

Guidelines for Coding of SAS® Programs

Thomas J. Winn, Jr.

Texas State Auditor's Office

Abstract

This paper presents a set of proposed guidelines that could be used for writing SAS code that is clear, efficient, and easy to maintain. The guidelines described herein are widely regarded as good practices for SAS programmers of all ability levels, and for all types of computing environments.

Introduction

Besides being very powerful, the SAS programming language also is very flexible. It allows programmers to write valid code in several different ways in order to accomplish the same result. However, just because you can write SAS code in a particular way does not mean that you should do so. Issues involving programming style involve more than just aesthetic interest. SAS code which is written with careful attention to certain principles is much easier to read, easier to correct if it contains mistakes, and easier to change afterward. In my own coding, I strive to find the right balance between computer efficiency and maintainability.

The idea behind conforming to a good set of standards is not to make our lives unnecessarily difficult, but actually to make the accomplishment of our work easier. Like all good habits, following good practices may keep us out of trouble later. On rare occasions, in which expediency is extremely important, I may decide not to follow all of my own standards but, even then, I recognize that I would be better off if I did follow them. There always are risks associated with “cutting corners”.

General Recommendations

First, I recommend that SAS programmers use a Software Development Life Cycle Methodology, whether developing complex systems or individual programs, whether manual or automated, and whether new or a modification of some existing project. The steps are:

- (1) Problem Definition – Identify the users' requirements – this is a statement of needs, from the users' perspective,
- (2) Preliminary Analysis and Planning – Refine the common understanding (between users and programmers) concerning WHAT the program or system is supposed to do – these are statements containing the functional specifications, the feasibility of the project, the proposed timeline, the resources needed, and a consideration of alternative solutions,
- (3) Design – Determine HOW the program or system will get it done -- create detailed specifications for each process, the development workplan, and a protocol for validation,
- (4) Coding and Testing of the components – programming activities,
- (5) Integration with other systems, and more Testing

- (6) Obtain User Acceptance – this involves more testing before user signoff for deliverables,
 - (7) Implementation, documentation, and user training,
 - (8) Post-Implementation Analysis, Maintenance, and Change Management.
- These steps may be compressed to include only those activities that would be appropriate for a particular project. However many steps there are, each of them would include testing, to ensure that the desired outcomes were achieved. And each step also would seek an agreement of all of the parties involved regarding the outcomes. The essential walkthroughs will help to clarify understandings.

Moreover, I make it a practice to create a physical file folder for each project I work on, containing written documentation regarding each step performed. The folder would include copies of the initial (and subsequent) requests, a narrative description of the work performed, information regarding program inputs and outputs, program listings, SAS logs, pertinent correspondence, and requestor responses. I do this because I tend to forget many details over time. I firmly believe that there is no such thing as disposable code, and I have noticed that very few of the projects that I work on ever completely go away – similar requests seem to just have a way of reappearing. I try to save enough information about the work that I do, so that I could re-construct my results with ease. Chances are, these observations about me and my work also apply to you. So develop the habit of writing essential things down, and of systematically storing them for later reference. And don't just stuff things into the folder, add plenty of explanatory remarks which will give context to the included materials.

Names

- Use meaningful, unambiguous names for libraries, programs, files, data sets, variables, formats, macros, and macro variables – avoid cuteness, single-letter names, and names that too closely resemble one another,
[Note – in Version 6 and earlier releases of the SAS System, the maximum length of SAS names is 8 characters; under Version 8, most SAS names may be up to 32 characters in length.]
- Names should be unique, short, and descriptive – in that order of importance. Underscores may be used to separate words, in order to enhance readability.
- If a user-defined format applies to only one variable, then name the format with a readily recognizable form of the variable-name plus the suffix FMT.
- Programs which are sequential should share the same name-prefix, but also have a numerical suffix which indicates their place in the sequence.
- Names of data files, which are shared by a group of individuals, should adhere to a naming standard which suits the needs of the group and the context of the data. For example, name components might include distinct identifiers for application, module, modifier, source, date-time, userID, sequence-number, and type-extension. Develop a standard that works, and use it consistently.

