ABSTRACT

This basic tutorial discusses tips for improving the overall efficiency of your SAS code.

Too often, talking about "efficiency" in coding raises unwelcome images of abstract nitpicking just to save a millisecond or two of CPU time. With the power of SAS software, however, users can sometimes fall into bad habits that have a tremendous negative impact on the amount of work they can get done. When efficiency considerations are totally ignored in SAS programming, bugs tend to proliferate and resource requirements can mushroom. Users then encounter what they falsely see as "limitations" in the SAS software, shrug their shoulders, and give up.

The goal of this tutorial is to show users specific ways to focus on tightening up their programs, leading to shorter code, less debugging, and less CPU time for their SAS jobs. Rather than delving into obscure efficiency "tricks," the tutorial presents generally applicable ways to develop ongoing self-discipline in coding.

INTRODUCTION

As a near-fourth-generation language, the SAS system was an especial boon to the great number of its users who were not pure programmers. The power provided by PROCs and DATA steps allowed the user to focus attention more on the task to be accomplished and less on the particular way in which data happened to be organized. This relative "data independence" is the special strength of the SAS system, and many of us who now program predominantly in the SAS language sometimes find it annoying when we have occasion to drop back to some other language where mundane considerations, such as record format and field locations, consume so much of our time.

Yet danger lurks in liberty. The very capabilities which make the SAS system so invaluable as a software development tool can also be misapplied, misunderstood, abused. New and even experienced SAS users, seduced by the sheer power of the tools at their disposal, can come to concentrate almost solely on results, ignoring how the SAS system—not to mention the resources of their installation—might be going through needless contortions to produce those results.

Such contortions, of course, however invisible they may be, always carry costs: time, money, productivity. The longer your SAS job runs, the fewer runs you can make over a given period of time. The longer and more intricate your SAS program becomes, the harder it is to shake out all the bugs in it. The more space your SAS data sets take up, the less room you have for others. At the end of this vicious circle lies an ironic truth: misuse of the freedom given to you by the capabilities of the SAS language ultimately fences you in, and you can't get the results you need.

This trap has never been so pernicious as in recent years, as we have seen the SAS language evolve from a powerful (but still basically ad hoc) tool for statistical analysis into a virtually unlimited software system designed to meet all the computing needs of its users. Large interrelated systems are now developed using SAS software. My installation routinely processes well over three million observations in a single job. This kind of large-scale use means that the impact of an undisciplined approach to coding can be potentially astronomical.

Not just large-scale SAS use requires self-discipline, though: the advent of the SAS language for personal computers has underscored the need to plan one's code to minimize resources. Lack of a disciplined approach to SAS on the PC can cause even more headaches than the same lack of discipline on the mainframe, where pure computing power and massive storage capability often compensates for programmer sloth.

The obvious question for many is where and how to begin to develop the necessary awareness and understanding to avoid the problem I have outlined. By their very nature, the many volumes of excellent documentation for the SAS system are more aimed at teaching the capabilities of specific language features than at discerning between good and bad ways of applying those features. What is obviously needed, then, is a guidebook through the pitfalls of SAS programming, a Baedeker showing a few of the pebbles along the route to self-discipline in SAS coding. That is the purpose of this tutorial.

PHILOSOPHICAL RULES

Most of the concrete recommendations in this tutorial have arisen from countless debugging sessions, from hours spent with SAS users at my installation poring over listings and PROC PRINTs of intermediate data sets in an attempt to find an elusive error. From these sessions, I have formulated several general rules to keep in mind when going about coding a new application or examining an old one. Although I
will present more specific rules later, the specific ones come from the general philosophical ones and it is perhaps best to state these early on.

First of all, don't let the SAS software do absolutely everything for you, at least not when you are developing code which will be used again and again. Basically, this rule involves limiting your reliance on the very data independence which is such a strength of SAS. It means that you will be exercising greater control over the steps of the task at hand, not just allowing SAS to make assumptions about what variables to keep, what data sets to process, how to output.

Secondly, let the SAS software do more for you. This sounds like a contradiction of Philosophical Rule number one, but the emphasis is different. The focus of this rule is on using the power of the SAS language to its fullest advantage. Many users don't know of basic capabilities (as outlined in the SAS documentation) and thus tend to re-invent PROCs or to make do without a function at times when a better tool was at hand.

Finally, know and understand what the SAS software is doing. I often see users take a trial-and-error approach to coding, rather than sit down with listings and manuals to figure out exactly what's going on in their program. An obvious rule, perhaps, but one which is followed all too seldom.

The specific examples which follow will use these rules as a springboard to show you ways to focus on tightening up your code. If you can make these techniques into ingrained habits, I guarantee you that you will reduce your debugging time and resource use in SAS, not to mention achieving greater confidence in your results.

