Paper 069-30

Stranded on a Deserted Island With Nothing But TITLE Statements

Rick M. Mitchell, Westat, Rockville, MD

ABSTRACT

Imagine being a SAS® programmer stranded on a desert island with no programming tools except for a handful of TITLE statements. When presented with the task of creating populated forms in SAS that closely resemble their original hardcopy format, many SAS programmers might wait years for a bottle to wash up on shore with the magical solution. There is no need to worry though, because this paper will discuss how one can achieve such a task using the simplest of programming techniques. By fully utilizing the TITLE statement along with basic CALL SYMPUT statements, users can avoid complicated programming and still create simple, decent looking reports that look just like they were intended to look.

INTRODUCTION

At first glance, the TITLE statement may be one of the most basic SAS tools available. However, there is enormous potential for use of the TITLE statement when one is faced with the challenge of producing output that looks just like the original data collection form with corresponding data. For the beginning programmer, the task of generating reports can be difficult and often overwhelming when first faced with complex procedures such as PROC REPORT or other similar SAS tools. In today's fast-paced working environment, there is often little time available for programmers to put their feet up on their desks and spend endless time tinkering with their programs to get a variety of coding techniques to do just the right thing. For this very reason, a fast and easy approach is offered that will allow even the most beginning programmer to accomplish a look and feel that is advanced in appearance. The approach presented in this paper will in essence use only TITLE statements to actually display the desired output. With a few tricks here and there and the use of PROC REPORT as more of a driver, but not as a reporting tool, one can wow their friends and coworkers with this simple approach. When one has little experience in the SAS world, he or she is somewhat in a similar situation as one stranded on a deserted island. With potentially limited knowledge of how to get to the finish line, one must take advantage of whatever tools are available at the time.

OVERVIEW

This paper will discuss several basic steps that the beginning programmer may take to produce high-quality replication of forms by ultimately using only TITLE statements. The following steps will be discussed:

- The task
- Using CALL SYMPUT statements to define form fields
- Using PROC REPORT without really using it
- Taking advantage of the TITLE statement
- The final product
- Figuring out how many forms to create
- Generating multiple forms

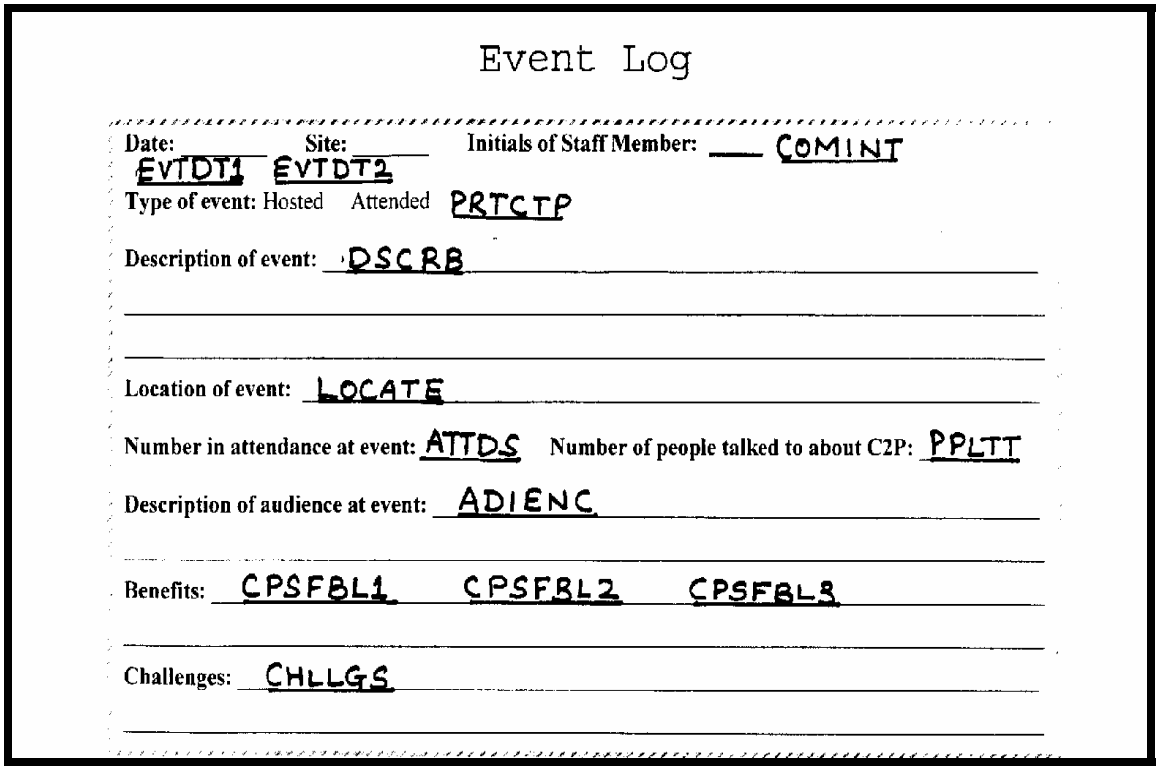
After following the basic process discussed in this paper, one can be assured that there are indeed feasible shortcuts to take as opposed to spending endless hours trying to tackle the wide variety of nuances within PROC REPORT or other similar SAS tools.

