Give Your Macro Code an Extreme Makeover:
Tips for even the most seasoned macro programmer

By: Russ Tyndall

Many times when writing code we revert back to our old and comfortable ways. A lot of times these ways may be simplified and written more efficiently. In this paper I will demonstrate new techniques of writing programs by implementing new macro features that can replace longer, older programs. By the end of this paper you should be able to write programs a lot faster and with a lot less code than in previous versions.

CALL SYMPUT VS. CALL SYMPUTX

Let's start by looking at common programming problems. In some situations you need to use CALL SYMPUT to create macro variables based on DATA step variables, but you want these macro variables to be global. The problem here is the uncertainty of how many macro variables are created, so you have to preprocess that data so that a %GLOBAL statement can be built, holding each of these variable names. Let's take a look at the old way of achieving this task versus the new and improved way.

The following code achieves the result described in the paragraph above:

```
data dusty;
  input dept $ name $ salary @@;
  cards;
bedding Watlee 18000   bedding Ives 16000
bedding Parker 9000    bedding George 8000
bedding Joiner 8000    carpet Keller 20000
carpet Ray 12000       carpet Jones 9000
gifts Johnston 8000    gifts Matthew 19000
kitchen White 8000     kitchen Banks 14000
kitchen Marks 9000     kitchen Cannon 15000
tv Jones 9000          tv Smith 8000
tv Rogers 15000        tv Morse 16000
;
```

Old Code:

```
%macro drive (class,var);
proc means noprint;
  class &class;
  var &var;
  output out=stats sum=s_sal;
run;

data _null_;
  set stats;
  if _n_=1 then call execute ('%global s_tot;');
  else call execute('%global s'||dept||';');
run;
data _null_;
  set stats;
  if _n_=1 then call symput('s_tot',trim(left(s_sal)));
```

```
  else call symput('s'||dept,trim(left(s_sal)));
run;
%mend drive;
%drive(dept,salary)
%put _user_;
```

Starting in SAS 9.0 you can use a new call routine, **CALL SYMPUTX,** to simplify this process. The four benefits of CALL SYMPUTX are:

```
1.) CALL SYMPUTX does not write a note to the SAS log when the second argument is
     numeric. CALL SYMPUT, however, writes a note to the log stating that numeric
     values were converted to character values.

2.) CALL SYMPUTX uses a field width of up to 32 characters when it converts
     a numeric second argument to a character value. CALL SYMPUT uses a
     field width of up to 12 characters.

3.) CALL SYMPUTX left-justifies both arguments and trims trailing blanks.
     CALL SYMPUT does not left-justify the arguments, and trims trailing
     blanks from the first argument only. Leading blanks in the value of
     name cause an error.

4.) CALL SYMPUTX enables you to specify the symbol table in which to store the macro
     variable, whereas CALL SYMPUT does not.  This is accomplished by an optional
     third argument to the SYMPUTX function.
```

New Code:

This program does the same as the one above except now we are using CALL SYMPUTX and you can see it took less code, as we were able to remove one of the data steps.

```
%macro drive(class,var);
proc means noprint;
  class &class;
  var &var;
  output out=stats sum=s_sal;
run;

data _null_;
  set stats;
  if _n_=1 then call symputx('s_tot',s_sal,'g');
  else call symputx('s'||dept,s_sal,'g');
run;
%mend drive;

%drive(dept,salary)

%put _user_;
```

LOG:

```
1    data dusty;
2      input dept $ name $ salary @@;
3      cards;
```

```
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.DUSTY has 18 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds


13   ;
14
15   %macro drive(class,var);
16   proc means noprint;
17      class &class;
18      var &var;
19      output out=stats sum=s_sal;
20   run;
21
22   data _null_;
23      set stats;
24      if _n_=1 then call symputx('s_tot',s_sal,'g');
25      else call symputx('s'||dept,s_sal,'g');
26   run;
27   %mend drive;
28
29   %drive(dept,salary)

NOTE: There were 18 observations read from the data set WORK.DUSTY.
NOTE: The data set WORK.STATS has 6 observations and 4 variables.
NOTE: PROCEDURE MEANS used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds


NOTE: There were 6 observations read from the data set WORK.STATS.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds


30
31   %put _user_;
GLOBAL S_TOT 221000
GLOBAL SKITCHEN 46000
GLOBAL SCARPET 41000
GLOBAL STV 48000
GLOBAL SGIFTS 27000
GLOBAL SBEDDING 59000
```

%SYSFUNC

One of the most exciting things to happen within the macro facility was the introduction of the function: %SYSFUNC. This opened up a new and exciting way to code within macro. Tasks that took many steps to achieve can now be accomplished with fewer lines of code. %SYSFUNC allows

you to use many SAS functions within the macro facility. These next few examples illustrate how this function can save time in writing macro programs.

