

Paper AD04_05
Systematic SAS® Development, A Disciplined Approach
John W. Davison, Jr., Factotum, Inc., Leesburg, VA

ABSTRACT

This paper presents my systematic approach to SAS® application design, based on nearly thirty years' experience in SAS development and maintenance. Without setting forth a rigid framework some broad guidelines are offered embodying the author's own rules that have served him well and may help others to develop their own. They represent a step toward formulation of a disciplined approach to more rapid development of more robust and maintainable applications and more reusable code.

INTRODUCTION

This paper is not intended to be a compendium of SAS® tips and tricks but seeks to organize and enunciate some guidelines for application design that I have found to be useful. I have worked in system and application development for four and a half decades with two thirds of that time predominantly in the SAS environment. Much, perhaps most, of what I present here restates principles set forth formally by others who are better qualified than I to teach good design. My viewpoint here is pragmatic, drawing on my own experience.

My notions have a twofold impetus. On the one hand, I seek to improve my work by drawing lessons from each project to apply subsequently. On the other hand, I have frequently had to maintain and enhance existing code, often my own but still more often by someone else. This has taught me the value of writing with an audience in mind that is unfamiliar with the original work. The saying has the ring of truth that one's own code viewed after any lapse of time may as well have been written by a stranger.

BACKGROUND

My own approach to these issues has been to cultivate a systematic organization for project structure and component design. Most of the principles that I have sought to apply are not specific to SAS development but to applications in any programming environment. They reflect concepts to which I was first exposed with my introduction in 1964 to higher language, in ALGOL as implemented on the Burroughs B5500, later acknowledged by the developers of PL/I as the model for their own work. In this environment typical program style was to write several subroutine or procedure definitions with the main program being little more than a sequence of procedure calls. Later in this period the term "structured programming" came into fashion and then gave way to "object oriented programming." Those with my background believed that essentially we were already doing the first and that the second could be viewed as a variation on the theme. Thus, to repeat myself, much if not most of what I have to say is a repetition of recommended practice, and perhaps even obvious or common sense.

This approach to program design was reinforced by another influence, the appearance in 1976 of the book *Software Tools* by Brian Kernighan and P. J. Plauger, some months before my first exposure to SAS. At Bell Labs, now Lucent, Brian Kernighan was near the center of modern thought about computing systems. He is the coauthor of several books on various aspects of the subject, any of which has much to offer. This book is considered by many to be one of the most important treatments of program design,

probably the dominant influence on my own thinking, and the inspiration for my second personal rule of good design. In my view only one other rule is more important.

MY FIRST RULE - BE CONSISTENT

There are in fact many ways to approach each problem, most if not all of which may be equally effective and so in a sense equally correct. I believe that whatever approach one takes, it is important to do so consistently. People may disagree on such conventions as indentation or semicolon placement but individually or in the work group they should establish and adhere to conventions of both organization and style in order that humans may recognize more readily the intent of code. Even if the work group tolerates individual differences, having a known style makes it easier for others to understand one's work. This is my most important rule.

To belabor the point, if the organization has established formal or informal style conventions, conform to them. If not, choose a style that is useful and stick to it. I will say more about my own personal style choices in a later section.

MY SECOND RULE – ATTACK IT IN DETAIL

The second rule is derived from my early experience programming in ALGOL and PL/I, and reinforced by the influence of *Software Tools*. This is to decompose every problem as completely as possible into its major and minor elements. Kernighan and Plauger introduce the concept of programs as filters that typically accept a single input and create a single output. They emphasize doing just one thing and doing it well. Higher level programs then string these together feeding the output of one to the input of the next. This concept is familiar to anyone who works in the UNIX environment or has access to POSIX compliant tools anywhere.

BOTTOM UP DEVELOPMENT

With a collection of such tools one may often assemble simple applications by little more than simply threading a selection of them together to transform an original input progressively to the desired result. This relies on the fundamental theorem, that if each component program returns a correct output for every input in its domain and rejects any input not in its domain, and if the range of one such component is entirely within the domain of a second, then piping the output of the first to the input of the second will also return a correct result

TOP DOWN DEVELOPMENT

Practical applications are usually too complex for treatment by such simple methods. We usually find ourselves with data in one hand and a desired result in the other, with a maze of unknown paths connecting the two and little clue to the path that does so correctly. Competing paths often require us to find one that is somehow better than others. I begin by decomposing the problem systematically into its component steps, first into major steps and then each of those in further detail seeking the most elemental sequence possible. An incidental benefit of this approach is that many of these elemental processes are much the same in function if not in detail from one program to another. This lends a common structure that can be set up in template form for reuse, enabling me to focus on each element independently of the others.

