Can Validating SAS Programs be Fun and Easy?

Sy Truong, Meta-Xceed, Inc, Fremont, CA

Abstract

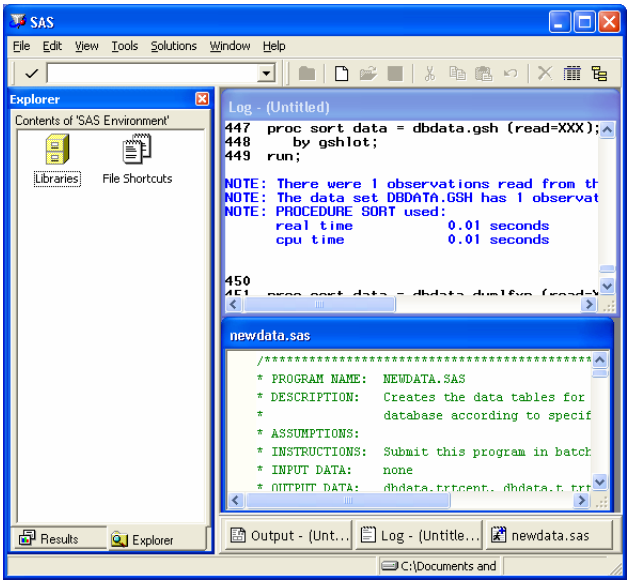
Validation is normally a laborious and arduous task. This paper will present new methodologies and tools developed in SAS that will make the process painless. The goal is to add little or no effort from the user's perspective, yet gain the benefit of having a secured audit trail of all SAS programs during the development and verification process. Some benefits are described below:

- comparing differences between different versions of programs
- adding notes describing edit changes to each version
- adding a validation checklist of tasks associated during verification and validation
- managing status of development to production by applying version numbers such as version 1.2
- generating reports for documentation and communication during validation

After you realize the ease of use and the amount of quality control that can be gained, the task of validation becomes transparent and fun.

Introduction

Validating SAS programs presents some unique challenges especially when working within a regulated environment such as the pharmaceutical industry. This paper explores the challenges specific to this environment though the examples can be useful in other environments as well. SAS programmers come from many different backgrounds that range from biology to statistics. The majority are not from a computer science background. This is normally due to the fact that they have expertise in the domain of the data in which they are analyzing. This is helpful for ensuring the outcome of the analyses but creates an unstructured environment for developing SAS programs. The work flow is driven by reports and therefore is usually done in an ad hoc manner. The analyst normally gets mockups of the report which describe what they need to produce. They often jump right into SAS programming with little or no data and programming design consideration. SAS has adapted to this work flow well compared to other more structured high level languages. Other languages such as C or Java are stronger typed. This means that the variables and tables have to be defined with proper variable type and length before they can be used. On the other hand, SAS programs can dynamically create variables as you go along lending itself to the ad hoc nature of the development process. This can be beneficial for creating exploratory analysis and conducting experiments with the data. However, it fosters software development that is riddled with maintenance challenges. The tools used to develop SAS programs such as display managers or text editors are further examples of ad hoc nature. Display manager gives some structure, but it is designed for exploration. Software development tools for other languages allow for the programmer to manage the source code as it relates to other programs and data. On the other hand, SAS programs are plain text files that any user can edit with a text editor of their choice. In a similar way, display manager leaves the programs stored on disk as text files and does not create any other structure upon that.



This is an example of how SAS programming is difficult to validate. Programs are inherently buggy by nature since there is great variability and complexity. It is written by humans but is interpreted by machine. Even though the syntax is set up with constructs to handle the parameters to help, humans do not think in complete logic. This leads to misinterpretations and bugs. SAS programming is often data driven which adds another dimension to the complexity since the data can be dynamic in content and structure. The changes in data drive changes in results and therefore changes in programs. The management of changes of each component and all of its interrelationships makes SAS programs an ever changing organism desperately in need of containment. The issue of change control will become a major strategy in taming the beast and is one of the primary themes of this paper.