Example 1:

Before **%SYSFUNC** a DATA step was required just to place the number of observations in a SAS data set into a macro variable. The example below illustrates the old method:

Old Code:

```
%macro numobs(dsn);
 %global num;
  data _null_;
   if 0 then set &dsn nobs=count;
   call symput('num',left(put(count,8.)));
   stop;
  run;
%mend numobs;

%numobs(dataset_name)
```

This will create a macro variable NUM that holds the number of observations in the data set.

Now with the use of the function %SYSFUNC along with the new ATTRN function the DATA step code can be eliminated from the macro and substituted with just these 3 lines of code:

New Code:

```
%macro numobs(dsn);
  %global num;
  %let dsid=%sysfunc(open(&dsn));
  %let num=%sysfunc(attrn(&dsid,nobs));
  %let rc=%sysfunc(close(&dsid));
%mend numobs;

%numobs(dataset_name)
```

The first %LET statement opens the data set. The second %LET uses the ATTRN function to check the number of observations via the NOBS attribute. The third %LET statement closes the data set.

Example 2:

A dramatic example of the value of %SYSFUNC is the code to check for the existence of a SAS data set. The old way of accomplishing this task is shown below:

Old Code:

```
%macro exist(dsn);
 %global exist;
  %if &dsn ne %then %str(
   data _null_;
     if 0 then set &dsn;
     stop;
```

```
   run;
   );
%if &syserr=0 %then %let exist=1;
%else %let exist=0;
%mend exist;

%exist(dataset_name_)
```

Now with the use of %SYSFUNC, along with the new EXIST function, the code above can be replaced with this single line of code:

New Code:

```
%let exist=%sysfunc(exist(dataset_name));
```

Macro variable EXIST will be set to 1 if the data set exists and 0 if it does not exist.

Example 3:

Another common task is to place each DATA step variable into a separate macro variable. The most common way to accomplish this task requires a PROC CONTENTS and a DATA step:

Old Code:

```
%macro test;
proc contents data=sasuser.class out=new(keep=name);
run;
data _null_;
 set new;
 call symput('name'||trim(left(_n_)),trim(left(name)));
run;
%put _user_;
%mend test;

%test
```

With the use of %SYSFUNC you can eliminate the PROC CONTENTS and the DATA step code and retrieve the same information by using macro logic only.

```
%macro test;
  %let dsid=%sysfunc(open(sasuser.class));
  %let cnt=%sysfunc(attrn(&dsid,nvars));
    %do i = 1 %to &cnt;
      %let name&i=%sysfunc(varname(&dsid,&i));
    %end;
  %let rc=%sysfunc(close(&dsid));
  %put _user_;
%mend test;
%test
```

The first %LET statement opens the data set. The second %LET uses the ATTRN function to check the number of variables via the NVARS attribute. The %DO statement is used to loop through the number of variables and placing each DATA step variable in its own macro variable via the %LET statement. The fourth %LET statement closes the data set.

Example 4:

Suppose you wanted to do something as simple as placing the current date and time within a title.

Old Code:

```
data _null_;
 call symput('date',put(today(),date.));
 call symput('time',put(time(),time.));
run;

proc print;
 title "Today is &date and the time is &time.";
run;
```

Now with %SYSFUNC this task is much easier because the function can be included in the TITLE statement. For example:

New Code:

```
proc print;
 title "Today is %sysfunc(today(),date.) and the time is %sysfunc(time(),time.)";
run;
```

NEW FEATURES PART A

In this next section I want to discuss some new macro features that can simplify the way you may have written code in the past.

Example 1: (%SYMDEL)

Let's say we want to delete all global macro variables. The old way of accomplishing this task would not delete the macro variables, but it would set them to blank values.

Old Code:

```
%let x=1;
%let y=2;
%put _user_;
%macro lst;
 %global list;
 %let list=;
 data _null_;
  set sashelp.vmacro;
   if scope = 'GLOBAL' then do;
    call symput('mac',trim(left(name)));
    call execute('%let &mac=;');
   end;
 run;
%mend;
%lst
%put _user_;
```

LOG:

```
    NOTE: SAS initialization used:
      real time           1.56 seconds
      cpu time            0.28 seconds

1    %let x=1;
2    %let y=2;
3    %put _user_;
GLOBAL X 1
GLOBAL Y 2
4    %macro lst;
5     %global list;
6     %let list=;
7     data _null_;
8      set sashelp.vmacro;
9       if scope = 'GLOBAL' then do;
10        call symput('mac',trim(left(name)));
11        call execute('%let &mac=;');
12       end;
13     run;
14   %mend;
15   %lst

NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.00 seconds

NOTE: There were 49 observations read from the data set SASHELP.VMACRO.

NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were generated.
16
17   %put _user_;
GLOBAL X
GLOBAL Y
GLOBAL LIST
GLOBAL MAC
```

Notice in the results above, the macro variables still exist, but they are now set to null.

