SAS1575-2015

New Macro Features Added in SAS® 9.3 and SAS® 9.4

Richard D. Langston, SAS Institute Inc.

ABSTRACT

This paper describes the new features added to the macro facility in SAS® 9.3 and SAS® 9.4. New features described include the /READONLY option for macro variables, the %SYSMACEXIST macro function, the %PUT &= feature, and new automatic macro variables such as &SYSTIMEZONEOFFSET.

INTRODUCTION

There are a number of new features that have been added to the macro facility during 9.3 and 9.4. You might not be aware of these features and you might find them useful. There are several new macro functions, several new automatic macro variables, and enhancements to the %PUT, %GLOBAL, and %LOCAL statements, all described herein.

THE &= SHORTCUT IN THE %PUT STATEMENT

You might be familiar with the named output feature of the PUT statement in the DATA step. For example:

```
data _null_; x=1; put x=; run;
```

produces

```
x=1
```

The %PUT statement now has a similar feature where &= is used before the macro variable. For example:

```
%let x=1;
%put &=x;
```

produces

```
X=1
```

Note that the macro variable name is normalized to uppercase when displayed.

READONLY ATTRIBUTE FOR GLOBAL AND LOCAL VARIABLES

Normally, any macro variable can be changed at any level of scope, and it can also be deleted. You might have macros that you are providing to users where you do not want macro variables to be changed or deleted. The new /READONLY option on the %GLOBAL and %LOCAL statements enable you to prevent changing values or deleting macro variables. For example:

```
%global/readonly stay_put=1;
%global          can_go;
%let can_go=2;

%symdel stay_put;
```

```
%symdel can_go;

%macro tryit;
%put &=mylocal;
%let mylocal=2;
%mend;

%macro localtest;
%local/readonly mylocal=1;
%tryit;
%mend localtest;

options mprint;
%localtest;
```

produces

```
ERROR: The variable STAY_PUT was declared READONLY and cannot be deleted.
MYLOCAL=1
ERROR: The variable MYLOCAL was declared READONLY and cannot be modified or
re-declared.
```

In this example, the macro variable STAY_PUT is initially set to 1 in a %GLOBAL statement with the /READONLY option. STAY_PUT cannot be subsequently deleted or changed. Trying to delete it with %SYMDEL produces an error message. The LOCALTEST macro has a local macro variable called MYLOCAL, also set with the /READONLY option. The TRYIT macro attempts to reassign MYLOCAL, but it fails with the error message shown.

Note also that LOCAL macro variables that are created with the READONLY attribute are deleted when the local scope is terminated. However, GLOBAL macro variables remain present for the duration of the SAS job.

If you want to see which variables are READONLY, use the following %PUT statement:

```
%put _readonly_;
```

%SYSMEXECDEPTH AND %SYSMEXECNAME FUNCTIONS

The %SYSMEXECDEPTH function tells you how many levels of execution are active. The %SYSMEXECNAME tells you the currently executing macro. This is best understood by a simple example:

```
%macro level1;
%put this is level1;
%put %nrstr(%sysmexecdepth)=%sysmexecdepth;
%put should be level1: %sysmexecname(1);
%put should be level1: %sysmexecname(%sysmexecdepth);
%level2;
%mend level1;

%macro level2;
%put this is level2;
%put %nrstr(%sysmexecdepth)=%sysmexecdepth;
```

```
%put should be level2: %sysmexecname(2);
%put should be level1: %sysmexecname(1);
%put should be level2: %sysmexecname(%sysmexecdepth);
%mend level2;

%level1
```

The macros Level1 and Level2 are defined. Level1 displays the current execution depth, then Level2 is invoked. Within Level2, you see that the execution depth is shown as well. The results in the log are shown as follows:

```
this is level1
%sysmexecdepth=1
should be level1:LEVEL1
should be level1:LEVEL1
this is level2
%sysmexecdepth=2
should be level2:LEVEL2
should be level1:LEVEL1
should be level2:LEVEL2
```

The %SYSMEXECNAME function tells you which macro is invoking at each level. When Level1 is first invoked, the execution depth is 1, which is displayed by %SYSMEXECDEPTH. Level1 is the macro that is invoked at the first level of depth. Once Level2 is invoked, you are at 2 depth levels, with Level2 being at depth level 2 and Level1 still being at depth level 1.

%SYSMACEXIST FUNCTION

The %SYSMACEXIST function is used to determine if a compiled macro exists in work.sasmacr. For example:

```
%put %nrstr(%sysmacexist(abc))=%sysmacexist(abc);
%macro abc; %put this is abc; %mend;
%put %nrstr(%sysmacexist(abc))=%sysmacexist(abc);
%abc;
%put %nrstr(%sysmacexist(abc))=%sysmacexist(abc);
```

produces the following:

```
%sysmacexist(abc)=0
%sysmacexist(abc)=1
%sysmacexist(abc)=1
```

The first %SYSMACEXIST indicates that the ABC macro does not exist. The second invocation indicates that the macro exists, because it has now been compiled via the %MACRO statement. The third invocation also indicates that the macro exists. iIt does not matter if the macro has not yet been executed in order for %SYSMACEXIST to determine if it exists.

Note that %SYSMACEXIST only checks for the existence of macros in work.sasmacr. If you have a macro catalog that is associated with the SASMSTORE= option or residing in SASHELP, those macro catalogs are not searched. Also, autocall macros are not considered to exist until they are referenced.

