jimmydefoor@gmail.com

Simple Ways of Using SAS Macros to Improve Your Programs

This paper and presentation will provide some examples of how to use SAS macros and SAS macro variables to make your SAS coding easier and enable your SAS programs to run more efficiently.

Before discussing SAS macros, we must create the data that will be used in the macro examples. This will allow an opportunity to discuss the reading of external data and the storage of that data in a SAS data set.

Using the special variable _infile_ and reading data from within the program.

This example shows that the special SAS variable _infile_ contains all the characters read on that line in the input data. It also shows that the SAS default field-delimiter is a space. That is, unless SAS is told otherwise, such as the delimiter is a comma, it will use a space as the delimiter between fields.

```
Data _null_;
   Input;        /* reads all characters in a line */

   put _infile_; /* writes the contents of the line that was read */

   first_field  = scan(_infile_,1); /* separates the line into single words */
   secnd_field  = scan(_infile_,2);
   third_field  = scan(_infile_,3);
   forth_field  = scan(_infile_,4);
   fifth_field  = scan(_infile_,5);

   put first_field= secnd_field= third_field= forth_field= fifth_field=;

   Output; /* Output statement defaults to execution statement in step. */

   Cards; /* Tells SAS to read data within the program and that data follows.*/

       10001        Red Truck           Dallas
       10012        Green Bicycle       Fort Worth
       10033        Blue Doll House     Houston
       10057        Orange Scarecrow    Austin
       10072        Yellow Songbird     San Antonio
;
Run;

/* Results of Put _Infile_. */
    10001       Red Truck           Dallas

/* Results of Put variables= */
first_field=10001 secnd_field=Red third_field=Truck forth_field=Dallas fifth_field=

    10012       Green Bicycle       Fort Worth

first_field=10012 secnd_field=Green third_field=Bicycle forth_field=Fort fifth_field=Worth

    10033       Blue Doll House     Houston
```

Setting the length of the field to be stored versus the length of the field to be read.

This example shows the difference between setting the length of the field to be stored in the SAS data set and setting the length of the field to be read from the input source.

```
Data work1;

   /* Length sets length of the field to be stored in the SAS Data Set. */

   Length prod_id 8 prod_desc $25   city $20;

   /* Input sets length of the field to be read, which defaults to the location   of
   the first blank space unless length is specified on Input or Informat statement. */

   Input  prod_id   prod_desc $25.0 city $20.0;

   put _infile_;

   put prod_id= prod_desc= city= /;

   Output;

   Cards;
       10001        Red Truck           Dallas
       10012        Green Bicycle       Fort Worth
       10033        Blue Doll House     Houston
       10057        Orange Scarecrow    Austin
       10072        Yellow Songbird     San Antonio
;
Run;

/* Results of Put _infile _. */
    10001       Red Truck           Dallas

/* Results of Put variables= . */
prod_id=10001 prod_desc=Red Truck city=Dallas

    10012       Green Bicycle       Fort Worth
prod_id=10012 prod_desc=Green Bicycle city=Fort Worth

    10033       Blue Doll House     Houston
prod_id=10033 prod_desc=Blue Doll House city=Houston

    10057       Orange Scarecrow    Austin
prod_id=10057 prod_desc=Orange Scarecrow city=Austin

    10072       Yellow Songbird     San Antonio
prod_id=10072 prod_desc=Yellow Songbird city=San Antonio
```

Creating Macro Variables That Contain Data Values.

This example shows how to transform data into the contents of SAS macro variables that can be evaluated or written within a Macro %Do Loop.

First, create a character count of each line so that it can be used to create a unique macro variable of each data value. Then concatenate the count variable to each macro variable so that each data value is a unique macro variable, but starts with the same string as the other macro variables of that characteristic, such as prod_id1, prod_id2, prod_id3, etc. .

