RULES WE FOLLOWED AND WISH WE HAD FOLLOWED IN MANAGING DATASETS, PROGRAMS AND PRINTOUTS

K. E. Muller, J. Smith and D. H. Christiansen, Univ. of North Carolina, Chapel Hill

ABSTRACT

This paper presents rules used for managing datasets, programs and printouts in a SAS-based data analysis project. First, the limits and nature of the type of project are considered The rules discussed have evolved in working with very small projects to medium size projects. A number of definitions are included to help specify the problem. File naming conventions are discussed in detail. Although trivial to implement, adherence to naming rules provides a sound basis for high quality documentation. Rules for printout management, keyed to program rules, are presented. Only a small amount of time is devoted to discussing programming style and strategies. A few simple programming rules require little effort to follow, but automatically provide some useful documentation. Finally, archiving and documentation (in the traditional sense) are discussed briefly. Both can be done in a straightforward way if the rules discussed earlier have been followed.

CONTEXT OF THE PROBLEM

Datasets, programs and printouts grow like well watered vegetables and weeds around any data analysis project. In this paper, we shall try to present rules for managing them in a SAS-based data analysis project. In order to do that, we shall first discuss the nature of the project to be considered. We shall discuss naming conventions in detail. Strict adherence to naming rules provides the basis of our approach to high quality documentation. We shall also discuss rules of printout management, keyed to the naming conventions. A few simple programming rules will be discussed which can be followed with little effort but which are important in producing useful documentation.

We address ourselves to what we consider a very common research environment. We would say that the type of research project considered is a middle sized project. We consider anything with fewer than one hundred observations to be a very small research study. A study involving up to one thousand observations may be a small study. The middle range includes studies of one to ten thousand observations. A study involving more than one hundred observations may be considered a large study. Certainly more than one million observations implies a large study. Project costs could also be used to classify projects. Useful cutting points might be $1000, $10,000, and $100,000. Staff size could also be used. Cutting points might be one person, ten people, and fifty people.

The projects in the middle range face a unique problem. Small studies are so small that typically one person has complete command over everything associated with the study. The very large study has the advantage of economy of scale. Explicit allocation of money and time is made toward documentation. Only the person in the middle is short-changed. Enough data is present to let things get out of hand, but not enough to support a "Department of Documentation." We think, therefore, that the middle-sized study is likely to be the most poorly managed with respect to datasets, programs, and printouts. Since this middle range study involves few people, and a relatively small budget, any efficient technique must be easy to use and not consume a great deal of time. Furthermore, it should not require too much police action on management's part if it is to succeed.

DEFINITIONS

Some definitions are necessary in order to ensure we share a common vocabulary. Since we restrict our attention to a SAS environment, we implicitly restrict ourselves to a large IBM computer, with some flavor of an OS based system. By "tape" we refer to a nine-track tape, typically 1600 or 6250 BPI density. Tapes are usually IBM standard label tapes. When we speak of a "disk pack" we refer to some variation of either a 3000 series (such as a 3330 or 3350) or a 2000 series (such as a 2314). A "datum" is any recorded observation, the "elementary particle" of data analysis. A "dataset" is any rectangular array of data. This reflects a research data analysis and SAS bias. Observation units such as subjects are often called "records," and form rows of the array. Columns of the array correspond to variables. A "variable" is a data item which is identified by a symbolic name and holds a value for each observation. Each unit has values for all variables. The number or character string recorded is the "value" of the variable for that unit. An important particular value is "missing." By "EBCDIC file" we mean a collection of raw data or source program, in an industry compatible form. This may be a card deck, a tape file, or a disk file. A "SAS dataset" is any data matrix which is in the internal SAS system format. A "SAS data base" is an operating system file (OS file) which contains one or more SAS datasets.

