**MWSUG 2017 – BB142**

# DOSUBL and the Function Style Macro

John Henry King, Ouachita Clinical Data Services, Inc. Caddo Gap Arkansas

## ABSTRACT

The introduction of the SAS® function DOSUBL has made it possible to write certain function style macros that were previously impossible or extremely difficult. This Beyond the Basics talk will discuss how to write a function style macro that uses DOSUBL to run SAS code and return an "Expanded Variable List" as a text string. While this talk is directed toward a specific application the techniques can be more generally applied.

## INTRODUCTION

The type of function style macro we will discuss is a macro that returns text that can be assigned to a macro variable or used in the generation of SAS code. For example the macro will appear to function in the same way as true macro functions %LENGTH or %SUBSTR simply returning a character string. The macro will use DOSUBL to execute SAS data and procedure steps that would normally generate step boundaries preventing the macro from behaving as a function.

## BASIC DOSUBL

If we look at the online documentation for DOSUBL we find a rather pithy summary, "*Imports macro variables from the calling environment, and exports macro variables back to the calling environment*." If we look a bit further down that page to details we find a bit more to chew on in this sentence, "*The DOSUBL function enables the immediate execution of SAS code after a text string is passed. Macro variables that are created or updated during the execution of the submitted code are exported back to the calling environment*." "Immediate execution of SAS code" sounds interesting as this allows an executing macro to pause and execute SAS code without causing the generation of a step boundary in the calling environment, allowing the macro to perform as a true function. Macro variables that are created or update allow communication with the calling macro which allows the macro to return the value of a macro variable, again in true function style. Other SAS objects, like SAS data files, remain in the library where they are created and are available in subsequent steps.

## DOSUBL BUG

There is a bug in DOSUBL with regards to GLOBAL/LOCAL macro variables with the same name. This SAS-L Thread discussed the problem and provides the example shown here. If a LOCAL macro variable has the same name as an existing GLOBAL variable the value of the GLOBAL variable is exported back to the calling environment. This happens whether the macro variable is modified or not by the code executed by DOSUBL. Consider this example where macro variables A B C are global and a macro defines local variables with the same names. After calling DOSUBL the local macro variables *magically* have the values of the global macro variables.

```
1  %let a = 1;
2  %let b = 2;
3  %let c = 3;
4
5  %macro testdosubl;
6      %local a b c;
7      %let a = 10;
8      %let b = 20;
9      %let c = 30;
10     %put NOTE: Before;
11     %put _LOCAL_;
12     %put NOTE: DOSUBL Called %sysfunc(dosubl(%str(data _null_;run;)));
13     %put NOTE: After;
14     %put _LOCAL_;
15     %mend testdosubl;
16
17 %testdosubl
```

```
NOTE: Before
TESTDOSUBL A 10
TESTDOSUBL B 20
TESTDOSUBL C 30
NOTE: DOSUBL Called 0
NOTE: After
TESTDOSUBL A 1
TESTDOSUBL B 2
TESTDOSUBL C 3
```

**SAS Log Output of DOSUBL Bug**

From the LOG output show here it is evident that local variables A=10, B=20, C=30 and have changed to the values from the global variables of the same name now local A=1, B=2, C=3.

However this can be easily worked around by creating a unique variable name and SAS provides for that with the function %SYMEXIST.

## THE APPLICATION

The application is a SAS Variable List expander; a program that resolves or expands the variable names represented by a SAS Variable List into individual variable names. The program minimally accepts as input a data set name and a SAS Variable List or a list of variable names or some mixture, e.g.