Readability & Appearance

- Write code that is readable. It will be easier to understand and to modify.
 - ♦ Programs should look neat and orderly,
 - ♦ The visual appearance of a program listing should mirror its logical structure,
 - ♦ Use white space liberally,
- Separate blocks of code, using indents and white space --
 - ♦ Insert a blank line between SAS program steps; that is, before each DATA or PROC step,
 - ♦ Be consistent with your indentation increments,
 - ♦ Indent all statements in a logical grouping by the same amount,
 - ♦ Left-justify all OPTIONS, DATA, PROC, and RUN statements. Indent all of the statements within a DATA or PROC step,
 - ♦ Indent conditional blocks and DO groups, and do it consistently, The logic will be easier to follow.
 - ♦ Align each END statement with its corresponding DO statement. This will make it easier to verify that they match.
 - ♦ Remember to preface major blocks of code with explanatory comments,
 - ♦ Consider inserting PAGE statements to force the SAS Log to begin tracing the execution of new modules on a new page,
- 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,
 - ♦ Break really complicated statements into a number of simpler statements,
 - ♦ Don't be too clever,
 - ♦ Even if you think that it may be more elegant to do so, refrain from using unconventional, obscure, and convoluted logic, except when you can't think of a simpler approach and, if you must, then insert lots of explanatory comments,
 - ♦ Avoid implicit coding -- know your defaults, but don't overdo invoking them implicitly (for example, provide names on DATA steps and PROC steps that create data sets, and specify data set names as inputs to SAS procedures),
- Don't place more than one programming statement on a single line.
- Make sure that titles and footnotes used in printed reports accurately reflect the information being presented.
- Make sure to use date formats when reporting SAS dates.
- Do not extend any statement beyond column 72.

Internal Program Documentation

- Use numerous explanatory comments within the program,
 - ◆ Use comments to provide a prologue / program header, which describes what the program was intended to accomplish, its author, and the circumstances of its having been written (what, why, for whom, by whom, and when); the prologue also should include a change control log, which describes subsequent modifications made to the program (what, why, for whom, by whom, and when),
 - ◆ Use comments to present module descriptions,
 - ◆ Use comments to explain subsetting or other conditional logic,
 - ◆ Use comments throughout the code to document the program,
 - ◆ For situations in which meaningful names would be too long, use inline comments (*/* . . . */*) to more fully identify names whose required brevity obscures their meaning.

Reusability

- Write code that can be re-used, with different parameters. Keyword parameters are preferable to positional parameters, because they are less likely to be specified incorrectly.
- Write the code you use repeatedly as a macro, and then, instead of repeating your code, invoke the macro.
- Avoid using global macro variables
- If a macro is used by more than one program, put it into an AUTOCALL macro library.

Efficiency

- 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,
 - ◆ End every DATA and PROC (except SQL) step with a RUN statement,
 - ◆ End every PROC SQL step with a QUIT statement,

- In a DATA step, place most of the non-executable statements before the executable statements – exceptions include the DROP or KEEP statements, which may be placed after the executable statements,
- Define INPUT and PUT variables one per line, using @ pointer control,
- Screen data for unusual circumstances,
- Reduce the number of times the data are read,
 - ♦ Minimize the number of passes through the data,
 - ♦ Minimize the number of DATA steps,
 - ♦ Read and store only the data that are needed,
 - ♦ Sort data only when absolutely necessary,
- When you read in an external file, use pointer controls, informats, or column specifications in the INPUT statement to read only those fields you actually need,
- Store only the variables you need by using DROP or KEEP statements, DROP= or KEEP= options (eliminate variables from the output data set which are needed only during DATA step execution, and not afterward),
- When only one condition can be true for a given observation, use IF ... THEN ... ELSE ... statements (or a SELECT group), instead of a series of IF ... THEN ... statements without ELSE statements (In a sequence of IF-THEN statements without the ELSE, the SAS System will check each condition for every observation),
- When using a series of IF ... THEN ... ELSE ... statements, list the conditions in descending order of probability,
- Use the LENGTH statement to reduce the storage space for variables in SAS data sets,
- Minimize workspace usage by using the DELETE statement in a PROC DATASETS step, to eliminate temporary data sets that are no longer needed by the program.
- Use the IN operator instead of a series of multiple logical OR operators.

Conclusion

By consistently using good coding standards for writing SAS programs, the code will be easier to follow, more efficient to run, and easier to maintain. As a result, programming errors will be reduced, and the quality of the results will be improved. Get yourself into the habit of following good coding standards, and always use them in writing your programs. You know what to do – so do it!

Author Information:

Tom Winn, Ph.D.
Senior Systems Analyst
Information Systems Team
Texas State Auditor's Office
P.O. Box 12067
Austin, TX 78711-2067

Phone: (512) 936-9735
E-Mail: twinn@sao.state.tx.us