Starting with SAS 8.2 you can use the new macro statement **%SYMDEL,** to delete macro variables. The code below modifies the previous example by using %SYMDEL. The CALL EXECUTE allows us to execute the %SYMDEL for every observation for which the IF condition is true.

New Code:

```
%let x=1;
%let y=2;
%put _user_;
%macro delvars;
  data vars;
    set sashelp.vmacro;
  run;
  data _null_;
    set vars;
    if scope='GLOBAL' then
```

```
      call execute('%symdel '||trim(left(name))||';');
  run;
%mend;
%delvars

%put _user_;
```

LOG:

```
NOTE: SAS initialization used:
      real time          0.93 seconds
      cpu time           0.37 seconds

1    %let x=1;
2    %let y=2;
3    %put _user_;
GLOBAL X 1
GLOBAL Y 2
4    %macro delvars;
5      data vars;
6        set sashelp.vmacro;
7      run;


8      data _null_;
9        set vars;
10       if scope='GLOBAL' then
11         call execute('%symdel '||trim(left(name))||';');
12     run;
13   %mend;
14   %delvars

NOTE: There were 47 observations read from the data set SASHELP.VMACRO.
NOTE: The data set WORK.VARS has 47 observations and 4 variables.
NOTE: DATA statement used:
      real time          0.04 seconds
      cpu time           0.01 seconds



NOTE: DATA statement used:
      real time          0.01 seconds
      cpu time           0.01 seconds

NOTE: There were 47 observations read from the data set WORK.VARS.

NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were generated.
15
16   %put _user_;
```

As you can see the macro variables have been completely removed from the symbol table.

There is also a NOWARN option that can be placed on the %SYMDEL statement, for cases where you may try to delete a macro variable that does not exist. For example:

SAMPLE CODE:

```
%symdel x;

%symdel x / nowarn;
```

LOG:

```
WARNING: Attempt to delete macro variable X failed. Variable not found.
 17   %symdel x;
 18
 19   %symdel x / nowarn;
```

Notice that you referenced a macro variable X that does not exist. The first SYMDEL, without the option, gives you a warning. The second statement, with the option NOWARN, gives no warning. There is also a DATA step call routine equivalent to %SYMDEL called CALL SYMDEL.

Example 2: (%SYMEXIST)

The following example illustrates the code needed to check if a macro variable exists without getting a warning in the log.

```
%macro check(mvar);
 %local i tmp;
 %let dsid=%sysfunc(open(sashelp.vmacro));
 %let num=%sysfunc(varnum(&dsid,name));
  %do %until(&ob = -1);
   %let i=%eval(&i+1);
   %let ob=%sysfunc(fetchobs(&dsid,&i));
   %let val=%sysfunc(getvarc(&dsid,&num));

 %if &val = %upcase(&mvar) %then %do;
 %let ob = -1;
 %let tmp=1;
%end;

 %else %do;
  %let tmp=0;
 %end;

 %if &ob=-1 %then %do;
  &tmp
 %end;

%end;
%let rc=%sysfunc(close(&dsid));
%mend check;

/** Check for the existence of the macro variable abc **/
%put Return 1 if macro variable exist:%check(abc);
```

Beginning with SAS 9.1, the **%SYMEXIST** function can be used to determine if a macro variable exists. The function returns a value of 1 if the macro variable exists and a value of 0 if it does not. The macro above can be replaced with this single line of code:

```
%put Return 1 if macro variable exist: %symexist(abc);
```

Note: The & (ampersand) does not precede the macro variable name in the argument to the function. There is also a DATA step function equivalent to %SYMEXIST called SYMEXIST.

Example 3: (%SYMGLOBL)

To take the example above a step further, what if you wanted to check to see if a macro variable was declared as global. The code below would accomplish this task:

```
%let x=1;
%macro check(name);
 %global ans;
 data _null_;
  set sashelp.vmacro;
  if upcase(scope)='GLOBAL' and name="%upcase(&name)" then do;
   call symput('ans',trim(left(1)));
   stop;
  end;
  else call symput('ans',trim(left(0)));
 run;
%mend check;
%check(x);
%put 1 if macro variable is global: &ans;
```

Starting in SAS 9.1 there is a new macro function called **%SYMGLOBL** which returns a 1 if the macro variable is global in scope or 0 otherwise. So the entire program above could be rewritten as:

```
%put 1 if macro variable is global: %symglobl(x);
```

There is also a similar function called **%SYMLOCAL** that checks for a macro variable on the local symbol table.

There are also DATA step functions equivalent to %SYMGLOBL and %SYMLOCAL called SYMGLOBL and SYMLOCAL.

