Paper HW07

Effective Ways of Validating SAS Programs

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

Abstract

Validation is normally a laborious and arduous task. This paper will present 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 tasks and benefits covered in this paper include:

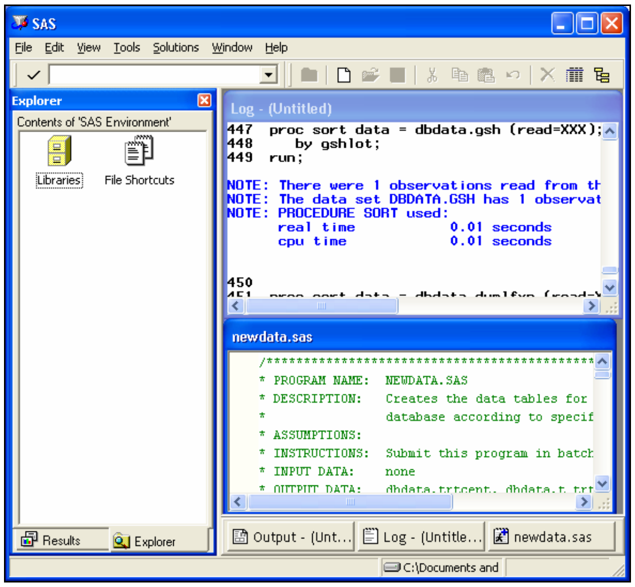
- preparing a requirement/functional specification and test plan
- comparing differences between different versions of programs
- adding notes describing edit changes to each version
- adding a validation checklist of tasks associated during verification and validation
- managing status of development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

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

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 environments, although 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 SAS programmers 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 programming environment. 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 considerations.

SAS has adapted to this work flow compared to other more structured high level languages. Other languages such as C or Java are stronger typed. This means that the variables and data 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 the 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.



```
447  proc sort data = dbdata.gsh (read=XXX);
448     by gshlot;
449  run;

NOTE: There were 1 observations read from th
NOTE: The data set DBDATA.GSH has 1 observat
NOTE: PROCEDURE SORT used:
      real time            0.01 seconds
      cpu time             0.01 seconds


450
451  proc sort data = dbdata.dumlfxn (read=X
```

```
/***********************************************
* PROGRAM NAME:   NEWDATA.SAS
* DESCRIPTION:    Creates the data tables for
*                 database according to specif
* ASSUMPTIONS:
* INSTRUCTIONS:   Submit this program in batch
* INPUT DATA:     none
* OUTPUT DATA:    dbdata.trtcent, dbdata.t trt
```

This is an example of how SAS programming becomes difficult to validate. Programs are inherently buggy by nature since there is great variability and complexity. They are written by humans but are interpreted by machine. Even though the syntax provides the constructs to handle parameters, humans still 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 the changes in output and therefore the 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 validation effort. There are many 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.
- Prevents 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 in 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 would 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.

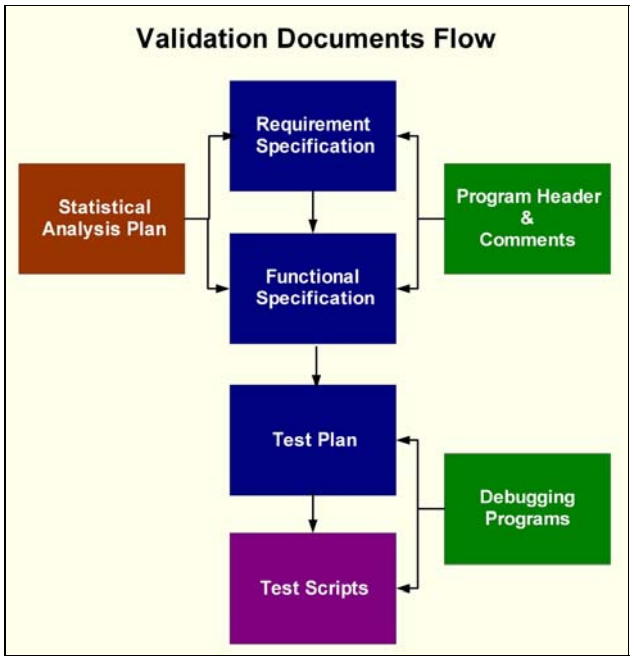
We developed a process and tools that would accomplish the versioning of SAS programs. In order to accomplish our validation requirements, we needed to do more than just make a backup copy. We identified the following tasks that needed to be done:

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 has failed or passed.