"Record" may refer to either the logical record size for a file or the collection of raw file logical records accessed by an input statement to produce one record in a SAS dataset. Note that the definition of "record" varies between data analyses. A "datastream" is a logically unified collection of data typically arising from a single common source or arriving at data processing as an intermingled collection. "Data generation" is an important concept. We feel a new generation is produced whenever any of the following six things occur: 1. any record is to be deleted, 2. any record is to be added, 3. any value is to be changed, 4. any value is to be added, 5. any files are to be merged or 6. any files are to be generated via splitting. A "source program" is any colections of statements which produces a complete run, a "job," on the computer. This may be any mix of JCL and source languages such as SAS or PLI. A "source file" is an operating system file whose data happen to be programs. Examples include card decks and TSO files. Data, of course, may be inside that source file. For us source files are industry compatible coded files, and hence exclude object modules and load modules.

NAMING CONVENTIONS

If we have had any success in managing our data analysis projects, it is because we have been very systematic in following naming conventions. We think that naming conventions should apply to everything. That includes disks, tapes, source files, raw data files, SAS files and job names. We do not consider the rules to be presented special in any single way. Any alternative rule system that will be used and retains the advantages inherent in the ones we use is acceptable. We do not accept replacing any of these naming conventions with no rule. A constructive alternate must be used. A rule for naming is required for each element of the data analysis path.

First we list some general rules that apply to all sequences. Then we mention each type of sequence that should exist in our system. Our naming system depends on the use of sequence numbers. Hence every name ends in a number of at least two digits.

A simple example is disk pack volume names. We operate with two disk packs for one particular project. The volume serial names are UEPA01 and UEPA02. Volume serial names are limited to eight characters. Our computer center requires that the first character be "U." The project involves research for a Federal EPA contract, hence the "EPA" stem. The last two characters are the disk pack number indicating simply their order of acquisition.

Naming conventions for tapes are simple for us. We have no choice, the computer center defines them. However, we have not exploited sufficiently the header label which is available. Presently we only include a textual description of the use of the tape. We should include our own serial number information.

File naming conventions for all files are an important part of our approach. We have rules both for the DDNAME and the DSN value for the DD statement. We shall consider the DSN specification first. We shall present two alternate naming conventions. Any project, and probably any shop, should follow just one of them. Example DSN's that could be used are
UNC.EP.F273S.MULLER.CHESS.SAS.DB01,
UNC.EP.F273S.SMITH.OAK.TAPE.DATA.FILE01,
and UNC.EP.F273S.MULLER.KIDS01.CNTL.
Our computer center requires a fixed initial stem for each dataset set name which contains account code information. In the first example this is UNC.EP.F273S.MULLER. The programmer name is required next. We suggest that one name be used for the entire project and suggest a fixed project mnemonic. In the first example it is CHESS. From the mnemonic on out the naming conventions can have a number of different options. One reasonable one is to have the third element be one of three things: TAPE, DISKON or DISKOFF. This indicates the device where the file resides. With this convention the next or fourth element indicates the type of file. For us this only has the values SAS, DATA or null (for source program files). The next piece gives the sequence number. For SAS databases this has to be of the form DB01. For raw data this should be FILE01. For a program series on children's data this might be KIDS01, as in the third example. For source files we add the extension CNTL. Only with the source file is any choice available to the programmer as to name. Even this should not change too often, perhaps not at all for a project. Otherwise the system is defeated. Obviously, if the project is large enough and involves enough people, allocation of the names must be done by someone designated as a data coordinator. Our target here is the project involving a few programmers. Any larger system allows time for such explicit management behavior. A small shop such as we are interested in can work with a system of this sort conveniently.

SAS dataset names also should be

chosen in a reasonable way. Here we have two possibilities also. One possibility is to follow the DATA01, DATA02, format. Note that we do not think one should ever use the default naming feature of SAS in this respect. Using names of this sort requires the user to use the excellent documentation facilities in SAS to indicate what is inside the dataset. A second reasonable alternative for naming SAS datasets is to use a mnemonic stem followed by a number of at least two digits. This might be of the form BADGE01, BADGE02 or MALE01, FEMALE01.