THE TASK

It is not so uncommon for one to want to display information in exactly the same format that it was collected in as shown below in Figure 1. This form, an Event Log for a behavioral study, was used to collect data for various events as related to the subject matter. After data entry has been completed, the user would then like to print out and view the data so that it can be compared to the original hardcopy form. A basic listing from PROC PRINT would somewhat meet this need, however, it is not going to give the optimal look. Furthermore, a programmer that is not familiar with more advanced SAS tools such as PROC REPORT, may have a difficult time figuring out just what to do. The task was as follows:

- Make the SAS output look like the data collection form
- Develop an approach quickly
- Develop an approach easily

So, the ultimate goal is to extract data from the database that can be presented similar to the original hardcopy form. One can achieve this goal by creating macro variables for the data fields and then taking advantage of TITLE statements to give a great look that is fast and easy to accomplish.


Event Log

Date: _____ Site: _____ Initials of Staff Member: ___ COMINT

EVTDT1 EVTDT2

Type of event: Hosted Attended PRTCTP

Description of event: DSCRB

Location of event: LOCATE

Number in attendance at event: ATTDS Number of people talked to about C2P: PPLTT

Description of audience at event: ADIENC

Benefits: CPSFBL1 CPSFBL2 CPSFBL3

Challenges: CHLLGS

Figure 1 – Original Data Collection Form (Annotated)

USING CALL SYMPUT STATEMENTS TO DEFINE FORM FIELDS

In order to make each data field available for use within the TITLE statement, CALL SYMPUT is utilized. In Figure 2 on the following page, the code is shown where macro variables are created for every field that is needed in the final output. Let us assume for now that we are working with a single record, and multiple records will be addressed later in the paper. Depending on the size of each variable, macro variables are generated with the TRIM and LEFT functions being utilized to left justify each of the data fields.

```
data elog;
  length blankline $ 200.;
  set extract.elog;
  if _n_=&i;
  blankline=' ';
  label blankline='Reviewer Comments';
  call symput ("ptnum",trim(left(pt)));
  call symput ("evtdt1", trim(left(evtdt1)));
  call symput ("evtdt2", trim(left(evtdt2)));
  call symput ("site",    trim(left(invsite)));
  call symput ("comint",trim(left(comint)));
  call symput ("prtctp",  trim(left(prtctp)));
  call symput ("dscrb",  trim(left(dscrb)));
  call symput ("locate", trim(left(locate)));
  call symput ("attds",   trim(left(attds)));
  call symput ("ppltt",    trim(left(ppltt)));
  call symput ("adienc",trim(left(adienc)));
  call symput ("cpsfbl1",trim(left(cpsfbl1)));
  call symput ("cpsfbl2",trim(left(cpsfbl2)));
  call symput ("cpsfbl3",trim(left(cpsfbl3)));
  call symput ("chllgs",  trim(left(chllgs)));
run;
```

Figure 2 – Code to Create Macro Variables For Output

Note that each CALL SYMPUT statement matches one of the variables on the annotated form shown in Figure 1 (with the exception of a few identifiers that are not shown).

