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('mvar1',argument2);
call symput(dsvar,argument2);
call symput('mvar'||left(_n_),argument2);
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*/
runt;
```

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:

```sas
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:

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

Example 1 - Data Dependent Titles

```sas
/* ------------------------------------------------
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";
```

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

```sas
/* ------------------------------------------------------
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
```

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:

```
<table>
<thead>
<tr>
<th>OBS</th>
<th>CARS</th>
<th>AGE</th>
<th>GENDER</th>
<th>SALARY</th>
<th>STATUS</th>
<th>CHILDREN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>F</td>
<td>8000</td>
<td>D</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>M</td>
<td>10000</td>
<td>D</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>M</td>
<td>10000</td>
<td>D</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```

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

```
DATA _NULL_;  
  if 0 then set oldrecs nobs=numobs;  
  call symput('append',left(numobs));  
  stop;  
RUN;  
```

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
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;
```

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
```
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'

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.

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

<table>
<thead>
<tr>
<th>PATIENT</th>
<th>TREATMNT</th>
<th>DAY</th>
<th>SUPSYS</th>
<th>SUPDIA</th>
<th>SUPPUL</th>
<th>STDSYS</th>
<th>STDDIA</th>
<th>STDPUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>-29</td>
<td>178</td>
<td>112</td>
<td>75</td>
<td>188</td>
<td>115</td>
<td>78</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>-14</td>
<td>168</td>
<td>109</td>
<td>72</td>
<td>147</td>
<td>107</td>
<td>84</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>0</td>
<td>154</td>
<td>95</td>
<td>60</td>
<td>149</td>
<td>108</td>
<td>69</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>14</td>
<td>164</td>
<td>101</td>
<td>56</td>
<td>142</td>
<td>98</td>
<td>60</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>28</td>
<td>160</td>
<td>100</td>
<td>54</td>
<td>185</td>
<td>103</td>
<td>54</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>42</td>
<td>137</td>
<td>92</td>
<td>51</td>
<td>143</td>
<td>91</td>
<td>60</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>56</td>
<td>157</td>
<td>92</td>
<td>60</td>
<td>170</td>
<td>116</td>
<td>60</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>84</td>
<td>170</td>
<td>100</td>
<td>58</td>
<td>170</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>110</td>
<td>177</td>
<td>100</td>
<td>56</td>
<td>180</td>
<td>103</td>
<td>58</td>
</tr>
<tr>
<td>203</td>
<td>DRUG B</td>
<td>140</td>
<td>160</td>
<td>97</td>
<td>66</td>
<td>161</td>
<td>103</td>
<td>66</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>-29</td>
<td>233</td>
<td>110</td>
<td>84</td>
<td>233</td>
<td>110</td>
<td>90</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>-14</td>
<td>209</td>
<td>100</td>
<td>84</td>
<td>214</td>
<td>103</td>
<td>78</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>0</td>
<td>211</td>
<td>91</td>
<td>72</td>
<td>209</td>
<td>89</td>
<td>72</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>14</td>
<td>215</td>
<td>87</td>
<td>66</td>
<td>224</td>
<td>92</td>
<td>72</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>28</td>
<td>180</td>
<td>85</td>
<td>75</td>
<td>205</td>
<td>93</td>
<td>84</td>
</tr>
<tr>
<td>204</td>
<td>DRUG A</td>
<td>42</td>
<td>159</td>
<td>80</td>
<td>60</td>
<td>155</td>
<td>84</td>
<td>78</td>
</tr>
</tbody>
</table>

...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.