We also think that DD names should be specified in a consistent manner. Again we present two alternatives. The first option is to use the last level of the dataset name as the DD name:

```
//DB01                    DD
DSN=UNC.EP.F239.TAPE.SAS.DB01,DISP=SHR.
```

A second reasonable alternative is to use a name which shows that only input is allowed on the dataset: //INPUT01 DD DSN=... With this convention //OUTPUT01 DD DSN=... indicates that only output processing is being done. Furthermore //INOUT01 DD DSN=... indicates that both input and output are done with the dataset. This may seem trivial, but in looking back at programs run even a week or two weeks earlier, we have found that this has been a great help in finding what happened in the job. Here again one should always append a two-digit number to the name. Note in all cases where we present two options the user should use just one consistently throughout the entire project or throughout an entire shop.

Perhaps the most important data management technique that we can talk about is the naming of jobs. An example job name is KIDS0101. The job would be stored in a TSO file with the name UNC.EP.F239R.MULLER.KIDS0101.CNTL. The larger the number of programs and the longer the interval between the beginning of a project and the time to access the datasets or printouts, the more we appreciate the advantages. Since we work in a TSO environment, we coordinate job names with source file names. Our computer system generates the accounting stem with the name of the programmer as in the examples of DSN's. All subsequent characters are optional to the user.

The job name follows the programmer's name, and is followed by an extension which indicates that the file is a program. Again the project stem, KIDS, is constant. We've found that the job name convention seems more helpful than anything else we do. The first four characters are a mnemonic indicating the datastream or the project, such as KIDS in the example. The last four characters are numbers in our scheme. We refer to the first two as a "series number." These two digits can change for a variety of reasons but in all cases we use them sequentially and exhaustively. We don't ever number 1, 2, 3, 5, but would always number 1, 2, 3, 4. In all cases then one knows that the highest number appearing is the last, the lowest number appearing is the first, and all programs have been run absolutely sequentially. We know no valid reason to circumvent the rule. If tasks must be done concurrently, then the two tasks deserve different series or perhaps even different project names. A series number can be changed for a variety of reasons, and if in doubt, change it. This is purposefully vague to give flexibility to the programmer. If 99 series are too few, a three digit series number could be used. Reasons for changing series include: 1. a task is completed or a major milestone is passed, 2. programmer responsibility is changed or 3. sequence numbers become too large. We refer to the last two digits as the "sequence number." Hence KIDS0101 is the very first program in the first KIDS series and KIDS0102 is the second program in the series. In general we feel we should change the series number after, at most, 10 to 20 programs have been run. In our experience this coincides rather nicely with the completion of a task for a major milestone in processing. When a new series is started, one implicitly is saying that the preceeding series will not be accessed except for information purposes. This may sound harder to achieve than it is. It actually is quite easy if one follows the rule of naming jobs in order.

ARCHIVING FILES

When a new series is started, the collection of programs in the preceding series should be archived. A very simple and easy process we use is to simply define a new file whose name ends with the series number. If the first program were KIDS0101, then we would collect all of those programs into a parent file and call it DSN=UNC.EP.F239R.MULLER.KIDS01.CNTL. The series name constitutes the parent file name. A good technique would be to let the parent file be a partitioned dataset with the members of the series being the members of the partitioned dataset. Our present operating and utility software makes this inconvenient and hence we have not done that. A good hierachal file structure, such as is available in UNIX (c) would certainly facilitate such an approach.

Sequential job name/numbers of this sort turn out to be wonderful. They have no character such as HUNT, GARBAGE, BONZO, NAPALM or any of the other fun names we all enjoy dreaming up. If at any point one makes a mistake then one can simply rerun all jobs with a number equal to or higher than the job number where the mistake was made. That is a beautiful characteristic. Error recovery is straightforward. The file naming conventions and the job naming conventions we present here automatically allow backtracking and avoid secondary errors.

In our opinion, one should have a separate series which does all file copying, archiving and creating. We usually use a separate job to create an empty dataset, then write to it with DISP=OLD. This allows rerunning the program without changing the JCL. We catalog all datasets at time of creation.

We keep all files within a series online whenever a series is being used. Once the series seems to be complete, we gather it into a parent file and archive it. Typically archiving for us involves copying to offline tape and an offline disk pack so that we have two copies. Once we have assured ourselves we have two copies, we then destroy the online copy. The online files are backed up regularly by the system. The goal of course is to never be caught without at least one copy, even after having made a very dumb mistake.