These steps comprised only part of the process. The highlighted steps above were automated. The other steps included tasks dealing with documentation which were optimized, but it was still a manual process.

Requirements/Functional Specifications

One of the first steps in validation is to have a test plan. The test plan is derived from a list of functional specifications. The functional specifications are, in turn, driven by the list of requirements. This is an interrelated set of documents that drive the process.


Validation Documents Flow
Statistical Analysis Plan
Requirement Specification
Functional Specification
Test Plan
Test Scripts
Program Header & Comments
Debugging Programs
Figure 1

The level of detail of documentation depends on the complexity of the set of programs being validated. The following levels distinguish the differing types of SAS programs.

- Exploratory – These are random sets of programs developed by analysts and statisticians to test out a hypothesis. They are not included in the final analysis or part of a submission.
- Stand Alone – These are one of the programs developed to generate specific reports or analysis files. They may be driver programs that call other macros but these driver programs are not used multiple times.
- Multi-Use – These are usually macro code or standard code segments that are used multiple times in more than one analysis. They can be stored at a global library where multiple users can access them.

It may be worthwhile to track exploratory programs, but they are not usually included in formal documents. However, it is recommended that both stand alone and multi-use programs are included. The multi-use programs deserve more detail and formal documentation, while the stand alone programs can be abbreviated to fit the complexity of the specific program.

The validation documents are analogous to the programming tasks that programmers commonly perform. These include:

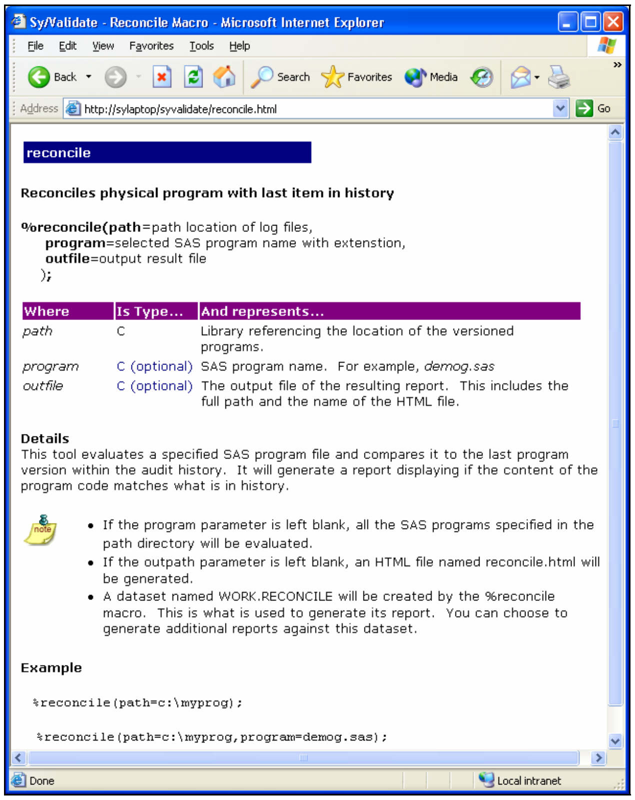
1. Program Header – At the top of a SAS program, there is a comment section including a description of what the program does along with who programmed it and when.
2. Section Comments – A SAS program has comments interspersed throughout the code. Usually each functional section contains comments describing how and what that particular section does.
3. Debugging – After the program is done, it is usually executed with various input data and parameters. The output and log are reviewed to ensure that it worked accordingly. This sometimes requires a separate test program to verify the results.

These three programming tasks are analogous to the three pieces of documentation. The descriptive comment in the program header of step 1 can have the same information in the requirement specification. This describes to the user what is expected of the program and what it is supposed to do. The functional specification includes functional details describing each component of the program, spelling out what each function does. The debugging process is analogous to the verification process captured in the test plan.

