intended to be processed, instead of allowing the SAS System to make its own assumptions about what you might have meant.

- d) Use a linearized, modular style of programming. Avoid jumping to statement labels by GO TO or LINK statements and RETURN statements. If possible, replace logic which jumps between subroutines with DO ... END and IF ... THEN ... ELSE ... logic.

Following is an example of a portion of a program which uses GO TO statements and statement labels in such a way as to cause the SAS System to jump to a new location in the program, and to continue execution from that point on.

```
LENGTH BEGINP 4 STOPP 4 SORTKEY $ 19;
IF BEGPER < 1000 THEN GO TO L350;
```

```

        ELSE GO TO L355;
RETURN;
L350:   STARTQTR = BEGPER;
        STARTMO = 1;
        BYY = INT(BEGPER/10);
        BQ = BEGPER - (10*BYY);
        IF BQ > 0 THEN BMM = (3*BQ) - 2;
            ELSE IF BQ = 0 THEN BMM = 1;
        GO TO L360;
L355:   BYY = INT(BEGPER/100);
        BMM = BEGPER - 100*BYY;
        STARTQTR = INT(((BMM + 2)/3) + (BYY * 10));
        IF      BMM=1 OR BMM=4 OR BMM=7 OR BMM=10 THEN STARTMO=1
            ELSE IF BMM=2 OR BMM=5 OR BMM=8 OR BMM=11 THEN STARTMO=2;
            ELSE IF BMM=3 OR BMM=6 OR BMM=9 OR BMM=12 THEN STARTMO=3;
        GO TO L360;
L360:   BEGINP = COMPRESS(STARTQTR || STARTMO);
        IF ENDPER < 1000 THEN GO TO L370;
            ELSE GO TO L375;
        RETURN;
L370:   ENDQTR = ENDPER;
        ENDMO = 4;
        EYY = INT(ENDPER/10);
        EQ = ENDPER - (10*EYY);
        IF EQ > 0 THEN EMM = 3*EQ;
            ELSE IF EQ = 0 THEN EMM = 12;
        GO TO L380;
L375:   EYY = INT(ENDPER/100);
        EMM = ENDPER - (100*EYY);
        ENDQTR = INT(((EMM + 2)/3) + (EYY * 10));
        IF      EMM=1 OR EMM=4 OR EMM=7 OR EMM=10 THEN ENDMO=1;
            ELSE IF EMM=2 OR EMM=5 OR EMM=8 OR EMM=11 THEN ENDMO=2;
            ELSE IF EMM=3 OR EMM=6 OR EMM=9 OR EMM=12 THEN ENDMO=3;
        GO TO L380;
L380:   STOPP = COMPRESS(ENDQTR || ENDMO);
        SORTKEY = COMPRESS(TPNUM || BEGINP || STOPP);
        KEEP TPNUM BEGPER ENDPER BEGINP STOPP
            SORTKEY TOTHR5 TRESULTS;

```

This is a legitimate programming style, but the approach can lead to convoluted statements which may be difficult to follow. Actually, the preceding example isn't too arcane — it *is* modular and fairly linear. (It could have been made a lot more confusing by making additional, separately-labeled subroutines out of the conclusions of each conditional statement, and then by shuffling all of the subroutines in the program. But you get the idea.) Here is the same program, where the execution is controlled by DO-END and IF-THEN-ELSE programming statements, which most SAS programmers find are easier to read and to understand.

```

LENGTH BEGINP 4 STOPP 4 SORTKEY $ 19;
IF BEGPER < 1000 THEN DO;
    STARTQTR = BEGPER;
    STARTMO = 1;
    BYY = INT(BEGPER/10);
    BQ = BEGPER - (10*BYY);
    IF BQ > 0 THEN BMM = (3*BQ) - 2;
        ELSE IF BQ = 0 THEN BMM = 1;
END;
ELSE DO;
    BYY = INT(BEGPER/100);
    BMM = BEGPER - 100*BYY;
    STARTQTR = INT(((BMM + 2)/3) + (BYY * 10));
    IF      BMM=1 OR BMM=4 OR BMM=7 OR BMM=10 THEN STARTMO=1;
        ELSE IF BMM=2 OR BMM=5 OR BMM=8 OR BMM=11 THEN STARTMO=2;
        ELSE IF BMM=3 OR BMM=6 OR BMM=9 OR BMM=12 THEN STARTMO=3;
END;
IF ENDPER < 1000 THEN DO;
    ENDQTR = ENDPER;

