Paper CC121

The %LetPut Macro, and Other Proactive Macro Debugging Techniques

Shane Rosanbalm, Rho, Inc.

ABSTRACT

Macro debugging can sometimes be difficult. Having ready access to the values of local macro variables is often quite helpful. This paper will introduce a simple macro, %LetPut, to assist in the displaying of the values of macro variables to the log. Other minimally invasive techniques for generating helpful messages in the log will also be presented.

WRITING VALUES OF MACRO VARIABLES TO THE LOG

Displaying the values of newly-created macro variables to the log is helpful for macro debugging. Consider the following statements, which create a macro variable &maxmsrp:

```
proc sql noprint;
   select   max(msrp)
   into     :maxmsrp
   from     sashelp.cars
   ;
quit;
```

A simple way to display the value of &maxmsrp to the log is with a %put statement.

```
%put &maxmsrp;
192465
```

While this satisfies the minimum requirement of showing the value in the log, it is often difficult to find the value amongst all of the source code and notes. Simply adding the name of the macro variable as static text before the value can be helpful.

```
%put maxmsrp = &maxmsrp;
maxmsrp =   192465
```

What is perhaps not immediately obvious from the above log line is that the value of &maxmsrp has leading spaces. This may or may not be a problem in your particular application. To make it easier to see these leading spaces, it is helpful to add square brackets around the value of the macro variable.

```
%put maxmsrp = [&maxmsrp];
maxmsrp = [  192465]
```

Lastly, we can add color and indentation to the log to make the value of &maxmsrp even easier to find. Both the color and indentation are accomplished through the use of the NOTE- option of the %put statement.

```
%put NOTE- maxmsrp = [&maxmsrp];
      maxmsrp = [  192465]
```

LEADING AND TRAILING BLANKS

Leading and trailing blanks in macro variable values are rarely essential and are often a nuisance. There are techniques for preventing these blanks from appearing at the initial creation of the macro variable: symputx, left/trim, the SQL trimmed option, etc. However, I don't like spending my limited brain power thinking about things like, "Is this macro variable that I'm about to create going to be generated with unwanted spaces?" Instead, I prefer to create macro variables quickly and then use the %let statement to remove potential leading and trailing blanks.

```
%let maxmsrp = &maxmsrp;
%put NOTE- maxmsrp = [&maxmsrp];
      maxmsrp = [192465]
```

THE %LETPUT MACRO

The %let and %put statements that I'm advocating for are a little bit of a typing liability.

```
proc sql noprint;
   select   max(msrp)
   into     :maxmsrp
   from     sashelp.cars
   ;
quit;

%let maxmsrp = &maxmsrp;
%put NOTE- maxmsrp = [&maxmsrp];
```

```
maxmsrp = [192465]
```

Not only do you have to type the name of the macro variable 4 times, but the square brackets are also somewhat awkward to type at speed. This keyboarding liability is why I have created a simple macro which will write the %let and %put statements for me. The macro is called %LetPut, and it is used as follows:

```
proc sql noprint;
   select   max(msrp)
   into     :maxmsrp
   from     sashelp.cars
   ;
quit;

%letput(maxmsrp);
```

```
maxmsrp = [192465]
```

The code inside the %LetPut macro is a mere 5 statements.

```
%macro letput(mvar);

   %if %symexist(&mvar) eq 1 %then %do;

      %let &mvar = &&&mvar;
      %put NOTE- &mvar = [&&&mvar];

   %end;

   %else %put NOTE- Macro variable %upcase(&mvar) does not exist.;

%mend letput;
```

Strictly speaking only 2 of these statements are necessary (%let and %put), though the other statements do help the macro to fail gracefully should the user mistype the name of the macro variable.

This concludes the %LetPut portion of the paper. We now move on to a couple of other proactive macro debugging techniques.

