Paper #1138-2017

User-Written vs. System-Generated SAS® Source Code

Is System-Generated Code a Boon or Bane to Programmers?

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

The traditional model of SAS® source code production is for all code to be directly written by users or indirectly written, i.e., generated by user-written macros, Lua code, or via DATA steps. This model was recently extended to allow SAS macro code to operate on arbitrary text (e.g., html) via PROC STREAM. In contrast, the modern SAS system includes many products that operate in the client-server environment and function as follows: 1) the user interacts with the product via a GUI to specify the processing desired, 2) the product saves the user-specifications in metadata and generates SAS source code for the target processing, 3) the source code is then run (per user directions) to perform the processing. Many of these products give users the ability to modify the generated code and/or insert their own user-written code. Also, the target code (system generated plus optional user-written) can be exported or deployed to be run as a stored process, in batch, or in another SAS environment. Here we review the SAS ecosystem contexts where source code is produced, the pros and cons of each approach, discuss why some system-generated code is inelegant, and make some suggestions for determining when to write the code manually, and when/how to use system-generated code.

SAS ECOSYSTEM IS LARGE & DIVERSE

Context is a constraint: the SAS® Ecosystem is very large, literally hundreds of vertically and horizontally integrated software products. Many of the software products generate and run SAS source code as part of their functionality. These products function within a context – an application or application area - and they may differ significantly in how they are used in practice.

For example, one code generating product may have a focus on generating code that might be run in batch (or as stored processes), while another product updates a transaction-type database, with significant interactive functionality and limited batch processing. Comments that are relevant to one code-generating product might not apply to or may be less relevant to a different product.

No one person has used **all** of the 200+ SAS software products, so the discussion here is constrained by the author's experience, i.e., this writer has direct experience with:

- SAS® Enterprise Guide®
- SAS® Data Integration Studio®
- SAS running in batch
- Windowing SAS aka "PC SAS"
- SAS Studio

and the discussion here reflects the author's experience with these products. Those who have used SAS products that have a large interactive component, e.g., SAS/AML: Anti-Money Laundering, may have different experiences/perspectives than are provided by the products above.

BACKGROUND

System-generated code is source code that is produced by a computer program, often based on inputs provided by a user/programmer. The concept is not new in software; see e.g., Roth (1982), an ACM AFIPS Conference paper for early examples. Czarnecki (2004) is a more current related reference on *generative software development*, a type of *automatic programming*.

In the SAS ecosystem, the tools that produce system-generated source code run in the client-server environment. Once source code has been generated by a SAS tool, it can be run in any of the supported SAS environments (batch, widowing, client-server), though significant code changes may be required when exporting such code to batch or windowing environments; see Billings (2015) for a discussion of this point. User-written source code can be developed in one SAS environment and ported to and run in another, with the porting effort required varying based on coding techniques.

Most of the SAS tools that generate code are based on a GUI and utilize metadata. The user interacts with a GUI to specify the desired processing to be performed. The options specified by the user are saved in metadata and used to generate SAS source code to perform the target processing. The code may be run multiple times during development, and can be exported to a stored process or as source code in a text file. Most of these tools allow the user to insert code into the generated code, and/or modify the generated code to varying degrees.

The process of using a GUI to produce system-generated code is an extension of the principles behind user-written macros. A programmer would specify options via macro parameters in a macro call, with the end result being the production of SAS source code to accomplish a target task. User-inputs provided via a GUI serve the same function: the tool produces SAS source code to accomplish a target task.

As mentioned above, system-generated code can be produced and exported into a program text file. That code can then be inserted manually into other system-generated code. This nesting of user & system-generated code can be multiple-levels deep, though such complexity should be avoided due to the difficulty in maintaining such complex code.

The SAS ecosystem environments and programming in them are illustrated in figures 1 & 2.

