Paper 4160-2020

Think Globally, Act Locally: Understanding the Global and Local Macro Symbol Tables

Michelle Buchecker, Independent Consultant

ABSTRACT

One of the most missed questions on the Advanced SAS® Certification exam during Beta testing dealt with global and local macro variables. Even people who got nearly every other question right missed these questions. In this paper, you learn when a local symbol table gets created, when a macro variable gets placed in the local table, what happens if you have a macro variable by the same name, and why does any of this matter. %LET, CALL SYMPUT, CALL SYMPUTX, and the INTO clause in SQL each have their own rules for addressing how they interact with the local and global tables. Techniques for debugging and displaying symbol tables are also discussed. Learn them here, live them when you leave.

INTRODUCTION

Macro variables allow you to do text, or more accurately, code substitution. You can take control of where macro variables are stored. If you don't, SAS takes control for you. Just like anything in your life where something else takes control, you don't always end up with the results you want.

In this paper you will learn what the default behavior is for macro variables in different situations. Knowing the defaults if the first step to understanding what is going on and if you want to change it.

BRIEF OVERVIEW OF SYMBOL TABLES

SAS variables are stored in SAS data sets. Macro variables are stored in an area of memory called the **symbol table**. This allows you to dynamically do code substitution with macro variables.

There are two types of symbol tables, Global and Local. There is only 1 Global symbol table, but there may be multiple local symbol tables.

A common question is "Can you have a macro variable by the same name in more than one table?" and the answer is Yes. Which is often followed by the question "How does SAS know which one to choose?" That answer is slightly more complicated and will be discussed throughout the paper.

GLOBAL SYMBOL TABLE OVERVIEW

WHEN THE GLOBAL SYMBOL TABLE GETS CREATED

The Global symbol table gets created in memory when your SAS session starts. In other words when you open SAS or a batch job begins running. At this point, SAS creates many system macro variables and puts them in the Global symbol table.

WHEN THE GLOBAL SYMBOL TABLE GETS DELETED

The Global symbol table gets deleted from memory when your SAS session ends. In other words when you close SAS or a batch job finishes.

Which means all macro variables you place in the Global symbol table are there for the duration of your SAS session. There is a way you can manually delete a macro variable from the table but that is outside the scope of this paper.

LOCAL SYMBOL TABLE OVERVIEW

WHEN THE LOCAL SYMBOL TABLE GETS CREATED

The local symbol table gets created if the macro has parameters. If the macro doesn't have parameters, it may or may not get created. The details below will discuss when it would get created.

WHEN THE LOCAL SYMBOL TABLE GETS DELETED

The local symbol table (if it exists at all) is removed from memory when **the macro finishes execution**. So macro variables placed in the local symbol table are only accessible while that macro is executing.

SAS DEFAULTS

Any time you create a macro variable and you are **outside of a macro definition**, that macro variable will **ALWAYS** be placed in the Global symbol table. Local symbol tables do not exist outside of a macro definition. So that makes sense then, right? Only the Global symbol table exists outside a macro definition, so that is the only place a new macro variable can be placed.

For example:

```
%let dsn=sashelp.class;

data _null_;
   call symput('vars', 'height weight');
run;
```

Since that code is outside of a macro definition, the macro variable DSN will be placed in the Global symbol table with a value of **sashelp.class**, and the macro variable VARS will be placed in the Global symbol table with a value of: **height weight**.

SAS Macro Variable Name	Value
dsn	sashelp.class
vars	height weight

Table 1. Global Symbol Table

Now let's talk about if you are **inside a macro definition**, what does SAS do by default?

Parameters

If your macro has parameters, they are **always** placed in the local symbol table for that macro.

For example:

```
%macro reports(dsn, vars);
   proc means data=&dsn;
     var &vars;
   run;
%mend;
```

%reports(sashelp.heart, weight)

The macro variables DSN and VARS will be placed in REPORTS local symbol table.

SAS Macro Variable Name	Value
dsn	sashelp.heart
vars	weight

Table 2. Reports Local Symbol Table

Does the Macro Variable already exist?

