An Information Centre SAS® Consultant’s Opinion

On HOW TO BUILD A "BAD" SAS® PROGRAM

F.W. Bruce Densmore, Bank of Nova Scotia

Abstract

After two years as an internal Information Centre SAS® consultant I have seen SAS code written by scores of different programmers. Some were new to SAS programming, while others had years of experience.

I have been thinking about what makes a "GOOD" SAS program. How should a SAS program be written so that it is readable, understandable, and easily maintained?

I found it easier to describe the things that constitute "BAD" techniques rather than define a formula for generating "GOOD" SAS code.

I plan a "tongue-in-cheek" review of "techniques" I have encountered, followed by my "rules of thumb" for generating what I consider reasonably "GOOD" SAS code.

This will not be an exhaustive discussion of how SAS software products should be used and I will only deal with components of the base SAS product. SAS software is a tremendously powerful programming tool and its proper use can be extremely beneficial in the area of program development and later program maintenance.

Introduction

I have been thinking about what makes a "GOOD" SAS program. How should a SAS program be written so that it is readable, understandable, and easily maintained? The actual complexity of the problem being addressed affects these criteria but I believe there are steps that can be taken by the SAS programmer that can simplify the matter.

My ulterior motive is to lay the groundwork for SAS programming standards that the Information Centre should expect from any SAS routines that they will be expected to maintain.

It is my opinion that most of the SAS programs written by Information Centre users can be generalised into four steps: read and select relevant data, sort that data, summarise the data, and produce the report(s). There are various ways to accomplish this using SAS software as the following table may illustrate. Other sorts, summarisations, and reports may be needed in one program.

How to build a "BAD" SAS program.

The following points should be taken "tongue-in-cheek". I found it easier to describe the things that constitute "BAD" techniques rather than define a formula for generating "GOOD" SAS code. I should point out, though, that in my two years as an Information Centre SAS consultant, I did see examples of every one of the "techniques" listed.

1. Use only one SAS step, a DATA step, especially a DATA _NULL_ step. This will allow you to defeat most of the built-in facilities of the base SAS software. The added complexity will frustrate anyone trying to maintain your program. You may even get lost yourself.

---

<table>
<thead>
<tr>
<th>DATA PROCESS</th>
<th>Method Type 1</th>
<th>Method Type 2</th>
<th>Method Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>read and select</td>
<td>DATA ...</td>
<td>INPUT ...</td>
<td>IF ...</td>
</tr>
<tr>
<td>sort</td>
<td>PROC SORT ...</td>
<td>PROC SUMMARY ...</td>
<td>PROC</td>
</tr>
<tr>
<td>summarisation</td>
<td>PROC MEANS ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>report</td>
<td>PROC PRINT ...</td>
<td>PROC PRINT ...</td>
<td>TABULATE ...</td>
</tr>
<tr>
<td>-or-</td>
<td>DATA <em>NULL</em> ...</td>
<td>DATA <em>NULL</em> ...</td>
<td></td>
</tr>
</tbody>
</table>
---

479
2. Go nuts with SAS arrays, explicit or implicit. Better yet, set up an array of arrays. By the time someone wades through the code needed to support this concept they will be in no shape to attempt understanding what your program does.

3. Ignore all the powerful SAS data manipulation functions and code your own. This will confuse you as well as anyone to follow, even if you spend all the time needed to get it right (including leap years).

4. Ignore the SAS summarisation procedures like PROC SUMMARY, PROC MEANS, PROC FREQ, and PROC TABULATE. You can literally introduce hundreds of unnecessary and confusing lines of code this way.

5. Go nuts with the SAS macro facilities (there are two). Better yet, build a really complicated macro that invokes other macros and then invoke the whole thing ONLY ONCE. This will make your program look twice as long as it need be, while hiding the process in a maze of interconnected code.

6. Above all else, ignore structured programming techniques and modular design. SAS software by its nature encourages and supports both techniques in tandem. Defeating this is an excellent way to write "BAD" SAS code.