```

```

ENDMO = 4;
EYY = INT(ENDPER/10);
EQ = ENDPER - (10*EYY);
IF EQ > 0 THEN EMM = 3*EQ;
    ELSE IF EQ = 0 THEN EMM = 12;
END;
ELSE DO;
    EYY = INT(ENDPER/100);
    EMM = ENDPER - (100*EYY);
    ENDQTR = INT(((EMM + 2)/3) + (EYY * 10));
    IF      EMM=1 OR EMM=4 OR EMM=7 OR EMM=10 THEN ENDMO=1;
    ELSE IF EMM=2 OR EMM=5 OR EMM=8 OR EMM=11 THEN ENDMO=2;
    ELSE IF EMM=3 OR EMM=6 OR EMM=9 OR EMM=12 THEN ENDMO=3;
END;
BEGINP = COMPRESS(STARTQTR || STARTMO);
STOPP  = COMPRESS(ENDQTR || ENDMO);
SORTKEY = COMPRESS(TPNUM || BEGINP || STOPP);
KEEP TPNUM  BEGPER  ENDPER  BEGINP  STOPP
    SORTKEY  TOTHR  TRESULTS  COMDATE;

```

- e) Don't place more than one programming statement on a single line.

The following syntax may be acceptable to the SAS System,

```
DATA COMBO; SET AUDITS FRAUDITS; BY SORTKEY; IF STATUS EQ '1';
```

But is more difficult to read than these lines:

```

*** RE-COMBINE (BY INTERLEAVING) REGULAR AUDITS AND FRAUD AUDITS *****;
DATA COMBO;
    SET AUDITS FRAUDITS;
    BY SORTKEY;
    *** ELIMINATE AUDITS THAT ARE NOT COMPLETED *****;
    IF STATUS EQ '1';

```

- f) Separate blocks of code. Insert a blank line before each step boundary. Preface major blocks of code with explanatory comments (see "b" and "e"). Consider inserting PAGE statements to force the SAS Log to begin tracing the execution of new modules on a new page.

Here are a few lines of code from a SAS job:

```

PROC SORT DATA=SLSAUDS NODUP OUT=COMPFIL.AUDS; BY SORTKEY; PROC PRINT
DATA=COMPFIL.AUDS N UNIFORM; ID SORTKEY; VAR TPNUM TPNAME SIC BEGPER ENDPER
COMDATE COMDATE TOTHR TRESULTS; SUM TOTHR TRESULTS; FORMAT COMDATE DATE7.
TOTHR COMMA12.0 TRESULTS DOLLAR15.2; TITLE1 "S T A T E   O F   T E X A S";
TITLE2 "COMPTROLLER OF PUBLIC ACCOUNTS -- AUDIT DIVISION"; TITLE3 "COMPLETED
SALES TAX AUDITS FOR PARTICULAR INDUSTRY CODES"; TITLE4 "WHERE AUDIT PERIODS
OVERLAP WITH INTERVAL OF INTEREST";

```

The following lines would be much easier to follow:

```

*** CREATE A TAPE FILE CONTAINING THE SELECTED AUDITS ***;
PROC SORT DATA=SLSAUDS NODUP OUT=COMPFIL.AUDS;
    BY SORTKEY;

*** PRODUCE A REPORT OF THE AUDITS UNDER DISCUSSION *****;
PROC PRINT DATA=COMPFIL.AUDS N UNIFORM;
    ID SORTKEY;
    VAR TPNUM TPNAME SIC BEGPER ENDPER COMDATE COMDATE TOTHR TRESULTS;
    SUM TOTHR TRESULTS;
    FORMAT COMDATE DATE7. TOTHR COMMA12.0 TRESULTS DOLLAR15.2;
    TITLE1 "S T A T E   O F   T E X A S";
    TITLE2 "COMPTROLLER OF PUBLIC ACCOUNTS -- AUDIT DIVISION";
    TITLE3 "COMPLETED SALES TAX AUDITS FOR PARTICULAR INDUSTRY CODES";
    TITLE4 "WHERE AUDIT PERIODS OVERLAP WITH INTERVAL OF INTEREST";

```

- g) Group the executable statements. Place most non-executable statements in a DATA step ahead of all the executable statements. In particular, place variable attribute and RETAIN statements near to the top of the DATA step, and ahead of the executable statements. On the other hand, place DROP, KEEP, and LABEL statements after the executable statements.

