Paper 148-2010

Boot Camp for Programmers:
Stuff you need to know that's not in the manual –
Best Practices to Help us Achieve Reproducibility

Elizabeth Axelrod, Abt Associates Inc., Cambridge, MA

ABSTRACT

Imagine that you completed a research project a year ago, and now your clients are challenging the results. They doubt your numbers, and they want to know how you came to your conclusions. Could you retrace your steps and reproduce your results? Could you describe what you did and why you made those decisions along the way? Could you find the data or the programs that you wrote? With the passage of time, it becomes more and more difficult to recall or recover the process, data, or results of a project.

As programmers, we're tempted to head straight for the keyboard, and start writing code. But first some basic training is in order. This paper is geared for programmers and programmer wannabes, focusing on specific guidelines, best practices, and techniques which can help us ensure the reproducibility or our results, and in doing so, improve the quality of our work.

INTRODUCTION

Welcome to boot camp! This paper presents a selection of concepts and organizing principles that provide a foundation for ensuring that you can retrace your steps, by helping you to keep track of information—both before and after it becomes a program. These practices are not tricks or fancy tools but fundamental guidelines for managing the flow of information from source data to final product.

Concepts presented here are an expansion of an earlier paper (Axelrod, 2003) and were developed while working in a government research environment, but the concepts are widely applicable in other settings.

Many data users learn the importance of documentation when data are challenged, and the need arises to explain or even reproduce the results of an analysis. At that point, everyone knows what should have been done and typically laments the day that a few simple procedures were ignored. In the sections that follow we discuss:

1. **Documentation.** How to keep track of what you're doing so you can retrace your steps.
2. **Processing.** How to run your jobs so you'll be able to reproduce the results.
3. **Programming guidelines**. How to write code that follows simple rules of style and coding that affect reproducibility and quality.
4. **File Storage**. Where to put your data, programs, and documentation so others can find them.

An exhaustive presentation of any of these topics is far beyond the scope of this paper. Instead the paper focuses on providing general guiding principles in each area and illustrating those principles with practical examples or best practices for making sure you can reproduce your work.

1. DOCUMENTATION: THE BIG 'D'

Preparing careful, comprehensive documentation—even in the face of time and budget constraints-- is perhaps the single most important practice for ensuring that you can retrace your steps.

"Documentation" is a broad term that covers a hierarchical array of information sources:

- **Comments in the code.** Useful to the program author as well as other programmers, comments help organize and explain the logic of a program.
- **Descriptive Documentation.** These are various documents that are developed during the process of file construction, and might include e-mails, memos and correspondence, specifications, descriptions of finished programs, user guides, etc. They are the building blocks of the overall project documentation, but are not typically organized in a useful format.
- **Process Documentation.** Process Documentation includes all of the above, as well as the 'recipe' used to construct the final file(s), or product. It is a comprehensive history of the project, recorded in sufficient detail to allow programmers and managers to retrace their steps, and reproduce results. This is what I refer to as the Big 'D'.

Typically, the project leader is responsible for the design of Process Documentation. The programmer is responsible for ensuring that the steps he or she takes in the handling of data can be recreated in the future. This includes traditional practices in code development and data handling, as well as the avoidance of certain practices that are inherently difficult to document.

THE DATA PROCESS

For the purposes of this paper, the "Data Process" consists of all the steps necessary to go from raw or source data, to a final product, whether it's a file, a table, or a report. These steps might include:

- Data receipt (receiving data from an external source, be it the internet, a client, or a colleague)
- Traditional batch processing (running a SAS® job on an input file, to create some sort of output);
- Moving data from one place to another (such as downloading data from the Internet, or uploading data from your PC to a mainframe);
- Transforming data interactively (in EXCEL spreadsheets, MS Access tables, or during an interactive SAS session).

It is the programmer's responsibility to record the data process.

KEY ITEMS TO BE RECORDED

At a minimum, you need to record the following items for each step or program along the way:

- Date the job was run
- Name & location of input file(s)
- Name & location of program
- Name & location of output file(s)
- Number of observations created in the output file(s)
- Brief comment describing the task.

METHODS OF RECORD KEEPING

There are many ways to maintain this information - some manual, some automated. I use a simple form, shown in Exhibit 1, which I call the *Docbox.* Each column of the Docbox corresponds to one of the minimum bits of information listed above. I record these items for each program I run, especially if the resulting output is a new file.

If you need to recreate the files, you've got all the information you need: a record of the sequence and location of the programs you ran, and the location and names of the source data. And if you want a reminder of what you did, you can just read down the right-hand column (comments), to see a descriptive summary of each step.

The Docbox is only one method for documenting a process. A flowchart is useful as a more visual representation of a data flow. Note, however, that if you do not include the actual paths and filenames, a flowchart only serves as a conceptual representation of what was done, and not a detailed document, or recipe, that could be used to redo the steps and reproduce your results.