For other parts of SAS, the rule is update what you can first, create as a last resort. Since you could have a macro variable by the same names in **both** the Global symbol table and a local symbol table. SAS will first look to the **local** symbol table for that macro variable. If it isn't found it moves to the "next closest table". This could be a local symbol table from which that macro was called. In other words, if you have macro A that makes a call to Macro B, SAS will first check the local symbol table for B, and if not found will check the local symbol table for A. If it's not found in any local symbol table, SAS will check for a macro variable by that name in the Global symbol table.

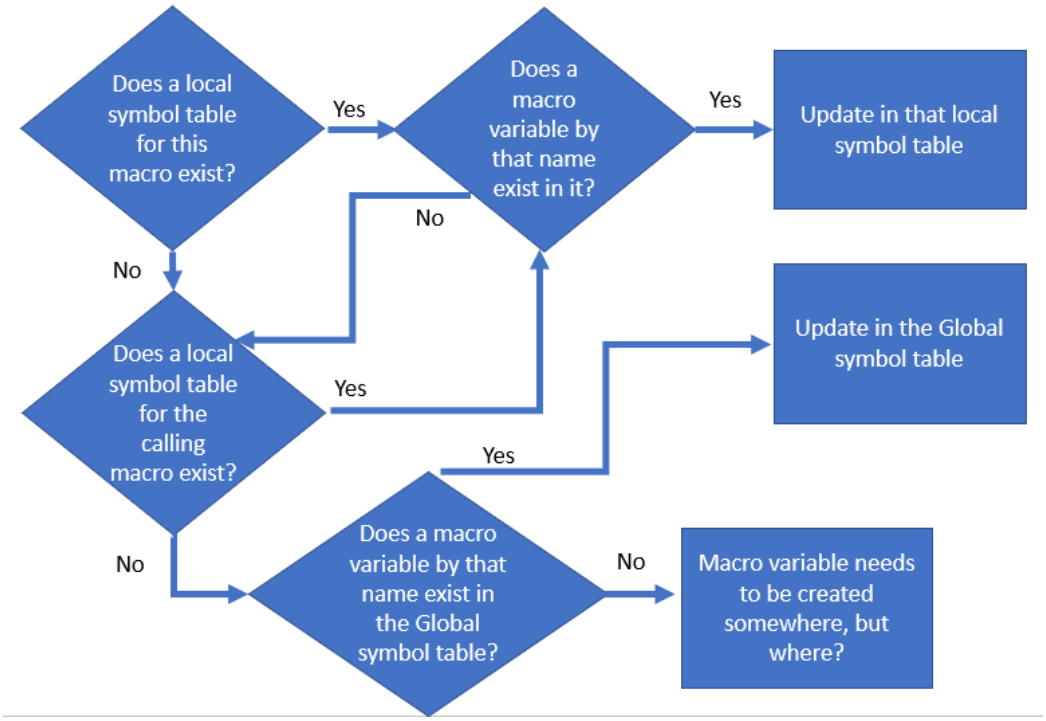

Figure 1. Attempting to Find an Existing Macro Variable to Update

Existing Macro Variable Not Found, Now What?

If a macro variable is not found in any symbol table to update, it must be created. Where it gets created depends on what statement you used to create that macro variable.

%LET Statement and the INTO Clause in PROC SQL

For both a %LET statement and the INTO clause in PROC SQL inside of a macro definition, if a new macro variable has to be created, it will be created in the **local** symbol table, even if a local symbol table doesn't exist. %LET and the INTO clause have the power to create a local symbol table.