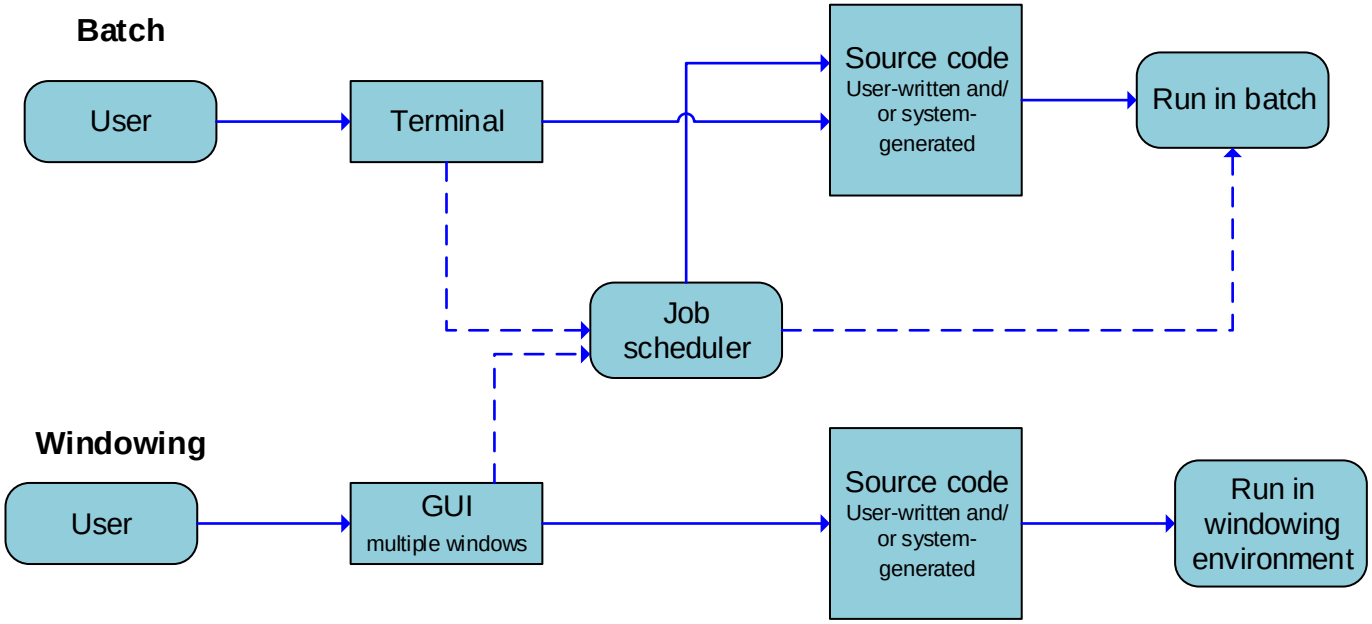

Figure 1: SAS Batch & Windowing Environments, programs and source code

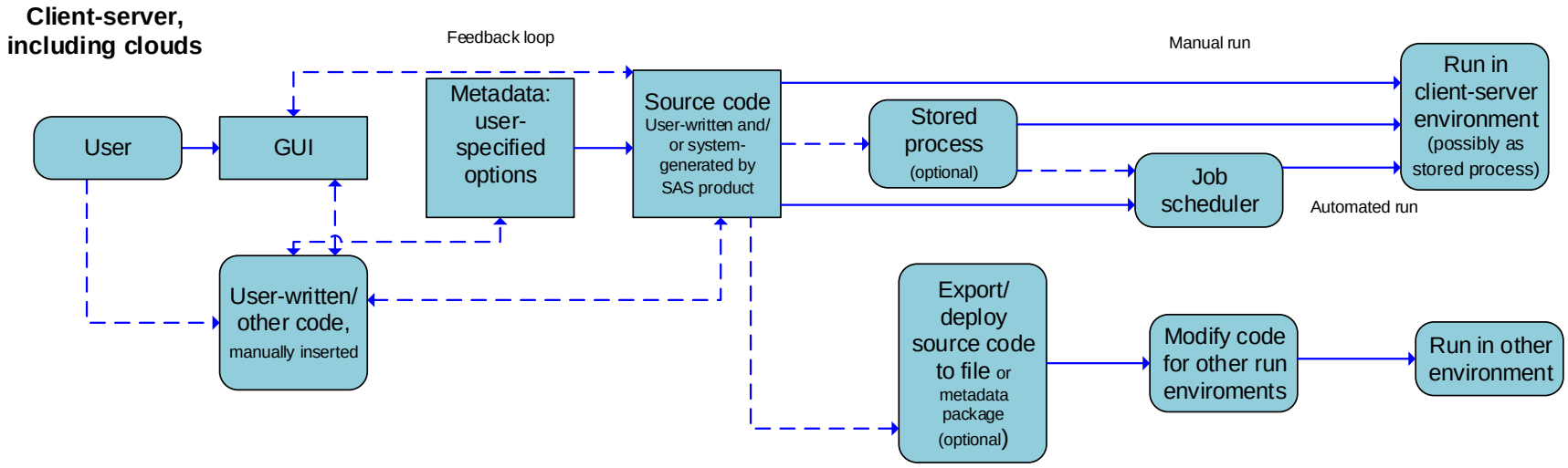

Figure 2: SAS Client-Server Environment, programs and source code

PROS & CONS, USER-WRITTEN VS. SYSTEM-GENERATED CODE

User-written code: PRO

- Total control by programmer
- Pride in ownership – can produce elegant *artisan code*
- Code functionality limited only by system constraints and development time/effort
- Code produced can be relatively portable
- Code produced may be relatively efficient in terms of system resources (CPU time, use of disk space)
- Code produced can meet production standards for error checking and error handling, restart after failures, etc.
- Code produced can meet the *Reproducible Research Standard* (Stodden 2009A, 2009B) and if desired can be published/released under open-source copyright licenses
- Provides opportunities for programmers to learn new techniques and methods

User-written code: CON

- Increased risk of errors, especially if written under time duress
- Writing code is labor-intensive process
- User-written code can be more expensive for the enterprise
- Development time (including testing) can be long.
- Writing code can be repetitive, even with use of macros and/or Lua
- Major changes in requirements can induce significant rework/effort
- May force a programmer to learn new techniques and methods, possibly under time duress

System-generated code: PRO

- Potential large improvement in productivity - rapid development/shorter development time:
  - SAS Enterprise Guide via GUI can produce reports & analyses in a very short time, compared to user-written code
  - SAS Enterprise Guide is a powerful and efficient tool (SQL Query Builder task) to quickly diagnose & research data problems/anomalies
- Jobs/projects can (in many cases) be modified quickly – at the metadata level - to accommodate major changes in requirements. This varies by tool, context, and magnitude of the changes. Adding or dropping a variable from a "spider-web" of SAS Enterprise guide tasks may require extensive manual change; while SAS Data Integration Studio has options to partially automate this type of metadata change.
- System-generated code can be lower cost for the enterprise
- Products based on system-generated code can provide important SAS-system-based services and functionality to users who do not have SAS programming skills (or have limited programming skills)
- System-generated code can be exported as stored process for reuse in other client-server environments
- For some products/applications, system-generated code can be promoted into production in the client-server environment [bypassing a potentially difficult conversion to batch]
- System-generated code can be enhanced – manually cleaned up and converted into macro-based code for use in other environments, e.g., batch