The three documents are more formal in their writing style but the contents contain analogous information. If the SAS programmers have these concepts in mind, they can perform good programming practices while also producing meaningful validation documents. The combination of documenting as you program can save time and produce quality programs. If you were to develop the program and then later go back and perform the formal documentation, the information may not be accurate and reflective. The documentation can clarify the logic and algorithm in the programmer's head before coding starts. This leads to clearer and more concise coding resulting in a faster development cycle.

Reference Manual as Functional Specifications

A good strategy to optimize the validation process is to identify other tasks that are normally done which are analogous to a validation task. The duplicate tasks can be combined to save time. The reference or user manual commonly contains similar information to the functional specifications document. By authoring these two at the same time, the same content can be used. Normally, the reference and user manuals are authored after all programming has been complete. By moving this process prior to coding, users can quickly grasp the functionality of the tool early on. Users can therefore communicate changes or other feedback to the programmer. This can act as a short hand for the functional specifications. In this example, the reference manual for the *%reconcile* macro is authored in HTML and distributed to all the users via an intranet.


Sy/Validate - Reconcile Macro - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sylaptop/syvalidate/reconcile.html Go
Done
Local intranet

reconcile

Reconciles physical program with last item in history

```
%reconcile(path=path location of log files,
    program=selected SAS program name with extenstion,
    outfile=output result file
   );
```

Where	Is Type...	And represents...
path	C	Library referencing the location of the versioned programs.
program	C (optional)	SAS program name. For example, *demog.sas*
outfile	C (optional)	The output file of the resulting report. This includes the full path and the name of the HTML file.

Details
This tool evaluates a specified SAS program file and compares it to the last program version within the audit history. It will generate a report displaying if the content of the program code matches what is in history.

- If the program parameter is left blank, all the SAS programs specified in the path directory will be evaluated.
- If the outpath parameter is left blank, an HTML file named reconcile.html will be generated.
- A dataset named WORK.RECONCILE will be created by the %reconcile macro. This is what is used to generate its report. You can choose to generate additional reports against this dataset.