AN EXAMPLE

Four to five years ago I was developing a system of reports for many different units of a company, supporting Finance, Sales, Collections, and multiple groups in Operations, among others. I needed to respond concurrently and rapidly to the requirements of them all and so was motivated to design as much

as possible for parallel structure and reusable code. This led me to design each report as a distinct subsystem with three major procedural components, “Initialize,” “Main,” and “Terminate,” reflecting the structure I first saw expressed in just that way in SAS/AF®, which was in fact the vehicle for the system. In my present engagement I am again writing similar report programs as separate systems, that I plan ultimately to incorporate into a single structure with a SAS/Intrnet® user interface.

“Initialize” comprises all preparatory steps performed once in the course of a project or report. This in turn has distinct steps, which I gave the same names in each subsystem:

- a. Establish server communication.
- b. Accept control parameters.
- c. Derive additional parameters that depend on the input.
- d. Assign file and library names.
- e. Set options.
- f. Create formats.
- g. Compile macros not already catalogued in usable form.

Today control parameter entry is usually interactive using SAS/Intrnet, SAS/AF, or DATA step windows to communicate with an application on a local workstation or a server. For many, however, particularly if they are long running and on a mainframe or server, it is appropriate to read a control file and free the user’s console. At the OCC analyses are typically performed on quarterly data with output to files whose names incorporate a form of the quarter date. My “Initialize” therefore usually accepts the quarter of interest as input and uses SAS date functions to represent it in forms needed in file and folder names. I prefer to use a null DATA step for this in order to use the full capacity of its language and functions to parse parameter strings, construct names and tokens, and assign them to library and file names and global macro variables. In passing, I might mention that I gave myself problems until I moved “set options” to follow “set parameters” and “assign library names”, because AUTOCALL libraries were not yet defined in the first version.

“Main” for each report comprises just two major steps: “Get Data” and “Generate Reports” as separate frames in the SAS/AF system. In open SAS code I invoke them with %INCLUDE statements. I think that SAS ETL® (Extract, Transform, Load) technology reflects my notion of “get the data” as a distinct phase of an application, although in name only since it properly alludes to SAS facilities for performing the function across platforms.

In this concept, “Get Data” generally comprises the steps of extracting data from a DBMS or data warehouse, transforming it in steps such as calculating aggregates, statistics, and ratios, and storing it in working tables for use by “Generate Reports.” If the underlying data do not change these working tables may persist through multiple runs of the reporting process; my “Get Data” module is therefore designed as a complete process incorporating everything up to the point of selecting subsets structured for particular reports. This of course glosses over the issue of data capture into the repository, which may take any of several forms depending on the source and nature of the data and the capture mechanism. In systems with a variety of subtasks, as in the reporting system example above, I do the same with the subordinate elements. As the level of detail increases the steps become more specific to the particular problem.

Finally, “Terminate” is used to clean up after each subsystem to restore the initial state, ready for invocation of another request. It may not be necessary in all cases, particularly if each task is executed independently with the entire system closed upon completion.

DON’T DO IT YOURSELF IF SOMEONE ELSE ALREADY HAS

We can travel much farther and faster by learning from the experience of others, and even copying their work if we are not violating their intellectual property rights. From its earliest availability, or very nearly so,

SAS itself has included in one or another form a sample library of completed examples, some immediately useful, others less obviously so or merely amusing. Even so, if one is adept at analogy even the latter can serve as models for us to apply to the solution of our own problems. This library has expanded together with SAS and SAS Institute as more products have been introduced to address specific areas of application.