- System-generated code can provide learning opportunities for some programmers

System-generated code: CON

- System-generated code is generalized and for some tools can be:

  - Written by & for a computer, hence less readable than user-written code, and harder for programmers to understand
  - More complex than user-written code, increasing the risk of errors if users modify it
  - Lengthy and voluminous – indeed, may be too voluminous for manual maintenance
- Code functionality is limited by tool and GUI; supplementation with user-written code is required in some instances or contexts
- System-generated code, in some cases, may be relatively **in**efficient in terms of system resources (CPU time, use of disk space). Users have to understand the limits of the tool and may need to create jobs/projects that overcome or avoid such limitations.
- Ability to modify system-generated code varies by tool; may be limited in some cases
- Some tools have limited or partially disabled error handling, hence are unsuitable (risky) for production runs. The client-server environment is not as safe for production as batch. Error handling varies by tool – virtually none for SAS Enterprise Guide while SAS Data Integration Studio has many error-handling features; see Billings (2014) for related information.
- System-generated code often requires extensive changes (wrappers, error handling) to be run in batch production
- The copyright status of system-generated code is unclear; assume *copyright all rights reserved* by *SAS Institute, Inc.* – hence cannot be released under an open source copyright license, and cannot meet the Reproducible Research Standard.

SYSTEM-GENERATED CODE IS OFTEN "UNATTRACTVE" – WHY?

One of the most common criticisms of system-generated code is that it can be unattractive and lacks elegance. This type of code is generalized and the requirements for these code snippets may be broader than one might expect. Here are some likely requirements, suggested by reverse engineering and review of some selected system-generated code:

- The code is embedded inside a complex tool and that induces some house-keeping requirements (often implemented via macros)
- Has to work with very large datasets
- Is written by a computer, for a computer so data set, view names don't have to be meaningful to users,
- If interrupted by the user or by failures due to errors, needs to be re-runnable with minimal user intervention required
- Some tools may allow you to work with metadata for data sources instead of physical files; this lets you develop prototype programs even if the data sources don't exist yet because they are still under development. (In the case of SAS Data Integration Studio, this feature makes some generated code voluminous and cumbersome – e.g., code to append files.)