One of the attempts to create structure around the chaos is the use of SAS macros. Macros are intended to isolate repeated tasks and parameterize them so that they can be repeated. However, the way macros are sometimes used leads to spaghetti code since one macro calls another macro in a nested loop. This sometimes results in more complexity and becomes more challenging rather than simplifying.

Validation Benefits

There are many challenges in creating an effective validation environment for SAS programs but there are also many benefits that can rationalize the effort. There are reasons to make a strong business case for performing validation. The most obvious is the requirements by the FDA spelled out in CFR part 11. In a regulated environment, it is not just a nice idea to perform validation, but it is a legal necessity. Here are some examples of other important benefits.

- Less Rollouts – Each time a program is rolled out, it is commonly followed by patches. This is to fix bugs that did not get caught during validation.
- Prevent Data Corruption – Bugs can be traced back to programs that have not been fully validated. Using these programs creates corruption in the data and reports.
- Facilitates Communication – The requirements and functional specifications along with the test scripts can

be developed with close collaboration with the end user. This leads to a clearer understanding between the user and the developer.

- Software Maintenance – During validation testing, versioning and an audit trail are created. This helps with tracing and attributing features and bugs. This audit trail leads to better tracking of programs between different releases which helps in the management of bugs and wish list items.

A little effort can go a long way. Validation can be viewed as an investment. At first, the amount of validation "capital" invested may not seem to have any immediate returns. However, as the process gets further into the development life cycle, the benefits are well worth it. This does require a long term vision with commitment for quality.

Validation Scenario

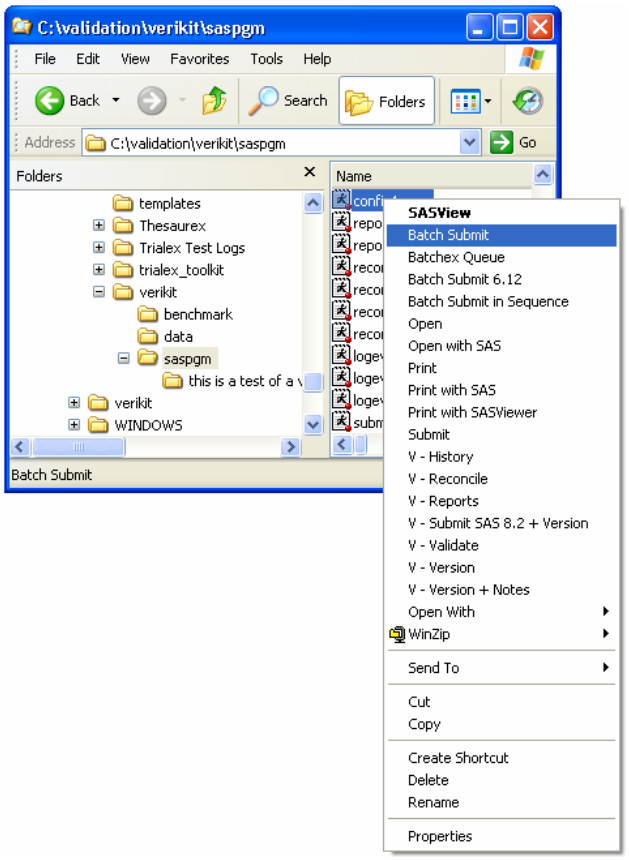
The following example was implemented with a small biotech company consisting of six SAS programmers and several statisticians. The SAS programmers and I met as a team on a weekly basis in formulating this validation process. The validation effort was part of a larger effort in creating a statistical computing environment which included a new four processor Windows 2000 server. The team had to put in extra efforts in developing the process, while at the same time performing analysis and reporting of clinical trials data.

There are several aspects to the validation process. Our team decided to automate the parts that will save the most amount of time. The team originally worked on a VMS legacy computing environment. In this environment, each SAS program was automatically versioned each time it was edited and saved. This works in a similar way to the GENMAX option for SAS datasets. The operating system allows users to specify how many generations or versions they would like to keep. Each time they edit a SAS program, the old version is kept as a separate file with a version number appended to its name. When the team moved into the Windows environment, there was no such auditing capability for SAS programs.

