

Learning the Basics: Everything Your Professor Should Have Told You

Veronica R. Walgamotte
Computer Sciences Corporation

Abstract

Most of us had our first introduction to SAS[®] from our statistics professor. He taught us just enough SAS to read the data into SAS data sets and run the STAT procedures. In other words, just enough to make us novice users. This tutorial intends to take the novice user one step beyond the basics. It will cover the KEEP=, DROP= and RENAME= options, the ELSE statement, the DO statement, the PUT statement, the INPUT and PUT functions, the WHERE= option, the OBS= and FIRSTOBS= options, the IN= option and, finally, the RUN statement.

Introduction

Generally, we learn by example. Our mentor gives us a program and asks us to modify it slightly. For example, we may be asked to change the variable name in a PROC PLOT from WEIGHT to HEIGHT. This turns out to be a simple change, there are other relatively simple changes which will help us improve our programs. This paper is intended as a beginner tutorial to explain some commonly used features of

SAS that can distinguish the novice from the more experienced programmer. Most of these features involve improving the efficiency of your programs; all these features are just good sound programming practices.

KEEP= and DROP= Options

Let's start with two basic data set options, KEEP= and DROP=. These options can be attached to any SAS data set, whether it is used as input or output. These options affect which variables are reserved or discarded from the data set. You may not be familiar with these options because the SAS programs you inherited used all of the variables in your data set. There are also KEEP and DROP statements which can only be used in the Data Step (example 1). The Data Step statements are similar to the KEEP= and DROP= options and perform the same basic functions. The difference is that the KEEP= and DROP= options are much more flexible than the Data Step statements. The options only affect the data set they are attached to while the Data Step statements affect all output data sets. In addition, the data set options can be used on input data sets to reduce the variables read into the step.

Example 1:

```

/* create copy of profdata with */
/* only the variables GROWTH,    */
/* RAIN & SUNSHINE                */

data mydata;
  set profdata;
  KEEP growth rain sunshine;
run;

```

Example 2:

```

/* create copy of profdata with */
/* only the variables GROWTH,    */
/* RAIN & SUNSHINE                */

data mydata
  (KEEP = growth rain sunshine);
  set profdata;
run;

/* print only the variable RAIN */
/* from mydata                  */
proc print data = mydata
  (DROP = growth sunshine);
run;

```

As a good programming practice, the KEEP= option is recommended over the DROP= option and the KEEP and DROP statements. Since the KEEP= option explicitly states which variables are needed, this helps to document the variable being passed from one data set to another or which variables are expected by the procedure.

RENAME= Option

The RENAME= data set option changes the name of a variable. If a variable doesn't have an appropriate name, the approach most novice programmers use is to simply create a new variable with the appropriate name and assign the value of the old variable to the new variable (example 3). This works fine if you need both the old and new variables, which will have identical

values, in the resulting data set. If not, use the RENAME= option (example 4). The syntax is:

```

(RENAME = (old_variable_name =
           new_variable_name
           <old_variable_name_2 =
           new_variable_name_2 > ))

```

Like the KEEP= option, the RENAME= option can be attached to any SAS data set (input or output). Remember if you use the RENAME= option on the input data set, use the new variable name during the processing. Conversely, if you use the RENAME= option on the output data set, use the old name during the processing.

Example 3:

```

data mydata;
  set profdata;
  newvar = growth;
run;

```

Example 4:

```

data mydata (RENAME = (rain =
                       raintot));
  set profdata (RENAME = (growth =
                           newvar));
  /* growth was renamed on input */
  /* remember use new name */
  if newvar le 10 then growcat = 1;
  /* rain is not renamed until */
  /* output - use old name */
  if rain gt 10;
run;

```

ELSE Statement

The ELSE statement states an alternative action to take when the previous IF-THEN statement is false. When you have a series of IF-THEN statement in a Data Step which

are mutually exclusive, using an ELSE statement will be more efficient. It is more efficient because once SAS reaches the first true condition, it does not continue to check the remaining ELSE statements. ELSE statements must always immediately follow the IF-THEN statement they are associated with, no other statements can be between these two statements. Either another IF-THEN statement or an assignment statement can follow the keyword ELSE. Example 5 has three IF-THEN statements that are mutually exclusive, they can be rewritten more efficiently with ELSE statements (example 6).

