Advanced SAS® Macro Language Techniques

Destiny Corporation, Wethersfield, Ct

ABSTRACT

This paper will demonstrate the advanced uses of the macro language within the SAS® system. These include interaction between the data step and SAS macro language, dynamic creation of programs (which write themselves based on the incoming data!), use of **%DO/%END** loops, and advanced **SYMPUT/SYMGET** techniques. You will emerge with an understanding of macro language internals and enough new jargon to impress your co-workers. We anticipate that you will have been using macros for a minimum of one year.

CALL SYMPUT

Of all the macro - data step interfacing functions and routines, the **CALL SYMPUT** call routine is undoubtedly the most important and most useful of all. With **CALL SYMPUT** we have yet another way of creating a macro variable and giving it a value, but this time from within data step execution.

Remember, this is a data step routine; like **SYMGET**, it cannot be used anywhere other than in a data step.

The general form is:

```
call
symput(argument1,argument2);
                  •         •
                  •         •
                  •
••••• Source
                  •

••••••••••••••• Target
```

Source or Target may be:

- Literal String (i.e. enclosed in quotes)
- Data Step Variables
- Data Step Expressions

SYNTAX

- argument1 is the target - the macro variable to be created. It can either be a literal value enclosed in quotes, or a data step variable, or a data step expression - particularly useful for creating a series of macro variables.

```
call symput('mvar1',argument2);

call symput(dsvar,argument2);

call
symput('mvar'||left(_n_),argument2);
```

- argument2 is the value given to argument1. Again, argument2 can be a literal string, a data step variable or a data step expression.

```
call symput(argument1,'literal');

call symput(argument1,datavar);

call
symput(argument1,put(date,worddate18.));
```

COMBINATIONS OF FORM

PROGRAM EDITOR

```
data work1;
  var1 = 'value1';
  var2 = 'value2';
  call symput('mvar1','newvalue');          /*Example 1*/
  call symput('mvar2',var1);                /*Example 2*/
  call symput(var1,var2);                   /*Example 3*/
  call symput(var1,'newvalue');             /*Example 4*/
run;
```

In Example 1, the macro variable is called **mvar1** and the value it takes is **newvalue**.

In Example 2, the macro variable is called **mvar2** and the value it takes is **value1**.

In Example 3, the macro variable is called **value1** and the value it takes is **value2**.

In Example 4, the macro variable is called **value1** and the value it takes is **newvalue**.

Of these forms, the most common is the one shown in example 2 above, where the macro variable name is programmer-defined, but the value it takes comes from the data, often in combination with some justification or formatting function.

QUESTIONS

(a) which symbol table does the macro variable go in?

Most macro variables created in this way by the use of **Call SYMPUT** are placed in the global table. However, the variable will be placed in the nearest symbol table in the current referencing environment of the data step, providing that symbol table is not empty. If it is empty, it will be placed in the symbol table higher, providing that is not empty and so on.

(b) when is the macro variable available for use?

The most common mistake with the use of Call SYMPUT is to forget that the macro variable is *only available after the data step completes execution!*

(c) how does **CALL SYMPUT** format character values?

The default format is $w. where w is the width of the variable. Hence trailing blanks may be transferred also. Avoid this by the use of the trim function with the second argument:

```
  call symput('mvar1',trim(datavar));
```

(d) how does **CALL SYMPUT** format numeric values?

The default format is BEST12. with the number being right justified. You may need to use the left function to get your desired result:

```
  call symput('mvar1',left(datavar));
```

Example 1 - Data Dependent Titles

PROGRAM EDITOR

```
/* -----------------------------------------------
   Call SYMPUT Example 1.
   Create a data dependent title.
-----------------------------------------------*/
proc means data=saved.demograf noprint mean;
   var age;
   output out=average mean=avage;
run;
data _null_;
   set average;
   call symput('meanage',left(put(avage,2.)));
run;
proc print data =saved.demograf;
   title "Average age of trial sample is &meanage";
run;
```

The MEANS procedure produces a one-variable (avage), one-observation data set (average). The DATA _NULL_ step produces no output SAS data set, but is simply a vehicle for getting an independently executable step that will transfer the value of avage into a macro variable meanage. This, of course, will go into the global symbol table. Once the DATA _NULL_ step has completed, the macro variable is available for use, and is simply used in a usual TITLE statement. The output produced is:

