

Techniques for Creating Reviewer-Friendly SAS® Programs

Mario Widel, Eli Lilly & Company, Indianapolis, IN
Jay Zhou, Amylin Pharmaceuticals, Inc., San Diego, CA

ABSTRACT

Based on the recent regulatory guidelines, FDA has been asking to include SAS program files in submissions that are executable in a PC environment. These submitted programs would save FDA statistical reviewers' time to understand how data are derived and analyses are conducted. Reviewer-friendly programs will likely speed up the review process. This paper will present several practical techniques and some examples to discuss approaches to create reviewer-friendly programs.

INTRODUCTION

In the pharmaceutical industry, a large portion of the SAS programming community is devoted to the production of Tables, Figures and Listings (TFLs) that will be somehow incorporated into Clinical Study Reports for regulatory purposes. The TFLs will also be the supporting evidence to conference posters and publication manuscripts not necessarily regulated. Whichever is the case, programmers need to render outputs with some desirable properties. Besides correctness and readability of the outputs, validation and/or verification of code has become a crucial step in the process. Vachal (2001) made a compelling case about making the review process easier on the reviewer.

When designing TFLs, a notable consideration is given to the intended audience, being the ultimate reader of the final report or manuscript, but before the output can actually be published the programs must have been validated. At this step, we could consider the individuals involved in the review process as the targeted audience, so why not make their life as easy as possible and, in doing so, most likely speed up the review process. This paper will review several practical techniques with some examples to discuss approaches on how to create reviewer-friendly programs.

CODING TECHNIQUES

What makes a program easier to review? Stylistic considerations apply. Readability would be a very (if not the most) important one. Naturally readability is just the name of a broader concept. A readable program implies maintainability and reliability and it is a component of efficiency if we accept the definition that an efficient program is the one that consumes less resource. Carpenter and Payne (1996) presented a deep and amusing set of tips that every SAS programmer should read it, and you may not like it but do not be surprised if you discover yourself using some of these practices.

STRUCTURED PROGRAMMING TECHNIQUES

The structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure. It will inherently provide a better frame to follow the logic built in the program. Therefore, readability is improved in terms of easier understanding of the intended objectives. Modularity, not to be confused with programming style, where a module would have ideally a single input, a single output and a single purpose plays an important role in both readability and maintainability (Glaser 2002).

Consistently adopting coding standards is another good practice. For instance, each `DATA` or `PROC` step should be ended with a `RUN` or `QUIT` statement, which provides a boundary so that each code block can be executed independently when doing spot-checking. However, a data step that spans three pages can very likely be split into logical parts.

Whenever possible, one should avoid the use of SAS defaults like `_LAST_` data set or automatic conversions. It could generate unexpected results when the code is executed partially.

At the end of the program or at strategic points, it is a good practice to use `PROC DATASETS` to delete unneeded data sets from the work library. This not only will improve performance, but more importantly will show the intention to the reader as well.

Rhodes (2004) recommended some practical suggestions about standards and style sheets. The paper showed examples of implementing a database of coding standards, styles, testing and review checklist that certainly could accelerate the peer review process. Another set of amusing good habits, not necessarily oriented to style, can be found in Mitchell (2005).

Dilorio (1996) suggested to avoid data steps with an excessive number of statements as well as excessive nesting of code. Unnecessary data steps should be avoided as well. Macros that call macros that define macros that call macros could be a nightmare even for the author of the program and sometimes it is hard to justify.

Choosing a meaningful variable or data set name that is self-explanatory will help the author and reviewer to easily follow the steps. It is important to keep the naming convention consistent across data sets and programs. From SAS version 7 or above, long names are allowed, which certainly increase the set of possible meaningful names for a variable. But one should not get carried away, names too long can become a drag for typing and increase the chance of typos, also may create transportability issues. For those data sets used for regulatory submissions, the length of variable names should be kept up to 8 characters. Otherwise, it becomes an obstacle when making SAS V5 transport files. Labels for variables and data sets provide reviewers further clarification information, which should not be omitted. To avoid automatic creation of variable attributes, use `ATTRIB` or `LENGTH` to declare new variables.