7. Use GO TO statements and subroutine calls. In other programming languages they often simplify the coding process, however, I believe they can greatly increase the complexity of your SAS code, and reduce its readability. If a SAS program is well thought out neither is really necessary. Since SAS programs are interpreted (rather than compiled), coding subroutines as simple macros can be the better way to go. Macros are available to all SAS steps, whereas subroutines exist within only one SAS step.

8. Hang on to useless data for as long as possible. The more variables you carry the harder it is to keep track of what is going on in the program. Think of the confusion that unnecessary variables can introduce. Think of the extra computing resources you can consume, especially in sorts and summarisation.

9. Use meaningless variable and SAS set names. Better yet, go out of your way to make the variable names misleading. Names like FILE1, FILE2, VAR1, etc., impart little understanding to the program. If you use names like NAME, ADDRESS, POODE, etc. for data that represents opening balance, interest, principal, you can reach a new low in program understanding. By the way, I consider BOTH of these naming techniques to be EQUALLY MISLEADING in the context of a SAS program.

10. Create a new SAS set with every SAS step. Here is an example. Read the data into FILE1, sort it into FILE2, merge it with FILE3 to create FILE4, summarise it into Files, and print a report. If the input file is large and you follow some of the other techniques listed above, you can literally use up all the work space available at your data centre. By the time you reach the report stage, all files except FILES are obsolete. In fact each successive step made the previous input file(s) obsolete.

11. Never get rid of SAS sets that are no longer needed. Along with the previous technique, these are among the fastest ways to run out of work space.

12. Read every possible input data field and keep them throughout your program whether you need them or not. This adds to the confusion and fills up the work space much faster.

"Rules of Thumb"

The following are my rules of thumb for generating what I consider reasonably "GOOD" SAS code. They are not necessarily in any particular order.

1. Learn about and use the powerful SAS procedures whenever possible. Your repertoire MUST include PROC SORT, PROC MEANS, PROC PRINT, PROC SUMMARY, PROC FREQ, and PROC TABULATE. In many respects you can think of these as "black boxes". You feed them data and certain control parameters to get the desired result. You do not have to "re-invent" the process, and anyone maintaining the code later will not have to pour through complicated logic. I can think of one case where only 68 lines of procedurised code replaced over 1100 lines of unintelligible "brute force" code.
It is important to generally understand how SAS procedures work in order to produce appropriate observations in your DATA steps. An understanding of the SAS statements BY and CLASS is essential.

2. Base SAS software supports the allocation of two variations of one-dimensional arrays, or vectors (e.g., implicit and explicit). With imaginative programming (there are examples in the SAS manual) the SAS product can be coaxed into simulating multi-dimensional arrays. This is the infamous array of arrays.

SAS arrays are really index structures superimposed on a list of variables that are not necessarily related. This can be a very powerful programming technique, if used properly. When misused it can be a disaster. The key is to realise that the SAS ARRAY statement defines an indexing mechanism for the the variables listed in the latter part of that statement. This is true for both implicit and explicit arrays.

With version 5, the SAS product introduced support for explicit array definitions thereby providing array indexing syntax that was similar to that used in many other computer languages.

This does not mean that arrays work the same way in SAS software as they do in other languages. In other languages you actually define the array variables which usually have a common name with an indexing mechanism appended and you can only refer to an array element that way. With SAS software, arrays are "index structures superimposed on a list of variables". Therefore those variables can be referenced TWO WAYS, by the actual variable name assigned, or by array indexing using your favourite implicit or explicit coding technique.

I personally have no preference but tend to use the older implicit variety in my programs. Earlier versions of SAS software exclusively used a separate independent integer variable to implicitly index the array structure.

The place of arrays is for grouping related, indexed data WITHIN an observation. Arrays have no place in accumulating data from several observations. That job should be left for the SAS summarisation procedures. To do otherwise is to introduce a totally unnecessary level of complexity.

I support and often encourage the use of arrays within a single DATA step and involving a SINGLE OBSERVATION. A proper understanding of SAS DATA step and PROC step processing should provide ample good ideas for avoiding multi-dimensional arrays that span SAS observations. This usually means taking another look at what you consider to be the key variables in your data. Use those key variables in BY or CLASS statements, and not as array indices.