1. A list of space delimited names; AGE GENDER WEIGH HEIGHT
2. A name prefix list, a name followed by a colon SALES:
3. A name range list, two names connected by double dash -- that may or may not include the intervening keywords -numeric- or -character-
4. Special SAS name lists, _CHARACTER_, _NUMERIC_, and _ALL_
5. Numbered range lists, X1-X100

The minimum output of the program consists of a string of individual variable names, (the expanded variable list) for example if the input data is SASHELP.CLASS and the SAS Variable List is _NUMERIC_ the program returns Age Height Weight. Also included are options to control which variables are available for expansion through the use of DROP and KEEP options. The output string can be varied from a simple list of words through the use of an expression, for example a rename list could be generated using `CATX('=',_name_,cats('NEW_',_name_))` to produce a new list of old-name=new-name rename pairs. Also, through the use of the WHERE option which specifies a WHERE data set option applied to the OUT data set from PROC TRANSPOSE the list may be further qualified perhaps using LIKE operator to output list of variables with a common suffix for example `_name_ like '%Status'`.

## BASIC CODE

The basic code to expand any “Sas Variable List” and store it in a macro variable is the following simple two-step process. I discussed this in some detail in “Atypical Applications of Proc Transpose”

```
proc transpose data=sashelp.class(obs=0) out=variables;
   var _all_;
   run;
proc sql noprint;
   select _name_ into :expvarlist separated by ' ' from variables;
   quit;
```

When PROC TRANSPOSE is run using OBS=0 no values are transposed (no observations are read), the only output is one observation for each time a variable is named in the VAR statement. The technique takes advantage of the fact that the VAR statement knows how to process the all the various types of “SAS Variables Lists” and importantly preserves the order of the variable names as stated in the VAR statement or implied by expansion of the list. The SQL step should be familiar to most; it simply places the output of the expression, for all rows of the data specified in FROM clause of the SELECT statement into a macro variable (EXPVARLIST) delimited by character(s). One delimited string for each row in DATA variables, created by OUT= from PROC TRANSPOSE. The macro will package and generalize this basic code into a powerful macro function.

## DESCRIPTION OF MACRO PARAMETERS

- DATA=_LAST_, specifies the SAS data set or view to use as input to PROC TRANSPOSE.

- VAR=_ALL_, specifies the “SAS Variable List” to expand.
- WHERE=1, is a WHERE expression that is applied to the OUT= data set allowing the expanded variable list to be modified. It is particularly useful to create a list based on a variable name suffix.
- EXPR=NLITERAL(&NAME), the expression used in PROC SQL SELECT statement to generate the delimited output string. The default uses the NLITERAL function to create name literals for any variable that needs to be written as a name literal. The expression lends a great deal of flexibility and extends the power of the macro far beyond a simple list expander. Using character manipulation functions syntactic constructs that can be quite varied.
- KEEP=, data set option applied to DATA in the PROC TRANSPOSE step which can be used to further qualify the variables that are “seen” by the VAR statement. For example to create a list of all character variables whose names begin with VISIT use KEEP=_CHARACTER_,VAR=VISIT:
- DROP=, data set option applied to DATA in the PROC TRANSPOSE step. See KEEP=.
- OUT=, this parameter is used to name the output data set created by PROC TRANSPOSE. Normally the data created by PROC TRANSPOSE is deleted by the macro, specifying a name saves the data set. This data set is indexed by _NAME_ and _INDEX_ allow a name to be looked up by variable number (_INDEX_) or variable number used to be looked up a variable name (_NAME_).
- NAME=_NAME_, specifies the PROC TRANSPOSE statement option NAME.
- LABEL=_LABEL_, specifies the PROC TRANSPOSE statement option LABEL.
- INDEX=_INDEX_, names the index variable (row number) in OUT. The macro creates this variable and adds it to OUT and creates an INDEX for the variable. The index can be used to lookup a name based on the row number.

## EXAMPLES

Consider the data set SASHELP.HEART which has the following CONTENTS the following examples will use the variables from this data.