Example 5:

```
data mydata;
  set profdata;
  /* each if condition will be */
  /* checked for each observation */
  if growth le 10 then growcat = 1;
  if growth gt 10 and growth le 50
    then growcat = 2;
  if growth gt 50 then growcat = 3;
run;
```

Example 6:

```
data mydata;
  set profdata;
  if      growth le 10
    then growcat = 1;
  ELSE if growth le 50
    then growcat = 2;
  ELSE   growcat = 3;
run;
```

DO Statement

If more than one statement needs to be executed for a particular condition, the DO statement can be used to reduce programming and increase efficiency. The DO statement can be used as part of an IF-THEN statement or alone. Each DO

statement must have a matching END statement. In the following examples, the number of times the variable growth must be compared to a constant is reduced from four to two by using the DO statement. The result is a more efficient program.

Example 7:

```
data mydata1 mydata2;
  set profdata;
  if growth le 10
    then growcat = 1;
  else if growth le 50
    then growcat = 2;
  if growth le 10
    then output mydata1;
  else if growth le 50
    then output mydata2;
run;
```

Example 8:

```
data mydata1 mydata2;
  set profdata;
  if growth le 10 then
    DO;
      growcat = 1;
      output mydata1;
    END;
  else if growth le 50 then
    DO;
      growcat = 2;
      output mydata2;
    END;
run;
```

PUT Statement

The PUT statement writes the value of a variable, by default, to the SAS log. This can be a handy debugging tool. In the following example, the first PUT statement conditionally prints the value of GROWTH if GROWTH's value is less than 2. The second PUT statement prints the value of RAIN and SUNSHINE to the log for every observation. In addition to being an excellent debugging tool, the PUT statement

can be used to write very sophisticated reports. Use the FILE statement to reroute the information written by the PUT statement. To write customized reports, see the PUT statement in "SAS Language: Reference, Version 6, First Edition" pages 450-459.

Example 9:

```
data mydata;
  set profdata;
  if growth lt 2 then PUT growth= ;
  PUT rain= sunshine=;
run;
```

INPUT and PUT Functions

SAS only recognizes two types of variables, character and numeric. Of course you can store numbers in a character variable, but if you try to do any calculations with the character variable, SAS will convert the variable to numeric before it does the requested calculation. The experienced SAS programmer will not let SAS convert the variable, but will take control of the processing and do the conversion with the appropriate function (INPUT or PUT). The INPUT function should be used to convert a character variable to a numeric variable and the PUT function can be used to convert a numeric variable to a character variable. The PUT function can also convert character values to other character values with formats. The syntax for these functions are:

INPUT(variable-to-be-converted,informat.)
PUT(variable-to-be-converted, format.)

(See "SAS Language: Reference, Version 6, First Edition" page 635 for a list of informats and page 673 for a list of

formats.) In the following example, assume the variable RAIN is a character variable and the variable SUNSHINE is numeric. By default, the variable ESTGROW will be numeric because it is the result of an arithmetic operation. The variable MSG, on the other hand, will be character because it is the result of a character string concatenation. In example 10, SAS does the conversion of RAIN to numeric in the first assignment statement and ESTGROW to character in the second assignment statement. This Data Step will generate notes in the SAS log which indicate the number of times the conversions took place. In example 11, the programmer takes control of the conversions using the INPUT and PUT functions, which improves efficiency and reduces the number of notes in the log.

Example 10:

```
data mydata;
  set profdata;
  estgrow = rain + (2 * sunshine);
  msg      = 'This plant is estimated
             to grow ' || estgrow;
run;
```

Example 11:

```
data mydata;
  set profdata;
  estgrow = INPUT(rain,8.) +
             (2 * sunshine);
  msg      = 'This plant is estimated
             to grow ' ||
             PUT(estgrow,8.);
run;
```

WHERE= Option

The WHERE= data set option can be used to restrict the observations read or processed. Only those observations that

meet the specified criteria are read. This option has a counterpart the WHERE statement which applies to all SAS data sets in the step (Proc Step or Data Step). The WHERE= option is recommended because it is attached to a single data set.