Example

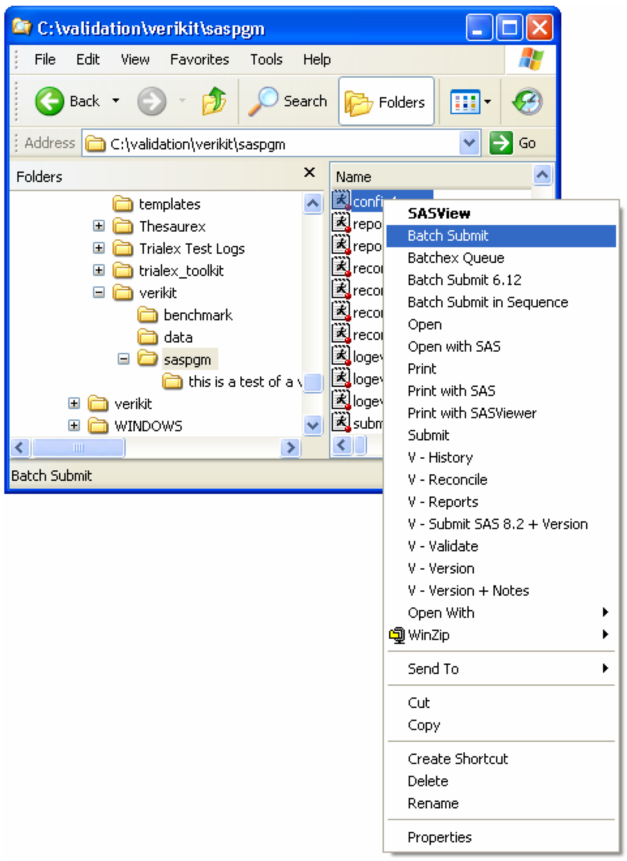
```
%reconcile(path=c:\myprog);

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

The documentation contains a description of the macro along with all the parameters. The content is organized in a way that is easy for the user to understand how the macro would be used. At this point, users can supply feedback to the programmer requesting clarification or changes to the features. Changes can be agreed upon and the reference documentation can be updated quickly without any major code changes since no programming has yet begun. Once all users and the programmer agree upon the functionality of the new tool, the same content will be paraphrased or pasted into the functional specifications. The programming effort will then commence.

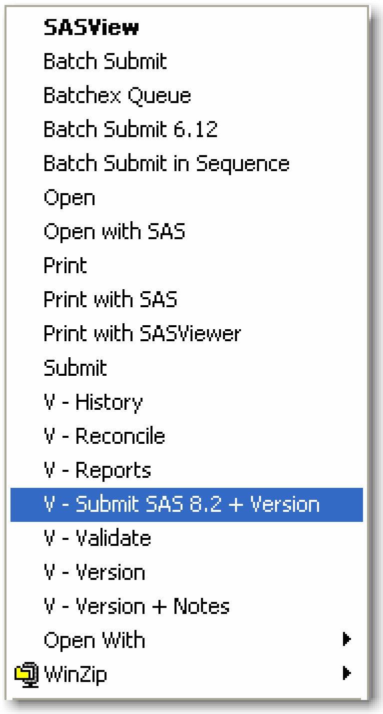
This approach brings the functional specification closer to the user since they have input. You can use the functional specifications document for this purpose, but the reference manuals are usually friendlier documents for the users compared to the formal text of functional specifications. Changes can still occur once the users start to use the tool but this makes version 1.0 of the tool much closer to the users' requirements. By using this approach, you are also accomplishing multiple things since upon completion, you will have both functional documentation and polished user documentation.

Automated Validation Tasks

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".

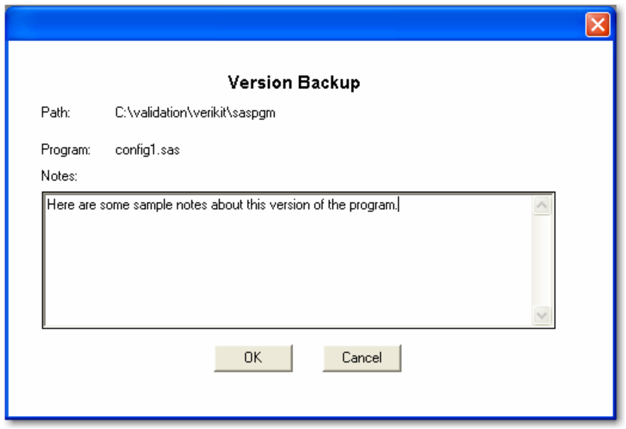


We extended the menu so that in addition to submitting the program, a version backup is also captured. The amount of effort from the user is the same. That is, they would right mouse click on the program and select a menu item.



In this step, the tool would automatically capture pieces of information of items 1 through 4 as mentioned in the validation scenario section 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 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.


Version Backup

Path: C:\validation\verikit\saspgm

Program: config1.sas

Notes:

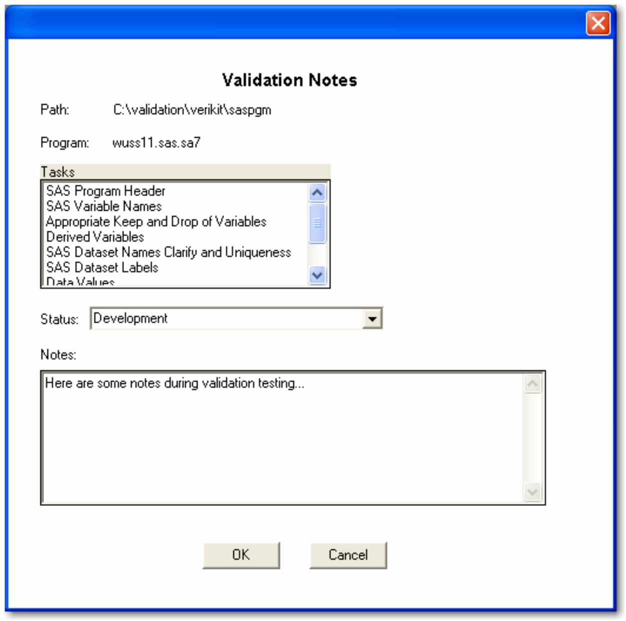
Here are some sample notes about this version of the program.

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".


Validation Notes

Path: C:\validation\verikit\saspgm

Program: wuss11.sas.sa7

Tasks

SAS Program Header
SAS Variable Names
Appropriate Keep and Drop of Variables
Derived Variables
SAS Dataset Names Clarify and Uniqueness
SAS Dataset Labels
Data Values

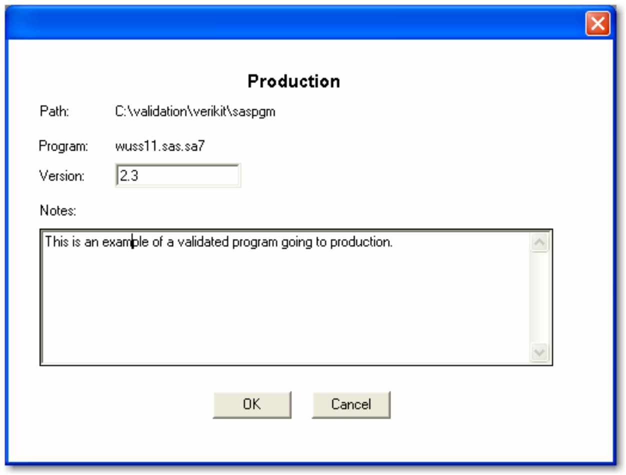
Status: Development

Notes:

Here are some notes during validation testing...

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…


Production

Path: C:\validation\verikit\saspgm

Program: wuss11.sas.sa7

Version: 2.3

Notes:

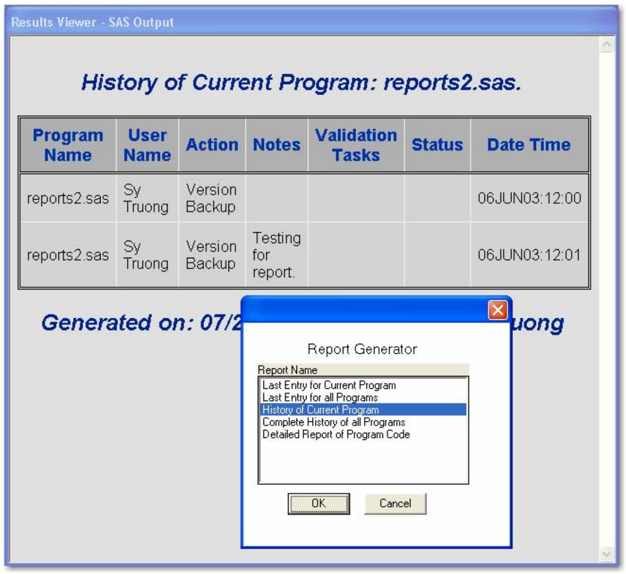
This is an example of a validated program going to production.

