WHATITDO: DOCUMENTATION AUTOMATICALLY PRODUCED BY YOUR SAS PROGRAM

Karen Crandall, Eastman Kodak Company, Rochester, NY

WHATITDO SAS is a program that writes documentation based on comments in your SAS program. It reads the comments and writes them to a LISTING file. You can produce documentation on demand by simply specifying the name of the SAS program for which you want the documentation and running WHATITDO SAS.

There is just one catch. The comments in the program must be well-constructed and well-placed. They must articulate the relevant details precisely and clearly. The focus of this paper will be making judicious use of comments in your SAS programs.

An Example

The intent of WHATITDO SAS is to store documentation right along with the program. Whenever the documentation is needed, just run WHATITDO SAS and the most up-to-date documentation for the program is at your fingertips in seconds.

Figure 1 shows the most recent version of documentation for a program named PRINNTDO SAS.

```
OPTIONS MPRINT QUOTE NOCENTER;
** PROC PRINTTO;
** PROGRAM NAME: PRINTTO SAS
** PURPOSE:
** Some PROCs do not generate output datasets;
** in a form you need, although the printouts;
** they generate do have what you need.
** You can still capture this information in;
** the form of a SAS dataset by some;
** finagling as illustrated in this program.
** REQUIRED INPUTS:
** A dataset to be analyzed by PROC FREQ.
** OUTPUTS:
** A SAS dataset that ACCUMULATES THE RESULTS;
** of a TABLE stmt which contains requests;
** a one-way table for each of several;
** variables.
** PROGRAMMER/ANALYST: Karen Crandall, 8/94
** PROGRAM-TYPE: General Utility / Example
** Steps:
** 1. Make up some test data
** 2. Set up options and define external file
** 3. Re-route normal SAS output to this
** external file
```

Figure 1

You should have a pretty good idea about what PRINTTO SAS does. The program that solves the problem AND writes the documentation is illustrated in Figure 2. The purpose for including it here is to show where to place the comments. They are scattered around the code as appropriate, just as you would normally do it.
practice. Your attention to spacing trivia results in a more readable LISTING — in my opinion, it is worth the little extra effort to produce it.

**Verifying your LISTING**

How useful was the documentation for PRINTTO SAS? Did it describe the problem to be solved? Could you tell what inputs were required? Did you know what kind of output to expect? Did you have a reasonable understanding of what approach was going to be used to solve the problem? If you had just a general interest in this program, did the LISTING give enough detail to satisfy your curiosity? These are some typical questions to ask when assessing the quality of the comments.

In developing this example, I ran WHATITDO a half-dozen times, each time tweaking the comments until I got them to describe the program to my satisfaction. I could have stopped after the first or second iteration, but my text would have had less polish and a less “natural feel” to it. Use your judgment to determine how much tweaking is necessary. If you write comments while you are developing the code, they normally look pretty good. They usually are designed to describe the next several lines of code. However, when you read them in WHATITDO LISTING, often they seem somewhat choppy — some details are over-described, some are under-described. The comments can be fine-tuned at this point to improve the continuity and clarity.

**Fine-Tuning Tips**

I have a few tips that I use as my guide to help me write well-articulated comments. Most programs I write are not of the small, utility-type I have illustrated here. They are far more complicated and large. I have found, however, no matter how big the problem, it can be broken into a number of smaller sub-problems. Each sub-problem can be described and solved in one of the “Steps”. The LISTING resembles that shown in Figure 1, except that each Step has sub-step bullets.

Another tip is to document “busywork” in short comments on the same line with a line of code. My rule of thumb is never to document an implementation detail comment on a line by itself. The intent of my documentation is to describe the problem and the algorithm to solve it. Knowing that I will “re-structure a dataset via PROC TRANSPOSE for easier merging later on;” is an implementation detail that I do not care about on a macro level. If I need to make an implementation detail comment, I document it to the right of a program statement. WHATITDO reads only
those lines in the program that start with an asterisk (or start with %* (for macro users)).

Does WHATTIDO LISTING explain the essence of the problem and its solution? Not sure? — ask a colleague to review your LISTING. If your colleague is left with some questions, go back and improve your comments. Another programmer should be able to read the LISTING and get the gist of what is going on in the program without looking at the code itself.

The most useful tip is also the most painful. Sometimes, I just cannot articulate the words to describe the algorithm in a straightforward, easy-to-follow way. The algorithm is just too convoluted. Convoluted comments are not only useless, they are annoying. When I get to this juncture, I realize that I have failed to define a subproblem and I go back to re-engineer that part of my code. I go to the trouble because I know my program will ultimately be easier to understand and maintain. I know, too, that my “documentation review” will be a far more pleasant experience if I do not have to do a lot of extra explaining.

The final tip is to resist the urge to skip WHATTIDO on very complicated programs. Yes, they are the most work to document soundly. However, the payoff is the greatest. You are ensuring not only the integrity of the documentation, but more importantly, you are ensuring the soundness and efficiency of the algorithms and the code to implement them.

Conclusion

Because you might be curious about the program that started this whole business, it is illustrated in Figure 3:

```plaintext
OPTIONS LS=80 NODATE NOUNBER;
%let pgmname = PRINTTO SAS;

******************************************************************************;
*** WHATTIDO SAS: ***
*** Reads the comments in a SAS program and***
*** prints them out in a report. ***
***
*** To run program =>
*** Type the name of the SAS program you ***
*** want to read on the %let pgmname= stmt.***
***
*** Programmer Name: KAREN CRANDALL ***
***
*** NOTE =>
*** Intent is to discipline you, the author***
*** of the program and comments, to write ***
*** thorough comments in your program. ***
******************************************************************************;
```

Figure 3

There is nothing fancy or innovative about the program, WHATTIDO SAS. It simply reads the comments in your SAS program and writes them to a LISTING file. Likewise, there need be nothing fancy about your program comments. If you have taken care to design them carefully, they will have the clarity normally associated with a fine instrument. And why not? — well-commented code is a fine instrument!

Contact Information:

Karen Crandall
Eastman Kodak Company
Kodak Park
Rochester, NY 14652-4608
Telephone: (716) 722-6512

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries.

® indicates USA registration.