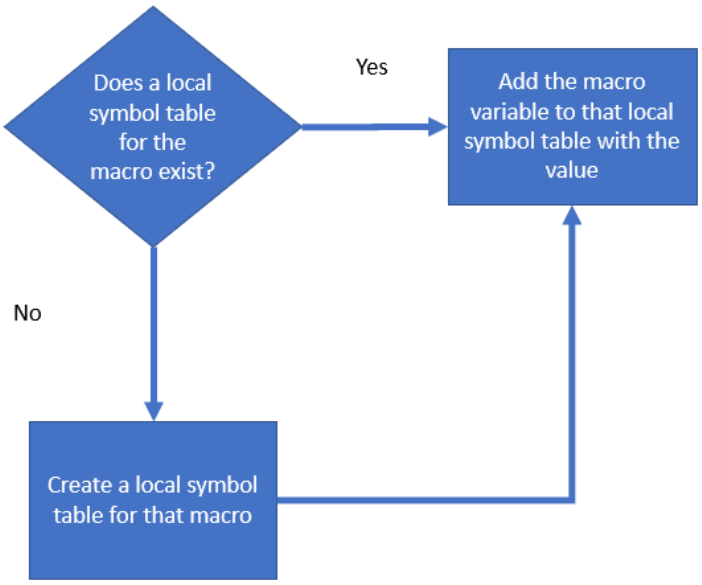

Figure 2. Creation with %LET or the INTO Clause

Example of INTO clause to create a macro variable inside a macro definition:

```
%macro test; /* no parameters, no local table to start */
   proc sql noprint;
   select avg(weight) into: avgwt
   from sashelp.class;
%mend;
%test
```

SAS Macro Variable Name	Value
avgwt	100.0263

Table 3. TEST Local Symbol Table

CALL SYMPUT

The CALL SYMPUT statement starts the same, update what we can first starting with our own closest table and moving on out. The difference is what happens when there is no local

macro variable by that name to update? CALL SYMPUT does not have the power to create a local symbol table. It is forced to put a new variable into an **existing** symbol table.

So, CALL SYMPUT has to perform an additional check: Does a local symbol table **exist**? Remember from earlier, local symbol tables only exist initially if there are parameters to a macro. So if there are no parameters and CALL SYMPUT is the first line of code in a macro definition, no local symbol table will exist at that point.

If a local symbol table **does not** exist, SAS will check to see if there is a local symbol table from the calling macro. For instance, if Macro A has parameters (therefore having a local symbol table), and then makes a call to Macro B that does not have parameters, and the first line of code in Macro B is a CALL SYMPUT, SAS will first check to see if that macro variable can be found anywhere. If not, SAS checks: Does B have a local symbol table? In our example, no. Then it checks: Does A have a local symbol table? In our example, yes. So the new macro variable will be places in the A's local symbol table.

If there are no local symbol tables, SAS will place the new macro variable in the Global symbol table.

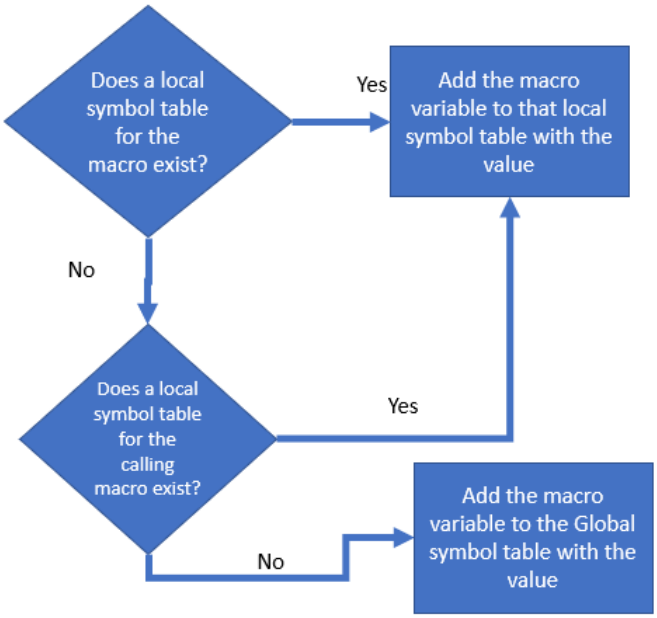

Figure 3. Creation with CALL SYMPUT

OVERRIDING SAS DEFAULTS

Now that you have a basic understanding of what SAS does by default, let's take a look at ways we can override the defaults.

%GLOBAL Statement

A %GLOBAL statement says "put this macro variable into the Global symbol table". If a macro variable by that name already exists in the Global symbol table, the statement is ignored. If a macro variable by that name does not exist in the Global symbol table, it is added there and given a NULL value.

For example:

```
%macro reports(dsn, vars);
   %global dsn;
   proc means data=&dsn;
     var &vars;
   run;
%mend;

%reports(sashelp.heart, weight)
```

In this case a macro variable named **dsn** is in the local symbol table for the reports macro because it is a parameter. Then a macro variable named **dsn** is placed in the Global symbol table because of the %GLOBAL statement. DSN in the local table has a value of **sashelp.heart** and DSN in the Global symbol table has a NULL value.