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 a 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



History of Current Program: reports2.sas.

Program Name	User Name	Action	Notes	Validation Tasks	Status	Date Time
reports2.sas	Sy Truong	Version Backup				06JUN03:12:00
reports2.sas	Sy Truong	Version Backup	Testing for report.			06JUN03:12:01

Generated on: 07/2 … uong

Report Generator

Report Name

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 choose 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 click on each program. In addition to this macro, there are other SAS macros developed 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.

Verifying SAS Data

The emphasis so far has been on SAS programs. Another component of the verification process pertains to SAS datasets. SAS datasets are commonly created as analysis files which contain the main business objective for the project. It is therefore important to thoroughly verify all datasets, such as those sent to the FDA, as part of an electronic submission. Some of the verification tests described in this section include:

- Standard Attributes – Consistent dataset, format and variable attributes across different datasets.
- Valid Dates – Partial character and missing dates are verified.
- CDISC Compliant – Dataset and variable names are consistent with CDISC guidelines where possible.

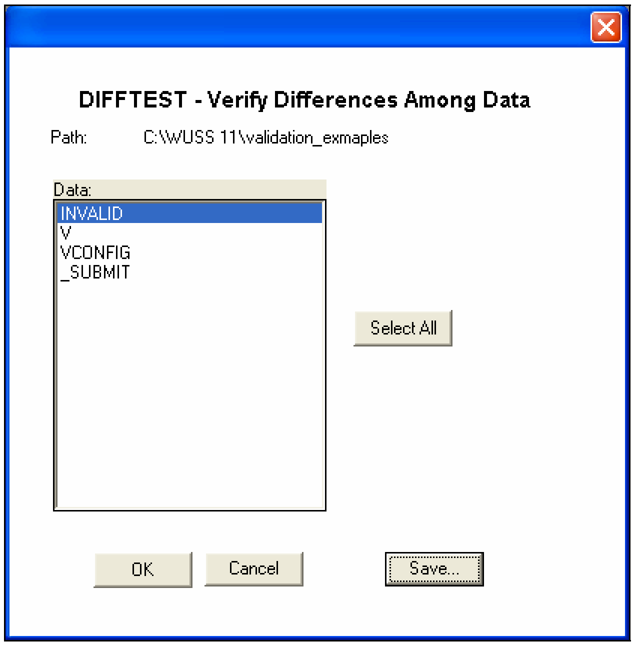
There are of course many more verification tests that can be performed to ensure the integrity of the data, but these are highlighted since they can be automated.

One of the most common mistakes made is to have two datasets within the same group of analysis files containing the same variable names, but having slightly different attributes. For example, a different label for two variables with the same variable name can cause some confusion for the reviewer. Even more confusing is when two different variables contain the same variable name, but have a different variable type or length. This creates problems when these variables are used as an index in a merge. It is therefore important to have all the attributes consistent across all datasets. Some of the verification tasks to consider while verifying data include:

1. For variables with the same name across different datasets, verify that the following attributes are the same:
    a. Type
    b. Length
    c. Label
    d. Format Name
    e. Informat Name
2. For variable labels that are the same, verify if the corresponding variable names are the same.
3. For format names that are the same, verify if the coded values of the formats are the same.
4. For format codes that are the same, verify if the format names are the same.
5. For dataset names that are the same, verify if the dataset labels are the same.
6. For dataset labels that are the same, verify if the dataset names are the same.
7. For variables with coded formats, verify if the values of the variables match up with the specified format codes.