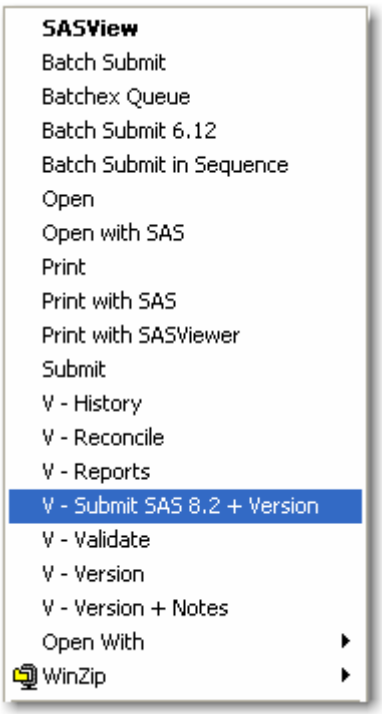
We developed a process and tools that would accomplish the versioning of SAS programs called Verikit™. In order to accomplish our validation requirements, we needed to do more than just make a backup copy. We identified the following tasks that needed to be done:

1. Backup – Make a copy of the current version of the SAS program.
2. User Name – Capture the user name of the person interacting with the program.
3. Date Time – Capture the date time at the moment of the transaction.
4. Action – Identify what type of action is being performed. This is defined as part of the validation process. Some examples include: version backup, locking for testing, validation testing, promoting to production.
5. Notes – It is optional to capture a short message explaining the current step. This adds meaning and context to the task.
6. Validation Tasks – If the step involves performing validation testing, the specific validation task is captured.
7. Status – A status associated with the SAS program to identify if the validation testing had failed or passed.

Once we had identified all the requirements, it became obvious that even the features of the legacy VMS operating system did not meet our validation needs. We wanted to develop a process in which all of the required information was captured, while adding little or no extra effort upon the user. One of the most common ways that a SAS programmer interacts with programs is submitting them. We decided that this would be a good time point to capture some of this information. From Windows Explorer, a user submits a program by right mouse clicking on the program and selecting "Batch Submit".

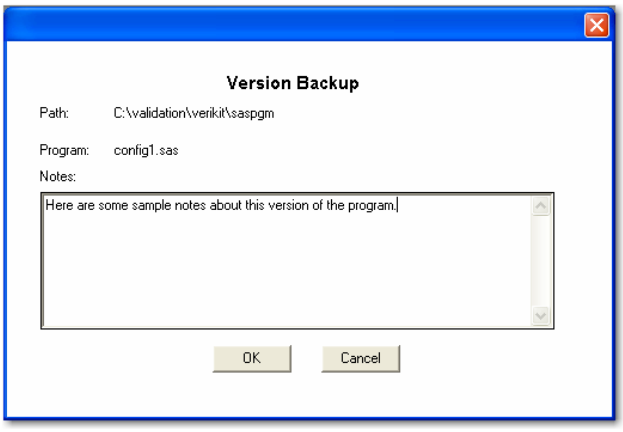


We extended the menu so that in addition to submitting the program, a version backup is also captured. The amount of effort from the user is the same. That is, they would right mouse click on the program and select a menu item.



In this step, the tool would automatically capture pieces of information of items 1 through 4 as mentioned above. In addition, it would assign a default status for item 7. After capturing and recording this information, it would then submit the program in the same way that the "Batch Submit" did before. In this case, we were able to capture about 70% of the required auditing information without any additional effort from the user.

We had determined that users did not need to capture every single version program during the development of their programs or validation test scripts. It is more realistic that only pivotal changes in the code would require a version backup. For smaller edits to the program, users would still use the "Batch Submit" selection. Once they decided that the code had changed significantly from the last time a version was captured, they would then choose the "V – Submit SAS 8.2 + Version" menu item. On some of these code changes, a note describing the change is required to add meaning to the audit trail. In this case, another menu item "V – Version + Notes" is selected.

