SELF-DOCUMENTING PROGRAMS AND DATA SETS
J. Meimei Ma, Quintiles, Research Triangle Park, NC, USA

ABSTRACT

Have you ever wondered what a program is supposed to accomplish? Or what a variable named X3 represents? This tutorial explores easy methods for avoiding such situations. You can easily create self-documenting programs and data sets when using the SAS® system. In the era of object-oriented programming, the basic principles of structured programming still make sense.

The target audience for this tutorial includes programmers, data managers, statisticians, and anyone else who writes SAS program code to create SAS data sets. Only basic knowledge of the SAS system is assumed. Examples of good and bad documentation choices are provided.

The presentation is divided into three sections. We begin with reasons for documentation and definitions of relevant terminology. Next, documentation strategies for programs are discussed. The final section covers how you can design SAS data sets that are easy for everyone to understand. The discussion is platform independent.

DOCUMENTATION

The reasons to invest time in documenting programs and data sets are simple. Even basic documentation will

• simplify and enhance communication
• allow validation
• provide implicit backup for programmers
• reduce frustration.

The benefits apply equally to projects with one person or a team of twenty people. Every project requires understanding the purpose of programs and data.

Terminology

For the purposes of this tutorial, let us define a few terms.

To start with, what do we mean by self-documenting? A self-documenting program contains enough information embedded in the code for a programmer to easily understand the program's purpose, logic, and outcome. A self-documenting data set contains basic information required to use the data. When you are working with a self-documenting entity, the level of internal documentation is high and the information provided is self-explanatory.

We will discuss two types of documentation: program descriptions and data descriptions. However, even a self-documenting program may require further documentation before you have a complete program description.

Additional documentation that is not stored in a program or a data set is called external documentation. For example, you may write a description—in English—about the overall logic of a series of programs or the relationships between a collection of data sets. Outlines, diagrams, or flowcharts are common types of external documentation.

The elements of external documentation most closely related to our topic are "system names." These include LIBNAME or FILENAME references; names of directories used for a project; or names for the files that contain code, output, or logs. Naming conventions for these items will not be discussed because they often depend on the computer environment.

Needless to say, internal and external documentation should match.
DOCUMENTING PROGRAMS

Consider the program in Figure 1. What do you think it would do?

```
libname what "c:sasdata";
proc sort data=what.x; by x2;
data stuff;
merge what.x what.zi by x2;
nex=x1-z;
if nex>=0 then s="Y"; else s="N";
proc freq;
tables x2*s;
run;
```

Figure 1. Mystery Program

Needless to say, this program does not follow structured programming guidelines. Before object-oriented programming (OOP), many basic computer courses stressed the objectives of structured programming described by Kernighan and Plauger. In short, a structured program is designed to be easy to understand and easy to modify. You achieve these goals by using modular logic and internal comments. Creating modular programs is easy in the SAS System with the appropriate mix of DATA steps and procedures. (The addition of the RUN statement made it even easier.)

To create self-documenting programs, start by deliberately developing standards for programming style, especially naming conventions and comments. All programs for a project should follow the same rules.

Programming Style

Programming style can be divided into two areas: form and content. Standards for form cover visual cues and certain aspects of naming conventions. Content standards cover logic guidelines, naming conventions, and testing strategies. We will concentrate on form and naming conventions.

The visual cues in a program enhance what you can glean about the program's intent. Look at Figures 2a and 2b. Which makes more sense? Do both programs do the same thing? What is different in Figure 2b?

```
data temp; set survey;
if area=1 & .<age<=25 THEN group=1;
else IF area=1 & age<98 then group=2;
ELSE if area=1 & age=98 THEN group=.A;
else if area=2 then group=3;
label age="original age"
group="analysis category";
PROC CONTENTS;
proc print data=survcode(obs=10);
run;
```

Figure 2a. Survey Program Without Style

```
DATA survcode(LABEL="Coded Surveys");
SET survey;
*
Create location group;
IF area=1 /* urban */
THEN DO;
IF 15<age<=25 THEN group=1;
ELSE IF 25<age<=97 THEN group=2;
ELSE IF age<15 OR /* invalid */
age=98 THEN group=A;
ELSE IF age=. OR /* unknown */
age=99 /* illegible */
THEN group=;
END;
/*rural*/
ELSE IF area=2 THEN group=3;
FORMAT group groupf.;
LABEL age = "Age from survey form"
group = "Analysis group";
RUN;
PROC CONTENTS DATA=survcode;
RUN;
PROC PRINT DATA=survcode(OBS=10);
run;
```

Figure 2b. Survey Program With Style
Any form standard should force programmers to take the time to

- use blank lines
- follow an indentation standard
- use consistent spacing
- use consistent capitalization (case).

Names in Programs

You should choose names for data sets and variables carefully. The more you can develop a habit, the easier it will become to invent self-documenting names. Of course, you should follow any existing standards regardless of your own preferences.