Caution should be taken when attaching formats to permanent data sets. Formats are a wonderful tool for consistency, efficiency and perhaps clarity as well, but they could cause potential problems if the data is reviewed without format catalogs supplied. Logical variables should have values like 'Y', 'N' instead of 0 and 1. For instance, having a variable `GENDER` with values 'F' and 'M' avoids the need of a format.

APPEARANCE

There are different ways to improve the readability through appearance as suggested by Levin (2003). Fecht and Stewart (2002) mentioned to avoid "wordy" coding and other suggestions to reduce programmer intervention. Winn (2004) addressed naming, indentation, documentation, reusability and efficiency considerations with examples. For example, inserting blank lines to separate data and proc steps will make a program more readable. One should use one statement per line, but split the statement into multiple lines by data step options with the statement too long to be viewed on monitors. Limiting the line space ensures that the program will not exceed the number of characters per line printable on network printers.

Another common way to gain readability is to indent SAS statements consistently throughout a program. While there is no standard for the number of columns to indent, at least 2 columns should be used as a minimum. It would be more useful something like "Every nesting level should be visibly indented from the previous level". For instance, groups of SAS codes, like `DO` and `END`, `IF` and `ELSE`, etc., should be indented to the same level to show logic and improve the readability of the code. Let's provide an example to illustrate what we have discussed:

```
PROC SORT DATA=DEMO(KEEP=PATIENT ITT AGE RACE GENDER BIRTHDAY FIRSTDOSE
LASTDOSE BMI) OUT=DEMO; BY PATIENT; RUN;
```

To improve the readability:

```
PROC SORT DATA=DEMO(KEEP=PATIENT ITT AGE RACE GENDER BIRTHDAY
                    FIRSTDOSE LASTDOSE BMI) OUT=DEMO;
    BY PATIENT;
RUN;
```

USING PROGRAM HEADER FOR DOCUMENTATION PURPOSES

Fehd (2005) explained the importance of documentation, in particular headers, by providing a computation of the cost of SAS code to address the importance of program header when recycling code. Hord and Zhou (2002) provided the methodology programmatically to automate the documentation process by utilizing the information provided in the standard program header.

The basic elements documented in the header should provide an idea of what the program is about just by a single glance (well... perhaps a good glance) in addition to the formal information required by the rules of the site. Very frequently headers contain a section for revisions, having a description of what is being changed and why, it would be convenient for the reviewer to have as well a reference where in the code the changes were actually applied. For example,

```
* REVISION HISTORY
-----
BY      REF   DATE      REASON
MW      #17   20061010  PER AMENDMENT XX REMOVE PATIENTS WITH BMI>35
-----;
```

The searchable reference number #17, cited in the code where the corresponding changed statements are located, would appear:

```
DATA PATIENTS;
  SET DEMO(WHERE=(BMI<=35)); *** #17 ***;
  ...
```

DOCUMENTATION AND COMMENTS

Self-documenting programs should be the preferred choice. A self-documenting program contains enough information embedded in the code for a programmer to easily understand the program's purpose, logic and outcome. Good self-documentation will reduce the need of comments, even though it will not completely go away. Some examples and definitions can be found in Ma (1996).

Comments should be provided throughout the program and appear at any stage when new logic is being applied. Typically, comments should appear before each major block of code, especially for complicated code blocks. In general, comments should be used to increase a program's readability and maintainability. However, unnecessary comments may become an enormous annoyance for the reviewer.

There are three styles to make comments in SAS:

- (a) `*** comment ***;`
- (b) `/** comment **/`, and
- (c) `%**comment ***;`

For non-macro programs, Style (a) should be used in preference to (b). This gives Style (b) of comments for commenting out large sections of code during program testing and revision. For macro programs, Style (c) should be employed in preference to Style (a) unless one wishes the comments to be visible in the LOG window when the MPRINT option is used. Calls to macros can be commented out with Style (c), by placing an asterisk after the '%' sign in the invocation and a semicolon after the closing parenthesis, e.g.,
`%*report(...);`

For convenience, programmers tend to place comments Style (b) to exclude unwanted variables from the middle of a SAS statement. For example, the BIRTHDAY variable is excluded from the keep= option by placing Style (b):

```
PROC SORT DATA=DEMO(KEEP=PATIENT ITT AGE RACE GENDER /*BIRTHDAY*/
  FIRSTDOSE LASTDOSE BMI) OUT=DEMO;
  BY PATIENT;
RUN;
```

This will result in a problem when using Style (b) to block out the entire above PROC step since SAS does not support nesting of comments.