OUTPUT

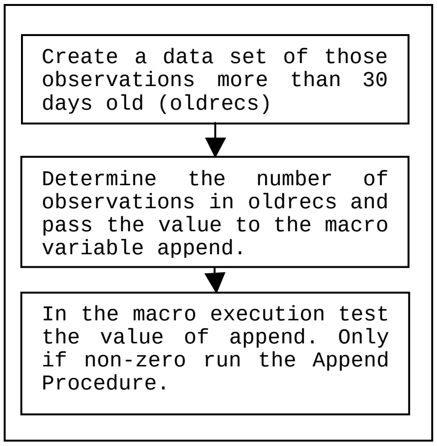
```
         Average age of trial sample is 33

OBS   AGE   GENDER   SALARY     STATUS     CHILDREN
CARS
 1    29     F          8000       D              2
1
 2    25     M         10000       D              2
1
 3    44     M         10000       D              3
1
```

Example 2 - Triggering a PROC based on a data step value

PROGRAM EDITOR

```
/* ----------------------------------------------------
   Call SYMPUT Example 2.
   Triggering a PROC based on a data step value
----------------------------------------------------*/
options mprint;
%macro archive;
   %if &append ^=0 %then %do;
      proc append base=arch data=oldrecs;
      run;
   %end;
   %else %put No archiving to be done;
%mend archive;
data new;
   input @1 date date7. reading1 reading2;
   cards;
01sep91  102 150
19aug91   98 143
05MAY90   98 142
07MAY90   90 140
21aug91   88 135
11MAY90   84 134
run;
data oldrecs;
   set new;
   if today() -date > 30 then output oldrecs;
run;
data _null_;
   if 0 then set oldrecs nobs=numobs;
   call symput('append',left(numobs));
   stop;
run;
%archive
```

MPRINT has been turned on to show the statement generated in the Log. The idea is to archive observations more than 30 days old. This example could be adapted to any dynamic file (say one under FSEDIT control) where it was important to get rid of old observations into some archive or backup file. The logic behind this routine is:



Create a data set of those observations more than 30 days old (oldrecs)

Determine the number of observations in oldrecs and pass the value to the macro variable append.

In the macro execution test the value of append. Only if non-zero run the Append Procedure.

Again a DATA _NULL_ step has been used to create the macro variable with **CALL SYMPUT**.

The line:

```
  if 0 then set oldrecs nobs=numobs;
```

uses the fact that the variable assigned to the nobs option is given its value at data step compile time. Therefore numobs holds the number of observations in the data set oldrecs without having to read an observation from it. Hence the dummy negative condition if 0. The stop is necessary. The normal way of terminating a data step is to reach an end-of-file marker on a raw data file or a SAS data set; if this is not present a STOP stops the data step trying to loop.

However, the whole point of the example is to test the number of observations in oldrecs. Only when this is greater than zero is the PROC APPEND step generated.

Here is the log from this job (run 19APR96):

LOG

```
70  /* ----------------------------------------------------
71     Call SYMPUT Example 2.
72     Triggering a PROC based on a data step value
73  ----------------------------------------------------*/
74  options mprint;
75  %macro archive;
76     %if &append ^=0 %then %do;
77        proc append base=arch data=oldrecs;
78        run;
79     %end;
80     %else %put No archiving to be done;
81  %mend archive;
82  data new;
83     input @1 date date7. reading1 reading2;
84     cards;

NOTE: The data set WORK.NEW has 6 observations and 3
variables.
NOTE: The DATA statement used 0.38 seconds.


91  run;
92  data oldrecs;
93     set new;
94     if today() -date > 30 then output oldrecs;
95  run;

NOTE: The data set WORK.OLDRECS has 6 observations and 3
variables.
NOTE: The DATA statement used 0.44 seconds.


96  data _null_;
97     if 0 then set oldrecs nobs=numobs;
98     call symput('append',left(numobs));
99     stop;
100  run;

NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
      98:30
NOTE: The DATA statement used 0.16 seconds.


101  %archive
MPRINT(ARCHIVE):     PROC APPEND BASE=ARCH
DATA=OLDRECS;
MPRINT(ARCHIVE):   RUN;

NOTE: Appending WORK.OLDRECS to WORK.ARCH.
NOTE: BASE data set does not exist.
      DATA file is being copied to BASE file.
NOTE: The data set WORK.ARCH has 6 observations and 3
variables.
NOTE: The PROCEDURE APPEND used 0.27 seconds.
```

Example 3 - Splitting a data set.

Consider the following extract from the data set SASUSER.BP1, concerning blood pressure measurements taken on various patients at various relative days over the course of a drug trial. Please refer to appendix 1 at end of paper.