**Variables in Creation Order**

| # | Variable | Type | Len | Label |
|---|---|---|---|---|
| 1 | Status | Char | 5 | |
| 2 | DeathCause | Char | 26 | Cause of Death |
| 3 | AgeCHDdiag | Num | 8 | Age CHD Diagnosed |
| 4 | Sex | Char | 6 | |
| 5 | AgeAtStart | Num | 8 | Age at Start |
| 6 | Height | Num | 8 | |
| 7 | Weight | Num | 8 | |
| 8 | Diastolic | Num | 8 | |
| 9 | Systolic | Num | 8 | |
| 10 | MRW | Num | 8 | Metropolitan Relative Weight |
| 11 | Smoking | Num | 8 | |
| 12 | AgeAtDeath | Num | 8 | Age at Death |
| 13 | Cholesterol | Num | 8 | |
| 14 | Chol_Status | Char | 10 | Cholesterol Status |
| 15 | BP_Status | Char | 7 | Blood Pressure Status |
| 16 | Weight_Status | Char | 11 | Weight Status |
| 17 | Smoking_Status | Char | 17 | Smoking Status |

Output 1 shows the most basic output produced by calling the macro and specifying only the input data set, a list of all variable names delimited by space.

```
Status DeathCause AgeCHDdiag Sex AgeAtStart Height Weight Diastolic
Systolic MRW Smoking AgeAtDeath Cholesterol Chol_Status BP_Status
Weight_Status Smoking_Status
```

**Output 1. Output from** `%expand_varlist(data=sashelp.heart)`

Output 2 shows how this list could be modified for use in an SQL select statement by changing the delimiter to a comma and a space. Adding the space after comma makes it a bit more readable and is still acceptable syntax for PROC SQL.

```
Status, DeathCause, AgeCHDdiag, Sex, AgeAtStart, Height, Weight,
Diastolic, Systolic, MRW, Smoking, AgeAtDeath, Cholesterol,
Chol_Status, BP_Status, Weight_Status, Smoking_Status
```

**Output 2. Output from** `%expand_varlist(data=sashelp.heart,dlm= ', ')`

Suppose we only want variables that end with the word "Status". Output 3 shows the addition of the WHERE parameter to subset the list using the LIKE operator. Note the word "Status" begins and ends with the substring "Status".

```
Status Chol_Status BP_Status Weight_Status Smoking_Status
```

**Output 3. Output from** `%expand_varlist(data=sashelp.heart,where=_name_ like '%Status')`

The output from PROC TRANSPOSE also includes the variable _LABEL_ which contains variable labels and could be used to generate a list that includes labels. In Output 4 the macro uses CATX in the EXPR parameter to generated variable-name = "variable-label" delimited by blanks.

```
Status=" " DeathCause="Cause of Death" AgeCHDdiag="Age CHD
Diagnosed" Sex=" " AgeAtStart="Age at Start" Height=" " Weight=" "
Diastolic=" " Systolic=" " MRW="Metropolitan Relative Weight"
Smoking=" " AgeAtDeath="Age at Death" Cholesterol=" "
Chol_Status="Cholesterol Status" BP_Status="Blood Pressure Status"
Weight_Status="Weight Status" Smoking_Status="Smoking Status"
```