USING SIMPLE DATA STEPS.

Even at the price of sacrificing some CPU efficiency and the additional cost of increasing the number of data steps, in general, not only the intended purpose but also the correctness of the logic of several simple data steps will be understood faster than if a single complex one were used. In other words, one should avoid

overly complex data steps, but at the same time avoid unnecessary data steps. If tasks logically belong together, then they should be programmed on the same data step or the steps following each other.

Before designing and writing a program, think of the end users or final audience. Smart or clever coding is welcome, but please provide enough comments to ensure that other people will understand. Using statement or data set options (e.g., `KEEP=`, `DROP=`, `IN=`, `WHERE=`, ...), whenever possible, will help to clarify reviewer's intentions. In particular, when identifying variables required the use of `KEEP` statement or "`(KEEP=)`" data set option should be used in preference to the `DROP` statement or `DROP=` option, the exact importance of the variables in the resulting data set is then clear. However, it is sometimes easier to name the few variables that are to be dropped from a data set (for example removing temporary variables required during the data step).

The `OUT=` option should be always used when sorting a permanent dataset to avoid overwriting that source dataset, especially when the `WHERE` statement or `WHERE=` option is used.

TESTING AND VALIDATION

Bentley (2005) stated well-founded reasons for doing formal testing on SAS programs. Clearly, reviewer friendly programs will help on the testing process. Sharlin (2005) also made a case on how the FDA approval time can be affected by validation issues. For debugging purpose, Delwiche and Slaughter (2003) provided some rules that would also help the review process by avoiding (whenever possible): warnings, uninitialized variables, automatic generation of missing values, and automatic conversion of data type. Howard and Gayari (2000) and Howard (2003) gave definition and clarification of validation terminology and of the System Development Life Cycle, good practices thought for the validation process that certainly facilitates reviews.

MACRO-FREE CODE

SAS Macro is a miraculous tool that gives you programming power and flexibility. As programmers, we cannot live without it. However, when reading SAS source code, it can become a serious burden to those readers who have less macro knowledge.

There are many ways to generate Macro-free code. Troxell (2001) mentioned briefly several ways to create macro free programs. Although not for the specific purpose of creating macro-free programs, Gau (2004) and Reading (2001) showed the method of generating SAS programs with another SAS programs. Beakley and McCoy (2004) shared more techniques for generating dynamic SAS code. Johnson (2004) presented another set of techniques to write SAS programs that can use macros and still write SAS programs that truly can be macro free. That is, still taking advantage of the iterative power and versatility of macros, to "generate" source code that eventually will be run to produce the desired output.

Edgington and Zhou (2002) took a different approach to produce macro-free SAS programs. Their method employs two SAS system options, `MFILE` and `MPRINT`, to save the resolved source code into an external ASCII file which will be later post-processed by a macro to create a macro-free program. To preserve the readability of the regenerated, macro-free program, standard programming practice has been built into the macro intelligently. The advantage of this approach is to add no restriction to the original programs. But the step of re-testing the regenerated program is needed to ensure it works as expected.

Naturally, writing and validating programs that create new programs will not be trivial. The ideal approach could be based on a set of validated modular macros that generate the target code in combination with some specific steps that include the use of `DATA _NULL_` and `PUT` statements. For standard tables and listings, it is feasible to develop a system to generate macro-free programs. But it will be challenging when dealing with more customized reports.

DISCUSSION

All the ideas have been discussed in the paper in the hope that it will spawn further discussions of this important topic and standardize the coding style to facilitate the cost-effective review process. More intelligent programming editing tools could be developed to aid the development of reviewer-friendly SAS programs.

