Viva la Resolución!
5 Things You Should Know About Macro Resolution

Daniel R. Bretheim, Towers Watson, Arlington, VA

ABSTRACT
This paper introduces five topics about macro resolution that are useful to every SAS® programmer:
1. Macro-Related Options
2. Understanding Macro Symbol Tables
3. When are macro variables created and accessible for use?
4. Changing Default Macro Variable Values
5. Forward Re-Scan Rule

#1: Macro-Related Options
There are several SAS system options that can display information about the execution of a macro in the SAS log. This information can be very helpful for debugging programs.

- Use the MPRINT option to -> print in the SAS log the text that is sent to the SAS compiler as a result of macro execution.
- Use the MLOGIC option to -> print messages in the SAS log that indicate macro actions that were taken during macro execution.
- Use the SYMBOLGEN option to -> see the results of resolving macro variable references in the SAS log. The SAS log will display the macro variable name and the resolved value.

Examples of all three options are presented below (see #4).

#2: Understanding Macro Symbol Tables
The value of a macro variable is stored in a symbol table. There are two types of symbol tables: Global and Local.

- Global Symbol Table – The global symbol table is created during the initialization of a SAS session and is deleted at the end of the session. Automatic macro variables are stored in the global symbol table and user-defined macro variables that you create with a %LET statement in open code are also stored in the global symbol table.

Macro variables in the global symbol table
  o are available anytime during the session
  o can be created by a user
  o have values that can be changed during the session (except for some automatic macro variables).

You can create a global macro variable with
  o a %LET statement (used outside a macro definition)
  o a DATA step that contains a SYMPUT routine (creates a macro variable and assigns to that variable any value that is available in the DATA step)
  o a DATA step that contains a SYMPUTX routine (also automatically removes leading and trailing blanks from both arguments)
  o a SELECT statement that contains an INTO clause in PROC SQL
  o a %GLOBAL statement.

- Local Symbol Table – The local symbol table is created when a macro that includes a parameter list is called or when a request is made to create a local variable during macro execution. The local symbol
table is deleted when the macro finishes execution. That is, the local symbol table exists only while the macro executes.

The local symbol table contains macro variables that can be

- created and initialized at macro invocation
- created or updated during macro execution
- referenced anywhere within the macro.

You can create local macro variables with

- parameters in a macro definition
- a %LET statement within a macro definition
- a DATA step that contains a SYMPUT routine within a macro definition
- a DATA step that contains a SYMPUTX routine within a macro definition
- a SELECT statement that contains an INTO clause in PROC SQL within a macro definition
- a %LOCAL statement.

When the %GLOBAL and %LOCAL statement are not used, macro variables are generally
- global when they are defined outside of a macro
- local to a macro when they are defined inside of a macro

There are a number of important, and oftentimes subtle, exceptions to these two rules. However, referencing scopes are easily managed if you either define macro variables outside of all macros or pass macro variable values into and out of each macro as parameters.

#3: When are macro variables created and accessible for use?

The following examples illustrate the availability of macro variable values. The %PUT statement and the following options will be used to write text and the current values of macro variables to the SAS log.

```
%global_
lists user-created macro variables that will be available in all of the referencing environments.

%local_
lists user-defined macro variables that are available only in the current or local referencing environment. This option must be used from within a macro.

%user_
lists all of the user-created macro variables in each of the referencing environments.

- Global macro variable – created outside of any macro. Note: The value of all AUTOMATIC and GLOBAL macro variables is stored in the view SASHELP.VMACRO.

```

<table>
<thead>
<tr>
<th>Obs</th>
<th>scope</th>
<th>name</th>
<th>offset</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GLOBAL</td>
<td>OUTSIDE</td>
<td>0</td>
<td>AAA</td>
</tr>
</tbody>
</table>

NOTE: There were 1 observations read from the data set SASHELP.VMACRO.

```

** Global macro variable - created outside of any macro. ;
%let outside = AAA;

proc print data=sashelp.vmacro;
   where scope='GLOBAL';
run;
```

Fig. 1
* Local macro variable – created inside of a macro.

** Local macro variable - created inside of a macro. ;
%macro one;
  %let inone = BBB;
  %put _local_;
  %put global_;
%mend;

%one
ONE INONE BBB
GLOBAL OUTSIDE AAA

* The local macro variable INONE resolves inside the macro, but outside ;
* the macro it does not exist and therefore does not resolve. ;
%put *** The value of macro variable %NRSTR(&INONE) = &INONE *** ;
WARNING: Apparent symbolic reference INONE not resolved.
*** The value of macro variable &INONE = &INONE ***

* Use the %GLOBAL statement to create a global macro variable for use outside of a macro. The macro variable &INTWO is globalized before it is defined. Although it is defined inside of a macro, it is now available outside the macro as well.