Example 4: (SAS_EXECFILENAME)

Another popular question is: how can I retrieve the name of the program that is currently executing? This is a simple process running in batch. Using %SYSFUNC in conjunction with the GETOPTION function makes it easy. Consider the following:

```
%put The current program is %sysfunc(getoption(sysin));
```

Since SYSIN is used for batch mode source files it requires a little more effort to retrieve this information interactively. The following code can be run interactively to retrieve the path and the name of the current program:

```
%macro pname;
 %global pgmname;
 %let pgmname=;
  data _null_;
   set sashelp.vextfl;
    if (substr(fileref,1,3)='_LN' or
        substr(fileref,1,3)='#LN' or
        substr(fileref,1,3)='SYS') and
     index(upcase(xpath),'.SAS')>0 then do;
     call symput("pgmname",trim(xpath));
     stop;
    end;
  run;
%mend pname;
%pname;
%put pgmname=&pgmname;
```

Beginning in SAS 9.0 the macro above is not needed. There is a new environment variable for the Enhanced Editor called: **SAS_EXECFILENAME.** This does not include the full path, only the filename.

The following will now return the executing program when running interactively:

```
%put %sysget(SAS_EXECFILENAME);
```

In order to get the full path, there is also an environment variable for the Enhanced Editor called **SAS_EXECFILEPATH** that contains the full path of the submitted program or catalog entry. The full path includes the folder and the filename.

NEW FEATURES PART B

This section discusses new macro features which extend the existing capabilities of the SAS System. You may find these features beneficial when programming within macro.

Example 1: (LIBRARY CONCATENATION)

Unlike formats and the FMTSEARCH option, prior to SAS 7 there is no method to concatenate Stored Compiled Libraries.

Starting in SAS 7 you have the ability to implicitly concatenate SAS catalogs. Because the LIBNAME statement allows you to logically concatenate SAS data libraries, SAS catalogs that have the same name are also implicitly concatenated. All Stored Compiled macros are stored in a catalog called SASMACR. This feature enables you to reference multiple SASMACR catalogs in different SAS data libraries that have been stored in the SASMSTORE= option. SAS searches for the SASMACR catalog that is located in each library that is referenced in the LIBNAME statement and specified in the SASMSTORE= option until the entry (invoked macro) is found.

Syntax:

```
LIBNAME libref <engine> (library-specification-1 <...library-specification-n>)<options>;
```

If the library specification is a path, then each path must be enclosed in single quotation marks. Here is an example:

```
  libname allmine ('c:\test1' 'c:\test2' 'c:\test3');
  options sasmstore=allmine mstored;
```

Suppose you were invoking a macro called TEST and was using the LIBNAME and OPTIONS statement above. SAS searches the SASMACR catalog in c:\test1, then c:\test2, etc. and uses the first one that is found.

Example 2: (CAPTURING LOG MESSAGES)

What if you have a situation where you want to grab the text of the last error/warning message from the log and place this in a variable. You have to process the log file, read in the text, locate the last error and store this in a macro variable. In this example, errmess contains the last error message:

```
%let subdir=c:\errors\;
filename dir pipe "dir &subdir.*.log /B";

data new;
  infile dir truncover;
  input filename $80.;
  filename="&subdir" || filename;
  length lname logfile $30;
  infile dummy filevar=filename filename=lname end=done truncover;
  do while (not done);
    input test $5. @;
    if test='ERROR' then do;
      input @6 msg $76.;
      logfile=lname;
    end;
    else input;
    end;
  call symput('errmess',trim(left(msg)));
run;
```

The log was saved to a directory called errors with the file having a .log extension. The first INPUT statement reads a log file from the subdirectory. The second INPUT statement reads a record from that log file searching for the word ERROR. If ERROR is found the third INPUT is used to create the data set variable called msg that contains the error message. This is later placed in a macro variable called errmess via the CALL SYMPUT.

This task is much easier starting in SAS 9.2 with two new automatic macro variables entitled: **SYSERRORTEXT** and **SYSWARNINGTEXT**. SYSERRORTEXT contains the text of the last error message generated within the SAS log. SYSWARNINGTEXT contains the text of the last warning message generated within the SAS log. This one line replaces the code above:

```
  %let errmess=&syserrortext;
```

Here is an example using these two automatic macro variables.

SAMPLE CODE:

```
data NULL;
   set doesnotexist;
run;

%put &syserrortext;
%put &syswarningtext;
```

LOG:

```
1   data NULL;
2   set doesnotexist;

ERROR: File WORK.DOESNOTEXIST.DATA does not exist.

3   run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.NULL may be incomplete. When this step was
         stopped there were 0 observations and 0 variables.
NOTE: DATA statement used (Total process time):
real time       11.16 seconds
cpu time        0.07 seconds

4  %put &syserrortext;

File WORK.DOESNOTEXIST.DATA does not exist.

5  %put &syswarningtext;

The data set WORK.NULL may be incomplete.  When this step was stopped there
  were 0 observations and 0 variables.
```

Example 3: (SECURE)

In previous releases of SAS you had no way to truly hide code within a compiled macro. This is no longer the case starting in SAS 9.2 with the introduction of a new %MACRO statement option called **SECURE.**

This option causes the contents of a macro to be encrypted when stored in a stored compiled macro library. This feature enables you to write secure macros that will protect intellectual property that is contained in the macros. This is done using the Encryption Algorithm Manager. The SECURE option can only be used in conjunction with the STORE option.