Data Set Names

Data set names always matter, whether the data set will be temporary (WORK.) or permanent. Avoid using the default provided by a procedure. Provide explicit DATA= information for all procedures. (Placing PROC CONTENTS; after creating a data set, which will change from temporary to permanent after testing, may be an exception.)

When using WORK data sets, resist naming them TEMP1 and TEMP2. Never depend on the default. Descriptive names help you and others understand the program's logic quicker.

If you are defining a new naming convention, think about expandability. Will the convention handle 10 data sets gracefully? 100 data sets?

Permanent data sets should have a label in addition to a descriptive name. Remember that labels can be 40 characters, while names are limited to eight. The best way to label a data set is to use the LABEL= data set option in the program that first creates the data set. For an existing large data set, you may prefer using the DATASETS procedure to add a data set label to avoid recreating it.

Variable Names

A program may contain several different types of variables. For instance, you could have variables that are

- temporary, in a WORK. data set
- temporary, dropped after being used
- permanent
- for macro purposes.

As with other names, make variable names descriptive. Think about whether using different conventions for the different variable types is worthwhile. An example of good and bad variable names is shown later.

A programming consideration for variable names is how easily lists of variables can be referenced. For example, most programmers prefer QUEST_C1-QUEST_C5 instead of QUEST1C, QUEST2C, QUEST3C, QUEST4C, QUEST5C.

Any list of variables should be lined up for easy viewing. This means that Line 1, shown below, is better than Line 2.

```
Format agecat agecatf.
quest1 ynf.
quest5 agreef.;
```

```
Format agecat agecatf. quest1 ynf.
quest5 agreef.;
```

Using an abbreviated variable list for a sequentially numbered set of variables adds clarity, but in most other situations explicit variable lists are better in the long run. A name range list based on position in the program data vector that looks like

```
name--quest01
quest02--NUMERIC-quest20
```

makes it difficult to know which variables are being referenced. For example, CITY would be used only if it occurred between NAME and QUEST01.
Useful SAS Features

The SAS System includes features that make documenting programs easier. Consider the following statements or options

- %LET=
- &sysdate, &systime
- PUT statement for SAS Log
- KEEP/DROP
- RENAME.

Thinking about how to use the macro facility to help documentation is beyond the scope of this basic tutorial. However, the %LET= statement deserves mention as a quick method for documenting code and output at the same time. For instance, you could define a project identifier at the beginning of a program that would also appear in the first TITLE line. Batch programs may benefit from adding &sysdate or &systime to a FOOTNOTE.

You can use KEEP/DROP or RENAME statements or data set options to keep DATA steps and data sets "clean." Drop stray variables even from temporary data sets for clarity.

Comments in Programs

You will always benefit from putting comments into your programs. Following a consistent format simplifies the process. The trick is to avoid asking for too much and to resist the temptation of allowing too little.

For one program, the basic types of comments are

- a header block
- major section descriptions
- minor section descriptions
- in-line details
- TITLEs and/or FOOTNOTEs.

The header block answers the questions: why, where, what, who, and when. This set of comments should follow a template and appear at the beginning of every program. In certain environments, the header block is called a documentation block. See Figures 3a and 3b for examples of two different styles of header blocks. Figure 3a shows the minimum information needed in any header block.

Two methods exist for putting comments into SAS programs. As shown in Figure 4, the statement (*message; ) may be used for section descriptions. In-line details are handled with slash-star pairs, which can replace any blank. Because slash-star pairs allow embedded semicolons, they may be used to "comment out" sections of code used only for testing purposes. Notice how names, TITLEs, and comments combine to provide internal documentation in Figure 4.

Look again at the mystery program in Figure 1. Can you see that the program in Figure 4 accomplishes the same task? Think about how the blank lines and descriptive names make it easier to understand the logic.

```
*FUNCTION:  Create final grades
*SOURCE:    c:\subjarea\program\c101ex04.sas
*INPUT:     c:\subjarea\sasdata\c101ex95.sd2
*           c:\subjfinl\sasdata\allc101.sd2
*OUTPUT:    c:\subjarea\sasdata\c101_f95.sd2
*NAME:      M. Ma
*DATE:      03MAR96
*;
```

Figure 3a. Header Block Example A
BEGINNING TUTORIALS

---

**Program Name:** c101ex04  
**Title:** Create final grades  
**Description:** Merge 1995 class grades with overall grades  
**Directory:** subjarea for c101 on Department X server  
**Input:** Class grades and overall grades  
**Output:** Merged grades for C101 in 1995  
**Date Created:** 03/03/1996  
**Requester:** B. Calvert (course instructor)  
**Programmer:** M. Ma  
**Modification:** 03/04/1996 M. Ma  
**Revise variable labels**