MARKING THE TOPS OF %DO LOOPS

Many macros are written to perform repetitive tasks. For instance, a macro might produce one plot for each lab test in a dataset. Such a macro might begin with an SQL step.

```
proc sql noprint;
   select   distinct param
   into     :param1-:param999
   from     adam.adlb
   ;
   %let param_n = &sqlobs;
quit;
```

Having created macro variables &param1, &param2, etc., we can now write a simple %do loop to produce the plots.

```
%do i = 1 %to &param_n;

   title "Scatter Plot of &&param&i over Time";
   proc sgplot data=adam.adlb (where=(param="&&param&i"));
      scatter y=aval x=ady;
      yaxis label="&&param&i";
   run;

%end;
```

Something that can be done to make the log easier to navigate is to mark the top of each iteration of the %do loop with a long dashed line and the values of key macro variables:

```
%do i = 1 %to &param_n;

   %put ------------------------------------------------;
   %letput(i);
   %letput(param&i);

   title "Scatter Plot of &&param&i over Time";
   proc sgplot data=adam.adlb (where=(param="&&param&i"));
      scatter y=aval x=ady;
      yaxis label="&&param&i";
   run;

%end;

------------------------------------------------
      i = [1]
      param1 = [Albumin (g/L)]
<notes>

------------------------------------------------
      i = [2]
      param2 = [Alkaline Phosphatase (mmol/L)]
<notes>

<...>

------------------------------------------------
      i = [37]
      param37 = [WBC Count (x10**9/L)]
<notes>
```

CHANGE &&PARAM&I TO &PARAM

When I write code to produce a set of plots, I typically start by writing code to produce a single plot.

```
title "Scatter Plot of Albumin (g/L) over Time";
proc sgplot data=adam.adlb (where=(param="Albumin (g/L)"));
   scatter y=aval x=ady;
   yaxis label="Albumin (g/L)";
run;
```

Once I have the code working for a single plot, I then attempt to partially macrotize the code using a %let statement.

```
%let param = Albumin (g/L);

title "Scatter Plot of &param over Time";
proc sgplot data=adam.adlb (where=(param="&param"));
   scatter y=aval x=ady;
   yaxis label="&param";
run;
```

Only after this %let version has run successfully do I attempt to add a %do loop and create the macro variables &&param&i.

Repeatedly typing &&param&i is inefficient and prone to typos (not to mention unsightly). Consider instead using a %let statement inside the %do loop to create a new macro variable &param that can hold the values of &&param&i.

```
%do i = 1 %to &param_n;

   %let param = &&param&i;

   title "Scatter Plot of &param over Time";
   proc sgplot data=adam.adlb (where=(param="&param"));
      scatter y=aval x=ady;
      yaxis label="&param";
   run;

%end;
```

This strategy not only reduces the overall typing burden and lessens the likelihood of making typos, but it also allows you to take advantage of the code you wrote as part of the initial partial macrotization with %let.

PUTTING IT ALL TOGETHER

Following is an enumerated list of the techniques recommended in this paper.

1. Use %put to display values of macro variables to the log.
2. Display the macro variable name along with the macro variable value.
3. Place square brackets around the value of the macro variable.
4. Use %let to remove leading and trailing blanks from the values of macro variables.
5. Use the %LetPut macro to accomplish all of the above tasks in a single line of code.
6. Use %put statements to add dashes and macro variable values to the log at the top of each %do loop.
7. Use the %let statement to transfer &&param&i into &param.

Putting all of this together in code results in the following:

```
%do i = 1 %to &param_n;

   %let param = &&param&i;

   %put ------------------------------------------------;
   %letput(&i);
   %letput(&param);
   %put ------------------------------------------------;

   title "Scatter Plot of &param over Time";
   proc sgplot data=adam.adlb (where=(param="&param"));
      scatter y=aval x=ady;
      yaxis label="&param";
   run;

%end;
```

```
-------------------------------------------------
      i = [1]
      param = [Albumin (g/L)]
-------------------------------------------------
<notes>

-------------------------------------------------
      i = [2]
      param = [Alkaline Phosphatase (mmol/L)]
-------------------------------------------------
<notes>

<...>

-------------------------------------------------
      i = [37]
      param = [WBC Count (x10**9/L)]
-------------------------------------------------
<notes>
```

CONTACT INFORMATION

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

Shane Rosanbalm
Rho, Inc
6330 Quadrangle Dr.
Chapel Hill, NC 27517
919.595.6273
E-mail: shane_rosanbalm@rhoworld.com
Web: www.rhoworld.com

APPENDIX 1: %LETPUT SOURCE CODE

```
%macro letput(mvar);

   %if %symexist(&mvar) eq 1 %then %do;

      %let &mvar = &&&mvar;
      %put NOTE- &mvar = [&&&mvar];

   %end;

   %else %put NOTE- Macro variable %upcase(&mvar) does not exist.;

%mend letput;
```