A NOSECURE option has been implemented to aid in doing a global edit of a source file or library to turn on security. If you are writing several macros that will need to be encrypted, you can create them using the NOSECURE option. When all macros are completed and ready for production, you can do a global edit and change NOSECURE to SECURE.

The following example shows using the STORE and SECURE options to create a macro that is encrypted.

```
Libname mylib ‘c:\mymac’;
options mstored sasmstore=mylib;
%macro secure/store secure;
    data _null_;
      x=1;
      put "This data step was generated from a secure macro.";
    run;
  %mend secure;
  %secure
  filename maccat catalog 'mylib.sasmacr.secure.macro';
  data _null_;
    infile maccat;
    input;
    list;
  run;
```

After executing the above code this compiled macro is encrypted and its contents cannot be viewed.

Example 4: (SOURCE and %COPY)

In previous releases of SAS, you were unable to retrieve the source code from a compiled macro. Starting in SAS 9.1, **SOURCE,** a new option exists for the %MACRO statement that, when used with the existing STORE option combines and stores the source of the compiled macro. The compiled macro code becomes an entry in a SAS catalog in a permanent SAS data library. The compiled macro and the source code are stored together in the same SASMACR catalog.

The SOURCE option requires that the STORE option and the SAS option MSTORED be set. You can use the SAS option SASMSTORE= to identify a permanent SAS data library. You can store a macro or call a stored compiled macro only when the SAS option MSTORED is in effect.

Note: The source code saved by the SOURCE option begins with the %MACRO keyword and ends with the semi-colon following the %MEND statement.

Now that you have a way to store the source code with the SOURCE option, you also need a way to retrieve this information. The answer is the new **%COPY** statement, which copies specified items from a SAS macro library.

Syntax:
%COPY *Macro-name* </options(s)>

Macro-name
name of the macro that the %COPY statement will use.

option(s)
can be one or more of the following options:

LIBRARY= <libref>
LIB=
specifies the libref of a SAS data library that contains a catalog of stored compiled SAS macros. If no library is specified, the libref specified by the SASMSTORE= option is used. Restriction: This libref cannot be WORK.

OUTFILE=<fileref> | <'external file'>
OUT=
specifies the output destination of the %COPY statement. The value can be a fileref or an external file.

SOURCE
SRC
specifies that the source code of the macro will be copied to the output destination.
If the OUTFILE= option is not specified, the source is written to the SAS log.

Below is an example using the new SOURCE option along with the %COPY statement:

SAMPLE CODE:

```
libname test 'c:\';
options mstored sasmstore=test;

%macro test(arg) / store source des="test of the source option";
 %put arg = &arg;
  data one;
   x=1;
  run;
%mend test;

%copy test / source;
```

LOG:

```
149  libname test 'c:\';
NOTE: Libref TEST was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\
150  options mstored sasmstore=test;
151
152  %macro test(arg) / store source des="test of the source option";
153   %put arg = &arg;
154    data one;
155     x=1;
156    run;
157  %mend test;
158
159  %copy test / source;
%macro test(arg) / store source des="test of the source option";
 %put arg = &arg;
  data one;
   x=1;
  run;
%mend test;
```

Limitations:
The SOURCE option cannot be used on nested macro definitions (macro definitions contained within another macro).

If the SECURE option is used in conjunction with the SOURCE option, the SECURE option takes precedence and the source cannot be viewed due to encryption.

Limitations continued:
Macro catalogs cannot be moved across different releases of SAS or across different platforms.

ADDITIONAL OPTIONS, STATEMENTS, AND AUTOMATIC MACRO VARIABLES

This final section contains other new macro features that you may also find helpful when coding within the macro facility.

New option for %MACRO statement:

MINDELIMITER: Assigns the value to be used as the delimiter for the IN (#) operator within the macro at macro compilation (starting in SAS 9.2)

Syntax:

```
%MACRO <macro-name> ( <arguments > ) / MINDELIMITER='<single character>';
```

The value of the /MINDELIMITER option must be a single character enclosed in single quotation marks. The value specified by the /MINDELIMITER option overrides the value of the MINDELIMITER= global option. The default value is a blank. The /MINDELIMITER= option may appear only once in a %MACRO statement.