There is a difference between declarative programming statements which deal with descriptor information (like ATTRIB, LENGTH, INFORMAT, FORMAT) or other declarative statements (like ARRAY, RENAME, RETAIN), and which take effect when the SAS System compiles the program, and programming statements which actually affect DATA step execution (like INPUT, INFILE, SET, MERGE, DO, IF, IF-THEN-ELSE, PUT, RETURN, and assignment statements). It is a good idea to group most of the declarative statements together at the beginning of the DATA step, to keep them from cluttering up the logic portion of the program. An additional advantage of following this convention is that one could get a good idea which variables are created and their characteristics just by scanning the first few lines in a DATA step. However, it also seems logical to place DROP, KEEP, and LABEL statements *after* the executable statements, instead of *before* them.

- h) Indent conditional blocks and DO groups. And vertically line-up all DO statements with their associated END statements. This makes it easier to identify the group of statements which are executed as a unit. (See the second listing in "d" for an example.)

- i) Explicit versus Implicit Coding -- Know your defaults, but don't overdo invoking them implicitly.

The SAS System certainly makes computing easy for the novice programmer. It provides data set names, as well as default types, widths, and formats for variables, even though the programmer may have neglected to code these items explicitly. Unless the programmer specifies otherwise, the SAS System assumes that all of the observations for all of the variables should be included in the output data set. And, if the programmer fails to identify a particular SAS dataset when invoking a SAS procedure, the SAS System assumes that the programmer intended that the PROC should be executed using the most recently-created dataset within the job. Each SAS procedure has default options. This is all very convenient, so long as the actual outcome is what was intended. The advice is, *know your defaults, but don't overdo invoking them implicitly*. There needs to be a balance between programmer convenience and program completeness and clarity. The basic idea is to be intentional about telling the SAS System what you want, and not just to rely passively on SAS defaults. A reader of your code should be able to fully and correctly comprehend what you wrote. *Don't be too clever*. The programmer should strive to make the program easy to understand, and care also needs to be taken to ensure that the wrong dataset isn't used, by accident.

- j) Simplify complicated code and complex expressions. Insert parentheses in meaningful places in order to clarify the sequence in which mathematical or logical operations are performed. Also, there are times when complicated statements could be broken into a number of simpler statements for clarity.

- k) Screen data for unusual circumstances. Errors often can be avoided by checking the data for invalid values. Unfortunately, erroneous or unforeseen data abound throughout the world, and your SAS program will find them. Experience can be your best teacher. Pay attention to the unexpected circumstances that aren't supposed to happen, but do, in the particular data files that you use. It would be a good idea for you to do some preliminary data exploring with PROC FREQ, to become better acquainted with your data, *before* you put that new, special report into production. If the subject-matter experts tell you that only n possible conditions can be applicable in some given situation, you should be skeptical enough to expect more than that, and code accordingly.

l) Regarding macros . . .

One way of simplifying your code is to write the code you use repeatedly as a macro, and then, instead of repeating your code, invoke the macro. But beware! The art of debugging macros can be fairly mysterious. Furthermore, since most of the operations of the SAS macro facility are carried out in the background, it is especially important for macros to be well-documented.

