CC014

The Best of Cheesy, Sleazy SAS® Tricks

Michael Davis, Bassett Consulting Services, Inc., North Haven, CT

ABSTRACT

Based on an off-hand comment, the author solicited examples of "cheesy" or "sleazy" SAS tricks from his friends on the SAS-L distribution list. To qualify, the trick might be a coding declaration or other construction that suppresses the over-solicitous manner in which SAS tries to protect coders from themselves. A trick might also qualify if it hinted at an unexpected SAS behavior that works against coding stereotypes. The following paper presents some of the best tricks received.

ADD DUMMY PARAMETERS TO MACROS

Here is the scenario. You write this nifty macro and deploy in all of your department's programs. Subsequently, it is discovered that the macro needs to be modified. An additional parameter is now required. Does that mean that you have find all of the invocations of the macro embedded in the department programs and add the additional parameter to the macro call? Ouch!

For macros for which you anticipate expansion, consider this trick from Ian and Marianne Whitlock. If the macro has no parameters, code it as:

```
%macro noparms(dummy=) ;
```

and call it with

```
%noparms() ;
```

If the macro already has parameters, add a dummy parameter at the end of the parameter list. It will save you from hunting for all instances a macro that has been widely deployed.

I CAN DO IT IN THREE (OR FOUR)

Most SAS programmers are familiar with the Boolean expression

```
if a=1 and b=1 ;
```

This expression is commonly used with IN= temporary variables created by a MERGE statement to keep only those observations where both input data sets contribute. It can be shortened to:

```
if a and b ;
```

However, thanks to Ray Pass, one could represent that Boolean expression as:

```
if a*b ;
```

This shortcut works since if either the variable a or b is false, the expression resolves to zero (false). However, at the cost of one additional keystroke, the Boolean expression could be represented as:

```
if a& b ;
```

In this expression, the ampersand represents a logical AND, not a macro variable. The contributor of this trick, Ian Whitlock, suggests that the meaning of this expression is more explicit and perhaps executes a bit faster. So the choice is yours.

THIS VARIABLE IS UNINITIALIZED

Sometimes when one is coding a specified data set structure, some of the variables will be uninitialized. This results in the SAS Log message similar to:

```
NOTE: Variable z is uninitialized.
```

This is not acceptable if one is a member of the "Clean SAS Log" club. Richard DeVenezia contributed the following trick:

```
data … ;
   attrib
   …
   ;
   %* prevent NOTE: Variable ***** is uninitial-
ized. messages ;
   retain _character_ '' _numeric_ . ;
stop ;
run ;
```

Richard supplied the following caution for his trick: "Only use this with impunity when creating a zero rowed table of desired structure as set forth in prior attrib statements(s)."

ERR-OR, WARN-ING

Whenever I see "ERROR" or "WARNING" in a SAS Log, a small shiver travels along my spine. Many SAS users (and their bosses) feel the same sense of trepidation when they see these two words in a SAS Log. So for custom control messages in a program, Christoph Edel uses the following trick:

```
data _null_ ;
 put "WARN" "ING: - bad thing! Check!" _all_ ;
run ;
```

By splitting the WARNING or ERROR between two literal strings, these tokens do not appear in the SAS Log unless the trigger condition emerges. Notice that no concatenation operator is needed between the two literal strings.

TWEAKING PROC FREQ SORT ORDER

Consider the following. You have a format generated by the following VALUE clause:

```
value myfmt
   1="High School"
   2="College"
   3="Grad School"
   ;
```

If you subsequently use the *myfmt* format in a PROC FREQ, the rows of the report would come out in the order of the format label values (College-Grad School-High School). To get the values to appear in the same order as the unformatted values (1-2-3), insert a leading blank in front of "High School":

```
value myfmt
   1=" High School"
   2="College"
   3="Grad School"
   ;
```

In the FREQ procedure, format labels with a leading blank sort ahead of the other values but the blanks do not appear in the report.

GLOBAL MACRO DECLARATION TRICK

You need to determine whether a global macro has been set. Macro variables are essentially character strings so we can check their state by examining the length with the %LENGTH macro function. However, when we run the following statement:

```
  %put The Length of NESUG is %length(&nesug) ;
```

and the value of the macro variable, *&nesug*, has not been set, we receive the following message in the SAS Log:

```
WARNING: Apparent symbolic reference NESUG not
resolved.
```

The cheesy, sleazy solution, courtesy of Roland Rashleigh-Berry, is to declare *&nesug* as a global macro variable ahead of the %LENGTH function:

```
   %global nesug ;
```

The SAS Log returns the desired result:

```
The Length of NESUG is 0
```

LEFT SIDE FUNCTIONS

Nearly every SAS programmer knows that functions on the right side of an assignment statement can create, set, or modify the variable on the left side of the statement. However, one of the cheesy, sleazy features of SAS is that functions can also be used on the left size of an assignment statement. Consider the following example suggested by a friend who prefers to remain anonymous:

```
substr(name, 2) = lowcase(substr(name, 2)) ;
```

In this example, if *name* was equal to "RALPH", the assignment statement would change it to "Ralph".

QUICK WAY TO READ MIXED-CASE TEXT

The next trick, courtesy of Mike Zdeb, is a cheesy, sleazy way to convert raw mixed-case text to uppercase text.

```
data xyz ;
   infile 'mydata.txt' ;
   input @ ;
   _infile_ = upcase(_infile_) ;
   input … ;
run ;
```

The reason why this trick qualifies as cheesy and sleazy is that one would not ordinarily think of modifying _infile_, which is the automatic variable that references the contents buffer created by the INFILE statement for the current raw text record.

However, the following test will prove that one can overwrite the contents of _infile_. Just cut and paste the above code into a SAS editor, create *mydata.txt*, and modify the second INPUT statement to read the text in *mydata.txt*. Submit the code and inspect the data set *xyz* via PROC PRINT or a Viewtable window. You will see that it has been converted to uppercase.

For those not familiar with the using of the trailing @ ("at" sign) in an INPUT statement, the trailing @ instructs the SAS System to not move the record pointer and to use the data just placed in the input buffer for the next INPUT statement. While the input record pointer is held, we can have our way with _infile_ as this trick illustrates.

DEBUG PROC REPORT COMPUTE BLOCKS

The last trick, again courtesy of Richard DeVenezia, helps solve the problem of how to debug PROC REPORT compute blocks. If you were to insert a PUT statement in a compute block, you would see a log message similar to:

```
ERROR:  PUT statement is not valid in this con-
text.
```

To see the value of a variable in the compute block, insert the following code:

```
   call execute('%put' || <variable>) ;
```

This trick qualifies as cheesy and sleazy since one would expect the CALL EXECUTE to reference a macro, not a %PUT.

CONCLUSION

As the following tricks illustrate, SAS can be a subtle language. The following tricks only scratch the surface of these features and how to exploit them. An implication of "cheesy" and "sleazy" is that one might fear that some of these tricks stop working in subsequent SAS releases. However, in the author's opinion, these features are well embedded in SAS and will probably continue to be available in subsequent releases and across platforms.

The author may revisit this subject in the future. So feel free to submit your candidates for "cheesy, sleazy" tricks to the author at the email address shown below.

ACKNOWLEDGMENTS

SAS is a registered trademark of the SAS Institute, Inc. of Cary, North Carolina. The author wishes to thank his SAS-L friends who contributed "cheesy, sleazy" tricks used in this paper. He also would like to thank Ralph Leighton, who suggested the idea for this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michael L. Davis
Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
Phone: 203-562-0640
Fax: 203-498-1414
Email: michael@bassettconsulting.com
Web: http://www.bassettconsulting.com