Beyond the sample libraries that come with SAS we have the resources of SUGI, and regional and industry specific user groups. The annual conferences offer a wealth of experience that can be applied directly or extended to our own problems, and often include code that like the sample libraries can be appropriated to our needs. I consider the Hands On Workshops to be among the most informative sessions available and try either to attend or review and work through at home every one offered at each conference. Also, some of our colleagues are recognized as particularly expert in one or another aspect of SAS. Wherever I am engaged I recommend to those about to attend their first SUGI or regional SUG conference that they hear Dan Bruns and Ray Pass on PROC TABULATE and PROC REPORT, and Lauren Haworth on ODS, to name just the first ones that come to mind. The availability of conference Proceedings in machine sensible and searchable form on line and on CD-ROM has greatly increased the value of this resource. I think it would be useful too if earlier Proceedings now available only on paper were one day to be scanned and made available in the same form.

Many of these recognized contributors have also written Books By Users treating topics both general and specific, which again usually include code examples that can be readily found on line at SAS Support and profitably adapted to our own purposes. Again naming but a couple of the most prominent, I especially collect books by Art Carpenter and Ron Cody, as well as most compilations I encounter of SAS tips and tricks. An extensive library of such works of course is expensive and can be difficult to justify personally or to one's employer. In my own case, as a contract developer going from one job to another, my stock in trade is less what I know than it is my ability to discover quickly what I need to know and put it to productive use, so I need good information ready to hand. Remember too that one need not buy the book to download the code examples from SAS Support.

I do not intend turn this paper into a commercial for SAS Press but cite these resources as places where one can discover whether someone has already solved the problem one is about to tackle. Another such resource is available on line at SAS-L and in the pages of consultants worldwide, both individual and corporate. Many of these offer collections of utilities and simple applications, often free of charge and very often useful in one or another context. To avoid the appearance of commercialism I will not name them here, but my bookmarks file is full of their URLs, which I will willingly share on request but without offering reviews.

Finally under this heading do not overlook the SAS documentation itself as a source not only of information on SAS facilities but also of models in the short examples provided. In some simple cases such an example may be sufficient to establish the framework of one's own invocation of a PROC step.

MAKE THE MACHINE DO THE WORK

As preface I would like to mention an experience of my early days as SAS site representative at the Federal Reserve Board. During the first several months I was effectively the sole source of SAS information of any kind, having been to a SAS basics course and initially possessing the only set of SAS manuals. It clearly demonstrated that in the valley of the blind the one eyed man is king. One of my duties was to prepare and conduct SAS in house training. My audience included two distinct types of person: programmers from my own division already proficient in COBOL, FORTRAN, and PL/I, and statisticians and research assistants from the line divisions for whom programming was a secondary activity. I soon observed that the former were quick to seize upon the DATA step as a new programming language, writing code to perform the same calculations they had done in the procedural languages, while the latter tended to think of the PROC steps as functions and focus directly on them to achieve their desired results. The second is of course a more effective use of SAS and offers an example of what I mean by

making the machine do the work, although today I am sure the culture has changed enough that the particular problem would no longer exist.

The SAS Enterprise Guide®, introduced comparatively recently as a standalone product but now a Base SAS component in version 9, strikes me as perhaps the most important tool available in support of my stepwise view of program development. It provides an integrated development environment to relate data to task and provide views of program and data flow. I confess that while I have extensive experience with such tools in other contexts, I do not yet fully avail myself of it. My only excuse for this is the proficiency I have developed with my earlier tools; established habits are difficult to change. I fully intend to improve my practice in this regard, and recommend that those less set in their ways than I take full advantage of this tool.

I do use and recommend the tools provided with earlier SAS versions, all of which are now represented or have equivalents or better in the SAS Enterprise Guide. Some of these are provided with all SAS installations and others are specific to particular SAS products. They are available through the Display Manager Tools and Solutions drop down menus and include the SQL Query, Table, Report, and other Editor windows under Tools, and ASSIST, Analysis and Reporting facilities under Solutions. There are also several collections of SAS code templates available. Their particular advantage is in convenient generation of syntactically correct code, working from a display of the variable lists of the actual SAS tables on which it is to work. I reiterate that the SAS Enterprise Guide incorporates such function and more.