Use the End= defined variable to write out a macro variable of the total line count so that it can be used to control the use of all macro variables in a %Do Loop.

```
Data _null_;
  Set work1 end=eof;
  cnt + 1;
  charcnt  = left(put(cnt,2.0));
  charprod = left(put(prod_id,6.0));

  /* Call Symput writes out a Macro Variable from inside a Data Step.
     The first argument is the macro variable to be created. The
     Second argument is the data value that will be stored in the
     macro variable. */

  call symput('prod_id'||charcnt,charprod);

  /* can now use strip instead of left and trim */
  call symput('prod_desc'||charcnt,left(trim(prod_desc)));
  call symput('city'||charcnt,left(trim(city)));

  /* write out number of macro variables at end of file */
  if eof then
    call symput('cnt',left(trim(charcnt)));
run;
```

```
NOTE: There were 5 observations read from the data set WORK.WORK1.
NOTE: DATA statement used:
      real time          0.12 seconds
      cpu time           0.00 seconds
```

```
Count is clearly five because five records of variables were read from work1.
An && resolves to a single & in the first pass of macro resolution. Thus,
&&prod_id&cnt first becomes &prod_id5, which then becomes 10072. This is
called double resolution.

/* %Put displays the content of macro variable in the SAS log */
%put cnt=&cnt
     prod_id1   =&prod_id1     prod_desc1   =&prod_desc1
     prod_id&cnt=&&prod_id&cnt prod_desc&cnt=&&prod_desc&cnt;
```

```
cnt=5 prod_id1=10001 prod_desc1=Red Truck prod_id5=10072  prod_desc5=Yellow Songbird
```

Compiling a Macro that Will Generate If-Then-Statements

Macros are code that are compiled and executed before SAS code is compiled and executed. Their purpose is to control when and where SAS code executes or to 'write' the SAS code that will be executed.

The keyword %macro starts a macro compilation. The keyword %mend closes a macro compilation.

First, the Ifstat macro is compiled.

```
%macro ifstat;
  %Do j = 1 %to &cnt;
    %if &j ge 2 %then
       %do;
          Else
       %end;
          If prod_id = &&prod_id&j then
            prod_desc = "&&prod_desc&j";
    %if &j = &cnt %then
       %do;
          Else prod_desc = 'Unknown';
       %end;
  %end;
%mend ifstat;
```

Second, it is placed in the Data Step in the location where the IF statements will be executed.

SAS starts compiling the Data Step when it sees the word Data. When it encounters the macro call, it executes the macro - which places the SAS code in the location of the macro. SAS then compiles the generated text as if a macro never existed.

```
data _null_;
  set work1;
  %ifstat ;
  output;
run;
```

```
183  data _null_;
184    set work1;
185    %ifstat ;
MPRINT(IFSTAT):   If prod_id = 10001 then prod_desc = "Red Truck";
MPRINT(IFSTAT):   Else If prod_id = 10012 then prod_desc = "Green Bicycle";
MPRINT(IFSTAT):   Else If prod_id = 10033 then prod_desc = "Blue Doll House";
MPRINT(IFSTAT):   Else If prod_id = 10057 then prod_desc = "Orange Scarecrow";
MPRINT(IFSTAT):   Else If prod_id = 10072 then prod_desc = "Yellow Songbird";
MPRINT(IFSTAT):   Else prod_desc = 'Unknown';
186    output;
187  run;

NOTE: There were 5 observations read from the data set WORK.WORK1.
NOTE: DATA statement used:
```

Compiling a Macro that Will Create a User Format

This macro will be named 'format' and will create a user format named Prddesc. User formats are very efficient ways of assigning values to variables and can be stored in a format library, if desired, so that they can be used in many programs.

This macro will use a %Do loop to generate an assignment statement for each prod_id and prod_desc. It will also assign a description of 'Unknown' when a prod_id is encountered that does not have a product description.

```
%macro format;
  /* Execute %Do loop according to the number of lines of data */
  %Do j = 1 %to &cnt;
    /* SAS statements between %macro statements are emitted in whole.*/
    %if &j = 1 %then
       %do;
          Proc Format;
            Value Prddesc
       %end;
            &&prod_id&j  =  "&&prod_desc&j"
    %if &j = &cnt %then
       %do;
             other = 'Unknown'
           ;
          Run;
       %end;
  %end;
%mend format;
```

Executing a Macro that Will Create a User Format

Options mprint guarantees that the SAS code generated by a macro will be displayed in the SAS log.

```
options mprint;
```

A compiled macro is called and executed by using the percent sign (%) in front of the name of the macro.

```
%format ;
```

When SAS code generated by a macro is displayed in the SAS log, Mprint and the name of the macro is displayed.

```
MPRINT(FORMAT):   Proc Format;
MPRINT(FORMAT):   Value Prddesc
                      10001 = "Red Truck"
                      10012 = "Green Bicycle"
                      10033 = "Blue Doll House"
                      10057 = "Orange Scarecrow"
                      10072 = "Yellow Songbird"
                      other = 'Unknown' ;
MPRINT(FORMAT):   Run;

NOTE: Format PRDDESC has been output.
```

User formats are executed with a PUT function that relates the description to the product id. Here is an example.

```
data work2;
  set work1;
  prod_desc = put(prod_id, proddesc.0);
  output;
run;
```

All of the assignment work is done in the background using a binary search that is quite efficient for assignments that have ten or more characteristics that could be assigned. Otherwise, an If-Then-Else assignment will do just as well.