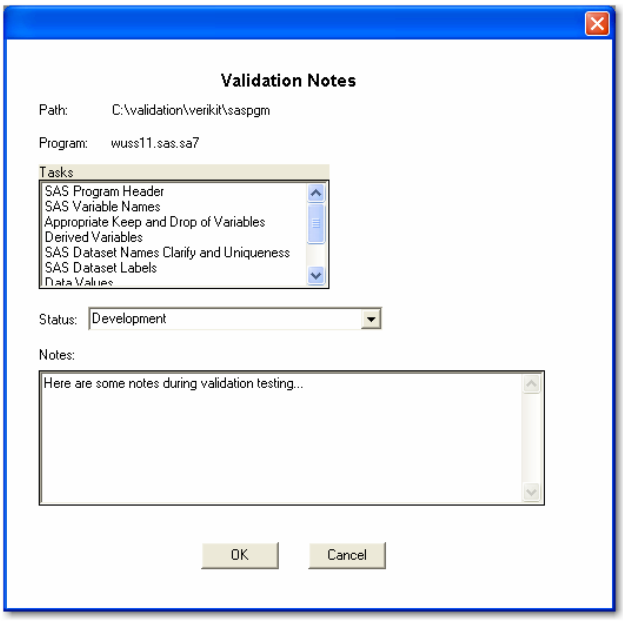


This step would capture all the information as the previous example including: backup program code, program name, user name, action, task, and date time. In addition, a short note can be entered describing the current code or logic change.

The features of creating a backup and capturing a descriptive note can be used during any type of SAS program development. However, in order to integrate this into our validation process, we needed a mechanism that would lock the program for performing validation testing.

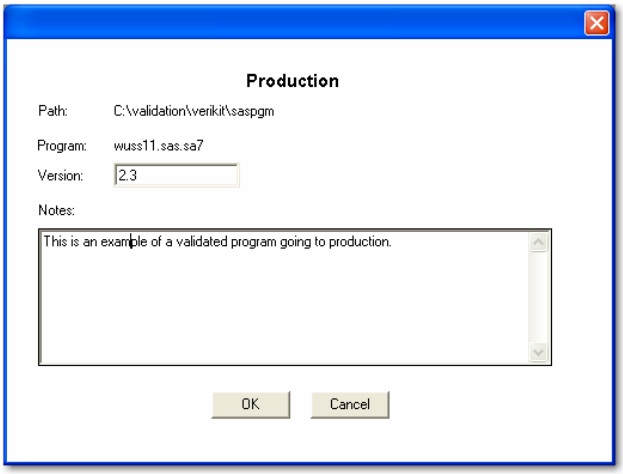
The process involved a verifier, who is a different person from the original author of the SAS program, to review the program and associated output and data. The verification may even include developing another SAS program to come up with the same results. In this case, the verifying program can use the same versioning technique for a complete audit trail. During verification, however, it made sense to lock the original code since we did not want to be verifying a moving target. When the user initiates the validation process by selecting the menu "V – Validation", a copy is made but it also changes the file extension so that it is clear that this is no longer a program to be edited.

Upon completion of verification, the verifier can record the findings by right mouse clicking on the locked program and selecting "V – Notes".



This allows the verifier to record specifically which verification tasks were performed and if the testing was successful or not. A status is recorded to determine what is to be done. If it failed, then the original programmer has to fix the problem and the verifier goes through the loop again. If it passes, it can be promoted directly to production. At each step of the way, information is captured including a descriptive note which gives context to the task at hand.

Upon promotion to production, the programmer can choose to assign a version number. This can follow the decimal conventions such as version 1.0, 1.1, 1.2 etc…