Despite the function afforded by such tools occasions will arise when one must pick apart existing SAS code to understand its function either to modify it or to use it as a model for a new project. For such purposes I have found it useful to turn to POSIX compliant tools, which can be found in both commercial and open source versions, or whose function can also be effectively emulated in SAS DATA steps using the wide variety of functions provided. Here familiarity with sources such as Kernighan and Plauger is very helpful. To me the choice of one's editor is highly personal. The SAS Enhanced Editor has good facilities for searching, sorting, and manipulation, and enhances readability by color coding keywords and the opening and closing of quotes, parentheses and comments. At the time it and the Display Manager were being developed I was using the XEDIT editor of the VM/CMS operating system, so when I moved from IBM mainframes to the PC I chose a commercial direct copy of XEDIT that offered still more extensive facilities, particularly block cut and paste operations. I have found them especially useful in hacking an application, to run PROC CONTENTS with its output to a file that I export to an spreadsheet, then block copy the names into the new code I am writing as a KEEP list, or the appropriate columns into an ATTRIB statement. Several such editors are available, including shareware and open source. Another tool that I have found very useful, especially when I am looking for common code fragments to make into macros, is a commercial file comparison utility that has a visual interface and does much more than the POSIX compliant DIFF utility.

One tool that I consider potentially very useful but that I have not found is a good SAS code formatter, or pretty print program. In the days when file names and SAS variables were limited to eight characters I found that I could make the freeware SNAP! utility serve the purpose very well. This was written to format dBase and Clipper code and generate cross references, but also permitted one to supply one's own list of keywords. This enabled it to work effectively on SAS code as well. Unfortunately it was never enhanced to support long names. To write one's own such program is a standard exercise but I have so far been too lazy to do so. If anyone knows where such a tool is available I would like to hear of it.

MY STEPS IN PROGRAM DEVELOPMENT

GET IT RUNNING

This and the following step may seem redundant but in my view they are not. My object at this point is to have code for each module that is syntactically correct and will run to completion, with no error messages.

I usually but not always test that the entire process also flows from beginning to end, but most often look no further than to have the “Get data” and “Generate Reports” sections each run in this fashion before continuing. Clearly, report generation cannot be expected to run correctly until it has correct input available, so I most often continue the “Get Data” process through correctness testing before devoting effort to report generation.

GET IT RUNNING CORRECTLY

This development step does focus on ensuring that each component and then the entire program does indeed produce correct results under all conditions, testing the output values for correctness. Here is where I devote serious effort to the layout of the output to ensure that it meets the specification if there is one, or applying a critical eye if I have been given a free hand in its design.

GET IT RUNNING EFFICIENTLY

Only after I am sure that I have a process that runs completely and correctly in a “push the button and stand back” mode do I direct any real effort to making the code efficient in the sense of eliminating unnecessary or redundant steps. I also seek opportunities to introduce macros and common execution blocks where they will improve performance or clarity. I do try to use good coding practice throughout and to be alert to recognize where I can use the same code, written once and parameterized, to perform the same function in more than one context. I have also accumulated a library of code fragments over the year that I can reuse with little change.

MAKE IT ELEGANT

Once the preceding three steps are accomplished the application is essentially ready for production use; however, wherever possible I try as a final step to go over it carefully seeking to eliminate awkward constructions and to identify those elements that can be written as macros. This is done not only to polish the application but to make it more comprehensible when it must be revised to meet changed requirements or be used as a model for another application. Very often when I first develop an application that includes several similar or related steps I will write each independently of the others. Then after completion I may discover that they are very similar so that with some thought I can make one module can serve in more than one place, perhaps through effective use of macros or global variables.

%MACROS AND %INCLUDE FILES

I have already mentioned how I decompose an application into steps that I write separately and invoke most often as %INCLUDE files. It is a common practice to write an application as a SAS macro, invoked with parameters to perform a common task on various data. This is an area where there is no one correct approach, and a matter of personal taste or local standard; however, my preference is to write plain SAS code rather than macros for such complete applications. In the first place, I find macros that contain their own extensive logic hard to read because the percent signs interfere with my reading scan. Also, it is not easy to shift mental gears between following the logic of the macro, effective during code generation, and that of the application itself, which occurs after compilation. My inclination therefore is to write straight SAS code, parts of which may be common expressions and statements or sequences that I insert into DATA and PROC steps as macros, with or without parameters. Such parameters often accommodate variations in code logic rather than apply the same code to different data. I also modify code using the full power of the SAS language and functions in a null DATA step to create statements as strings written by CALL SYMPUT to global macro variables and then used in a subsequent DATA or PROC step by inserting the macro variable for the statement.