Once referenced, they are compiled behind the scenes and are subsequently found by %SYSMACEXIST. For example:

```
%put %nrstr(%sysmacexist(trim))=%sysmacexist(trim);
%let x=%trim(abc);
%put %nrstr(%sysmacexist(trim))=%sysmacexist(trim);
```

produces the following:

```
%sysmacexist(trim)=0
%sysmacexist(trim)=1
```

This is because TRIM is a standard autocall macro. It is not compiled when first queried by %SYSMACEXIST. But, once it is referenced in the %LET statement, it is found, read, and compiled. So then %SYSMACEXIST finds it when subsequently queried.

Note that if you need the functionality to check for SASMSTORE, SASHELP, and autocall macros, you can use the special macros that are described in my paper, “A Macro to Verify a Macro Exists,” from the *Proceedings of the SAS Global Forum 2013 Conference*.

%SYSMACDELETE STATEMENT

You can delete a macro from work.sasmacro with the %SYSMACDELETE statement. For example:

```
%macro temp; %put this is temp; %mend;
%temp
…
%sysmacdelete temp;
```

Of course, you can replace a macro with a new definition of the same name and the previous macro definition will be deleted.

Do not confuse %SYSMACDELETE with %SYMDEL, which is used to delete macro variables.

Note also that %SYSMACDELETE only deletes macros from the work.sasmacr catalog.

%SYSMSTORECLEAR STATEMENT

To understand the use of the %SYSMSTORECLEAR statement, you need also to understand how the SASMSTORE= option works. If you have this code:

```
libname mylib 'somelib';
options sasmstore=mylib;
/* abc is not in WORK.SASMACR, but is in SOMELIB.SASMACR */
%abc
```

The SOMELIB.SASMACR catalog is not opened when the SASMSTORE option is given, but instead only once the %ABC macro is referenced (because it is not in WORK.SASMACR).

Likewise, if you have this code appearing later:

```
options sasmstore=mylib2;
libname mylib 'somelib2';
```

You will find that even though you have revised the SASMSTORE option, the MYLIB libref cannot be reassigned because it is still in use by the macro facility. This is because the SASMACR catalog remains open until the next apparent macro reference not found in WORK.SASMACR is attempted to be resolved.

At which time, the change in the SASMSTORE option is processed and the previously opened SASMACR catalog in MYLIB is closed and the SASMACR catalog in MYLIB2 is opened.

In order to get the macro facility to close the SASMSTORE catalog without a macro reference, use the new %SYSMSTORECLEAR statement, as in:

```
%sysmstoreclear;
options sasmstore=mylib2;
libname mylib somelib2;
```

AUTOMATIC MACRO VARIABLES

A number of automatic macro variables have been added between SAS 9.2 and 9.4. Here is the list of these variables and their values. Note that they can be referenced at any point in the SAS program, but they might have 0-length values until they are initialized.

SYSENCODING	Same as ENCODING= option (lowercased)
SYSERRORTEXT	The text of the most recent error message
SYSHOSTNAME	Host name running single TCP/IP stack
SYSLOGAPPLNAME	Same as LOGAPPLNAME= option
SYSODSPATH	Current ODS pathname
SYSTCPIPHOSTNAME	Host name running multiple TCP/IP stack
SYSWARNINGTEXT	The text of the most recent warning message
SYSDATASTEPPHASE	Current DATA step processing phase
SYSHOSTINFOLONG	More verbose host information
SYSODSGRAPHICS	1 if ODS graphics is in effect
SYSPRINTTOLOG	The LOG= value of the active PROC PRINTTO
SYSPRINTTOLIST	The PRINT= value of the active PROC PRINTTO
SYSTIMEZONE	Same as TIMEZONE= option
SYSTIMEZONEIDENT	Timezone identifier
SYSTIMEZONEOFFSET	Offset in seconds from GMT
SYSADDRBITS	Number of bits (32 or 64) of the SAS implementation
SYSNOBS	Number of observations in the most recently created data set of a procedure or DATA step
SYSODSESCAPECHAR	Current ODS escape character
SYSSIZEOFPTR	Size of a pointer in SAS (8 for 64 bit or 4 for 32 bit)
SYSPROCESSMODE	Current SAS session run mode

The &SYSPRINTTOLOG and &SYSPRINTTOLIST options are especially useful if you are providing macros that need to run PROC PRINTTO. You can verify if PROC PRINTTO is already in effect, and whether it can handle that situation properly.

&SYSSIZEOFPTR is useful if you are running on Windows but cannot remember if you are using the 32-bit or 64-bit version of SAS. (If you are calling SAS Technical Support, this information will be requested).

ABOUT NEW FUNCTIONS AND STATEMENTS

All new statements and functions have the %SYS prefix and all new automatic macro variables have the &SYS prefix. Do not use the SYS prefix for naming your own macros or macro variables. The macro facility does not prevent you from creating them, but you should avoid doing this in order to prevent problems with future releases of the macro facility. For example, if you created a macro called

%SYSMACEXIST, it would not be usable in SAS 9.4 because the new internal function would be used instead.

REFERENCES

Langston, Rick. 2013. “A Macro to Verify a Macro Exists.” *Proceedings of the SAS® Global Forum 2013 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings13/339-2013.pdf.

CONTACT INFORMATION

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

Rick Langston
SAS Institute Inc.
Rick.Langston@sas.com