EXHIBIT 1

SAMPLE DOCBOX

PROJECT: *ACME project*			TASK: *Process the Wave I ACME survey*		
DATA: * = h:/ACME/data/survey/ PROGRAMS: # = h:/ACME/pgmdir/survey/					
DATE	INPUT	PROGRAM	OUTPUT	N OBS	COMMENTS
6/1/07	*ACME_Survey.txt	#surv001	*ACME_surv001	12,346	SASify the ACME wave1 survey data. Look for duplicates.
6/1/07	*ACME_surv001	#surv002	Printout		Examine the duplicates.
6/1/07	*ACME_surv001	#surv003	*ACME_surv002	12,340	Remove duplicates.
6/2/07	*ACME_surv002	#surv004	Printout		Run preliminary formatted freqs and means. Check skip patterns.
6/3/07	*ACME_surv002	#surv005	*ACME_surv003	12,340	Apply recodes: Set legitimate skips to special missing values.

BEAUTY IN A BOX

Using a *Docbox* has several advantages, as you will see below. But whatever method you choose, it is useful to bear in mind a few general principles for creating a Docbox, or equivalent documentation of the data process:

Document as-you-go. If you wait until the end of the data process to do your documentation, it will likely be a painful chore, and you will probably just not do it at all. But if you document as you go, it's not so bad, and, in fact, has some collateral benefits. By taking a step back from coding and running your jobs, and really looking at your SAS logs, you might see an error or inconsistency you would not otherwise have caught. Or if you have to re-run the last few programs, you don't have to try to remember which programs to run, or the order to run them in -- it's all in your Docbox, and you can follow it like a recipe.

Step away from your code. There are certainly ways to help you automate the process of maintaining this kind of documentation, and many programmers I talk to have asked me if I have developed any macros or background processing that will grab internal comments and 'dump' them into the documentation. I prefer to do it manually, because it forces me to really examine the results of each of my programs, and to make sure that the results I'm getting are consistent with what I expected. I have developed some simple macros within WORD to simplify the data entry part of the task. The method you use is not important - as long as you do *something* to keep track of your data process.

Code is Truth! Documentation is not truth… it's only what you *think* you did and it's only as accurate as your diligence in maintaining it. Keep in mind that if you want to know the exact details of what you did, you will need to look at your code.

Organize the "stuff". As you progress through the many steps in your data process, you will likely accumulate lots of 'stuff' – both hardcopy and electronic. Printouts can be kept in folders or binders, and it is always advisable to maintain the SAS LOG and LIST together (including any JCL or system messages, if applicable). An isolated SAS LIST, without its associated LOG and file references, does not provide a complete picture of what was actually done. An additional organizing tool for hardcopy is a 'project bible', containing all descriptive documentation, including pertinent e-mail correspondence, memos, and programming specifications. It is useful to maintain electronic versions of Docboxes (or equivalent job summary documents) on a network, so that others can find and share this resource.

EMAIL: A MIXED BLESSING

How could we live without e-mail, at this point? But relying on email to provide a record of specifications and decisions is a dubious practice. Since multiple people may be contributing to a decision process, those communications typically live in separate email baskets, making it difficult to assemble all the contributions from a project team in any sensible order. If you have access to collaborative software, this is a great way to replace the invasive spread of e-mail messages. If not, it is important to develop and maintain a running history of decisions made along the way. This can be done simply using a single document that is accessible by the team.

2. PROCESSING: BATCH VS INTERACTIVE MODE

Choices you make in the data process affect the Big 'D'. Programmers have options when it comes to running SAS jobs: we can use the display manager, and run our jobs interactively, or we can use batch mode, where we submit a job and run it in the background.

Running jobs interactively provides several advantages: You get quick turn-around and immediate results. It is nicely suited to 'playing' with your data, to try out new things, and develop, test, and debug your code. But it has some severe disadvantages, too. Even if you religiously save the logs from these sessions, they will not necessarily provide an accurate record of what was done. For example, if you change some system options during the session, your final log might not show the options that were in effect the last time the code was submitted. Therefore, interactive mode is great for developing, 'playing', and testing, but when you are ready for a production run, use batch mode to ensure that your results can be reproduced.

Using the display manager in SAS isn't the only interactive application that can pose problems. Changing data in a spreadsheet is a common but treacherous practice as there is no record of the changes that were made. If possible, use SAS and a transaction file to apply changes to a dataset. For presentation purposes you can then put the resulting data into a spreadsheet. There is a lot of literature about spreadsheet quality (or lack thereof!). Although it does not speak directly to the issues of reproducibility, it is, of course, related and important. Readers are referred to an article by Philip Bewig (http://www.eusprig.org/hdykysir.pdf) that provides a detailed discussion of best practices for using spreadsheets.