SAS Macro Variable Name	Value
dsn	

Table 4. Global Symbol Table

SAS Macro Variable Name	Value
dsn	sashelp.heart
vars	weight

Table 5. Reports Local Symbol Table

%LOCAL Statement

A %LOCAL statement says "put this macro variable into the local symbol table for this macro". If a macro variable by that name already exists in that local symbol table, the statement is ignored. If a macro variable by that name does not exist in that local symbol table, it is added there and given a NULL value. NOTE: If a local symbol table does not yet exist for that macro, one is created to put this local macro variable into it.

For example:

```
%let dsn=sashelp.cars; /* outside of a macro definition = Global table */
%macro reports; /* no parameters, no local table to start */
   %local dsn; /* Place DSN in a local table, creating a local table as we
                  do so. DSN will have a null value */

   %let dsn=sashelp.heart; /* Update DSN. Follow rule of update from local
                              table first. */
   proc means data=&dsn;
   run;
%mend;

%reports
```

The first %LET statement is outside of a macro definition, so the macro variable DSN is added to the Global symbol table and given the value **sashelp.cars**. The %local statement will create a local symbol table for the reports macro and add the macro variable DSN to it and give it a NULL value. When that %LET statement is encountered, remember the rules update what you can first, starting with the local table and moving outward. Since a macro variable named DSN exists in the local symbol table for the reports macro due to the %LOCAL statement, SAS will update the NULL value to a value of **sashelp.heart**.

At this point there are now two macro variables named DSN, one in the Global symbol table with a value of **sashelp.cars** and one in reports local symbol table with a value of **sashelp.heart**.

SAS Macro Variable Name	Value
dsn	sashelp.cars

Table 6. Global Symbol Table

SAS Macro Variable Name	Value
dsn	sashelp.heart

Table 7. Reports Local Symbol Table

CALL SYMPUTX

The CALL SYMPUTX statement is similar to CALL SYMPUT with a few differences. One is that it strips off leading and trailing blanks from the value before assigning it to the macro variable. Secondly, CALL SYMPUTX supports an optional third argument that can help control in which symbol table a macro variable is created.

For example:

```
%macro create; /* no parameters, no local table to start */
   data _null_;
     set sashelp.class;

     /* Create macro variables and put them in the default table.
        Since a local table does not exist, that is the Global table */
     call symputx ('dsn', 'sashelp.class');
     call symputx ('name' !! left(_n_), name);
   run;
%mend;

%create
```

With no third argument and no parameters, all macro variables are placed in the Global symbol table because no local table exists.

SAS Macro Variable Name	Value
dsn	sashelp.class
name1	Alfred
name2	Alice
name3	Barbara
...	...
name19	William

Table 8. Global Symbol Table

If you are interested in what that 2nd CALL SYMPUT does, it creates a series of macro variables, name1, name2, etc. The !! means concatenate, and using the automatic variable _n_ to add consecutive numbers. The left function removes leading blanks when SAS does an automatic numeric to character conversion

By adding a third argument of 'L' that says place this macro variable in the local symbol table. If a local symbol table does not exist, one is created.

Using L as the third argument:

```
%macro create; /* no parameters, no local table to start */
   data _null_;
     set sashelp.class;

     /* Create a macro variable named dsn and put it in the Local Symbol
        table */
     call symputx ('dsn', 'sashelp.class', 'L');

    /* Create a series of macro variables and put them in the default
       table. Since there is no parameter, but the previous statement did
       create a local table, that will be the local symbol table. */
     call symputx ('name' !! left(_n_), name);
   run;
%mend;

%create
```

In the above example, the macro variable DSN will be placed in the local table for the create macro. Because that first CALL SYMPUTX statement ends up creating a local symbol table, the macro variables from the second CALL SYMPUTX statement follow the rules from above: update if macro variables by that name exist, if they don't create in the local table if one exists, otherwise create in the next closest table until you reach the Global symbol table. So, assuming macro variables name NAME1, NAME2, etc. did not exist in the Global symbol table, these macro variables will be placed in the **create** local symbol table since one now exists.