SAMPLE CODE:

```
%macro test / mindelimiter=',';
  %if &x in (1,2,3) %then %put this is true;
  %else %put this is false;
%mend test;
%let x=3;
%test
```

LOG:

```
1   %macro test / mindelimiter=',';
2    %if &x in (1,2,3) %then %put this is true;
3    %else %put this is false;
4   %mend test;
5   %let x=3;
6   %test

this is true
```

New Automatic macro variables:

&SYSMACRONAME : Returns the name of the currently executing macro (started in SAS 8.2)

Here is an example of using SYSMACRONAME:

SAMPLE CODE:

```
%macro test2;
 %put &sysmacroname is executing;
 %put this is another test;
%mend test2;
%macro test;
 %put this is a test;
 %test2
%mend test;
%test
```

LOG:

```
103  %macro test2;
104   %put &sysmacroname is executing;
105   %put this is another test;
106  %mend test2;
107  %macro test;
108   %put this is a test;
109   %test2
110  %mend test;
111  %test
this is a test
TEST2 is executing
this is another test
```

&SYSPROCNAME: Contains the name of the procedure (or DATASTEP for DATA steps) currently being processed by the SAS Language Processor (started in SAS 8.2). This must be checked between the procedure statement and the next step boundary.

SAMPLE CODE:

```
proc print data=sashelp.class;
 %put &sysprocname;
run;

data one;
 x=100;
 put "&sysprocname";
run;
```

LOG:

```
143  proc print data=sashelp.class;
144   %put &sysprocname;
PRINT
145  run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

```
146
147  data one;
148   x=100;
149   put "&sysprocname";
150  run;

DATASTEP
NOTE: The data set WORK.ONE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

&SYSNCPU: Contains the current number of processors available to SAS for computations (started in SAS 9.0)

This macro variable was created to help determine whether you are taking advantage of the new parallel-processing abilities in SAS, &SYSNCPU contains the current number of CPUs that SAS can use during the current SAS session.

&SYSNCPU is an automatic macro variable that provides the current value of the CPUCOUNT option. For more information about CPUCOUNT system option, see the SAS Language Reference: Dictionary.

&SYSENCODING: Contains the name of the current session encoding (starting in SAS 9.2)

The following statement displays the encoding for the SAS session.

%put The encoding for this SAS session is: &sysencoding;

Executing this statement writes the following to the SAS log:

The encoding for this SAS session is: wlatin1

Note: The value ‘wlatin1’ may be different on your OS depending on the encoding you are using.

New macro system options for debugging:

MEXECNOTE: Displays macro execution information in the SAS log at macro invocation. The default value is NOMEXECNOTE. (started in SAS 9.0)

SAMPLE CODE:

```
options mexecnote;

%macro test;
 %do i = 1 %to 3;
   %put &i;
 %end;
%mend test;
%test
```

LOG:

```
2    options mexecnote;
3
4    %macro test;
5     %do i = 1 %to 3;
6       %put &i;
7     %end;
8    %mend test;
9    %test
NOTE: The macro TEST is executing from memory.
1
2
3
```

MCOMPILENOTE: Specifies that a NOTE be issued to the SAS log when the compilation of a macro is completed and also writes out the size and number of instructions upon the completion of the compilation of any macro. MCOMPILENOTE is set to none by default. (started in SAS 9.0)

Syntax:

```
MCOMPILENOTE=<NONE | NOAUTOCALL | ALL>

NONE
prevents any NOTE from being written to the log.

NOAUTOCALL
prevents any NOTE from being written to the log for AUTOCALL macros, but does issue a
NOTE to the log upon the completion of the compilation of any other macro.

ALL
Print compilation about all macros that are being compiled
```

MPRINTNEST: Enables the macro nesting information to be displayed in the MPRINT output in the SAS log. Default value is NOMPRINTNEST. (started in SAS 9.0)

This has no effect on the MPRINT output that is sent to an external file. For more information, see the MFILE system option.

The setting of MPRINTNEST does not imply the setting of MPRINT. You must set both MPRINT and MPRINTNEST in order for output (with the nesting information) to be written to the SAS log.

SAMPLE CODE:

```
%macro outer;

data _null_;
    %inner
run;
```

```
%mend outer;
%macro inner;
   put %inrmost;
%mend inner;
%macro inrmost;
   'This is the text of the PUT statement'
%mend inrmost;

options mprint mprintnest;
%outer
```

LOG:

```
MPRINT(OUTER):   data _null_;
MPRINT(OUTER.INNER):   put
MPRINT(OUTER.INNER.INRMOST):   'This is the text of the PUT statement'
MPRINT(OUTER.INNER):  ;
MPRINT(OUTER):   run;
This is the text of the PUT statement
NOTE: DATA statement used (Total process time):
      real time          0.10 seconds
      cpu time           0.06 seconds
```

MLOGICNEST: Enables the macro nesting information to be displayed in the MLOGIC output in the SAS log. Default value is NOMLOGICNEST. (started in SAS 9.0)

The setting of MLOGICNEST does not affect the output of any currently executing macro.