Admittedly, a good programmer can produce code that is more elegant and more efficient than system-generated code, but the cost of that code is often much higher to the enterprise.

BOON, BANE, OR STRATEGIC USE OF TOOLS THAT GENERATE SOURCE CODE?

Your experience as a programmer working with system-generated code may be pleasant and productive, or not so pleasant, depending on a number of factors and circumstances:

- Tool capabilities and functionality
- Application requirements -- interactive vs batch
- Complexity of tool and application – can the system code be modified and maintained, manually?
- Criticality of system

- Risk tolerance for the subject system in the enterprise, i.e., can delays or downtime be tolerated?
- User skills with the tool GUI vs. user programming skills

To expand on the above, consider the requirements your software must meet – not just for basic processing, but including operational, security, and use of the derived system products by downstream groups. Next, how well does the code produced by the generators, meet these requirements? You may find that the tool meets your requirements as-is or with limited code changes.

Alternately, it is possible that the code produced by the tool has to run in batch and needs wrapper code added, the error handling features of the tool should be used, and/or the architecture of the system to be developed must be carefully designed in advance to produce a robust system that can tolerate errors. The most important factors here are how well the generated code meets your requirements (tool function vs. application) and whether you have the relevant skills to use the tool to produce code that meets – or comes close to meeting – your overall requirements.

Rather than limiting work to 100% user-written or 100% system-generated code, programmers are encouraged to find the optimal mix of user-written plus system-generated code that will produce working, reliable systems that meet requirements at the lowest cost to the enterprise. In other words, both methods of code production are good, but they need to be managed and balanced to achieve the optimum state.

ACKNOWLEDGEMENTS

This paper is an expanded version of the author's notes prepared for a panel discussion on the topic at the 2016 *Western Users of SAS Software* conference in San Francisco, California. Other members of that discussion panel included:

- Jack Hamilton, Kaiser Permanente, Oakland, California
- Scott Leslie, STATisfy Analytics, San Diego, California
- Robert Springborn, Healthcare Outcomes Center, Sacramento, California.

Also, thanks to the panel discussion coordinator, Kimberly LeBouton, KJL Computing, Rossmoor, California. Thanks to the panel members and Kimberly for an interesting and thoughtful panel discussion. Any errors herein are solely the responsibility of the author.

REFERENCES

Note: all URLs quoted or cited herein were accessed in September 2016.

Billings, T. Differences in Functionality of Error Handling Features, SAS® Enterprise Guide vs. Batch. *WUSS Conference Proceedings,* 2014. URL:
http://www.wuss.org/proceedings14/13_Final_Paper_PDF.pdf

Billings, T. Enhancing the Portability and Reusability of SAS® Enterprise Guide® Projects. *WUSS Conference Proceedings,* 2015. URL:
http://wuss.org/Proceedings15/11_Final_Paper_PDF.pdf

Czarnecki, K. (2004) Generative Software Development. Software Product Lines, volume 3154 of the series Lecture Notes in Computer Science, pgs. 321-321. Springer-Verlag. URL:
http://link.springer.com/chapter/10.1007/978-3-540-28630-1_33

Stodden, Victoria (2009A). "Enabling reproducible research: open licensing for scientific innovation." *International Journal of Communications Law and Policy.* 13, 1-25.

- Abstract URL: http://ijclp.net/old_website/article.php?doc=1&issue=13_2009
- Full text on author's website: http://www.stanford.edu/~vcs/papers/Licensing08292008.pdf

Stodden, Victoria (2009B). "The legal framework for reproducible scientific research: licensing and copyright." *Computing in Science and Engineering*. 11, 35-40; 2009B.
- Abstract URL: http://www.computer.org/csdl/mags/cs/2009/01/mcs2009010035-abs.html
- Full text on author's website: http://www.stanford.edu/~vcs/papers/LFRSR12012008.pdf

Roth, R. L. (1982) . Program generators and their effect on programmer productivity. *AFIPS '82 Proceedings of the June 7-10, 1982, National Computer Conference;* pgs. 351-358. ACM, New York, NY, USA URL: http://dl.acm.org/citation.cfm?id=1500817&CFID=664219319&CFTOKEN=29425029

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at:
http://www.sascommunity.org/wiki/Presentations:Tebillings_Papers_and_Presentations
or use this alternate short URL: http://goo.gl/uocYNc

Thomas E. Billings
MUFG Union Bank, N.A.
Basel II - Retail Credit BTMU
350 California St.; 6th floor
San Francisco, CA 94104

Phone: 415-273-2522
Email: tebillings@gmail.com