Suppose we wish to take a data set such as SASUSER.BP1 which has multiple observations per patient, and construct a data set for each patient - to split up the data set by each patient for others to do separate analyses for individual patients.

To do this in normal open code would require a data step of the form:

PROGRAM EDITOR

```
data pat203 pat204 pat205....;
  set saved.bp1;
       if patient=203 then output pat203;
  else if patient=204 then output pat204;
  .
  .
  run;
```

The problems here are:

- The number of output data sets can vary
- The patient variable is numeric, so a character prefix is required for the output data set name
- The first IF statement is plain, all the others need an ’ELSE’

PROGRAM EDITOR

```
/* ----------------------------------------------------
  Call SYMPUT Example 3.
  Splitting a data set.
-------------------------------------------------------*/
%macro split(inputds,byvar,prefix);
  proc freq data=&inputds;
    tables &byvar / noprint out=numbys(keep=&byvar);
  run;
  data _null_;
    set numbys end=eof;
    call symput(’mvar’||left(put(_n_,2.)),
         left(put(&byvar,3.)));
   if eof then call symput(’numobs’,put(_n_,2.));
  run;
  data
    %do i = 1 %to &numobs;
      &prefix&&mvar&i
    %end;
  ;
    set &inputds;
  %let else=;
  %do i= 1 %to &numobs;
  &else if &byvar=&&mvar&i then
    output &prefix&&mvar&i;
    %let else=else;
  %end;
run;
%mend split;
%split(saved.bp1,patient,ds)
```

Note the parameters are:

- the input data set
- the variable to be used for the split
- the prefix to the output data set names

The output from the original PROC FREQ step gives one observation per value of **&byvar**.

OUTPUT

```
 OBS    PATIENT


   1      203
   2      204
                 ...etc
  68      302
  69      303
```

The DATA _NULL_ step creates multiple macro variables, one per each observation of the output data set from the PROC FREQ step. The second **CALL SYMPUT** gives a macro variable containing the number of observations in the data set, so as to give a variable ceiling to the subsequent loops.

Global Symbol Table

Variable	Value
mvar1	203
mvar2	204
mvar3	206
etc to...	
mvar67	301
mvar68	302
mvar69	303
numobs	69

Because the number of observations is variable, the number of macro variables generated is variable, and indirect reference to them must be used.

Note that the first %DO loop generates a variable DATA statement, the second generates the IF..THEN..ELSE statements, omitting the ELSE from the first one.

Creating a Macro Variable from Proc SQL

Another way of creating and passing a value to a macro variable is via a query in Proc SQL. The result of the query must be one value to be assigned to the macro variable:

PROGRAM EDITOR

```
proc sql;
  select avg(salary)
    into :mvar1
    from saved.demograf;
quit;
title "Average salary of the sample is &mvar1";
```

As each Proc SQL statement executes separately, the macro variable (here called mvar1) created is available for immediate use.

```
PATIENT TREATMNT  DAY SUPSYS SUPDIA SUPPUL STDSYS STDDIA STDPUL

  203    DRUG B   -29   178    112    75     188    115    78
  203    DRUG B   -14   168    109    72     147    107    84
  203    DRUG B     0   154     95    60     149    108    69
  203    DRUG B    14   164    101    56     142     98    60
  203    DRUG B    28   160    100    54     185    103    54
  203    DRUG B    42   137     92    51     143     91    60
  203    DRUG B    56   157     92    60     170    116    60
  203    DRUG B    84   170    100    58     170    100    60
  203    DRUG B   110   177    100    56     180    103    58
  203    DRUG B   140   160     97    66     161    103    66
  204    DRUG A   -29   233    110    84     233    110    90
  204    DRUG A   -14   209    100    84     214    103    78
  204    DRUG A     0   211     91    72     209     89    72
  204    DRUG A    14   215     87    66     224     92    72
  204    DRUG A    28   180     85    75     205     93    84
  204    DRUG A    42   159     80    60     155     84    78
  ....and many more to 69 patients.
```

SAS is a registered trademark or a trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Copyright © 2002 Destiny Corporation. All rights reserved.

Prepared by

Destiny Corporation
100 Great Meadow Rd Suite 601
Wethersfield, CT 06109-2379
Phone: (860) 721-1684 - 1-800-7TRAINING
Fax: (860) 721-9784
Web: www.destinycorp.com
Email: info@destinycorp.com
Copyright © 2002

****