The SAS® Data step/Macro Interface
Lawrence Heaton-Wright, Quintiles, Bracknell, Berkshire, UK

ABSTRACT
The SAS® macro facility is an extremely useful part of the SAS® System. However, macro variables and the macros themselves do not have to be exclusively referenced from within the macro facility. The SAS® data step can create, resolve and execute macros using 2 functions: SYMGET and RESOLVE and 2 routines: CALL SYMPUT and CALL EXECUTE.

This paper aims to demonstrate the usefulness and relative ease of two of these functions:

The CALL SYMPUT routine allows the creation of SAS macro variables from within a SAS data step
The CALL EXECUTE routine allows the execution of a SAS macro from within a SAS data step

The audience for this paper does not require a high level of technical expertise but should be familiar with the workings of the SAS macro facility. The contents are not restricted to one particular operating system.

INTRODUCTION
The SAS® macro facility is an extremely useful part of the SAS® System. However, macro variables and the macros themselves do not have to be exclusively referenced from with the macro facility.

The data step interface consists of 2 routines and 2 functions that can create, resolve and execute macros using 4 functions: CALL SYMPUT, SYMGET, RESOLVE and CALL EXECUTE.

The CALL SYMPUT routine allows the creation of macro variables from within a data step.

The CALL EXECUTE routine allows the execution of a macro from within a data step.

The RESOLVE and SYMGET functions allow the conversion of macro variables into data step variable values.

This paper will be concentrating on the two routines: CALL SYMPUT and CALL EXECUTE.

THE CALL SYMPUT ROUTINE
This routine is used by the programmer to create macro variables using either data step variable values or data set information.

The basic syntax for this routine is:

CALL SYMPUT (macro-variable , value);

Macro-variable is the name of a macro variable to be initialised with value. This can be a character string that is a valid SAS® name enclosed in quotes, the values of a character variable in a data step or a character expression.

Value is the value to be assigned to macro-variable. This can be a character string enclosed in quotes, the value of a data step variable which can be character or numeric or a data step expression.

FIXED MACRO VARIABLE CREATION

CALL SYMPUT ('exvar', 'testing');

The character string testing is assigned to the macro variable EXVAR.

CREATED FROM DATA STEP VARIABLES

DATA _p_values;
  INPUT test : $8. label : $25.;
  CALL SYMPUT (test, label);
CARDS;
  Exact Fisher's Exact
  CMH Cochran-Mantel-Haenzel
;RUN;
%PUT _USER_;

LOG file:
GLOBAL CMH Cochran-Mantel-Haenzel
GLOBAL EXACT Fisher's

The character string Fisher's Exact is assigned to the macro variable EXACT.
The character string Cochran-Mantel-Haenzel is assigned to the macro variable CMH.

DYNAMIC CREATION USING EXPRESSIONS

DATA totals;
  INPUT trt total @@;
  CALL SYMPUT ("trt" || PUT(trt,1.),
               COMPRESS("N="||PUT(total,3.)));
CARDS;
  1 100 2 120 3 85
;RUN;
%PUT _USER_;

LOG file:
GLOBAL TRT1 N=100
GLOBAL TRT2 N=120
GLOBAL TRT3 N=85

The character string N=100 is assigned to the macro variable TRT1.
The character string N=120 is assigned to the macro variable TRT2.
The character string N=85 is assigned to the macro variable TRT3.

LOG file:
GLOBAL TRT1 N=100
GLOBAL TRT2 N=120
GLOBAL TRT3 N=85

TIMING OF MACRO VARIABLE CREATION
Remember the macro variable created using CALL SYMPUT is not available for use until the data step has executed.

DATA not_available;
  test='23rd October 2002';
  CALL SYMPUT ('NotAvail',test);
  retrieve="&NotAvail";
RUN;

LOG file:
1
DATA blanks;
LENGTH test $8.;
test='a';
CALL SYMPUT ('TrailBl',test);
CALL SYMPUT ('TrimBl',TRIM(test));
RUN;

FORMATTING OF MACRO VARIABLES
I would recommend using the TRIM, LEFT, COMPRESS and COMPBL functions as appropriate when creating macro variables, otherwise the macro variables created will possibly have trailing or leading blanks.

DATA blanks;
LENGTH test $8.;
test='a';
CALL SYMPUT ('TrailBl',test);
CALL SYMPUT ('TrimBl',TRIM(test));
RUN;

%PUT Macro variable with trailing blanks = ***&TrailBl***;
%PUT Macro variable without trailing blanks = ***&TrimBl***;

Submitting this section of code produces this in the log file:

Macro variable with trailing blanks = a       ***
Macro variable without trailing blanks = a***

THE CALL EXECUTE ROUTINE
A programmer can use this routine to enable execution of macros from within a data step.