The setting of MLOGICNEST does not imply the setting of MLOGIC. You must set both MLOGIC and MLOGICNEST in order for output (with nesting information) to be written to the SAS log.

SAMPLE CODE:

```
%macro outer;
   %put THIS IS OUTER;
   %inner;
%mend outer;
%macro inner;
   %put THIS IS INNER;
   %inrmost;
%mend inner;
%macro inrmost;
   %put THIS IS INRMOST;
%mend;
options mlogic mlogicnest;
 %outer
```

LOG:

```
MLOGIC(OUTER):  Beginning execution.
MLOGIC(OUTER):  %PUT THIS IS OUTER
THIS IS OUTER
MLOGIC(OUTER.INNER):  Beginning execution.
```

MLOGIC(OUTER.INNER): %PUT THIS IS INNER
THIS IS INNER
MLOGIC(OUTER.INNER.INRMOST): Beginning execution.
MLOGIC(OUTER.INNER.INRMOST): %PUT THIS IS INRMOST
THIS IS INRMOST
MLOGIC(OUTER.INNER.INRMOST): Ending execution.
MLOGIC(OUTER.INNER): Ending execution.
MLOGIC(OUTER): Ending execution.

MAUTOLOCDISPLAY: Specifies that the source location of the autocall macro be displayed in the SAS log when the autocall macro is invoked. Default value is NOMAUTOLOCDISPLAY. (started in SAS 9.0)

SAMPLE CODE:

options mautolocdisplay;

%let x=%trim(abc);

LOG:

```
26   options mautolocdisplay;
27
28   %let x=%trim(abc);
MAUTOLOCDISPLAY(TRIM):  This macro was compiled from the autocall file
                        C:\SASv9\sas\dev\mva-v920\shell\auto\en\trim.sas
```

New macro system options for operational needs:

MEXECSIZE: Specifies the maximum size macro to be executed in memory. Default value is 65536. (starting in SAS 9.2)

Syntax:
MEXECSIZE=n | nK | nM | nG | nT | hexX | MIN | MAX
n specifies the maximum size macro to be executed in memory available in bytes.
nK specifies the maximum size macro to be executed in memory available in kilobytes.
nM specifies the maximum size macro to be executed in memory available in megabytes.
nG specifies the maximum size macro to be executed in memory available in gigabytes.
nT specifies the maximum size macro to be executed in memory available in terabytes.
MIN specifies the minimum size macro to be executed in memory. Minimum value is 0.
MAX specifies the maximum size macro to be executed in memory. Maximum value is 2,147,483,647.
hexX specifies the maximum size macro to be executed in memory by a hexadecimal number followed by an X.

Use the MEXECSIZE option to control the maximum size macro that will be executed in memory as opposed to being executed from a file. The MEXECSIZE option value refers to the compiled size of the macro. Memory is only allocated when the macro is executed. Once the macro completes, the memory is released. If memory is not available to execute the macro, an out of memory message is written to the SAS log. Use the MCOMPILENOTE option to write to the SAS log the size of the compiled macro. The MEMSIZE option has no relation to the MEXECSIZE option.

MINOPERATOR: Recognize special operators (IN, #) when evaluating logical or integer expressions. Default setting is NOMINOPERATOR. (starting in SAS 9.2).

NOMINOPERATOR: Do not recognize special infix operators when evaluating logical or integer expressions. This option is useful for jobs where IN needs to be treated as text rather than a special operator. (starting in SAS 9.2)

MINDELIMITER: Specifies the character to be used as the delimiter for the macro IN operator. Default value is a blank. (starting in SAS 9.2)

Syntax:
MINDELIMITER=<"option">

option:
is a character enclosed in double or single quotation marks. The character will be used as the delimiter for the macro IN operator.

Example:

options mindelimiter=',';

The option value is retained in original case and can have a maximum length of one character. The default value of the MINDELIMITER option is a blank.

You can use the # character instead of IN.

Note: When the IN or # operator is used in a macro, the delimiter that is used at the execution time of the macro is the value of the MINDELIMITER option at the time of the compilation of the macro. The option MINOPERATOR must be on to use the IN operator starting in SAS 9.2. The setting of MINDELIMITER does not imply the setting of MINOPERATOR.

SAMPLE CODE:

```
%put %eval(a in d,e,f,a,b,c);

%put %eval(a in d e f a b c);

option mindelimiter=',';
%put %eval(a in d,e,f,a,b,c);

%put %eval(a in d e f a b c);
```

Notice in the log below how setting the value of MINDELIMITER affects the outcome.