** The macro variable &INTWO is globalized before it is defined. ;
* Although it is defined inside of a macro, it is now available outside the macro. ;
%macro two;
  %global intwo;
  %let intwo = CCC;
  %put _local_;
  %put _global_;
%mend;

%two
GLOBAL OUTSIDE AAA
GLOBAL INTWO CCC

%put *** The value of macro variable %NRSTR(&INTWO) = &INTWO *** ;
*** The value of macro variable &INTWO = CCC ***

* The example below (Fig. 4) shows that when %last is called, the macro variable &INONE will not resolve because it is local to macro %one. When %one finished execution, the local symbol table that contains &INONE is deleted, which happens before the %put, making &INONE not available.
The example in Fig. 5 demonstrates the use of the CALL SYMPUT routine in a DATA step. Note that the macro variable &FIRSTN is not created until the DATA step is executed, at which time it will be written to the global symbol table for use in a subsequent step. As a result, the intended value for the variable NICKNAME will not be created and the log will contain a warning message.

```sas
data new;
  fname = 'Dan';
  call symput('firstn',fname);
  nickname = "&firstn";
  WARNING: Apparent symbolic reference FIRSTN not resolved.
run;

proc print data=new;
run;

Obs  fname   nickname
1    Dan     &firstn

proc print data=sashelp.vmacro;
  where scope = 'GLOBAL';
run;

Obs  scope   name     offset  value
1    GLOBAL  FIRSTN   0       Dan

data _null_;  
  set new;
  nickname = "&firstn";
  put _all_;  
run;

fname=Dan nickname=Dan _ERROR_=0 _N_=1
```
#4: **Changing Default Macro Variable Values**

A macro definition can include keyword parameters. Like positional parameters, keyword parameters create macro variables. However, when you use keyword parameters to create macro variables, you list both the name and the value of each macro variable in the macro definition.

Keyword parameters can be listed in any order. Whatever value you assign to each parameter (or variable) in the %MACRO definition becomes its default value. Null values are allowed.

When you call a macro whose definition includes keyword parameters, you specify both the keyword and the value for each parameter, in any order. If you omit a keyword parameter from the macro call, the keyword variable retains its default value.

Also, note the use of the macro options MPRINT, MLOGIC, and SYMBOLGEN in this example.

```sas
options mprint mlogic symbolgen; run;

data ds1;
  begin='c';
  location='X';
run;
%
macro attend(opts=,start=a,stop=z);
  proc freq data=ds1;
    where "&start" <= begin <= "&stop";
    table location / &opts;
    title1 'Letters from &start to &stop';
    %put _local_;
    run;
%mend;
%
attend()
%MLOGIC(ATTEND):  Beginning execution.
%MLOGIC(ATTEND):  Parameter OPTS has value
%MLOGIC(ATTEND):  Parameter START has value a
%MLOGIC(ATTEND):  Parameter STOP has value z
%MPRINT(ATTEND):   proc freq data=ds1;
%SYMBOLGEN:  Macro variable START resolves to a
%SYMBOLGEN:  Macro variable STOP resolves to z
%MPRINT(ATTEND):   where "a" <= begin <= "z";
%SYMBOLGEN:  Macro variable OPTS resolves to
%MPRINT(ATTEND):   table location / ;
%SYMBOLGEN:  Macro variable START resolves to a
%SYMBOLGEN:  Macro variable STOP resolves to z
%MPRINT(ATTEND):   title1 'Letters from a to z';
%MLOGIC(ATTEND):  %PUT _local_
ATTEND START a
ATTEND OPTS
ATTEND STOP z
%MPRINT(ATTEND):   run;

NOTE: There were 1 observations read from the data set WORK.DS1.
  WHERE (begin>='a' and begin<='z');
NOTE: PROCEDURE FREQ used (Total process time):
  real time 0.01 seconds
  cpu time 0.00 seconds
%MLOGIC(ATTEND):  Ending execution.
```

---

**Fig. 6a**
* Changes the default macro values to d and y;  
* which does NOT include the value of 'c'.  
%attend(opts=,start=d,stop=y)

MLOGIC(ATTEND):  Beginning execution.
MLOGIC(ATTEND):  Parameter OPTS has value
MLOGIC(ATTEND):  Parameter START has value d
MLOGIC(ATTEND):  Parameter STOP has value y

MPRINT(ATTEND):   proc freq data=ds1;
SYMBOLGEN:  Macro variable START resolves to d
SYMBOLGEN:  Macro variable STOP resolves to y
MPRINT(ATTEND):   where "d" <= begin <= 'y';
SYMBOLGEN:  Macro variable OPTS resolves to
MPRINT(ATTEND):   table location / ;
SYMBOLGEN:  Macro variable START resolves to d
SYMBOLGEN:  Macro variable STOP resolves to y
MPRINT(ATTEND):   title1 "Letters from d to y";
MLOGIC(ATTEND):  %PUT _local_
ATTEND START d
ATTEND OPTS
ATTEND STOP y
MPRINT(ATTEND):   run;

NOTE: No observations were selected from data set WORK.DS1.
NOTE: There were 0 observations read from the data set WORK.DS1.
   WHERE (begin>='d' and begin<='y');

MLOGIC(ATTEND):  Ending execution.