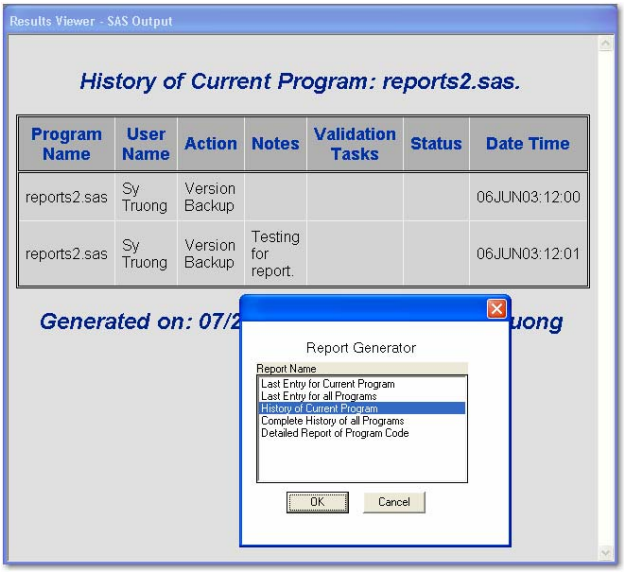


If the verifier promoted a program directly from the verification process with the selection of "Verication Passed + Production", this will increment the version number by one integer value automatically. By performing the promotion in a separate step, the user can increment the version number to their custom value.

If during verification, problems were identified, the original programmer will need to unlock the program to perform the fix. This is available through the menu item "V – Unlock". The tool allows the user to record a note pertaining to the unlocking and then it renames the file back with the (.sas) file extension for further edits.

At any point, a user can generate reports to see progress of the validation effort. They can select the menu item "V – Reports". The following choices are presented:

- Last Entry for Current Program
- Last Entry for all Programs
- History of Current Program
- Complete History of all Programs
- Detailed Report of Program Code



These reports can be attached or pasted into emails for communicating status. They can also be used as documentation of an audit trail as part of a validation test plan. These canned reports will deliver most of what is needed. The data from which these reports are derived are stored as a SAS dataset and catalog. Since the tasks being performed are done by a SAS programmer, it is a natural format to make the information available in SAS format. The user can therefore use their favorite reporting procedure such as PROC REPORT to generate their own custom reports.

Even though we take the precaution of recording versions and locking files, it is still possible to have a SAS program get out of sync. Since the program is a text file, another team member can accidentally open the file and inadvertently add some additional text. This issue is resolved through the reconcile process. By selecting the menu item "V – Reconcile", it will check to see if the physical program stored on disk is the same as the last version that has been captured. In case you have promoted a program to production mode. Reconcile will inform you if the physical program is the same as what has been recorded in production.

The reconciliation process is commonly applied to a group of programs rather than one at a time. As an alternate to the interactive process, a SAS macro named *%reconcile* can be used.

```
%reconcile(path=c:\myprog,
           program=demog.sas);
```

In this case, a set of programs can be reconciled at once and a report is generated without having to clicking on each program. In addition to this macro, there are other SAS macros in Verikit that automate validation tasks. The items highlighted in this paper are not comprehensive but give a flavor of one approach to the challenge of SAS program validation.

Conclusion

SAS programming can be unstructured at times. The data being processed drives business decisions and are key to any organization. However, this lack of structure and formal validation process can sometimes lead to erroneous results. The ad hoc nature of SAS programming creates an environment that is not conducive to consistency and accuracy. This leads to the development of an uncontrolled environment which produces programs that are difficult to understand.

Since performing validation can sometimes be a mundane and resource intensive process, it is challenging to get team members motivated to perform validation. Using the right tools which lessen the burden upon users is the solution to getting the job done. The tools presented in this paper, Verikit, allow users to perform many common validation steps with just one click from within a familiar environment. They don't have to adjust to a new complex system with a steep learning curve. The FDA regulations require "validation of systems to ensure accuracy, reliability, consistent intended performance, and the ability to discern invalid or altered records." This is accomplished through consistent recording of changes throughout the process. Change control is a significant part of the validation process. Verikit attempts to automate this and other tedious steps in this process to make the work bearable and even fun.

References

FDA, Guidance for Industry: Providing Regulatory Submissions in Electronic Format – General Considerations, 1999

Guidance for Industry: Providing Regulatory Submissions in Electronic Format - NDAs , January 1999

Verikit™ information at: http://www.meta-x.com/verikit

About the Author

Sy Truong is a Systems Developer for Meta-Xceed, Inc. They may be contacted at:

Sy Truong
48501 Warm Springs Blvd. Ste 117
Fremont, CA 94539
(510) 713-1686
sy.truong@meta-x.com