Paper AD08

Building A SAS® Application to Manage SAS Code

Phillip Michaels, P&Ls, Saint Louis, MO

ABSTRACT

In spite of SAS's power and flexibility, it is rarely used for anything more than extracting, analyzing, and reporting. IT professionals are resistant to using SAS for major corporate applications because of its lack of structure for managing complex application development and maintenance. This paper discusses how to create this structure, addressing such questions as:

- How can IT manage the running of regular nightly/weekly/monthly SAS jobs?
- What must be included in SAS applications to monitor IT operator-run SAS programs for failure.
- Why should each line of the SAS code be put into a SAS accessible database?
- How is SAS uniquely positioned to document the integration of data and programs in an application?

INTRODUCTION

The process for managing SAS applications presented in this paper was worked out during the course of building a supply chain management system in SAS for a $2 billion consumer products company. This home-grown set of applications significantly lowered inventories, increased sales, and resulted in vendor-of-the-year awards from Target and Wal-Mart the year it was introduced.

Regular SAS users readily understand why SAS was chosen for this application. SAS is a highly flexible programming language with one of the most extensive software toolsets. It is one of the few application languages that do not necessitate the incorporation of a second, or third, language or application software to meet required functionality. You can extract data from multiple formats easily, analyze the data in sophisticated ways, and report the findings in extensive graphics. All of this can be done in a single job that can be submitted to a mainframe, a desktop PC, a networked server, or any combination of these resources. The only thing that rivals SAS's reputation for completeness is the clarity of its documentation and its depth of technical support.

Why, then, are so few major applications built in SAS? Have you ever wondered why system administrators are so reluctant to bring SAS into their shops, why these administrators are even more resistant to allowing the number of SAS users to increase, or why very few IT shops (outside of drug and insurance companies) ever build major corporate applications in SAS? It doesn't make sense that a language with the power and flexibility of SAS (even though it is only available for lease) is mostly used for extracting, analyzing, and reporting, does it?

More than a few IT professionals point to the fact that SAS is a leased product, or that SAS's programs can demand a lot of hardware resources. But the reality is that IT objects to SAS because SAS is designed for users, not the IT operations group.

SAS lacks a comprehensive environment for managing complex application development and maintenance. One person using SAS to work on a problem can do amazing things. Ten people trying to cooperatively build and maintain a SAS-based application can easily get lost. Corporate IT groups often have substantial home-grown processes in place to avoid just such problems. They have learned, usually through painful experience, that to ensure reliable and effective application development and maintenance the development and maintenance process must have a structure.

The basic requirements for creating an effective environment for SAS application development and maintenance are not difficult to meet. One good manager, some discipline in SAS code writing, and a few small SAS applications can go a long way to making SAS the in-house powerhouse development environment. However, three areas in SAS require additional structure to assure adequately controlled SAS program development and maintenance:

1. documenting new applications;
2. controlling change implementation; and
3. creating an IS-operations-SAS-batch environment.

DOCUMENTING NEW APPLICATIONS

There is nothing a new programmer hates more than to be handed a 50-page, single-spaced document that details the corporate process for creating and documenting a new application. However, to be truly successful, applications must stand the test of time and nothing increases application life expectancy more than effective documentation.

Application life expectancy is largely determined by how easily an application can be read and understood. A truly elegantly written SAS program ("elegant" meaning written in as few lines of code as possible) that can not be easily understood will always be regarded with some suspicion. Future programmers maintaining the code will eventually despair of understanding how it works and try to write code around it, sometimes quickly turning elegant into ugly.

Making programs easier to read, with transparent data organization and readily understood application structures, is a very complex task. Many programmers think that inserting comments at the beginning of a program to tell what the code is supposed to be doing satisfies the requirement to document their work. Although these written explanations are helpful, in order to be truly successful, documentation must be built into the very structure of the code, the data, and the application itself.

Programming Structure Rules

Following is a section of code involved in adding and coding new records to a sales forecast. This example code (Figure 1 below) makes use of SAS's ability to handle code in almost any way a programmer types it in.

```
data ds1;retain v1;attrib v2 length=$9;set myco.file1(keep=v2 v3 v4
v5);if _n_=1 then v1=round((datetime()),1.); proc sort data=ds1
out=fid(keep=v3 v2) nodupkey;by v2 v3;data ds2;set ds2;v6=v2;if v6 in
('FAM' 'MEF' 'KAU' 'LOR' 'FOL' 'FLS' 'HCT' 'ROE') then v6='MYC';else
if v6 in ('BUR' 'LAZ' 'MAC' 'MCN' 'STI' 'BNO' 'BLD') then
v6='FED';else v6='ROM';
```

Figure 1. **Unstructured Code**

While code written in this manner (admittedly extreme for purposes of example) may be a little easier to type and may save a little paper, it is very hard to decipher what it's doing. Adding just six rules to the way the code is structured makes a dramatic difference in how it can be read. The rules are as follows:

1. Choose variable names, dataset names, and SAS library names that have appropriate meaning and can be easily understood by any subsequent programmer.

