

UPDATING SOME TIPS AND TECHNIQUES TO VERSION 7 - PART 2

Deb Cassidy, Cardinal Distribution, Inc., Dublin, OH

ABSTRACT

There was so much material to cover for this presentation that it became two presentations. Each are written independent of the other.

The SAS® Applications Guide, 1987 Edition stated that it is “a collection of examples that illustrate how you can use base SAS software to solve a variety of data entry, analysis, management and report generation problems.” It was written as a Version 5 and 6 update for the 1980 edition. Now that another 12 years have passed, and we’re using Version 7 or 8 with rumors of Versions 9 and 10, are the examples still valid? Do they run? Are there alternative ways to do the same thing that are more efficient? This presentation will walk through some of the examples that are covered in the Diagnosing Errors chapter of the guide.

This paper is based on Version 7 on a Windows operating system with Base, SAS/STAT, SAS/GRAPH, SAS/FSP, SAS/AF, SAS/ASSIST, SAS/EIS, SAS/ACCESS to PC file formats and SAS/ACCESS to ODBC. You may notice differences on other operating systems or with a different suite of products installed.

One thing that has changed in Version 7 is the terminology. SAS has changed terminology to be more consistent with other languages. Variables are fields, observations are records and data sets are tables. While I still use data set in this paper, I typically used the terms fields and records unless the point was easier to make using the old terminology.

CHAPTER 8 **DIAGNOSING ERRORS**

Any user who has ever followed the manual and still can’t get the program to run will realize this chapter can be more helpful than the ones showing the correct way to do things. Many times, the only problem is a typographical error - catch is, the SAS System thinks you want something else.

Errors in Syntax

The first section covers errors in syntax. You will normally see a statement start with a SAS keyword and end with a semi-colon. Two exceptions are the assignment statement and sum statement.

Assignment and Sum Statements

The assignment statement is in the form

`variable=expression;`

The SAS System will evaluate the expression following the equal sign and place the result in the variable.

The sum statement looked very similar except instead of an equal sign you have a plus sign.

`variable+expression;`

Once again, the SAS System will evaluate the expression following the equal sign. However, in this case, the result is added to the value already in the variable. This will accumulate the total across all the records you are processing. If you wish to subtract an expression, you must write it as

```
variable+(-expression);
```

because SAS only recognizes the plus sign and not the minus sign as a sum statement.

Now what happens if you incorrectly type the statement as

```
X+Y=Z;
```

In this example, we are also assuming that X and Y are in the data set but Z is not. You will get a NOTE that says that Z is uninitialized but no other error messages. Why? SAS thinks you want

```
variable+expression;
```

where X is the variable and Y=Z is the expression. The latter is a perfectly valid expression. If Y=Z then a value of 1 is returned to show the expression is “true”. Otherwise, the expression is “false” and a value of 0 is returned.

You received the uninitialized note because Z didn’t exist. A NOTE does not stop processing - it is basically just telling you a fact. However, good programmers always review the NOTES to determine what cause them so “hidden” problems such as this will be discovered.

What happens if you really do have Z in your data? You will not receive any notes. If, for example, X=1, Y=5 and Z=6, your data set will still have those values. However, if both Y and Z are the same, the expression is “true” so it is a 1. X becomes its original value plus 1.

Keywords

Since so many statements begin with keywords, it is important that they are

spelled correctly. The guide gave the following example.

```
data two;  
  set one;  
  dorp x;  
run;
```

The guide shows that you receive an error message stating the “statement is not valid or it is used out of proper order”. This stops the rest of the job from running. However, one of the enhancements that has been made in the last 12 years includes making some assumptions about what you intended to type. In Version 7, the above example causes the WARNING message “Assuming the symbol DROP was misspelled as dorp.” The job continues running. Many misspellings are now being caught but not everything will be. For example, the system doesn’t make any assumptions about “brop” so you receive the error message as noted above. You should always review such messages because the assumption may not be what you really wanted.

Semicolons

You’ve probably heard a thousand times “Don’t forget the semi-colon!” However, there are cases where the job runs even when you forget a semi-colon. But, in most cases, you won’t get what you expected. Here’s an example.

```
data a  
  input x;  
  y=x+1;  
cards;  
your data  
;  
run;
```

The job runs but you do get some NOTES. They tell you that X is uninitialized. You’ll also get a note about the missing values. The notes that are often ignored are the ones

with the clue as to what happened. When you create a data set, you always get a note telling you how many observations and variables were in the data set. The clue in this case is the fact that you received three such notes - one for data set A as expected but also ones for data set INPUT and data set X. Without the semi-colon after the A, SAS thought you really wanted three data sets even though you put two of them on the next line which is perfectly acceptable.