Generating comments in the SAS log from a Macro.

There may be times when displaying comments is a SAS log from a macro may be helpful for understanding the actions taken by a macro, such as when user format is used to assign descriptions in SAS code. This is done by using the %Put statement to display the comment and ending semicolon (;). A %STR statement is used to differentiate the semicolon for the comment from the semicolon for the %Put statement. It hides the semicolon from the Macro processor.

```
%macro comment;
   /* toy prod descriptions. */
   %put;
   %put * This comment shows the descriptions assigned
          to each product id %str(;) ;
   %do j = 1 %to &cnt;
      %put * "&&prod_id&j" = "&&prod_desc&j" %str(;) ;
   %end;
   %put * Other = "Unknown" %str(;) ;
%mend comment;

%comment ;
```

```
* This comment shows the descriptions assigned to each product id ;
* "10001 " = "Red Truck" ;
* "10012 " = "Green Bicycle" ;
* "10033 " = "Blue Doll House" ;
* "10057 " = "Orange Scarecrow" ;
* "10072 " = "Yellow Songbird" ;
* Other = "Unknown" ;
```

Using a Macro to Generate a Variable List for Keep statements or Where Clauses or If Statements.

First, compile the macro.

```
%macro vargrp;
   %do j = 1 %to &cnt;
      %let varlist = %trim(&varlist &&prod_id&j);
   %end;
   ;
%mend vargrp;
```

Second, set the macro variable for the varlist to blank.

```
%let varlist = ;
```

Third, execute the macro to populate the varlist.

```
%vargrp;

MPRINT(VARGRP):
```

Fourth, verify that the varlist was generated using a %Put statement.

```
259  %put varlist=&varlist;

varlist=10001 10012 10033 10057 10072
```

Fifth, use the varlist where appropriate.

a) In a Set statement where clause that limits records read into a Data Step.

```
261  /* Select only matching prod_ids from work1 */
262  Data work3;
263    set work1(where=(prod_id in (&varlist)));
264    output;
265  run;

NOTE: There were 5 observations read from the data set WORK.WORK1.
      WHERE prod_id in (10001, 10012, 10033, 10057, 10072);
NOTE: The data set WORK.WORK3 has 5 observations and 3 variables.
NOTE: DATA statement used:
      real time           0.12 seconds
      cpu time            0.01 seconds
```

b) In a Proc SQL where statement that limits records read from a table.

```
266
267  /* Select only matching prod_ids from work1 */
268  Proc sql;
269    create table work3 as
270      select * from work1
271        where prod_id in (&varlist);
NOTE: Table WORK.WORK3 created, with 5 rows and 3 columns.
```

c) In a Data Step to limit the records output from the step.

```
275  /* Output only matching prod_ids to work3 */
276  Data work3;
277    set work1;
278    if prod_id in (&varlist) then
279      output;
280  run;

NOTE: There were 5 observations read from the data set WORK.WORK1.
NOTE: The data set WORK.WORK3 has 5 observations and 3 variables.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

d) In a Proc SQL to limit the records output to a table.

```
281
282  /* Output only matching prod_ids to work3 */
283  Proc sql;
284    create table work3 as
285      select * from work1
286        having prod_id in (&varlist);
NOTE: Table WORK.WORK3 created, with 5 rows and 3 columns.

287    quit;
NOTE: PROCEDURE SQL used:
      real time           0.01 seconds
```

Final Note, because Proc SQL can employ in-line views, the function of a &varlist can be replaced with an In-Operator that references a Select statement.

Here is an example that uses a Select statement in a Where clause.

```
38   Proc sql;
39     create table work3 as
40       select * from work1
41         where prod_id in (select distinct prod_id from work1);
NOTE: Table WORK.WORK3 created, with 5 rows and 4 columns.

42     quit;
```

Here is an example that uses the Select Statement in a Having Clause.

```
52   Proc sql;
53     create table work3 as
54       select * from work1
55         having prod_id in (select distinct prod_id from work1);
NOTE: Table WORK.WORK3 created, with 5 rows and 4 columns.

56     quit;
```