An important comment need to be made about managing programs. A cliche exists: "Never scavenge a deck." People who know it treat it as gospel, and those who don't eventually will or will continue to pay the price. In an interactive context the rule may be translated as "Never edit a program that produced output which was kept." Even if only one card is changed, create a new file or deck. Even if four programs differ only by one card, have four programs. If large blocks of code are reused store the code as a macro library. This is easy and convenient for both card and interactive systems. Also, if an output is not kept then the source file should not be either. Obviously a balance should be kept between being too neat and throwing away things that are later useful. A recycle stack can be used judiciously as a first in first out buffer.

PRINTOUT MANAGEMENT

This brings us rather naturally to the issue of printout management. Here we have a relatively simple and perhaps more primitive approach. Any printout that is kept must be associated with the source and not just any source but the exact source that was kept. Such an approach is easier in an interactive system where one doesn't end up with cases of cards. A number of advantages accrue. First, if one has made an error it is easy to go back and rerun all of the appropriate jobs. The process of scavenging need not be duplicated. Third, it's the ultimate documentation. If one has the exact source then one can easily recreate the analysis without even understanding the original program! We are involved in environmental research which is closely scrutinized work. The style of management we have suggested has made it easy for us to respond to requests such as subpoenas.

Any printout that is kept is placed in a binder. Typically we include all members of a program series in one binder, in numerical order. In accessing such a collection of binders it is obvious which programs ran first, and which programs depended upon previous programs. It is a very friendly system, and the friendliness is derived from those seemingly trivial numbering and naming schemes. The programs may be terrible but if they are it is easy to find where they are and easy to see the flow. We are dealing with data flow, information management, a concept somewhat broader than the idea of data management.

That the printouts are in binders is typically not enough. Minimally, a separate documentation text file is created. We include a description of every file used as input or created by any program, a list of the programs, and locations for all copies of all files. We include this file on every archive tape or disk pack. Hence finding any copy of the master documentation file allows complete reconstruction of the entire project and data analysis, from any intermediate point, as long as at least one copy of the files needed exist and the software is still available.

It may be very helpful to produce a program run notebook. Ideally, one would have one sheet per program, with a standard mimeograph form being used. It should contain, at least, specification of the input file, specification of the output file, some simple description of the dataset, such as number of observations, job name, a job task description, programmer name and date.

PROGRAMMING STYLE

Programming style deserves some mention. First, take advantage of the protection available in JCL. A useful rule is to always use DISP=SHR on a file except when one explicitly must write to a dataset. If one does make a mistake

and accidently tries to write to the wrong dataset, this will protect the programmer from him or herself. Secondly, simply looking at the JCL is then sufficient to indicate to a subsequent reader that the program does no output. If one is interested in determining where changes have been made to files it can be very helpful.

We use TITLE statements to help implement our numbering system. Job names are included at the end of the TITLE1 statement in each job. TITLE2 (or the bottom most title) should have information describing characteristics of the pieces of the job. File name information in titles can be quite useful.

Another programming style suggestion is to use macros extensively. We use macro libraries in a primitive way by storing them in a file and concatenating them to SYSIN. We define all macros at the beginning of the program, which helps reading the program.

Variable names and variable labels stand somewhere between naming conventions and programming techniques. First, one should use long informative variable names. Second, one should include a LABEL statement, a LENGTH statement and a FORMAT statement with every variable when the variable is created. Consequently, the information will follow the variable throughout its life. In doing so, one again takes a step in the direction of self documenting programs.

CONCLUSIONS

It would be easy to dismiss our suggestions as trivial. They are trivial in the sense that they are simple to understand and easy to implement. Discipline is required since it takes a willingness to go back and rerun programs that do not conform. Even a wrong title statement requires a new run. The payoff cannot be overemphasized. We have used most of the rules extensively over the last few years and find that we can still go back and access programs with understanding. We have not talked about documentation within the program itself. That is an entirely separate issue. We have discussed documentation with the outer shell of the program, and we think that it has been overlooked, and greatly undervalued.