2. Each data step or procedure, except the very first one in a program, is preceded by a blank line.

3. Each SAS statement starts a new line. A statement begins with a SAS keyword or a variable name and ends with a semicolon.

4. If a statement runs over the end of a single line, then any subsequent line of text required to hold the rest of the statement begins indented one space inside the first line of the statement.

5. Use indenting to show general code hierarchy. The first statement of a proc or data step is always positioned the farthest to the left. All subsequent statements are at least one tab indented to the right.

6. Indent to show hierarchy in "If-Then-Else" and "Do-end" statements. As far as practically possible, indent by one tab an ELSE statement inside the starting position of the associated "If-Then" statement.

Applying these rules to the code example in Figure 1 has the following effects on the code

```
data adds;
     retain updatime;
       attrib custcode length=$9;
       set myco.augbuy(keep=custcode style quantity date);
       if _n_=1 then updatime=round((datetime()),1.);

proc sort data=adds out=fid(keep=style custcode) nodupkey;
     by custcode style;

data miss1;
     set miss1;
     custprox=custcode;
     if custprox in ('FAM' 'MEF' 'KAU' 'LOR' 'FOL' 'FLS' 'HCT' 'ROE') then
      custprox='MYC';
       else if custprox in ('BUR' 'LAZ' 'MAC' 'MCN' 'STI' 'BNO' 'BLD')
        then custprox='FED';
              else custprox='ROM';
```

Figure 2. **Structured Code.**

Comment lines are still needed to make clear exactly what the code is doing, but now program flow is easy to follow.

Observance of minimal programming structure rules should be a documentation requirement for all programmers that is enforced by peer review. (N.B. Peer review is nothing more than having other programmers look through the code to make certain it meets minimum design standards. "Peers" should be selected from those programmers most likely to have to support the code at some later date. Code that cannot pass peer review should not be implemented.)

Data Structure Rules

The payoff for adding a few structuring requirements to the way SAS datasets are created comes even sooner than for structured programming rules. For example, SAS labels are automatically used by the software wherever appropriate, so carefully labeling variables as they are introduced often pays off with the first program using the dataset. Although in the heat of trying to create a new application this task is an easy one to put off, don't. Here is a list of minimal data structuring rules required for a disciplined SAS programming shop:

1. Use meaningful literal variable values wherever possible. This helps keep the database and the resulting application code something anyone can understand without deciphering a cryptic code. So, for example, MALE and FEMALE are better literal values for a variable called GENDER than 0 and 1 even though the database will require more space and more time to load into memory as a result. The savings is in building applications that can use the data easily without having to resort to building reformatting efforts into each program.
2. Use SAS labels for short meaningful descriptions of any new permanent variable as it is being created.
3. Permanently assign a pre-existing SAS format to each new permanent variable as it is being created, if appropriate.
4. Create and permanently assign a user-defined format to any new variable where an appropriate pre-existing SAS format does not exist.
5. Use SAS's dataset label field to label each dataset with a more descriptive statement as the dataset is created.
6. Create a meta-dataset for all SAS variables (a dataset containing information about variables). Information on every variable in every permanent and/or widely-used SAS dataset should be included in this dataset. Meta-data on variables can be partially loaded from a PROC DATASETS output.

Although building a dataset about data may appear to be unnecessary work, the advantages of this last requirement are many. A well-structured meta-dataset can make it easier to create new datasets as an

application grows, easier to identify needed changes when an upstream source of data is being restructured, easier to rebuild a lost or damaged dataset, and easier to automate the creation of new reports.

Documenting data so thoroughly and storing it in an easily accessible dataset at the beginning of an application creation process makes it much more likely that the full powers of SAS's automated features are leveraged. It also makes it much easier to keep the meaning and use of a variable consistent throughout an application. In this way many conflicts that otherwise tend to become visible at the end of creating an application are visible in the early stages, making them much easier to deal with.

Application Structure Rules

Rarely does an application consist of a single program, or just one dataset. In fact, even one application can often involve hundreds of programs. And while understanding how all these programs fit together is second nature to the people creating it, the structure may be far from obvious to others, even seasoned programmers. In fact, most successful large applications may have no more than one, or two, people who have a complete understanding of all its parts. For obvious reasons, it is dangerous for any organization to build programs or processes that are completely dependent on one person. This makes it imperative to have a way to capture application structure so that it can easily be understood by others.

If the basic code in a program is written following the coding structure rules given earlier, the functioning of any one program will be decipherable by most programmers. However, understanding how the various datasets and programs fit together can be very difficult to visualize, even after reading most of the code.