SAS Macro Variable Name	Value
dsn	sashelp.class
name1	Alfred
name2	Alice
name3	Barbara
...	...
name19	William

Table 9. CREATE Local Symbol Table

Using G as the third argument:

```
%macro create(vars); /* parameter, local table to start */
   data _null_;
     set sashelp.class (keep=&vars);
     /* Create a macro variable named dsn and put it in the Global Symbol
        table */
     call symputx ('dsn', 'sashelp.class', 'G');

    /* Create a series of macro variables and put them in the default
       table. Since there is a parameter, that will be the local symbol
       table. */
     call symputx ('name' !! left(_n_), name);
```

```
   run;
%mend;

%create(name weight)
```

If you use a G as the third argument to CALL SYMPUTX that says put that macro variable in the Global symbol table and give it this value. This holds true **even if a macro variable by that name exists in the local symbol table**. This is a way to **override** the rule of update first, create as a last resort. This will **always** create in the Global symbol table.

SAS Macro Variable Name	Value
vars	name weight
name1	Alfred
name2	Alice
name3	Barbara
...	...
name19	William

Table 10. CREATE Local Symbol Table

SAS Macro Variable Name	Value
dsn	sashelp.class

Table 11. Global Symbol Table

DISPLAYING SYMBOL TABLES

Even with knowing the above, sometimes you just want to confirm which macro variable is in what symbol table. There are a couple of techniques you can use to do this which is useful for debugging purposes.

USING %PUT

The %PUT statement is a macro statement that can be placed anywhere in your code and it executes immediately. There is a handy keyword you can use on the %PUT statement of _user_.

%PUT _user_; writes to the log all of the user defined macro variables, their value, and which symbol table it is in. This is a great way to debug your SAS programs if your macro variables do not resolve to what you expect.

For example, adding %PUT _USER_ after the RUN statement:

```
%macro create(vars); /* parameter, local table to start */
   data _null_;
     set sashelp.class (keep=&vars);
     call symputx ('dsn', 'sashelp.class', 'G');
     call symputx ('name' !! left(_n_), name);
   run;
   %put _user_;
%mend;
```

```
%create(name weight)
```

Results in this partial log:

```
CREATE NAME1 Alfred
CREATE NAME10 John
CREATE NAME11 Joyce
CREATE NAME12 Judy
CREATE NAME13 Louise
CREATE NAME14 Mary
CREATE NAME15 Philip
CREATE NAME16 Robert
CREATE NAME17 Ronald
CREATE NAME18 Thomas
CREATE NAME19 William
CREATE NAME2 Alice
CREATE NAME3 Barbara
CREATE NAME4 Carol
CREATE NAME5 Henry
CREATE NAME6 James
CREATE NAME7 Jane
CREATE NAME8 Janet
CREATE NAME9 Jeffrey
CREATE VARS name weight
GLOBAL CLIENTMACHINE 10.0.2.2
GLOBAL DSN sashelp.class
```

Output 1. Output from a %PUT _USER_ Statement

Notice that there is an extra macro variable named CLIENTMACHINE. SAS will create macro variables during your SAS session that will show as User created macro variables. I just ignore those.

USING SASHELP.VMACRO

SASHELP.VMACRO is a SAS data set that contains data about your macro variables: name, value, and symbol table (aka SCOPE). You can simply open up this data set or submit PROC PRINT DATA=SASHELP.VMACRO**.**

Total rows: 107 Total columns: 4

	scope	name	value
101	GLOBAL	DSN	sashelp.class
102	GLOBAL	_EXECENV	SASStudio
103	GLOBAL	_CLIENTAPPABREV	Studio

Display 1. Partial results of opening the SASHELP.VMACRO table

CONCLUSION

The macro facility is a great way to write dynamic, maintenance free programs. However, deep knowledge of where macro variables get placed in which circumstances is key to writing good macro code.

ACKNOWLEDGMENTS

Thanks to Bret Smith at SAS Institute for proofreading and making valuable suggestions for this paper. Thanks to Marje Fecht of Prowerk Consulting for teaching me the ins and outs of macro programming.

CONTACT INFORMATION

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

Michelle Buchecker
https://www.linkedin.com/in/michellebuchecker/