STARTING TENETS TO LEARN

Of the many bugs in SAS programs which I have encountered, the cause of a significant number—perhaps as high as thirty to forty percent—boiled down to the vital need to follow three practices in virtually every piece of SAS code you produce, no matter how fleeting its anticipated use. These tenets are important not only because they help avoid bugs, though; following the first two tenets, at least, is almost certain to reduce the amount of storage and time you use.

I) Use (KEEP=) for almost every data set, both for input and output.

II) Use explicit LENGTH declarations whenever possible.

III) Learn (and keep relearning) the use and intricacies of the SET, MERGE, and UPDATE statements, as well as BY and SORT.

Tenet I (using the (KEEP=) option on input and output data sets) seems an especially harsh blow to the treasured data independence of the SAS system. Users tend to resist following it, simply because it requires planning ahead, and few of us really like to plan ahead. But remember our philosophy: don't let the SAS system do everything for you. Left to its own devices, SAS will keep every variable you will ever dream up. Here again, reject the default.

Tenet I serves at least two purposes, even apart from the reduction of resources it effects: first, it increases the self-documenting nature of the SAS program. When you know exactly which variables came from which data set, and which of those variables were kept in the output data set or sets, it is much easier to follow the code and make changes to it later. Secondly, if you follow this rule it is much harder to be waylaid (in a MERGE, for example) by an unexpected variable coming from a data set and overwriting another variable which you had named the same: in short, with this practice there are few surprises.

Tenet II (using explicit LENGTH statements whenever possible) again encroaches on the turf of data independence. Am I trying to turn this language into PL/I? No, but a lot of users don't realize that the SAS default of 8-byte precision will retain 16 significant digits. Most users would be quite content with 7 significant digits, which can be achieved in half the storage (4 bytes). Users who process thousands (or millions) of observations can save themselves untold amounts of space and time by remembering to be wary of defaults.

Tenet III (learn and keep relearning SET, MERGE, UPDATE, BY, and SORT) points at the most powerful—and least understood—basic tools in the SAS system. A significant number of programming errors occur because users haven't really understood how these tools work in different situations. My constant advice to every SAS user I know, both novice and advanced, is to re-read the SAS manual's excellent sections on SET, MERGE and UPDATE regularly.

Absorbing and implementing these tenets in your SAS code will do a great deal to steer you in the direction of a healthy self-discipline in SAS. But remember, these are simply starting tenets to learn. There are two more advanced ways you can improve your SAS programming and get more done.

SECRETS TO IMPROVING SAS PROGRAMMING

I) Zero in on, and try to remove, repetition in your code.

II) Read as few data sets as possible as seldom as possible.
The first secret is a way to isolate sections of your programs where it may be that things could be better done. In almost every case, if you are doing something over and over, in the same or nearly the same way, there is some SAS language feature that will help you to do it better or faster or less painfully—sometimes all three. If you focus on this repetition and try to think of other ways to approach the task at hand, you are aiming at our Philosophical Rule number two, which was to let SAS do more for you.

There are numerous ways in the SAS language to avoid repetitive coding, including loops and arrays, functions, use of format for table lookup, PROCs themselves, and of course the macro language. Sometimes whole pages of SAS code can be slashed and replaced with a single PROC which will do the same thing better with no debugging. I have seen experienced SAS programmers repeat blocks of almost identical code for pages and pages, simply because they didn't feel quite comfortable with the SAS macro facility. The real lesson here, of course, is to learn the tools of your trade, at least to the point of saving yourself grief.

The second secret (read as few data sets as possible as seldom as possible) points at the tendency of many SAS users to read and reread a data set within the course of the same job but for different purposes. One corollary of this rule is that if you have a very short DATA step, scrutinize it carefully; there's probably a way to combine it with another DATA step. For example, many users don't know or remember about multiple output data sets from a single DATA step, so they read the same data set three times in order to create three different subsets. Don't do this. Use the power available in the SAS language.

Another major corollary to this secret to improving your SAS programming has to do with SORT. Don't sort a data set when you don't need to. Typically, users get, early in their SAS experience, the rather taunting error message of

```
DATA SET IS NOT SORTED.
TO SORT THE DATA SET,
USE PROC SORT.
```

Almost every beginning SAS user I have ever seen responds to this little reminder by using PROC SORT after every single data step, just to be safe. Don't do this. Realize that once a data set is in sorted order (whether it got there by PROC SORT or simply through circumstance), there is no need to sort it again unless you change the actual variables named in the BY statement. In other words, a SET statement or a MERGE statement with a BY statement will preserve the sort order specified by the BY statement.

CONCLUSION

Years of SAS programming and debugging have convinced me that following the rules I have outlined in this tutorial will almost certainly reduce the data processing and human resources (e.g. sanity) you are currently spending on SAS programming.

Comments are welcome. Contact

Peter Kretzman
Rate Department, Room B932
77 Beale Street
San Francisco, CA 94106
(415) 972-8481

NOTE: SAS is a registered trademark of SAS Institute, Inc., Cary, NC, USA.