USING PROC REPORT WITHOUT REALLY USING IT

PROC REPORT is indeed a very powerful and flexible reporting tool. However, even the most advanced programmer must sometimes rely on help documentation or examples in order to get PROC REPORT to do just the right thing. Using PROC REPORT only periodically may put the user in the same boat each time where he or she is unfamiliar with the vast array of statements and options that are available. For the beginning programmer, this can be difficult to get a grasp of, if not overwhelming.

So, the “trick” that is being presented in this paper is one in which we take advantage of PROC REPORT as somewhat of a driver to the process where it is not actually doing anything. This trick can be accomplished by including a single variable within the COLUMN and DEFINE statements and then utilizing the NOPRINT option to suppress the printing of this information as shown in Figure 3 below. While tools such as PROC PRINT and PROC FREQ will not allow this approach (i.e. the entire output is suppressed), PROC REPORT still allows use of the TITLE statement when generating no relevant output.

```
proc report data=elog;
  columns blankline;
  define blankline   / NOPRINT;
```

Figure 3 – Code to “Trick” PROC REPORT

TAKING ADVANTAGE OF THE TITLE STATEMENT

While a few tricks are used here and there, the foundation of this approach to generating high-quality form replication is simply the use of TITLE statements. Within the TITLE statement, one can provide text for the form field labels and then follow those labels with the actual data as generated through macro variables via the CALL SYMPUT statement. The user can take advantage of fonts, colors, sizes, and other features transferable to ODS output to give resulting output that can look very similar, if not exactly the same, as the original hardcopy form along with its corresponding data (see Figure 4 on the following page). Note that many additional spaces are provided to force ODS to carry information to the next line when exceeding the line size. While trailing spaces may be ignored in some situations, the user can also take advantage of the color option to trick the TITLE statement into printing some symbol (e.g. “.”) and then giving it a color of

WHITE resulting in nothing actually showing up (unless of course you decide to use some type of fancy paper for your output that is not white!). The code for this approach is shown in Figure 4 below. To simplify this example, all of these TITLE statements (with the exception of the height in TITLE1) have used the same specifications (font=arial, color=darkblue, and h=1.2), but one can use his or her imagination to consider the enormous potential for dazzling up one's output. Italics, superscripts, and bold letters, here we come!

```
  title1 j=center font=arial color=darkblue h=2 "Connect to Protect - Event Log (ID Number=&ptnum)"
    "                                                  " color=white "                                  . "
    "                                                  " color=white "                                  . "
    "                                                  " color=white "                                  . "
    "                                                  " color=white "                                  . ";
  title2 ' ';
  title3 font=arial color=darkblue h=1.2 "Date:  &evtdt1 &evtdt2     Site:  &site      Initials of Staff Member:  &comint "
    "                                                  " color=white "                                  . ";
  title4 font=arial color=darkblue h=1.2 "Type of event:  &prtctp                                                                  "
     color=white "                                  . ";
  title5 font=arial color=darkblue h=1.2 "Description of event:  &dscrb"
    "                                                  " color=white "                                  . ";
  title6 font=arial color=darkblue h=1.2 "Location of event:  &locate"
    "                                                  " color=white "                                  . ";
  title7 font=arial color=darkblue h=1.2 "Number in attendance at event:  &attds       Number of people talked to about
    C2P:     &ppltt                                          " color=white "                                  . ";
  title8 font=arial color=darkblue h=1.2 "Description of audience at event:  &adienc"
    "                                                  " color=white "                                  . ";
  title9 font=arial color=darkblue h=1.2 "Benefits:  &cpsfbl1   &cpsfbl2   &cpsfbl3                                                 "
     color=white "                                  . ";
  title10 font=arial color=darkblue h=1.2 "Challenges:  &chllgs";
quit;
```

Figure 4 – Code For Output that is Nothing But TITLE Statements

The first question that one may ask is what about the limitation of only being able to use 10 TITLE statements? The answer is actually quite simple. All one needs to do is generate multiple PROC REPORT runs where each run would include 10 more TITLE statements, and since PROC REPORT is not actually printing any information, the TITLE statements from one procedure to another could be given the appearance of flowing smoothly together.