REFERENCES

- Beakley, Steven D. and McCoy, Suzanne D. (2004). "Dynamic SAS Programming Techniques, or How NOT to Create Job Security". *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*. Paper 78-29.
- Bentley, John E. (2005). "Software Testing Fundamentals – Concepts, Roles and Terminology". *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Paper 141-30.
- Carpenter, Arthur L. and Payne Tony (1996). "Programming for Job Security Revisited". *Proceedings of the Twenty-first Annual SAS Users Group International Conference*. Paper 275-23.
- Delwiche, Lora and Slaughter, Susan (2003). "Errors, Warnings, and Notes (Oh My). A Practical Guide to Debugging SAS Programs". *Proceedings of the Twenty-eighth Annual SAS Users Group International Conference*. Paper 58-28.
- Dilorio, Frank C. (1996). "The Elements of SAS Programming Style". *Proceedings of the Twenty-first Annual SAS Users Group International Conference*. Paper 59-21.
- Edgington, Jim and Zhou, Jay (2002). "Post-Processing MPRINT Outputs to Generate Macro-Free Code". *PharmaSUG2002*. Paper TT14.
- Fecht, Marge and Stewart, Larry (2002). "Don't be a Slave to Your SAS Programs". *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*. Paper 65-27.
- Fehd, Ronald (2005). "Journeymen's Tools: The Writing for Reading and Reuse Program Header". *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Paper 67-30.
- Gau, Linda (2004). "Write SAS Code to Generate Another SAS Program. A Dynamic Way to Get Your Data into SAS". *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*. Paper 175-29.
- Glaser, Allan MS (2002). "A Set of Quality Criteria for Statistical Programming", *Drug Information Journal*, Vol. 36, pp 565-570.
- Graebner, Robert W. (1998). "Generating SAS source Code with SAS Macros". *Proceedings of the Twenty-third Annual SAS Users Group International Conference*. Paper 81-23.
- Hord, Chris and Zhou, Jay (2002). "SAS® Macros to Help Relieve Common Program Documentation Pain". *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*. Paper 208-27.
- Howard, Neil (2003). "Beyond Debugging: Program Validation". *Proceedings of the Twenty-eighth Annual SAS Users Group International Conference*. Paper 58-28.
- Howard, Neil and Gayari, Michelle (2000), "Validation, SAS and the Systems Development Life Cycle: An Oxymoron?". *PharmaSUG*
- Johnson, Jim (2004). "Programming Squared (Writing Programs that Write Programs)". *PharmaSUG04*. Paper HW05.
- Levin, Lois (2003). "SAS Programming Conventions". *Proceedings of the Twenty-eighth Annual SAS Users Group International Conference*. Paper 241-28.
- Ma, J. Meimei (1996). "Self-documenting Programs and Data Sets". *Proceedings of the Twenty-first Annual SAS Users Group International Conference*. Paper 60-21.
- Mitchell, Rick M. (2005). "Fast and Easy Ways to Annoy the Job Security Specialist". *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Paper 148-30.
- Reading, Pamela L. (2001). "Using SAS to Write SAS – Automate Your Programming Tasks". *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference*. Paper 92-26.

Rhodes, Dianne Louise (2004). "Programming Standards, Style Sheets, and Peer Reviews: A Practical Guide". *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*. Paper 135-29.

Sharlin, Joshua (2005). "Using an Automated Flowcharting Tool for SAS Programs to Improve Part 11 Compliance and Reduce FDA Approval Time". FOI Services

Troxell, John K. (2001). "Complex Macros v. Simple Programs: Resolving the Conflict". *PharmaSUG2001*. Paper TT05.

Vachal, Rich (2001). "FDA Reviewers as Ultimate Users: Using the SAS System to Construct e-Submissions that Actually Facilitate the NDA/BLA Review Process". *PharmaSug2001*. Paper FDA09.

Winn, Thomas J. Jr (2004). "Guidelines for Coding SAS Programs". *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference*. Paper 258-29.

CONTACT INFORMATION

Mario Widel
Eli Lilly and Company,
Indianapolis, IN
(317) 433-3556
Email: widel_mario@lilly.com

Jay Zhou
Amylin Pharmaceuticals,
La Jolla, CA
Email: Jay.Zhou@amylin.com