LOG:

```
NOTE: SAS initialization used:
      real time           1.02 seconds
      cpu time            0.63 seconds

  %put %eval(a in d,e,f,a,b,c);

0
```

```
 %put %eval(a in d e f a b c);

1

  option mindelimiter=',';
  %put %eval(a in d,e,f,a,b,c);

1

  %put %eval(a in d e f a b c);
0
```

New Macro Statements:

%ABORT: Stops the macro that is executing along with the current DATA step, SAS job, or SAS session (started in SAS 9.1)

Syntax :
%ABORT <ABEND | RETURN <n>> ;

%RETURN: Causes normal termination of the currently executing macro (started in SAS 9.1)

Comparison: The %ABORT and %RETURN statements are similar to the ABORT and RETURN statements found in the DATA step.

SAMPLE CODE:

In the sample code below, if the error variable is set to 1, then the macro will stop executing and the DATA step will not execute.

```
%macro checkit(error);
   %if &error = 1 %then %return;

    data a;
       x=1;
    run;

%mend checkit;

%checkit(0)
%checkit(1)
```

LOG:

```
29   %macro checkit(error);
30      %if &error = 1 %then %return;
31
32       data a;
33          x=1;
```

```
34       run;
35
36   %mend checkit;
37
38   %checkit(0)

NOTE: The data set WORK.A has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           1.01 seconds
      cpu time            0.12 seconds


39   %checkit(1)
```

For complete documentation on new 9.2 features, see the 9.2 OnlineDoc when it becomes available.

See next page for a quick reference of all these new features.

Here is a quick reference of all the new macro features with their definition and the release of SAS in which they were introduced:

New macro feature for SAS 6.12

%SYSFUNC -- macro function to execute SAS functions or user-written functions.

New macro features for SAS 8.2

%SYMDEL -- macro statement that deletes the specified variables(s) from the macro global symbol table.

&SYSMACRONAME -- automatic macro variable that returns the name of the currently executing macro.

&SYSPROCNAME -- automatic macro variable that contains the name of the procedure (or DATASTEP for DATA steps) currently being processed by the SAS Language Processor.

New features for SAS 9.0

CALL SYMPUTX -- DATA step call routine that assigns a value to a macro variable and removes both leading and trailing blanks, also allows you to select the symbol table where the macro variable will be placed.

SAS_EXECFILENAME -- environment variable for the Enhanced Editor that contains only the name of the submitted program or the catalog entry name.

SAS_EXECFILEPATH -- environment variable for the Enhanced Editor that contains the full path of the submitted program or catalog entry. The full path includes the folder and the filename.

&SYSNCPU -- automatic macro variable that contains the current number of processors available to SAS for computations.

MEXECNOTE -- system option that specifies whether to display macro execution information in the SAS log at macro invocation.

MCOMPILENOTE=< ALL | NOAUTOCALL | NONE > -- system option that issues a NOTE to the SAS log upon the completion of the compilation of a macro.

MPRINTNEST -- system option that allows the macro nesting information to be displayed in the MPRINT output in the SAS log.

MLOGICNEST -- system option that allows the macro nesting information to be displayed in the MLOGIC output in the SAS log.

MAUTOLOCDISPLAY -- system option that displays the source location of the autocall macros in the SAS log when the autocall macro is invoked.

New macro features for SAS 9.1

%SYMEXIST -- macro function that returns an indication as to whether the named macro variable exists.

%SYMGLOBL -- macro function that returns an indication as to whether the named macro variable is global in scope.

New macro features for SAS 9.1 continued

%SYMLOCAL -- macro function that returns an indication as to whether the named macro variable is local in scope.

SOURCE -- %MACRO statement option that combines and stores the source of the compiled macro with the compiled macro code as an entry in a SAS catalog in a permanent SAS data library.

%COPY -- macro statement that copies specified items from a SAS library.

%ABORT < ABEND | RETURN > < n > -- macro statement that stops the macro that is executing along with the current DATA step, SAS job, or SAS session.

%RETURN -- macro statement that causes normal termination of the currently executing macro.

New macro features for SAS 9.2

MINDELIMITER= -- system option that specifies the character to be used as the delimiter for the macro IN operator.

MINOPERATOR -- Recognize special infix operators when evaluating logical or integer expressions.

NOMINOPERATOR-- Do not recognize special infix operators when evaluating logical or integer expressions.

MINDELIMITER= -- %MACRO statement option that specifies a value that will override the value of the MINDELIMITER= global option.

&SYSERRORTEXT -- automatic macro variable that contains the text of the last error message formatted for display on the SAS log.

&SYSWARNINGTEXT -- automatic macro variable that contains the text of the last warning message formatted for display on the SAS log.

SECURE -- %MACRO statement option that enables you to write secure macros which will protect intellectual property contained in stored complied macros.

&SYSENCODING -- automatic macro variable that contains the name of the current session encoding.

MEXECSIZE -- system option that specifies the maximum macro size that can be executed in memory.

Table of Contents