3. PROGRAMMING GUIDELINES – STYLE AND CODING

For achieving reproducibility, issues of both programming style and coding are important factors to consider. *Style* refers to things that don't matter to the computer while *coding* refers to issues that DO matter. This does not imply that stylistic issues are less important. To the contrary, they can be an enormous help or equally large hindrance as you try to retrace your steps. Three stylistic practices are especially important:

DOCUMENTATION WITHIN CODE

It is fairly obvious that you want to document within the code itself—a practice so sensible that failure to do so could be considered a capital crime. Remember, you WILL forget why you did what you did and someone else (or you!) might need to examine or rerun your work. It is also advisable to put a comment box at the top of each program that contains all the fundamental information—name and author of the program, a brief description, and notes about modifications (note that these are the same things that go in a Docbox).

```
************************************************;
*   project:     MY_PROJECT
*   program:      MP003.SAS
*   path:        h:\projects\my_project
*   author:      E. Axelrod
*   input:       2008 project data file - raw data
*   ouput:       SASified 2008 project data
*
*   purpose:      SASify the 2008 raw data file.
*                Keep an extract of vars.
*                Check for negative reimbursements.
************************************************;
```

SAS listings should include titles and footnotes that provide a concise description of the output. You can make your code 'self-documenting' by using the 'get option' function to retrieve the program name and pathname of your batch-submitted code, and include this in footnotes or titles. For example:

```
%let PROGRAM = %sysfunc(getoption(sysin));
TITLE1 "brief description of program goes here";
FOOTNOTE "&PROGRAM - &SYSDATE9 - &SYSTIME";
```

This code will place the full pathname and program name in a footnote, along with the date and time the job was run.

READABILITY AND APPEARANCE

It's more than just a pretty face: Code that looks good is easier to follow, debug, and maintain. Of course to a certain degree, style is a matter of personal preference, but here are some examples of basic guidelines that are generally accepted as best practices.

Use one line per statement. This code:

```
data two; set one; var1=x+y; label var1='This is hideous code' x='truly repulsive'
y='makes me sad'; run;
```

… is much easier to read when it looks like this:

```
data two;
    set one;
    var1 = x+y;
    label  var1   ='This is much nicer looking code'
           x      ='It will be easier to debug'
           y      ='This code makes me happy'
           ;
run;
```

Indenting nested loops and if/then//else statements also greatly improves the readability of your code, and can help in debugging.

```
data two;
    set one;
    by ID;
    if ID='blah-blah-blah' then do;
         if begin_dt = '31Dec2003'd then do;
              /* do something here */
         end;
    end;
    if last.ID then
         /* do something here, too */;
    else
         /* do something else here */;
    output two;
run;
```

NAMING CONVENTIONS

Using consistent naming conventions for variables names and array names also improves readability. A nice way of distinguishing an array name from a variable name is to preface your array names with the underscore:

```
array _newvars  {*} new1-new10;
array _oldvars  {*} old1-old10;
```

A good practice is to confer with your project team to come up with naming conventions for variables, labels, and file names that you're all willing to follow.

CODING ISSUES

SAS is a powerful tool that can provide elegant results but it can also go badly wrong in the hands of the uninitiated. There are many best practices for coding that go beyond the scope of this paper, but here are a few guiding principles that will not only minimize mistakes at the keyboard, but also ensure that your code will be easier to follow, debug, and maintain:

- Use Loops and arrays for repetitive processing to minimize the typing necessary and to ensure that changes only have to be made in one place.

- Using macros for repeated code serves a similar purpose in reducing the amount of typing and allowing you to parameterize items that might change for various iterations of your program. (Note that this advice is for the robotic duplication of executable code…. not for writing documentation or comments, as described above, which relies on your thoughtful participation.)

- Minimize hard-coding. Again, parameterize or use macros and put them near the top of your program if possible.

4. FILE STORAGE

OK, you've got all your stuff: your data, your programs, and your documentation. Now where are you keeping it all? Your properly backed up hard drive may be fine for storage if you are the only person accessing these files. But more likely, your work will need to be shared with others, in which case, network storage, or some other sharable device is the obvious choice. Have you ever tried to find a document in someone else's directory? Unless there is an agreed-upon structure, everyone will come up with their own method of (dis)organization, making it very difficult to find things. Designing and adhering to naming conventions is not only advisable for variable names, but also for path names and directory structures. Work with your team to develop a system that will make your data, programs, and documentation easier to find.

CONCLUSION

Follow the simple principles outlined in this paper and you will leave a trail of crumbs behind so that you – and your team – will know how you got here from there, and can find your way back again. If you have written your code in ways that make it easy to follow, debug and maintain, you'll be in good shape when it comes time to test the quality of your code and results.

REFERENCES

Axelrod, Elizabeth. "Data Quality and the Big 'D'. Proceedings of the NorthEast SAS® Users Group Conference (NESUG) 2003, September 2003.

Bewig, Philip. "How do you know your spreadsheet is right? Principles, Techniques, and Practice of Spreadsheet Style. July, 2005. http://www.eusprig.org/hdykysir.pdf

ACKNOWLEDGMENTS

The author wishes to thank Joan Mullen, Richard Sheppe, and Paul Grant for their editorial contributions to this paper.

RECOMMENDED READING

Levin, Lois. "SAS® Programming Guidelines". Proceedings of the Thirty-First Annual SAS® Users Group International Conference. March 2006, http://www2.sas.com/proceedings/sugi31/123-31.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Elizabeth Axelrod
Abt Associates
55 Wheeler Street
Cambridge, MA 02138
Phone: (617) 349-2458
E-mail: elizabeth_axelrod@abtassoc.com