The guide also gave another example of how you can have a semi-colon but the system doesn't see it. Those of you too young to remember punch cards but are inheriting legacy code will want to read this section closely. Punch cards were 80 columns wide. Many of the early editors also kept this restriction. Some languages and editors actually had a restriction of 72 characters for your code so the last 8 slots could be used to hold a line number. Some operating systems may still have this limitation but SAS under Windows doesn't. However, that doesn't mean someone can't set the option to restrict how many columns will be read. The option is still present in Version 7. A Windows user is most likely to encounter the problem if they are bringing in legacy code which contains the option. I set the option much smaller just so show you how it works.

```
options s=30;

data two;
  set one;
  keep a   b   c   d   e   f   g   h;
run;
```

If you count the columns, you will see that the semi-colon is beyond column 30. When you run this code in display manager, nothing will happen. You'll need to submit another data step, a proc step or another RUN. You'll then see a WARNING that says "The variable run in the DROP, KEEP,

or RENAME list has never been referenced." You'll also notice that your log has a gap in the middle of your keep statement. Your data set doesn't contain all the variables. SAS simply followed your instructions and only read the first 30 columns. This means the last variables and the semi-colon were never seen. Since SAS keeps going until it finds a semi-colon, it assumed RUN; was part of your KEEP statement. Since display manager doesn't process the code until it encounters a data step, proc step or RUN, nothing happened until you provided one.

Between Keyword and Semicolon

When SAS sees a keyword, it expects to find certain things before it encounters a semi-colon. The following example was provided in the guide.

```
Data b;
  set a;
  drop x+y;
run;
```

The guide says you'll get an error message stating the "word does not start with a letter or underscore". However, in Version 7, the same code produced the error message "The symbol is not recognized". In both cases, SAS stops the job. SAS thinks you are referring to a variable named "+" and that is not a valid name.

When you use a KEEP option on a data set, you follow the KEEP with an equal sign. However, when you use a KEEP statement, you don't. What happens if you do by mistake as shown below?

```
data two;
  set one;
  keep=x;
run;
```

You'll discover the job runs just fine. However, when you look at data set two you will see all the original variables plus a variable called "keep". It will contain the same value as X. If you look back earlier in the paper, you'll see your keep statement looks just like an assignment statement. KEEP is a perfectly valid variable name - although it is a very poor choice.

IF-THEN Statement

An IF-THEN statement is written in the form:

`IF expression THEN statement;`

Here there are some restrictions. The *expression* can be lots of things including a comparison such as $X < 10$. However, the *statement* must be an executable program statement. Examples of unacceptable statements include KEEP, RENAME, TITLE, LENGTH, and ARRAY. The general rule is to ask yourself if the statement can be executed conditionally. That is, can it apply to some variables or observations without applying to them all? For example, if you KEEP a variable for some observations, you must keep it for all of them in the same data set. The statement

`if x=1 then keep y;`

will give you an error message stating "Statement is not valid or it is used out of proper order."

Expressions

The following statement looks quite simple.

`if x=1 or x=2 then output;`

You simply want all the observations that have a value of either 1 or 2 to be written to the new data set. However, what happens if you write the statement as

`if x=1 or 2 then output;`

Some people who are used to working with other languages may think this is exactly the same thing. You do not get any error messages. What you do get is all the observations instead of just the values of 1 or 2. The statement is interpreted as though you had written

`if x=1 or 2^=0 then output;`

Since 2 is never equal to 0, the second condition is always true. This is an OR statement so the entire statement is always true.

What about a statement like

`if x then output;`

This works the same way. It assumes you wrote

`if x^=0 then output;`

Many people do use this as a shortcut. However, others who inherit the code may end up confused.

Parentheses and Quotes

Many times it is shorter to write one statement combining several functions rather than several statements. For example, the following shows two ways to get the sum of X and Y and then to take the integer portion when you divide that sum by 2.

`a=int((x+y)/2);`

`z=x+y;
a=int(z/2);`

The first combines both statements and doesn't require you to create an extra

variable. However, you need to make sure you have matched sets of parentheses. The guide says you will receive an error message such as “right parenthesis for function expected” but the Version 7 message is simply “expecting an)”.

Another concern about parentheses was not mentioned in the guide. You won’t get any warning or error message but you can get wrong results. This will happen if you have matched sets of parentheses but they are in the wrong place. The following look the same except for one parenthesis.

```
z=int((x+y)/2);  
q=(int(x+y)/x);
```

If x=4 and y=5, you’ll get z=4 but q=4.5. Why? Parentheses are evaluated from the inside to the outside. The first statement goes through the following logic.

```
x+y=9  
9/2=4.5  
int(4.5)=4
```

The second does the following instead.

```
x+y=9  
int(9)=9  
9/2=4.5
```

When you look closely, the outer parentheses in the second example really don’t do anything. You could omit them and get the same results.