3. The SAS product contains an impressive set of built-in date and time manipulation facilities, INFORMATS, FORMATS, and functions. I know of no programming language that even comes close. THERE IS VERY LITTLE REASON, beyond ignorance, TO WRITE SPECIAL CODE TO MANIPULATE DATE INFORMATION. You should be aware of such things as DATE, INFORMATS, FORMATS and constants; date conversion functions (e.g., DAY, MONTH, YEAR, MDY, DATEJUL, and JULDATE); and date processing functions (especially INTNX, and INTCK). Be aware that similar facilities exist for processing time data.

Internally, SAS programs store dates as simple numbers, the number of days since 01 Jan 1960. Therefore, date variables are available for simple arithmetic. (e.g. '01MAR88'd - '31DEC87'd = 1. Do that in some other language!) This is the key to the SAS date handling facilities.

4. Because of the SAS summarisation procedures there are few cases where a SAS programmer need worry about accumulating or manipulating data from related observations. Only in rare occasions, usually involving very large external files where only a few summary observations and variables are needed, does it pay to avoid the summarisation procedures and accumulate data "manually" in a DATA step. Generally, PROC SUMMARY, PROC FREQ, PROC FREQ, PROC TABULATE, PROC UNIVARIATE, etc. will provide whatever summarisation is necessary, more efficiently and understandably than can be programmed in a DATA step.

If you get to know and understand the processing of these SAS procedures, you will produce better, more efficient, understandable code.
5. Subsequently, your code will be easier to produce, document, and maintain.

I find that when I exploit the SAS product and its many procedures that the number of lines of "critical code" is greatly reduced. There are fewer places where my logic can go wrong, or where modifications must be made in order to affect changes. This greatly increases my ability to write complex routines quickly with reasonable confidence that they will work shortly after coding is completed.

5. The SAS macro facility is very powerful, to the point of being a language unto itself. It should be treated with respect and used cautiously. Limited use of macros greatly increases the flexibility of the SAS code that can be written.

The primary purpose of the macro facility is to provide a vehicle whereby SAS routines, under program control, can dynamically generate SAS code to be subsequently executed as part of the same processing stream. In fact, it is conceivable that the code ultimately processed bears little resemblance to code actually submitted.

There is a time and place for the "heavy-duty" use of the SAS macro facility but I suspect it has little to do with the types of end-user systems the Information Centre is likely to support. Any programmer that can figure out the convoluted logic required to dynamically build SAS code from macros should be able to simplify the process to make it less dependent on macros. The resultant code will be longer, but more straightforward, readable, understandable, and maintainable.

6. The SAS product is a modular language. A true SAS program should contain several modules or steps that logically break the program into manageable pieces. It therefore makes great sense to approach a problem by breaking it into its components and addressing each one. Knowledge of the SAS procedures will greatly simplify the process of determining the component parts of a SAS-based solution.

6. I am not totally against using the SAS macro facility. It is a useful way to produce repetitive code only once, where that code will be unconditionally substituted into the processing stream at several different points. Some of the macro commands can be used independently to carry data throughout a program stream without the problems of passing that data via files. It is this technique that makes the use of subroutines less savoury in the context of a SAS program.

When macros are used to conditionally generate the SAS code to be executed, the complexity of that SAS package increases dramatically, often beyond the ability of the programmer involved. That complexity is also very difficult for an Information Centre consultant to deal with easily. The successful use of this kind of "heavy-duty" macro application demands in depth knowledge of internal SAS substitution, compilation and execution processing. IN MOST CASES IT SHOULD BE AVOIDED.

8. When it comes to dealing with external input files and variables I have the following thoughts. As usual my concerns lie in readability, understandability, and maintainability.

When reading an external file, pick up only the fields that are needed. Your program will be less cluttered and run more efficiently.

Use meaningful variable names. This adds to the intuitive understanding of your code. This not only helps others that may have to use or enhance the program, but IT MAKES THE PROGRAM EASIER FOR YOU TO UNDERSTAND.