THE FINAL PRODUCT

Below is the final report (Figure 5) that represents information entirely provided in TITLE statements. The report has the same look and feel as the original hardcopy form, and actually may even look a little better! Now that the program has been completed to generate the desired output, one may want to read further regarding the automation of the process to work on multiple records and/or forms.

Adolescent HIV Prevention - Event Log (ID Number=R8072)

Date: 20040307 Site: 75 Initials of Staff Member: RM/DO/MR

Type of event: HOSTED

Description of event: YOUTH MEETING

Location of event: ROCKVILLE ADOLESCENT CRISIS CLINICAL CENTER

Number in attendance at event: 25 Number of people talked to about C2P: 18

Description of audience at event: YOUTH FROM THE COMMUNITY BETWEEN THE AGES 14-24

Benefits: WE ARE ABLE TO EDUCATE YOUTH FROM THE COMMUNITY AROUND ISSUES PERTAINING TO HIV/AIDS. WE ALSO USED THIS TIME TO BRAINSTORM ON DIFFERENT WAYS WE CAN EFFECTIVELY TARGET YOUTH IN HIGH RISK COMMUNITIES.

Challenges: N/A

Figure 5 – The Final Product

FIGURING OUT HOW MANY FORMS TO CREATE

To setup a macro to run this process on any given number of forms for any given study, one can run some simple code to capture and store this number, NUMRECS, as shown below in Figure 6. The code merely reads in the existing dataset and keeps track of the number of records until it gets to the last record. Then, CALL SYMPUT is utilized to create a macro variable with the information to repeat the loop as many times as necessary.

```
data _null_;
  set extract.elog end=eof;
  sum+1;
  if eof then call symput("numrecs",sum);
run;
```

Figure 6 – Code to Calculate Number of Forms

GENERATING MULTIPLE FORMS

After the program has identified how many forms are needed, one simply needs to run this basic macro shown in Figure 7 below. The macro will run for as many times as there are forms needed (from 1 to the total number of records). Basically, the macro variables generated with the CALL SYMPUT statements will be regenerated for each record in the dataset. For example, if there were 10 variables for 20 records then the result would be 10x20=200 macro variables that are generated (10 macro variables with the same name being regenerated 20 times each). Finally, enclosing the macro within a designated ODS output file will give the user an electronic replication of the original hardcopy form along with all of the corresponding data.

```
%macro elogrep;
  %do i=1 %to &numrecs;
     .
     .
     {DATA step with CALL SYMPUT statements}
     {PROC REPORT with TITLE statements}
     .
  %end;
%mend;

ods rtf file='c:\c2p_elog.rtf' bodytitle style=minimal;
  %elogrep
ods rtf close;
```

Figure 7 – Code to Generate Multiple Forms

CONCLUSION

The TITLE statement can be utilized in a variety of ways ranging from basic documentation of SAS procedure output to exclusive generation of fancier output by itself. The approach discussed in this paper will be of use to all skills levels of SAS programmers, but it is most likely the beginning SAS programmer who will receive the greatest benefit. Working within the constraints of knowledge, skill, time, and money, the beginning SAS programmer can take a fast and easy way out from having to figure out higher level SAS tools and still perhaps not getting the optimal results. The example presented in this paper is very basic, however, one can probably imagine that there are endless possibilities where one can obtain just the right look for many types of output requests. By utilizing the TITLE statement to its fullest extent, the beginning SAS programmer takes a huge step toward the next skill level on the SAS career ladder by thinking outside of the box. It is actually the advanced SAS programmer who can turn a basic and otherwise uninteresting TITLE statement into magic that will wow clients, coworkers, and friends!

ACKNOWLEDGEMENTS

CONTACT INFORMATION

Rick M. Mitchell
Westat
1650 Research Boulevard, WB 496
Rockville, MD 20850
(301) 251-4386 (voice)
(301) 738-8379 (fax)
RickMitchell@Westat.com