Probably one of the best uses of this routine is when using the data step iteration language [DO loops, etc.] to execute a macro using the values from the iteration.

To illustrate the simple usage of this routine, the following DATA _NULL_ steps will execute this macro:

%MACRO PrnDSet (DataSet/report, Where=treat IS NOT MISSING);
  TITLE1 "CALL EXECUTE EXAMPLE";
  TITLE2 "DATASET=&DataSet, WHERE CLAUSE='&Where'";
  PROC PRINT DATA=&DataSet;
    WHERE &Where;
  RUN;
%END PrnDSet;

THE BASIC CALL

DATA _NULL_;
  SET report END=eod;
  IF eod THEN DO;
    CALL EXECUTE('%PrnDSet (DataSet=report)');
  END;
RUN;

Producing a null report when there are no records in a data set
Check the SASHELP library for the VTABLE SAS View for no observations in a report data set.
Create a macro variable containing the number of observations in a report data set.
If number of observations equals zero then invoke the null report macro.

DATA _NULL_;
  SET SASHELP.vtable
    (WHERE=(&libname='WORK' AND
             memname='REPORT'));
  CALL SYMPUT
    ('Nobs',COMPRESS(PUT(Nobs,BEST.)));
RUN;

%IF %EVAL(&Nobs)=0 %THEN %DO;
  invoke null report macro
%END;
%ELSE %DO;
  invoke report macro
%END;

PRODUCING PATIENT NARRATIVE LISTINGS
Use CALL EXECUTE to initiate patient printouts from a selected list of patients:

```sql
PROC SQL;
    CREATE TABLE subjid AS
        SELECT DISTINCT subjid FROM RAW.ae
            WHERE serious='Y'
        ORDER BY subjid;
QUIT;
```

```data
DATA _NULL_;
    SET subjid;
    BY subjid;
    CALL EXECUTE('%PrnData
        (subjid='||PUT(subjid,4.)||')');
RUN;
```

**OTHER MACRO INTERFACES**

One of the other ways of creating a macro variable without using %LET or CALL SYMPUT is using PROC SQL. This is an extremely useful method of creating macro variables. In fact for the earlier example of accessing the SASHELP data views, I would normally use the PROC SQL methodology as this appears to be the most efficient method (both in terms of CPU and real time) of accessing these data views.

```sql
PROC SQL NOPRINT;
    SELECT DISTINCT PUT(nobs,BEST.)
        INTO :nobs
    FROM SASHELP.vtable
        WHERE libname="WORK" AND
            memname="REPORT";
QUIT;
```

The macro creation is carried out by the use of the INTO : clause. The NOPRINT option is useful here as it will stop the value of NOBS [from SASHELP.vTABLE] being printed to the output window.

The INTO clause can also be used to create a macro variable containing all values of a column by using the SEPARATED BY clause.

```sql
PROC SQL NOPRINT;
    SELECT DISTINCT subjid INTO :listsubj
        SEPARATED BY "," FROM RAW.ae;
QUIT;
```

In this case the macro variable, LISTSUBJ, would contain a unique list of subject ids from the adverse event data set delimited by a comma.

The SYMGET function can be used to retrieve macro variable values into data step variables.

```data
DATA temp;
    SysProcessName=SYMGET('SysProcessName');
RUN;
```

TEMP data set contains:

<table>
<thead>
<tr>
<th>Obs</th>
<th>SYSPROCESSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DMS Process</td>
</tr>
</tbody>
</table>

**IN CONCLUSION**

In conclusion, we can use CALL SYMPUT to create macro variables that can contain data step variable values. CALL EXECUTE can be used to invoke macros from within a data step.

I would suggest that the use of these two functions would enhance programs by allowing users to interface successfully between macros and the data step.

**REFERENCES**

SAS® Macro Language: Reference, Version 8

Data Set used in Call Execute example programs:

```data
DATA report;
    LENGTH outvar $15.;
    INPUT treat count @@;
    total=322;
    percent=(count/total)*100;
    outvar=PUT(count,3.) || "/" || 
        PUT(total,3.) || 
        " (" || 
        PUT(percent,2.) || "%)");
CARDS;
 1 100 2 123 3 99
RUN;
```

**CONTACT INFORMATION**

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

Lawrence Heaton-Wright
Quintiles Limited
Station House, Market Street
Bracknell, Berkshire, RG12 1HX
United Kingdom
Work Phone: +44 1344 708320
Fax: +44 1344 708106
Email: lawrence.heaton-wright@quintiles.com
Web: www.quintiles.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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