Example 12:

```
/* notice you don't need a      */
/* separate data step to create*/
/* a data set with those obs    */
/* that have heavy rainfall.    */
proc print data = profdata
    (WHERE = (rain gt 5));
run;
```

OBS= and FIRSTOBS= Options

Another way to restrict which observations are processed is to use the OBS= and/or FIRSTOBS= data set options. These options tell SAS which observations should be read from the input data set. The FIRSTOBS= option identifies the observation number at which the step should start reading the data set. The OBS= option identifies to SAS the last observation to be read from the data set. The syntax for these options are:

```
(FIRSTOBS = n)
(OBS = m)
```

where *n* and *m* are observation numbers

Both the *n*th and the *m*th observations are included. You are not allowed to use either of these options with the WHERE= option. You can use both options on the same data set, if *m* (the ending observation number) is larger than *n* (the starting observation number).

Example 13:

```
/* mydata is a copy of the first */
/* 10 observations of profdata */
data mydata;
    set profdata (OBS = 10);
run;

proc print data = mydata
    (FIRSTOBS = 5
     OBS = 7);
title 'Print of 3 obs from mydata';
run;
```

IN= Option

The IN= data set option is used on input data sets, usually, with the SET or MERGE statement. This option can tell you which input data set contributed to the current observation. The syntax for this option is:

```
(IN = sas-variable-name)
```

The *sas-variable-name* will contain a value of 0=false or 1=true to indicate whether the data set contributed to the current observation. In the following example, if the current observation came from PROFDAT1, the variable DSN1 would have a value of 1 and DSN2 would have a value of 0. It's important to note the new variables (DSN1 and DSN2) are NOT transferred to the output data set, MYDATA.

Example 14:

```
data mydata;
    set profdat1 (IN = dsn1)
        profdat2 (IN = dsn2);
run;
```

RUN Statement

In a batch environment, the RUN statement can be left out of the program and the program can execute without errors. In the interactive environment (Display Manager) if no RUN statements are used, the program will not finish execution of the last step. The RUN statement should be placed at the end of every step to signal the end of the step. The RUN statement help to isolate errors, therefore it aids in debugging. It also good programming style. For these reasons, the RUN statement is recommended for every environment. Without the RUN statement in example 16, the title on both the PROC PRINT and PROC FREQ output will be 'Frequency of Sunshine in My Data'. SAS seemly ignored the first title because it didn't see the step boundary until the word PROC in the PROC FREQ statement. The second title line wrote over the first title line before the PROC PRINT executed. Example 17 shows the correct use of the RUN statement.

Example 16:

```
data mydata (keep = growth
                rain sunshine);
    set profdata;
    title 'Sample Print of My Data';
    proc print data = mydata;
    title 'Frequency of Sunshine in My
    Data';
    proc freq data = mydata;
        table sunshine;
```

Example 17:

```
data mydata (keep = growth
                rain sunshine);
    set profdata;
run;
title 'Sample Print of My Data';
proc print data = mydata;
run;
```

```
title 'Frequency of Sunshine in My
Data';
proc freq data = mydata;
    table sunshine;
run;
```

General Notes

Notice all these data set options (KEEP=, DROP=, RENAME=, WHERE=, OBS=, FIRSTOBS=, and IN=), which are attached to SAS data sets start with a left parenthesis. More than one option can be used for any given SAS data set. To use multiple options, start with ONE left parenthesis, list the options, and then end with a single right parenthesis.

Example 18:

```
data mydata;
    set profdata
        ( KEEP = growth rain sunshine
          WHERE = (rain gt 5)
          RENAME = (sunshine = sunvar)
        );
run;
```

Conclusion

Some of these techniques allowed us to improve efficiency by restricting the variables read from or written to the SAS data set (KEEP=, and DROP= data set options); restricting the observations to be processed (WHERE=, OBS=, and FIRSTOBS= data set options); or controlling the conversion of variable types (INPUT and PUT functions). Others are helpful as debugging tools (PUT and RUN statements). The proper use of these techniques will not only improve efficiency of your programs, but will show good programming habits; indicating that you are

no longer a novice programmer but a true SAS whiz. These options and statements are just a few of the techniques available to SAS programmers to make their programs more efficient. In most cases, I have only touched on the power of these options. For more information refer to "SAS Language: Reference, Version 6, First Edition".

Acknowledgements

The author would like to thank; first, her many mentors who taught her SAS; and second, the reviewers of this paper for their helpful advice. In particular, the author would like to thank Chris Toppe for lassoing her into doing this paper. You can contact the author for more information at:

Computer Sciences Corporation
1301 Piccard Drive
3rd Floor
Rockville, MD 20850
(301) 258-5535

SAS is a registered trademark of SAS Institute Inc.,
Cary, NC, USA.