Improving Computer Efficiency of SAS Code

Now let's think about some ways of improving the *computer efficiency* of the SAS programs which we write. Computer efficiency has several aspects, including the utilization of such resources as time (both CPU and I/O), memory, and data storage. Here are a few things you can do to make better use of these resources:

- a) Decreasing the number of input/output operations. The most important thing most SAS users can do to improve the computer efficiency of their programs is to reduce the number of times the data are read in SAS programs. Efficiency-minded SAS programmers strive to *minimize the number of passes through the data*, as well as to *minimize the number of DATA steps*. If multiple data sets are to be created from the master data set, then create them all with one DATA step. The use of intermediate DATA steps to subset the data, to transform the data, or to perform calculations should be curtailed. Instead of using a subsetting IF statement in a DATA step as a preliminary to running some SAS procedure, eliminate the need for a DATA step entirely by performing the subsetting in the PROC step, through the use of a WHERE statement. Wherever possible, take advantage of procedure output datasets as inputs to other PROCs, to avoid having to read the master data set again. Read and store only the fields that you need. *Sort data only when absolutely necessary*.
- b) Use the IN operator instead of a series of multiple logical OR operators. This programming tip has the advantage of saving CPU time, as well as being easier to write and maintain.

For example, instead of coding a DATA step like this:

```
DATA AUDITS;
  INFILE FAIS;
  INPUT @7 TAXCD $2.
        @91 SIC $4. @;
  *** SELECT ONLY SALES TAX RECORDS WITH DESIRED SICS ***;
  IF TAXCD='02' AND (SIC='4841' OR SIC='4899' OR SIC='7389' OR SIC='7812')
  THEN DO;
    INPUT @1 TPNUM PD6. @9 BEGPER $4.
          @13 ENDPER $4. @38 ASSIGN $2.
          @70 COMDATE ?? MMDDYY6.
          @84 STATUS $1. @95 RESULTS PD5.;
    ... other SAS statements
  END;
```

Alternatively, it is suggested that the following lines be used:

```
DATA AUDITS;
  INFILE FAIS;
  INPUT @7 TAXCD $2.
        @91 SIC $4. @;
  *** SELECT ONLY SALES TAX RECORDS WITH DESIRED SICS ***;
  IF TAXCD='02' AND SIC IN ('4841','4899','7389','7812')
  THEN DO;
    INPUT @1 TPNUM PD6. @9 BEGPER $4.
          @13 ENDPER $4. @38 ASSIGN $2.
          @70 COMDATE ?? MMDDYY6.
          @84 STATUS $1. @95 RESULTS PD5.;
    ... other SAS statements
  END;
```

- c) When using nested IF . . . THEN . . . ELSE . . . statements, always check for conditions in the order in which they are likely to occur; that is, list them in descending order of probability. Considering the conditions in descending order of probability saves CPU time.

Summary

When developing SAS programs, be sure to code in the clearest manner possible, and streamline the input-output operations in your program so as to use the least amount of computer resources. Just do it! The time you save may be your own.

References

- Jim DeFoor (1991), "Building Code That Is Easy to DeBug and Maintain", Proceedings of the 1991 South-Central SAS Regional Conference, Arlington, TX: The South-Central SAS Users' Group, pp. 51-67
- Alfred W. Ey (1991), "Evaluating Efficiency Techniques Used in the Processing of Large Data Files", Proceedings of the 1991 South-Central SAS Regional Conference, Arlington, TX: The South-Central SAS Users' Group, pp. 69-85
- Steven First (1994), "Developing Easily Maintained SAS Code", Proceedings of the Nineteenth Annual SAS Users Group International Conference, pp. 1350-1357
- Clarence W. Jackson (1993), "SAS Programming for Efficient Processing", Proceedings of the 1993 South-Central SAS Regional Conference, Houston, TX: The South-Central SAS Users' Group, pp. 114-119
- SAS Institute Inc. (1990), Chapter 17 - "Optimizing Performance", SAS Companion for the MVS Environment, Version 6, First Edition, Cary, NC: SAS Institute Inc., pp. 473-493
- SAS Institute Inc. (1990), SAS Programming Tips: A Guide to Efficient SAS Processing, Cary, NC: SAS Institute Inc.
- Steven A. Wilson (1995), "Techniques for Efficiently Accessing and Managing Data", Proceedings of the Twentieth Annual SAS Users Group International Conference, pp. 474-483

The author can be contacted at:

Audit Division Headquarters
Texas Comptroller of Public Accounts
L.B.J. State Office Building, 111 East 17th Street
Austin, TX 78774

Voice: (512) 463-4907
FAX: (512) 475-0349
E-Mail: 0007095870@mcimail.com
twin504@audit.cpa.state.tx.us

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