---

Figure 3b. Header Block Example B

```sas
libname FILE1 "c:\subjarea\sasdata";
libname FILE2 "c:\subjfin\sasdeta\d";
title "Check Grades for Course 101 - Fall";
proc sort data=FILE1.C101EX95 out=FINAL; by examsect;
run;
data DIFFS;
merge FINAL FILE2.ALLC101;
by examsect;
   scordiff = score - median; /* difference */
   length above $1;
   if scordiff >= 0 then above="Y";
   else above="N";
   label above = "Above Median";
output;
return;
run;
```

**FOR CHECKING**

```sas
proc contents position;
title2 "Check of DIFFS";
run;
proc freq data=DIFFS;
tables examsect * above;
run;
```

Figure 4. Self-documenting Program Example Without Header Block
DOCUMENTING DATA SETS

Suppose you receive an unknown data set and the CONTENTS procedure gives you the information shown in Figure 5. What can you guess about the data? What do you know?

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR3</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>FROM</td>
<td>Char</td>
<td>20</td>
</tr>
<tr>
<td>VAR1</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>VAR2</td>
<td>Num</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5. Mystery Data Set

The major elements of internal documentation for a data set include:

- data set name and label
- variable names and labels
- variable formats and informats.

Ideally, you want to have information about data sets that readily let you answer audit questions about when, who, why, what, and where. Understanding how the data set relates to other data usually requires external documentation.

CONTENTS Procedure Information

The CONTENTS procedure is the primary tool for reviewing the internal documentation stored with a SAS data set. Let us divide the information into four categories:

- data set, system-generated
- data set, user-defined
- variables, system-generated
- variables, user-defined.

Consider the standard information provided by CONTENTS, as shown in Figure 6. (Engine/Host information is not shown.) For the data set itself, the system provides information such as Member Type, Engine, and Last Modified. The user-defined information is the Label. For variables, Position can be thought of as partially system-generated (column numbers) and partially user-defined (order). The other information about variables is user-defined in the sense that the source can be found in a SAS program somewhere.

The special information that help SAS data sets be self-documenting include:

- number of observations and variables
- sort status summary
- index status summary
- compress status.

Now compare Figure 5 with Figure 6. How much difference do the descriptive names and labels make? Which do you prefer? How does the listing by position help?

Naming Data Sets

Always name permanent data sets deliberately. You want to define names that are easy to understand. Follow existing standards. However, if those rules are too general then go further as needed. For instance, you can improve on a standard that only specifies that data set names should be seven characters long. Perhaps the team should consider beginning a set of related data set names with a consistent prefix.

An important consideration is how the data set names will sort alphabetically. For example, remember to account for leading zeros when numbering data sets sequentially. How would DESC1, ..., DESC12 sort compared to DESC01, ..., DESC12? Sort order may seem unimportant for a project with five data sets, but what will happen when it grows into a project with fifty?

The features worth reading about in the SAS manuals for naming data sets are:

- LABEL= data set option
- CONTENTS, _ALL_ option
- DATASETS procedure.
Naming Variables

When you name variables, the objectives are essentially the same as for data set names. However, you usually have many more variables than data sets, which means unclear names cause confusion much quicker. A project with a good variable naming standard has the following characteristics.

- Variable names are easy to understand.
- The naming convention is easy to remember and follow.
- All project data sets contain consistently named variables.
- Enough flexibility exists to allow for new situations.
- A written standard is available for quick training of new people.

Naming Conventions

Note that the existence of a standard does not guarantee that all variable names are consistent. If the standard is viewed as impractical then few people will follow it. A balance must be struck between forcing standardization and allowing programmers to work creatively.

Any variable naming convention must adjust to certain realities. For instance, variable names are limited to eight characters in Version 6.

You may need prefixes or suffixes to organize large collections of variables. The CONTENTS procedure sorts variable names alphabetically; most data dictionaries are sorted alphabetically. For example, variable
names of the form jJDESCnn might look like DENAME, DEADDR1, DEADDR2, DECITY, FAQUEST1-FAQUEST5, FBSCHL1, FBSCHL2 for a project that had three data collection forms (DE, FA, FB).

The best way to choose variable names and labels is to consult a representative of every group of people that will ever use the data. Of course, this is rarely possible. What is important is that you account for different viewpoints when naming variables. A name that is obvious to a programmer may be less useful to a statistician who works at another company. Labels have become even more important as more non-programmers access data using SAS/ASSIST®, SAS/INSIGHT®, or customized interactive applications. Do not ask non-programmers to define names for a long list of variables. Just ask them to review proposed variable names carefully.

Variable Name Example

Here is an example that illustrates how consistency and sort order make variable names easier to understand. Suppose a data set will contain variables for the name, birthdate, and various phone numbers of people being surveyed. Compare the following two sets of variable names, which are sorted alphabetically.