Choose the names of your SAS files carefully. Again this will make things easier for you. It is perfectly proper in the SAS product to use the same name for the input (SET statement) file and the output (DATA statement) file. This reduces the number of spurious intermediate files. SAS software also provides the ability to delete the remaining intermediate files that are no longer needed; do so it. Typically, you then intuitively understand which files contain the current data at any point in the program.

Some of these naming conventions may seem trivial, but when you get into maintaining numerous programs, every little trick helps if it makes things more readily understandable. A case could be made for standardising variables. (eg. name variable for branch transit number
might be TRANSIT, rather than BRNUM, TRAN, BRANCH, TRNUMB, or TRCODE and it is ALWAYS NUMERIC, never character.) Such a standard would make some things instantly recognisable to an Information Centre consultant or program maintainer. Additionally, programs and data could be interchanged and/or combined more readily.

**Tips and Techniques**

The following thoughts may prove useful as you choose the SAS procedures needed to process your data.

1. **Learn how SAS processes observations and structure your data selection and/or creation to exploit it.** At first this will seem strange, but it pays off in the long run.

2. **For each input record, deal with it entirely at the time you read it.** If necessary, generate multiple observations that will be processed later. Remember, you can conditionally create several SAS sets in a single DATA step.

3. **If you are selecting records based on data they contain, read the record in two pieces, using the "trailing @" facility.** First read the fields that form the basis of your selection. Use the SAS "SUBSETTING IF" statement to test the selection criteria. Only then process the rest of the required data from the input file. This technique can greatly reduce the time spent reading unwanted data.

4. **Sorting data is incredibly expensive, so do not sort data unnecessarily, and sort as few records as possible.** Do your data selection **BEFORE** you sort. If the last time the data was used it was in the order you now need, just use it, **DO NOT SORT IT AGAIN.** If the data you are about to use is the result of a DATA or PROC step that contains a BY or CLASS statement equal to the sort order you need, and those variables were not changed within the body of the DATA step, then that data is already sorted, **DO NOT SORT IT AGAIN.** For efficiency, **ALWAYS code the NOEQUALS option on your PROC SORT statements.**

5. **Use the SUMMARY procedure whenever multiple or successive levels of summarised data are needed.** A subsequent DATA step can be used to separate and/or combine the data for reporting purposes. I find that "bit testing" the TYPE variable, based on the - desired presence/absence (1/0) of the CLASS statement variables is the clearest technique. Set missing key variables to values that complement the desired sort order when sorted or interleaved.

6. **When you think you have the data you want, use PROC PRINT to dump it to paper.** This is a powerful simple procedure that may produce a satisfactory report with virtually no effort on your part. It will even produce simple totals and subtotals. Remember, this report is practically free. If you want to get fancy, you are going to have to write many lines of code. Are the layout, programming, and subsequent maintenance efforts **REALLY WORTH IT?**

7. **Invest some time! Study PROC TABULATE!** Along with the documentation in the basics manual SAS has a separate manual on this amazing routine. In about 10 lines you can define a detail/summary report that otherwise could require many hundreds of lines of code comprised of data selection, sorting, summarisation, transposition, and report formation. Once you understand PROC TABULATE, you will wonder why nobody told you about it before.

**Conclusions**

I do not believe that this has been an exhaustive discussion of how SAS software should be used. I have only dealt with components of the base SAS product. The base SAS manual is currently 1300 pages. There is no way to condense that bulk into this report. I have tried to highlight problems I encountered in two years of SAS consulting for the Information Centre. Hopefully I have explained what I believe are relevant solutions and techniques.
The SAS product is a tremendously powerful programming tool. Its proper use can be extremely beneficial in the area of program development and later program maintenance. As we embark into the arena of Host Applications development, we should consider setting minimum acceptable standards for the SAS programs we create. Those programs should be readable, understandable, and easily maintained. They should be as intuitive as possible. If I have at least initiated that process, then I have accomplished something with this document.

Trademark Citations

* SAS is a registered trademark of the SAS Institute Inc., Cary, NC, USA.

F.W. Bruce Densmore,
Model Branch & Host Application Services

Bank of Nova Scotia,
Scotia Plaza,
44 King Street West,
Toronto, Ontario
Canada, M5H 1H1

(416)-866-7015