The manual approach to performing these tasks would be to produce listings from PROC CONTENTS and PROC FORMAT and eye ball the differences. You can automate this process to a certain extent by sending the output of PROC CONTENTS and PROC FORMAT to datasets and then performing PROC COMPARE. This will help catch things that may be missed by the human eye.

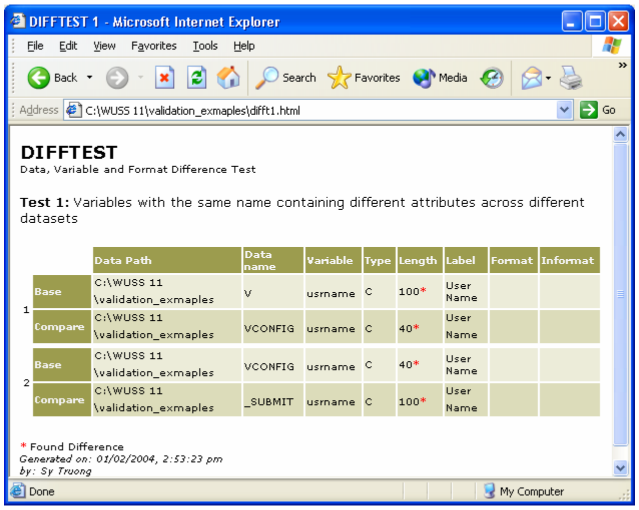
This can be automated further through the Sy/Data tool which can be accessed through the Windows Explorer window. You would need to right mouse click on a SAS dataset and select the menu item "D – DIFFTEST". An interactive menu appears displaying all the datasets in that specified directory.


DIFFTEST - Verify Differences Among Data

Path: C:\WUSS 11\validation_exmaples

Data:
INVALID
V
VCONFIG
_SUBMIT

You can then select all or a subset of the datasets to which you wish to apply the difference tests. The results will be automatically summarized and enumerated by test number. The following example shows the test criteria item 1 as listed above.


DIFFTEST 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\WUSS 11\validation_exmaples\difft1.html
Go
Done
My Computer

DIFFTEST
Data, Variable and Format Difference Test

Test 1: Variables with the same name containing different attributes across different datasets

		Data Path	Data name	Variable	Type	Length	Label	Format	Informat
1	Base	C:\WUSS 11 \validation_exmaples	V	usrname	C	100*	User Name		
	Compare	C:\WUSS 11 \validation_exmaples	VCONFIG	usrname	C	40*	User Name		
2	Base	C:\WUSS 11 \validation_exmaples	VCONFIG	usrname	C	40*	User Name		
	Compare	C:\WUSS 11 \validation_exmaples	_SUBMIT	usrname	C	100*	User Name		

* Found Difference
Generated on: 01/02/2004, 2:53:23 pm
by: Sy Truong

Besides the data and variable attributes, it is recommended that all character date variables be verified. Some of the date related tasks to be considered include:

- Partial Date – capture all date values which are missing either month, day or year.
- Missing Date – capture all date variables which have missing values.
- Invalid Date – capture invalid date values such as months greater than 12 or days greater than 31.
- Future Date – capture all dates that are greater than the current date.

The date values presented do not mean that the values are discrepant but it informs the verification person of the abnormalities for further review. A manual approach is to perform a PROC PRINT of all the character date variables and perform a visual inspection. An automated approach can be accomplished similar to the DIFFTEST, but in this case, it is named DATETEST.

The latest CDISC guidelines for submission datasets can be viewed at: http://www.cdisc.org/pdf/V3CRTStandardV1_2.pdf. This document recommends everything from variable naming conventions to variable types. If your dataset uses any of the same variable names that are part of the CDISC recommendations, it is advisable that your corresponding variable labels match the guidelines. The following verification tasks should be considered when your data attributes partially match CDISC guidelines. They are listed with the associated CDISC document number in parenthesis.