<table>
<thead>
<tr>
<th>Set A</th>
<th>Set B</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIRTHDAY</td>
<td>DATB</td>
</tr>
<tr>
<td>NAMEFIRST</td>
<td>FAX</td>
</tr>
<tr>
<td>NAMELAST</td>
<td>FIRST</td>
</tr>
<tr>
<td>PHONE_W</td>
<td>HOME</td>
</tr>
<tr>
<td>PHONE_H</td>
<td>LASTNAME</td>
</tr>
<tr>
<td>PHONE_F</td>
<td>WORK</td>
</tr>
</tbody>
</table>

Which set of names is more obvious? Less ambiguous? How would these names work for a data set with 15 variables? With 150 variables?

Note that the names in Set A tend to be longer then those in Set B. You get eight characters; do not waste any.

Useful SAS Statements

The statements that attach documentation to variables also serve to document the program that creates the data set. Document permanent variables when they are created with either an ATTRIB statement or LABEL, FORMAT, and LENGTH statements. Of course, not every variable requires an explicit format or length.

For temporary and permanent variables, use the DROP/KEEP statement or data set option to help eliminate unnecessary variables. No one likes to see variables called I and J floating around. If you do not see a label in CONTENTS output then review whether the variable should be kept at all.

Permanent Formats

Attaching formats to variables in permanent data sets provides additional internal documentation but may cause problems unless external documentation exists, e.g., a data dictionary with format values. Naming the formats presents another standardization challenge, which will not be discussed here.

Permanent formats link variables to external documentation and aid standardization across a project. Obviously output from procedures is easier to understand when people see "Yes" or "No" instead of "1" or "0."

The disadvantages of permanent formats become more obvious when data sets are transferred. Then a data set no longer can stand alone. Without the associated format catalog, you cannot even print out a few observations without getting an error message. Quality control listings may be unwieldy if the formats are significantly longer than the unformatted values. However, you can work around these potential problems. You can create fake formats (VALUE 1='1';) and/or remove formats temporarily with "null" FORMAT statements.
Organizing Variables

Creating a self-documented set of variables takes time. But in the long run you may actually save time. You will certainly help someone (perhaps yourself) avoid frustration if the project lasts longer than six months.

Consider whether the position of the variables is significant. Not only can you get a listing from CONTENTS by position, other procedures use the internal order as a default. For instance, a PRINT procedure without a VAR statement will print the variables in the order in which they are stored. This order is determined by the DATA steps that lead to the creation of the data set. You can force the order to change by putting a LENGTH statement that explicitly names every variable before a SET statement. However, this is impractical when you have lots of variables. If position matters, then design DATA steps accordingly. (Hint: learn how the program data vector gets built.)

Defining labels for variables will be easier if standards or guidelines are created first. Think about capitalization (all uppercase or mixed case) and acceptable abbreviations. Find out if users will be selecting variables based on variable names or labels. You might ask "Will the alphabetical sort order of labels matter?"

Assorted Hints

Here is a short list of other ideas related to self-documented data sets.

• Use output from CONTENTS to generate basic data dictionaries.
• Use DATASETS cautiously. You want to have an audit trail for changes.
• Look up the FMTLIB option of the FORMAT procedure. (You will like it.)
• Look up the YEARCUTOFF= system option if you worry about dates after 1999.

CONCLUSIONS

Questions to Ask

When you want to decide whether or not your programs and data sets are documented properly, ask the following questions.

• How easily can I change program logic?
• Are the comments and labels still correct?
• How long would it take to explain this to ... [name of a colleague]?
• Have existing standards been followed? (Yes, no, not exactly because...)
• What are the other programmers doing?
• Will this make sense six months from now? Two years?

Self-Documenting Projects

The advantages of having a habit of developing self-documenting programs and data sets are

• more efficient programming, long-term
• easier archiving
• better team communication
• less frustration.

If you think thorough documentation is unnecessary or too time-consuming, try to broaden your viewpoint. Imagine a project with 10 programs and two data sets that lasts six weeks. Now imagine 100 data sets, a team of 10 people, and a six-month timeline. Finally, what about a project with 1000 data sets, hundreds of programs, a team of 50 people, and a budget that lasts six years?

Invest the time required to create

• naming conventions that people use
• enlightening comments in programs.

In many cases, programs and data sets take on a life of their own. Poorly documented programs will come back to haunt you. Carefully labeled data sets are a joy to receive.
RECOMMENDED READING

Books or Manuals


User Group Proceedings


CONTACT INFORMATION

J. Meimei Ma, Ph.D.
Associate Director, Statistical Programming
Quintiles
P. O. Box 13979
Research Triangle Park, NC 27709
Phone: (919)941-2888
Fax: (919)941-0972
Internet address: mma@quintiles.com

SAS, SAS/ASSIST, and SAS/INSIGHT are registered trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.