* Resets the macro values to their default values (a and z).  ;
%attend()

MLOGIC(ATTEND):  Beginning execution.
MLOGIC(ATTEND):  Parameter OPTS has value
MLOGIC(ATTEND):  Parameter START has value a
MLOGIC(ATTEND):  Parameter STOP has value z

MPRINT(ATTEND):   proc freq data=ds1;
SYMBOLGEN:  Macro variable START resolves to a
SYMBOLGEN:  Macro variable STOP resolves to z
MPRINT(ATTEND):   where "a" <= begin <= 'z';
SYMBOLGEN:  Macro variable OPTS resolves to
MPRINT(ATTEND):   table location / ;
SYMBOLGEN:  Macro variable START resolves to a
SYMBOLGEN:  Macro variable STOP resolves to z
MPRINT(ATTEND):   title1 "Letters from a to z";
MLOGIC(ATTEND):  %PUT _local_
ATTEND START a
ATTEND OPTS
ATTEND STOP z
MPRINT(ATTEND):   run;

NOTE: There were 1 observations read from the data set WORK.DS1.
   WHERE (begin>='a' and begin<='z');
NOTE: PROCEDURE FREQ used (Total process time):
   real time 0.01 seconds
   cpu time 0.00 seconds

MLOGIC(ATTEND):  Ending execution.
#5: Forward Re-Scan Rule

The Forward Re-Scan rule can be summarized as follows:

- When multiple ampersands or percent signs precede a name token, the macro processor resolves two ampersands (&&) to one ampersand (&), and re-scans the reference.
- To re-scan a reference, the macro processor scans and resolves tokens from left to right from the point where multiple ampersands or percent signs are codes, until no more triggers can be resolved.
- You need to use three ampersands in front of a macro variable name when its value matches the name of a second macro variable, i.e., when you want to store the name of a macro variable in another macro variable. This indirect reference resolves to the value of the second macro variable.

The macro %SUMS is used to dynamically generate a single assignment statement. The following call to %SUMS:

```
%let total=4;
%let numtreat=3;
%sums(pref, total, numtreat)
```

will generate the following assignment statement:

```
pref4=sum(of pref1-pref3);
```

Note that when %SUMS is called, the parameters TOTAL and NUMTREAT are actually the names of the macro variables that contain the number of treatment groups.

The macro %SUMS expects to be passed the root portion of the name of the variable to be created (&PREFIX), the name of the macro variable that contains the number of the new variable (&SUFFIX), and the name of the macro variable that contains the number of variables in the list (&STOPMUM). It's the need to pass macro variable names that requires the use of triple ampersands – see Carpenter pg. 351.

```
%macro sums(prefix, suffix, stopnum);
 &prefix&&suffix=sum(of &prefix.1-&prefix&&stopnum);
 %put _local_;
 %mend sums;
```

The macro is called from within a DATA step.

```
data ds1;
   input @1 pref1 1.
      @3 pref2 1.
      @5 pref3 1.;
data lines;
1 2 3
run;

data count;
   set ds1;
   %sums(pref, total, numtreat)
   put _all_;
run;

cproc print data=sashelp.vmacro;
   where scope ne 'AUTOMATIC';
run;
```
For the macro call %sums(pref, total, numtreat), the triple ampersands are resolved as shown here:

<table>
<thead>
<tr>
<th>Start with =&gt;</th>
<th>&amp;&amp;prefix&amp;&amp;&amp;suffix=sum(of &amp;prefix.1 - &amp;&amp;prefix&amp;&amp;&amp;stopnum);</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st scan</td>
<td>&amp;&amp;prefix &amp;&amp;&amp;suffix =sum(of &amp;prefix.1 - &amp;&amp;prefix&amp;&amp;&amp;stopnum);</td>
</tr>
<tr>
<td>Resolves to</td>
<td>&amp; prefix &amp; total =sum(of pref1 - &amp; prefix &amp; numtreat)</td>
</tr>
<tr>
<td>2nd scan</td>
<td>&amp;prefix &amp;total n/a &amp;prefix &amp;numtreat</td>
</tr>
<tr>
<td>Resolves to</td>
<td>pref 4 pref 3</td>
</tr>
<tr>
<td>Final result</td>
<td>pref4 =sum(of pref1 - pref3);</td>
</tr>
</tbody>
</table>

REFERENCES

ACKNOWLEDGMENTS
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 registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
For additional information please contact:

Dan Bretheim
Towers Watson
901 North Glebe Road
Arlington, VA 22203
daniel.bretheim@towerswatson.com