1) Required Fields – (2.4.5) Required identifier variables including: DOMAIN, USUBJID, STUDYID and --SEQ.

2) Subject Variable – (3.5.1.2.8) For variable names, labels and comments, use the word "Subject" when referring to "patients" or "healthy volunteer".

3) Variable Length – (3.5.1.2.6) Variable names are limited to 8 characters with labels up to 40 characters.

4) Yes/No – (3.5.1.3.18) Variables where the response is Yes or No (Y/N) should normally be populated for both Yes and No responses.

5) Date Time Format – (3.5.1.4.19) Use yymmdd10. but yymmdd8. is acceptable.

6) Study Day Variable – (3.5.1.4.22) Study day variable has the name ---DY.

7) Variable Names – (3.5.2) If any variable names match that of CDISC, the associated label has to match.

8) Variable Label – (3.5.2) If any variable labels match that of CDISC, the associated variable has to match.

9) Variable Type – (3.5.2) If any variables match that of CDISC, the associated type has to match.

10) Dataset Names – (3.5.2) If any dataset names match that of CDISC, the associated data label has to match.

11) Dataset Labels – (3.5.2) If any dataset labels match that of CDISC, the associated dataset name has to match.

12) Abbreviations – (3.5.2) The following abbreviations are suggested for variable names and data sets.

    a) DM Demographics

    b) CM Concomitant Medications

    c) EX Exposure

    d) AE Adverse Events

    e) DS Disposition

    f) MH Medical History

    g) EG ECG

    h) IE Inclusion/Exclusion Exceptions

    i) LB Labs

    j) PE Physical Exam

    k) SC Subject Characteristics

    l) SU Substance Use

    m) VS Vital Signs

13) SEQ Values – (4.3.2.1) When the --SEQ variable is used, it must have unique values for each USUBJID within each domain.

Similar to the other tasks, you can manually produce a PROC CONTENTS and review each condition by visual inspection. You can also automate this task and have a report generated pointing out the differences for the verification person to evaluate changes in the data for compliance with the recommended standards.

By automating many of the tasks related to reviewing data, it frees the verification analysts from repetitive tasks and allows them to perform other verification tasks with much greater accuracy and speed. This removes some of the mundane aspects of verification and makes the process less of a burden.

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 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. The tools and process presented in this paper attempt to automate this and other tedious steps in the 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

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) 226-1209
sy.truong@meta-x.com