What about unmatched quotes? Unfortunately, SAS isn’t so kind as to tell you “expecting an “” like it does for the parentheses example. The system keeps going until it finds that matching quote. If you have multiple unmatched quotes, you may end up with a really confusing log. I

tried the following example in Version 7 to see if the error messages were the same only to discover something totally different. I then tried it in Version 6 and realized there are more possibilities than I thought.

```
data one;  
  x='no;  
run;  
title 'this is a test';
```

In many cases, you will get an error message which says “character literal has more than 200 characters”. My error message was “invalid date/time/datetime constant ‘no;run;title ‘t.” SAS keep going until it found the next quote. However, since there was a T immediately following the next quote, the system thought I was trying to create a SAS time. You also get the same error if the next letter is a D. This is because you create SAS date and time constants in the following form.

```
date='01jan99'd;  
time='12:15't;
```

Since Version 7 no longer restricts character fields to 200 bytes, I was very curious what the error message would be when I didn’t happen to do the date/time case shown above. Of course, when you are trying to write a presentation that needed to be done by a certain time in order to be in the proceedings, many authors find something will go wrong. In this case, I never figured out what happens. I actually filled my log numerous times and never saw an error message! I’m sure I’ll may a typo in some real work and discover what really happens.

Errors in Logic

I actually mentioned some logic errors above in relation to syntax errors.

Deleting before first. and last. Variables

Many people have heard about first-dot and last-dot variables and think it will make their work easier. Catch is, they don't understand how it works so they end up writing code which runs but is not logically correct.

First-dot and last-dot variable are automatically created by the SAS system. Suppose you have the following data.

Name	Course	Score
Joe	1	56
Joe	1	89
Joe	2	75
Tom	2	67
Tom	2	95
Tom	2	89

First-dot and last-dot will identify the first and last records WITHIN a BY group. The code

```
data one;
  set students;
  by name course;
  first_name=first.name;
  last_name =last.name;
  first_course=first.course;
  last_course=last.course;
run;
```

will create the automatic variables. The automatic variable are not available outside the data step unless you save them under

new variable names. That is what the assignment statements do. (In case you didn't know, Version 7 permits names longer than 8 characters.) The data set is shown on the following page. You can then use the automatic variable to identify the first or last record for a "name", the first or last "course" within a "name" or even the records which are not the first or last.

However, the mistake people will make is to use code such as the following.

```
data one;
  set students;
  by name course;
  if score>90;
  count+1;
  if last.name then output;
run;
```

The intent of the above code was to count all the times that each student scored above a 90. However, you've totally deleted Joe in the process which should be fairly obvious. What isn't so obvious is why you won't get a record for Tom either. The "last.name" record for Tom is the one with the score of 89. You deleted it so it never triggered the output to happen. Until people understand how this works, most will expect to find that Tom had a total of 1 since he did have a score of 95 on one of the records.

Data set for first. and last. example

Name	Course	Score	First. name	Last. name	First. course	Last. course
Joe	1	56	1	0	1	0
Joe	1	89	0	0	0	1
Joe	2	75	0	1	1	1
Tom	2	67	1	0	1	0
Tom	2	95	0	0	0	0
Tom	2	89	0	1	0	1

Compound Expressions in IF-THEN Statements

Suppose you want to keep all the records where $x=1$ or $x=2$ and delete everything else. The following code might seem logical.

```
if x^=1 or x^=2 then delete;
```

However, it is really equivalent to

```
if not (x=1 and x=2) then  
delete;
```

This one is more obvious that it will delete all the records since X will never be equal to both 1 and 2 at the same time so the expression is always true.

The code really should be

```
if x^=1 AND x^=2 then delete;
```

SUMMARY

This paper covers some of the examples covered in the Diagnosing Errors chapter in the SAS® Applications Guide, 1987 Edition. These examples are still valid in Version 7 but you'll notice you may not always get the same error messages. Each version brings enhancements which requires such changes. The enhancements may also make the system "smarter" about what you were attempting to do so the messages are clearer. However, it is still true that many times you need to

clearly understand what the system is doing as well as what you wanted it to do to avoid errors.

BIBLIOGRAPHY & TRADEMARK

SAS Institute Inc. SAS® Applications Guide, 1987 Edition. Cary, NC: SAS Institute Inc., 1987.

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

CONTACT INFORMATION

Deb Cassidy
Cardinal Distribution
7000 Cardinal Place
Dublin, OH 43016
614-757-7136
deb.cassidy@cardhealth.com