**Output 4. Output from** `%expand_varlist(data=sashelp.heart,`
`expr=catx('=',nliteral(_name_),quote(trim(_label_))))`

## MACRO CODE

```
 I.   %macro
         expand_varlist /*Returns an expanded variable list and optionally creates an indexed data set of variable names*/
            (
               data    = _LAST_,  /*[R]Input data*/
               var     = _ALL_,   /*[R]Variable List expanded*/
               where   = 1, /*[R]Where clause to subset OUT=, useful for selecting by a name suffix e.g. where=_name_ like '%_Status'*/
               expr    = nliteral(&name), /*[R]An expression that can be used to modify the names in the expanded list*/
               keep    = ,        /*[O]Keep data set option for DATA=*/
               drop    = ,        /*[O]Drop data set option for DATA=*/
               out     = ,        /*[O]Output data indexed by _NAME_ and _INDEX_*/
               name    = _NAME_,  /*[R]Name of the variable name variable in the output data set*/
               label   = _LABEL_, /*[R]Name of the variable name label variable in the output data set*/
               index   = _INDEX_, /*[R]Name of the variable index variable in the output data set*/
               dlm     = ' '      /*[R]List delimiter*/
            );
 II.   %local m i;
    a)  %let i=&sysindex;
    b)  %let m=&sysmacroname._&i;
 III.  %do %while(%symexist(&m));
    a)  %let i = %eval(&i + 1);
    b)  %let m=&sysmacroname._&i;
    c)  %end;
 IV.   %put NOTE: &=m is a unique symbol name;
 V.    %local rc &m code1 code2 code3 code4;
 VI.   %if %superq(out) ne %then %let code3 = %str(data &out(index=(&index &name)); set &out; &index+1; run;);
 VII.  %else %do;
    a)  %let out=%str(work._deleteme_);
    b)  %let code3 = %str(proc delete data=work._deleteme_; run;);
    c)  %end;
 VIII. %let code1 = %str(options notes=0; proc transpose name=&name label=&label data=&data(obs=0 keep=&keep drop=&drop)
       out=&out(where=(&where)); var &var; run;);
 IX.   %let code2 = %str(proc sql noprint; select &expr into :&m separated by &dlm from &out; quit;);
 X.    %let code4 = %str(options notes=1;);
 XI.   %let rc=%sysfunc(dosubl(&code1 &code2 &code3 &code4));
 XII.  &&&m.
 XIII. %mend expand_varlist;
```

I. Macro definition, see Description of macro parameters for details.
II. Define two local variables &I, a counter, and &M an arbitrary name.
    a. &I is assigned the value of &SYSINDEX an automatic macro variable maintained by SAS and incremented each time a macro is called. The combination of this value and &M will be the starting point for finding a unique
    b. &M is the concatenation of &SYSMACRONAME, an automatic macro variable that contains the name of the executing macro, i.e. EXPAND_VARLIST.
III. Test if &M exists. This test is for GLOBAL or LOCAL variable. If the name in &M is a macro variable the WHILE loop executes, if not the value of &M is unique.
    a. The value of &M is not unique so Increment the value of &I.
    b. Create a new name and store it in &M.
IV. The loop continues until a unique name is found and that name is printed on the LOG. Perhaps EXPAND_VARLIST999.
V. Create more local macro variables including one for the name contained in &M. RC is the return code from DOSUBL and CODE1-CODE4 are code fragments passed as the argument to DOSUBL. These variables are not required to be unique as they are not used after the call to DOSUBL so we don't if they might have been affected by the DOSUBL bug.
VI. When OUT is specified (not blank) the program creates a saved version of the output from PROC TRANSPOSE. A new variable _INDEX_ is added and the data set is index on both _NAME_ and _INDEX_.
VII. When OUT is not specified.
    a. Assign a value to &OUT
    b. Assign CODE3 a step to delete &OUT
VIII. CODE1 the PROC TRANSPOSE.
IX. CODE2 is the PROC SQL step.
X. CODE4
XI. Put CODE1-CODE4 macro variables into DOSUBL function and call using SYSFUNC. RC will be an indication of success or failure of DOSUBL.
XII. &&&M is resolved to &EXPAND_VARLIST999 which resolves to the output created by SQL INTO and is returned to the caller as described in the online documentation for DOSUBL.
XIII. Macro End.

## CONCLUSION

We've seen what I believe to be an interesting example of DOSUBL perhaps this will inspire you to try it.

## REFERENCES


Langston, Rick. 2103. "Submitting SAS® Code On The Side". SAS Global Forum 2013, San Francisco, California.
Available at https://support.sas.com/resources/papers/proceedings13/032-2013.pdf.

Rhoads, Michael. 2012. "Use the Full Power of SAS in Your Function-Style Macros". North East SAS Users Group 2012, Baltimore, Maryland.
Available at https://www.nesug.org/Proceedings/nesug12/bb/bb14.pdf.

King, John. 2012. "Atypical Applications of Proc Transpose". PharmaSUG 2012, San Francisco, California.
Availabe at http://www.lexjansen.com/pharmasug/2012/TF/PharmaSUG-2012-TF12.pdf