The solution is to put together a "programs and data" dataset that maps out these relationships. This dataset stores the structure of how programs and data are connected. Each record in the dataset describes one connection between a dataset and a program, or a program and another program. With this data PROC NETFLOW can be used to draw pictures of all the connections between programs and data. Every SAS batch program and every individual item in a SAS application catalog goes into this database. With a NETFLOW diagram of any application, a good programmer has a roadmap that documents how the programs and data fit together.

CONTROLLING CHANGE IMPLEMENTATION

The Developing, Testing, and Production Environments

Ordering change—whether it is the changes involved in minor modifications to an existing application or the implementation of a completely new application—requires that development occur in a structured environment. In the largest shops there are frequently three different areas where programs are stored: development, testing, and production. Smaller shops may merge development and testing together, although production always remains a separate entity.

The function of each area is implied in its name. "Development" is where program development occurs. This may well be contained completely on the desktop PC of the programmer. It requires its own set of data, and the rules for changing programs, data, names, etc. are very relaxed. This is where the programmer has the freedom to modify the application. Test datasets here need to be as complete as possible, but not very large. SAS's capabilities to select data on a random basis can easily be built into an application that builds small but robust test datasets.

Next is the testing environment. In those operations where testing is a separate area, it is typically used for testing by the ultimate application users. Datasets in the testing environment may contain most, if not all, of the data in the production environment, but they are typically not updated as frequently. It is here that the application's users can test drive the software with no concern about affecting production data. Errors, problems, and changes identified here go back to the programmer(s) and are fixed in the development environment. Moving a copy of a batch program, a changed element of a catalog, or a whole new catalog into testing may be under control of the programming group.

Moving to Production

The production environment is only for production quality code that has met the demands of peer groups, users, and IT operations staff. Moving changes, or a new application, to production must be carefully

controlled. Ideally, this would be done by a person outside the programming group (usually IT operations) who is a stickler for seeing that the new code meets all requirements before being moved.

Moving new code to production is always a risk. Therefore, there must be an easy and risk-free way to return to the just replaced code. This process still falls to the IT operations person who should maintain a careful written process of how to "back-out" changes and return the original code. In many cases the backup code would consist of saved copies of the last several versions of the program(s) or catalog(s) backed up in a physically separate area. Creating the backup and the steps needed to restore a backed-up copy to production should be part of a written procedure.

SAS Code Database

A critical part of tracking code that is moving into production is the building and loading of a last-resort code restoration tool. Every line of production SAS code goes into this dataset as it is being moved to production. In addition to current versions of the code, the last several versions (at least five) are stored here. While this dataset gives the operation a failsafe method for rebuilding a previous version of the code, this searchable dataset also is an invaluable resource for sorting out which programs a proposed change would have an effect on, for tracking how many lines of code each programming group is responsible for, and for analysis to identify effective programming techniques.

CREATING A BATCH PROCESS

IT-Initiated SAS Batch Jobs

Part of the support of every large application is "firing off" batch jobs that must be run on a specific schedule: nightly after 10 PM, weekly on Saturday, monthly at business calendar month-end, or nightly after job JX093FSB runs to completion. A good SAS IT operation has an application that makes this easy. This "IT Operations SAS Batch" application must accomplish several missions, including:

- Automatically showing any pre-conditions that must be met before running the job
- Automatically showing the batch operator when the next scheduled SAS job needs to run
- Tracking and logging the actions of the operator
- Identifying the programmer (and his contact information) on call to support each batch program
- Displaying specific job errors to the operator when a job doesn't run successfully
- Recording when jobs have run successfully
- Using the record of successfully completed SAS batch jobs to halt the running of a batch job stream until the failing job can be corrected.

SAS's ability to deal with complex date and time issues makes it easy to display batch production programs that must be run on a specific schedule. It is in generating specific error messages to the Operator that some careful programming is required.

Directing a batch computer operator to a SAS log is not an acceptable way of reporting errors in batch SAS jobs. Instead, programmers must write in macro tools to test each data or proc step to determine successful completion. If a step fails to run correctly, the job must automatically post a message to the error log database, halt production of the job, issue the error message to the operator, and halt the running of any other SAS jobs in this job stream until the error is corrected and this job has been run successfully.

CONCLUSION

In summary, the minimum applications, datasets, and processes needed to make SAS into an effective tool for general application development are:

- Applications
  - A Batch Scheduling Tool
- Datasets
  - A SAS variables meta-dataset (to track variable names and variable uses)
  - A programming and data links dataset (to make an application overview easier to visualize)
  - A SAS code dataset (to track changes in the code)
- Procedures
  - Programming structure rules
  - Data structure rules
  - A procedure for moving code from development to the testing environment
  - A procedure for moving code from the testing environment to production

Trademark Citation

Contact Information

Email: phillipmichaels@sbcglobal.net
Phone: 314-862-3217
Address: P.O. Box 16122, Saint Louis, MO 63105