I use macros extensively for lengthy variable lists, even for a single use, in order to make my code more compact and improve readability. I do this rather than assign the list to a global variable because the latter will include all spaces and returns while the macro processor will compress it. In this way I make the

string easy to read and modify in its extended form in my editor but it does not make the log difficult to read if the macro is displayed with MPRINT.

Finally, I use macros where I want to take advantage of iterative control structures. Examples of this occur in my present engagement. The group I serve is concerned with analysis and reporting of the condition of the banking system, based on data received in the quarterly Call Report required of all banks. The quarter is thus the natural unit for most of the applications, whether an analysis spans several quarters or the most recent is being reported. I have found it effective to design an extraction or a report for a single quarter and provide for its repeated execution over the required quarters in a macro %DO ... %WHILE loop that iterates from the first quarter through the last, invoking the application as an %INCLUDE file.

MY STYLE CHOICES

My own style, that suits me but is only one choice among many alternatives, goes beyond simple rules for indentation, semicolons, and white space to such seeming trivia as writing sequences of statements, no more than one per line, so that not only do equal signs line up vertically and with the right hand side expressions aligned to the right of the equal sign when they span multiple lines. Similarly, when such sequences are of similar form, for instance a sequence of assignments involving the same function I also align corresponding elements of the expression. My purpose in so doing is mainly to take advantage of the pattern recognition ability of the human eye, so I can run my eye down a page of code and see breaks in the pattern that indicate an change of logic or possible error.

I seek an effective compromise in my style of spacing between providing white space for clarity and clustering closely statements so their proximity illustrates the relationship. Some people double space all their code but I find that impairs my ability to grasp sequences that form a logical whole.

I start global statements, particularly DATA and PROC, in the leftmost column and indent all subordinate statements through the next RUN by at least two columns and by the same amount for all statements at the same logical level, and indenting further for subordinate elements such as file options or DO; ... END; and SELECT ... END; statement groups. I may not always use the same indentation levels in different applications but I always maintain the same style throughout a given application. In IF ... THEN; ELSE ...: statements it seems to me most logical that THEN and ELSE be aligned. I try to transform cascading IF ...; THEN ...; ELSE IF ...; sequences into SELECT ...; END; structures wherever possible, for clarity of the relationship between THEN and ELSE but also because over the years I have never been able to satisfy myself as to the proper indentation style for the cascade, which tends to run off to the right of the page.

I have mentioned that I often use macros as containers for sequences like KEEP lists. When I do not, I usually put each list item on its own line separate from keywords, parentheses, and closing semicolons. This makes it easy to edit the list if I find it necessary, inserting and deleting elements without having to change the layout of my code.

I put short statements on a single line with their semicolons at the end. If they span multiple lines I put the semicolon in the same column as the first character of the statement, however many lines higher it may be. This helps me to keep track of the scope of a block of code. In spite of my desire to keep things short it is often necessary that my code span many pages or screens, particularly if KEEP lists or SQL SELECT clauses involving several hundred banking variables are present. If my eye cannot span them then with my editor I can highlight a block one character wide from start to finish to check. I have not found the practice, suggested by some, of putting all semicolons in the first column to be particularly helpful in preventing dropped semicolons, but that it again interferes with readability. In fact, I have yet to find an absolutely certain human way to avoid missing semicolons in all cases.

CONCLUSION

In summary, my approach to SAS application development reduces to a very few simple rules, most of which reflect application of principles that have been around in one form or another for a good forty years. These, restated briefly, are:

1. Establish and conform to consistent standards.
2. Break the problem down into the smallest possible steps.
3. Have each step do one thing, and do it completely and correctly.
4. Do the same things the same way.
5. Wherever possible design each step to be comprehended in a glance.
6. Profit from the work of others.
7. Know your tools and learn to use them effectively.

REFERENCES:

Kernighan, Brian, and Plauger, P.J., *Software Tools*, Addison-Wesley, Boston, 1976

ACKNOWLEDGMENTS

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John W. Davison, Jr.
Factotum, Inc.
15265 Montresor Road
Leesburg VA 20176-5811
Work Phone: (202) 874-4586
Fax: (301) 433-6618
Email: johndavison@earthlink.net (h)
johndavison@occ.treas.gov (w)